NTNU – Trondheim
Norwegian University of
Science and Technology

Interoperability at the Application Layer in the Internet of Things

Hilde Marita Oen

Master of Science in Communication Technology
Submission date: June 2015
Supervisor: Frank Alexander Krämer, ITEM

Norwegian University of Science and Technology
Department of Telematics

Title: Interoperability at the Application Layer in the Internet of Things

Student: Hilde Marita Oen

Problem description:

Devices in the Internet of Things can interconnect using protocols such as Bluetooth and Wi-Fi, but that does not necessarily mean that they are able to understand each other. There exists no open standard language that enables devices from different vendors to communicate.

This thesis will investigate initiatives that work on top of IP such as, but not limited to, AllJoyn, which is managed by the AllSeen Alliance, and IoTivity, which is managed by the Open Interconnect Consortium (OIC). These initiatives are designed for seamless interconnection between different devices, and the thesis will emphasize on the ease of interoperability. Further, differences between the traditional Internet and the Internet of Things, and how they differ in terms of interoperability, will be discussed.

The goal of this thesis is to study, compare, and exemplify ongoing initiatives in the Internet of Things. The thesis should aim to conclude what are the best existing solutions for interoperability between different vendor devices. Further, possible improvements to these solutions should be discussed.

Responsible professor: Frank Alexander Kraemer, ITEM

Supervisor: Frank Alexander Kraemer, ITEM

Abstract

The Internet of Things consists of heterogeneous devices from a range of uncoordinated vendors, and this causes an interoperability problem. This thesis studies potential solutions to these challenges on the application layer of the Internet. Different approaches are studied, including, but not limited to: alliances, frameworks, consortia, open source projects, and closed projects.

The thesis looks at three different strategies: standardization, code generation, and extending the current web. These strategies are discussed regarding the requirements of the Internet of Things as well as future prospects. Next, the thesis presents four approaches on how to include semantics in order to achieve interoperability. The study reveals that not all approaches are able to solve the interoperability issue, but some can.

Based on the study, the thesis presents a tripartite division of the application layer. Protocols, data, and semantics are regarded separate problems that all require solutions in order to achieve an interoperable Internet of Things. This thesis does not provide a common language for the Internet of Things, but discusses the different requirements of such a language. The study provides guidelines on what need to be considered when developing an interoperable Internet of Things.

Sammendrag

Tingenes internett består av heterogene enheter fra en rekke, ukoordinerte leverandører og dette forårsaker et interoperabilitetsproblem. Denne avhandlingen studerer potensielle løsninger til disse utfordringene på applikasjonslaget i internett. Forskjellige tilnærminger studeres, inkludert, men ikke begrenset til: allianser, rammeverk, konsortier, åpen kildekodeprosjekter og lukkede prosjekter.

Avhandlingen ser på tre forskjellige strategier; standardisering, kodegenerering og utvidelse av den nåværende weben. Disse strategiene er drøftet med tanke på kriteriene til tingenes internett i tillegg til fremtidens prospekter. Deretter presenterer avhandlingen fire tilnærminger til hvordan semantikk kan inkluderes for å oppnå interoperabilitet. Under studiet kommer det frem at ikke alle tilnærmingene kan løse interoperabilitetsproblemet, men noen kan.

Basert på studiet presenterer avhandlingen en tredeling av applikasjonslaget. Protokoller, data og semantikk er ansett som separate problemer som alle krever løsninger for å kunne oppnå et samhandlende tingenes internett. Denne avhandlingen foreslår ikke et felles språk for tingenes internett, men drøfter ulike krav et slikt språk må oppfylle. Studiet foreslår ulike retningslinjer for hva som må vurderes når man utvikler et samhandlende tingenes internett.

Preface

This document serves as my master's thesis that marks the conclusion of a five-year study on communications technology at Norwegian University of Science and Technology (NTNU).

First I would like to thank my supervisor Frank Alexander Kraemer who guided me through my research and introduced me to the Internet of Things. I would also like to express my gratitude towards Martin Kirkholt Melhus for serving as discussion partner, spell checker, and moral support through the writing of this thesis. Finally I would like to thank the project contributors that have patiently answered my questions and provided me a better understanding for their projects: Klaus Birken from the Franca project, Alexander Edelmann and Olaf Weinmann from the Vorto project, and Matteo Collina from the Ponte project.

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

3G Third Generation.

4G Fourth Generation.

6LoWPAN IPv6 over Low power Wireless Personal Area Networks.

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

ARM Architectural Reference Model.

BLE Bluetooth Low Energy.

BSON Binary JSON.

CoAP Constrained Application Protocol.

CRC Cyclic Redundancy Check.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance.

DSL Domain Specific Language.

DTLS Datagram Transport Layer Security.

EDGE Enhanced Data rates for Global System for Mobile Communications (GSM) Evolution.

EMF Eclipse Modeling Framework.

ETSI European Telecommunications Standards Institute.

EU European Union.

EXI Efficient XML Interchange.

Gb/s Giga bit per second.

GPRS General Packet Radio Service.

GSM Global System for Mobile Communications.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

IBSG Internet Business Solutions Group.

ID Identifier.

IDL Interface Definition Language.

IEC International Electrotechnical Commission.

IEEE Institute of Electrical and Electronics Engineers.

IETF Internet Engineering Task Force.

IoE Internet of Everything.

IoT Internet of Things.

IoT-A IoT - Architecture.

IP Internet Protocol.

IPC Inter-Process Communication.

IPSO Internet Protocol for Smart Objects.

IPv4 IP version 4.

IPv6 IP version 6.

ISA International Society of Automation.

ISO International Organization for Standardization.

IT Information Technology.

ITU International Telecommunication Union.

JSON JavaScript Object Notation.

LAN Local Area Network.

LLN Low-powered and Lossy Network.

LOV Linked Open Vocabularies.

LoWPAN Low-power Wireless Personal Area Network.

LR-WPAN Low-Rate Wireless Personal Area Network.

LWM2M Lightweight M2M.

M2M Machine-to-Machine.

M3 Machine-to-Machine Measurement.

MAC Media Access Control.

Mb/s Mega bit per second.

MIB Management Information Base.

MTU Maximum Transmission Unit.

NAT Network Address Translation.

NFC Near Field Communication.

NTNU Norwegian University of Science and Technology.

OASIS Organization for the Advancement of Structured Information Standards.

OIC Open Interconnect Consortium.

OMA Open Mobile Alliance.

OMNA Open Mobile Naming Authority.

OWL Web Ontology Language.

PSM Protocol State Machine.

RDF Resource Description Framework.

REST Representational State Transfer.

RF Radio Frequency.

RFC Request for Comments.

RFID Radio-Frequency Identification.

SenML Sensor Markup Language.

S-LOR Sensor-based Linked Open Rules.

SMS Short Message Service.

SOAP Simple Object Access protocol.

SSN Semantic Sensor Network.

STAC Security Toolbox: Attack & Countermeasure.

SWoT Semantic Web of Things.

TCP Transmission Control Protocol.

TV Television.

UDP User Datagram Protocol.

UK United Kingdom.

URI Uniform Resource Identifier.

US United States.

W3C World Wide Web Consortium.

Wi-Fi Wireless Fidelity.

WLAN Wireless Local Area Network.

WoT Web of Things.

WPAN Wireless Personal Area Network.

WSD Web Services Description.

WSDL Web Services Description Language.

XML Extensible Markup Language.

XMPP Extensible Messaging and Presence Protocol.

Chapter 1
Introduction

In 2014 the Internet of Things (IoT) topped Gartner's hype cycle[1] [hypb], which suggest that it is on its 'peak of inflated expectations'. What is important for the IoT to succeed is to find the solutions that will help the field to mature and reach the 'plateau of productivity'. This chapter will define the IoT and give a brief history of the IoT up until today.

1.1 Internet of Things

1.1.1 Definition and Explanation

The Oxford dictionary [iotc] defines the term IoT as follows:

> The interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.

In other words we can say that the IoT is all everyday objects connected to the Internet that are able to communicate with each other. Examples of such objects can be lamps, washing machines, cars, industrial robots, food, clothes, cities, and the list goes on. Generally the IoT can be everything people surround themselves with.

The critical part is that the objects are able to communicate. Just being connected to the Internet does not necessarily make an object more desirable, but by being able to communicate with other objects, they have the possibility of making everyday life easier, more efficient, and cheaper. For instance, a coffee maker start brewing when the alarm clock goes off in the morning or a printer orders new ink when it is running low, it could even check where to buy the cheapest ink.

[1] A graphic representation of the maturity and adoption of technologies and applications, and how they are potentially relevant to solving real business problems and exploiting new opportunities. [hypa]

Other, similar terms that appear when studying the IoT are Web of Things (WoT) and Internet of Everything (IoE). These terms are harder to define as they are less used, but we can say that IoE is much the same as IoT in that it is a philosophy in which everything, objects, devices, people, animals, plants, etc., are connected to the Internet and able to communicate with each other. The WoT is a philosophy in which everything is integrated with the web. The WoT is considered as a subset of the IoT and the focus is on using well-known web standards such as Representational State Transfer (REST), Hypertext Transfer Protocol (HTTP) and Uniform Resource Identifiers (URIs) in IoT devices.

1.1.2 History

The term 'Internet of Things' first appeared in a presentation held by Kevin Ashton in 1999 [iotd], but there are many earlier examples of 'things' being connected to the Internet. For instance, in 1982, at Carnegie Mellon University, they had a Coke machine connected to the Internet [cok]. The machine was able to report on its inventory and whether newly loaded drinks where cool; they named it 'The "Only" Coke Machine on the Internet'. One can find many similar examples of computer engineers connecting coffee pots, toasters, and other everyday items to the Internet before the Internet went commercial in 1995. There have also been many presentations and articles before 1999 presenting the idea of things being connected and able to communicate, without it being called the Internet of things.

However, after 1999 the term has been adopted and the technology has evolved. In 1999 it was the idea that Radio-Frequency Identification (RFID) tags were the connection points between the Internet and the objects. In 2000 LG announced its plans of making an Internet refrigerator [lgf] that should be able to show prices and availability of groceries as well as give the consumers the possibility to use their refrigerator as a Television (TV), radio, videophone, and calendar among other things.

In 2003-2004 the term started to be mentioned in publications such as The Guardian [Dod03], Scientific American [GKC] and the Boston Globe [Wei04]. In 2005 the International Telecommunication Union (ITU) published the report 'The Internet of Things' as the 7th report in their series of reports on the Internet [ITU05]. Between 2006 and 2008 the EU recognized the IoT and the first European IoT conference was held [eui].

According to the Cisco Internet Business Solutions Group (IBSG), the Internet of things was 'born' between 2008 and 2009 as this was the breaking point where more objects or 'things' where connected to the Internet than people [Eva11].

In 2008 United States (US) made IoT one of the 6 'Disruptive Civil Technologies'

that may have an impact on the US interest out 2025 [Cou08]. In 2010 Google introduced a self-driving vehicle project. The same year Bluetooth released the Bluetooth Low Energy (BLE) that enabled applications in the fitness, health care, security, and home entertainment industries. In 2011 the IP version 6 (IPv6) was launched and made the IoT possible to realize.

After 2011 the IoT domain has grown; standards, organizations, platforms, projects, and alliances have appeared over the years, and some of these will be discussed in this thesis.

1.2 Motivation

The motivation behind this thesis is the need for a common language that enables all IoT devices to communicate. The current IoT is diversified and different vendors are using different standards. This is creating information silos and vendor lock-in. Consumers need different applications for every device they surround themselves with and none of the devices are able to talk together, they only talk to some server on the Internet. This is not making everyday life easier for the consumer, rather more complex. Silos may become the death of the IoT, and as Tan and Wang [TW10] state in their article: 'Only if we can solve the interoperability problem we can have a real the Internet of Things.'

1.3 Problem and Scope

The objective of this thesis is to find the road to one or more solutions to the interoperability issue in the hope that devices from different vendors in the future will be able to communicate. The thesis focus on how to achieve interoperability on the application layer of the Internet and investigates initiatives that work on top of Internet Protocol (IP) and are designed for seamless interconnection between different devices. This is done through studying ongoing projects, frameworks, organizations, alliances, and proposed standards for this field. These initiatives are then discussed in combination with different strategies the IoT can take. This involves looking at extending the existing web, creating a new standard and use code generators. What many of the initiatives lack are semantics and a common understanding of data, thus a chapter on different approaches on describing data is included. Semantics in the IoT can both be described through web services, metamodels, and ontologies, as well as other approaches that are not discussed in this thesis.

The result is a tripartite division of the application layer that enables considering data transfer protocols, data representation, and semantics on different layers. The thesis will show that there exist solutions enabling mapping between different protocols, thus proposing a solution where data representation and semantics are the two

aspects that need to be agreed upon when creating an interoperable IoT. The thesis aims to conclude what are the different requirements for a common language in the IoT and provide an insight in the ongoing interoperability initiatives. The thesis will not have one solution to the problem, but rather focus on considerations that need to be taken when trying to achieve interoperability.

Note that also security and privacy are big issues in the IoT, but those issues are outside the scope of this thesis.

1.4 Method

I have collected references and strived to find relevant scientific papers and approved standards where possible. As many of the projects are relatively new, not all have sufficient information available. In these cases, I also tried to directly contact project members in order to get a better understanding of their projects. I experimented with some of the projects' proposals. Initiatives were studied by looking at them from different perspectives as presented in Chapter 2. This included looking at the initiatives from the vendor's perspective as well as from the consumer and developer's perspectives.

1.5 Related Work

There are many projects with similar goals as this thesis. They all want everything in the IoT to be able to communicate across vendors. However, not all of them concentrate on the application layer.

The Internet Protocol for Smart Objects (IPSO) [ips] alliance is working on establishing the Internet Protocol as the basis for the connection between devices in the IoT. They also organize interoperability tests to show that devices that use IP for Smart Objects can work together and meet industry standards for communication. They are performing the groundwork for this thesis as they aim for IP to be the underlying protocol for all Smart objects.

European Telecommunications Standards Institute (ETSI) is working on making standards for Information and Communications Technologies and is recognized by the EU as a European Standards Organization [etsa]. They have a technical committee working on Machine-to-Machine (M2M) communications [etsb] that aims to provide an end-to-end view of M2M standardization. They are providing standards on all levels, also the application level of the IoT. This is highly relevant for this thesis as these standards are likely to be used by European vendors.

IoT - Architecture (IoT-A) is a project initiated by the EU and it has released a set of proposed guidelines for the architecture of the IoT. This is not a guarantee for interoperability in the IoT, but a tool to help achieve it.

The Allseen Alliance is a cross-industry effort that wants all objects to be connected in simple transparent ways to enable seamless sharing of information [alla]. AllSeen is managing the AllJoyn project, which is 'an open, universal, secure, and programmable software connectivity and services framework that enables companies and enterprises to create interoperable products that can discover, connect and interact directly with other AllJoyn-enabled products' [allb].

The Open Interconnect Consortium (OIC) [oic] was made as a competitor to the AllSeen Alliance and has a goal of defining connectivity requirements and ensuring interoperability of the devices that makes up the IoT. The OIC is also sponsoring the IoTivity [iota] project, which is delivering an open source reference implementation of the OIC standard specifications.

HyperCat [hypc] is a project that is aiming for interoperability in the IoT by using web annotations such as JavaScript Object Notation (JSON) and HTTP. It is designed to be easy to work with and is already used for creating smart cities in the United Kingdom (UK). They want applications to be able to discover and make sense of data automatically. This is a developed solution to the problem studied in this thesis and it is more about this solution in Chapter 4.

Eclipse is hosting many projects that are working in the IoT field. One of them is Vorto, which aims to enable a global standardization by creating a repository of IoT device meta information models and offer code generation to assist developers. Eclipse is also hosting the Ponte project, which wants to bridge the gap between application layer protocols such as Constrained Application Protocol (CoAP), MQTT and HTTP.

Machine-to-Machine Measurement (M3) is a framework to semantically annotate and interpret IoT data. This is designed to help developers develop for a Semantic Web of Things (SWoT) without needing to know semantic web technologies. The aim of the framework is to use ideas from the semantic web in order to create a SWoT and an interoperable IoT.

1.6 Outline

In the eight chapters that follow I discuss initiatives in the IoT and how these can be used in order to achieve interoperability. Chapter 2 present an example of an IoT device and three different view points for this device. This example is referred to through the whole thesis.

Chapter 3 looks at the Internet protocol stack and present protocols that are relevant to the IoT on the different layers. Chapter 4 present initiatives that are interesting regarding interoperability in the IoT and Chapter 5 present experiments conducted with four of these initiatives.

Chapter 6 present three different strategies for how the interoperability problem can be solved. This includes looking at which of these strategies the initiatives in Chapter 4 use. Chapter 7 presents four interoperability approaches that are more specific and on a higher level than the strategies. Chapter 8 is a discussion of what has been presented in Chapter 6 and Chapter 7. Chapter 9 contains concluding remarks.

Chapter 2
An Internet of Things Example

This chapter describes some personas that are used later in this thesis to explain how different applications and protocols work in order to better understand what advantages and disadvantages they bring into the IoT.

The first section describes an IoT device and its functionalities. The following section describes what different personas are expecting from this device.

2.1 The Device

The device is a smart thermostat that are learning the consumer's usage patterns and adapting to changes that happen in the environment it is placed. The thermostat should have the following functionalities:

- The thermostat should be able to be controlled through dedicated remote controllers and wall-mounted controllers as well as through smartphones, tablets or laptops.
- The thermostat should control temperature and humidity.
- The thermostat should announce to its users when it increase or decrease temperature due to changes it has noticed.
- The thermostat should learn the usage patterns of the consumer and be able to adjust to them.
- The thermostat should aim to keep a stable temperature in a room and notice if devices that may affect the temperature are turned off/on.
- The thermostat should notice if there are people present in a room.
- The thermostat should show how much time it is until a desired temperature is reached.

- The thermostat should aim to save power.
- The thermostat should guide the consumer to choose temperatures that save energy.
- The consumer should be able to set desired temperatures and for what time he wants them, also away from home.
- The consumer should be able to get an overview over energy use, temperature and humidity on a smartphone, tablet or laptop.

The thermostat is to be used in a home environment and a consumer can have many thermostats in a home, placed in different rooms and they should be able to communicate with each other.

2.2 Personas

2.2.1 The Vendor

The vendor wants to sell the thermostat and the main selling point is the energy savings.

The vendor does not care too much about which technologies that are used as long as it is not expensive and everything work as expected.

The vendor does not see interoperability with other vendors' devices as necessary. However, the vendor would like the thermostat to be able to communicate with devices he does not produce himself. Thus, the vendor agrees to use open technologies that can lead to interoperability.

The main goal of the vendor is to increase his sales.

2.2.2 The Developer

For the developer it is important that the technologies that are to be used are well documented and easy to understand. The developer is not afraid of trying new technologies, but know that new technologies require more development time as it is less documented and the developer has less experience. Thus, new technologies tend to be more expensive to use.

The developer is an open source supporter and believes that open source technologies are most useful.

The developer is concerned about the speed of the thermostat's communication, how often it is going to fail and what consequences that will have.

2.2.3 The Consumer

The consumer is the end-user of the thermostat and he wants it to work as soon as it is mounted on the wall. It is important for the consumer that the thermostat works as the specifications say and that it is easy to use it.

The consumer wants the thermostat to work perfectly at every moment and the down time should be little to non-existing. The consumer wants the thermostat to update itself without the need for human interaction.

The consumer wants the thermostat to work together with the rest of the IoT products he has as well as the ones he is going to buy in the future. The consumer does not want to perform manual configuration to make it all work together; this should happen automatically.

Chapter 3
The Internet Protocol Stack

The following chapter gives a brief explanation on how the Internet is built and provides some information about relevant protocols on the different layers. In the end of each section the protocols are summarized regarding why they are relevant for the IoT. This chapter also provides a reasoning for the assumption that everything has IP as an underlying protocol.

To avoid any confusion, a definition of the term protocol from [KR10] is provided:

> A protocol defines the format and the order of messages exchanged between two or more communicating entities, as well as the actions taken on the transmission and/or receipt of a message or other event.

3.1 Protocol Layers

In order to provide structure to the Internet, network designers organize protocols in layers. Figure 3.1 depicts the traditional five-layer Internet protocol stack. All protocols belong to one of the five layers and provide services to the layer above. Several protocols inside a layer can create their own stack and provide services within the layer. The services provided from one layer to another are called the service model of the layer. All layers (except the physical layer) are performing their services by using services provided by the layer below and performing actions within the layer.

3.2 Application Layer

The application layer is the uppermost layer and the layer, which is visible to the end-user; it is where the applications and their application-layer protocols reside. An application-layer protocol is distributed over multiple end systems, with the application in one end system using a protocol to exchange packets of information

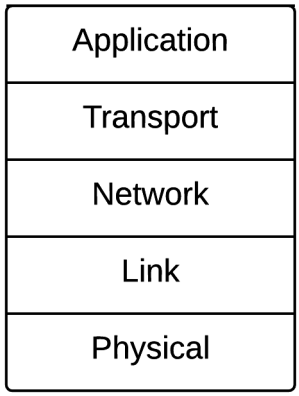

Figure 3.1: Five-layer Internet protocol stack.

with the application in another end system. In this thesis, the main focus is on application layer protocols and applications. In the following section three common application layer protocols used in the IoT are presented.

3.2.1 Hypertext Transfer Protocol (HTTP)

HTTP is the most widely adapted application layer protocol on the World Wide Web. The Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) [FGM+99] have done the standardization of HTTP. The abstract of the Request for Comments (RFC) says that:

> The HTTP is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

By using the REST pattern where every resource is globally identified by a URI, HTTP can be used to integrate different software applications. HTTP is a text-based protocol, which means that many data types are transferred in text format. HTTP uses Transmission Control Protocol (TCP) as the underlying transport layer protocol.

	Transport protocol	Messaging
HTTP	TCP	Request/response
CoAP	UDP	Request/response
MQTT	TCP	Publish/subscribe

Table 3.1: Summary of the application layer protocols.

3.2.2 Constrained Application Protocol (CoAP)

CoAP is a specialized web transfer protocol for use with constrained nodes and constrained networks in the Internet of Things [coa]. CoAP is specified in RFC 7252. See [SHB14] for a detailed description of CoAP.

As HTTP, CoAP is an implementation of the REST pattern. However, one important difference is that CoAP is by default implemented over User Datagram Protocol (UDP) and not TCP like HTTP. It is also binary and not text-based. CoAP was created in order to extend web technology to support small, constrained, and embedded devices on Low-powered and Lossy Networks (LLNs).

3.2.3 MQTT

MQTT is a publish/subscribe protocol with a central broker, designed to be lightweight, power efficient, and simple. It is based on TCP (like HTTP). MQTT is standardized by Organization for the Advancement of Structured Information Standards (OASIS) and the standard and full description of MQTT can be found in [mqt].

3.2.4 Summary

An overview of the protocols presented in this section can be found in Table 3.2.4.

There are also many more protocols, for instance Advanced Message Queuing Protocol (AMQP), Extensible Messaging and Presence Protocol (XMPP), Simple Object Access protocol (SOAP), etc., but they have not been widely adapted in the IoT.

As for the three protocols mentioned, it is not obvious which is better suited for the IoT. HTTP may be considered too extensive for many IoT devices because it is originally developed for the web. However, because it is developed for the web it could be favorable to use it in order to make the web and the IoT interoperable. Developers already know HTTP and development time would be short. At the same time, it is possible to use CoAP, which translates to HTTP. CoAP is generally very similar to HTTP so it is easy for web developers to start using it. However, both HTTP and CoAP are request/response patterns, which can become very complex

and impractical if many devices want to observe the same service. MQTT addresses the problem by being a many-to-many protocol that is a more efficient solution than the 'observe' mechanism implemented in CoAP.

Since MQTT is using TCP it is reliable and thus favorable to use in IoT environments where data is expensive and reliability is important. CoAP is using UDP and this can be a problem in Network Address Translation (NAT) environments, this can be solved with tunneling or port forwarding, but at the cost of added complexity.

CoAP and HTTP provide support for content negotiation and discovery, which allow devices to examine each other in order to find ways of exchanging data. MQTT does not provide any support for labeling messages and thus clients need to know the message formats beforehand in order to allow communication.

All three protocols have both advantages and disadvantages; which one to choose depends on the device and application requirements.

If we look at the thermostat from Chapter 2 it would want to subscribe to many sensors around the house such as thermometers, moist sensors, windows, doors, and ovens in order to have a good overview of the temperature situation in the house. This is making MQTT the most suitable protocol as it will not require that many one-to-one connections. The thermostat tells the broker which devices it wants to subscribe to and the broker is responsible for communicating the information. The thermostat will only have a few subscribers; probably some smartphones of those living in the house, there is no need for a broker to administer this so CoAP would be sufficient. Also the thermostat is fixed on the wall and most likely using the power-line and not a battery. Thus the thermostat could use HTTP without being drained of battery. This would also make it easy to make a web interface for it. But if the thermostat is only using HTTP it would be difficult for it to communicate with the other devices around the house that are smaller and battery driven and using CoAP or MQTT. Thus it is not a straightforward answer to what the thermostat should use, but because a house is small and does not (as of 2015) have thousands of sensors and devices, it would be sufficient to use CoAP.

3.3 Transport Layer

The transport layer carry application layer messages between application endpoints. There are two transport-layer protocols in the Internet; TCP and UDP. The transport layer offers the application layer services such as: reliability, congestion avoidance, flow control, multiplexing, and connection-oriented data stream support.

	TCP	UDP
Connection	Connection-oriented	Connectionless
Header size	20 bytes	8 bytes

Table 3.2: Summary of the transport layer protocols.

3.3.1 Transmission Control Protocol (TCP)

TCP provides a connection-oriented service to its applications that includes guaranteed delivery of application layer messages to the destination. TCP requires a three-way handshake in order to set up a connection. The advantages of TCP are obvious; it offers a reliable service, flow control, and multiplexing as well as other services.

3.3.2 User Datagram Protocol (UDP)

The UDP protocol provides a connectionless service to its applications. It does not offer a reliable service, so on lossy links, there is a high risk that not all packets reaches the receiver. However, UDP has a smaller header than TCP, which requires less computational power in the end-devices. UDP is also faster because there is no error checking for packets.

3.3.3 Summary

Table 3.2 shows a brief overview of the transport layer protocols; TCP and UDP.

Regarding the IoT it is not clear what transport protocol should be used. CoAP is using UDP while MQTT is using TCP. These are two application layer protocols that are used a lot in the IoT.

If we look at the thermostat from Chapter 2 we see that it is supposed to continuously control the temperature and have to send and receive messages with the temperature several times in the minute. For this task UDP would be sufficient, as it would not be critical if a temperature measurement was lost. However, the thermostat is also supposed to take orders from consumers and a consumer would not be satisfied if he had to set the temperature several times because the information got lost on the way. Thus, TCP could be a good solution. One problem with the IoT is that it often uses LLNs and would still like to have both a fast service and no loss.

TCP is the most adopted protocol in the Internet, except for real time services such as video and audio, and thus many developers tend to use it because that is what they are used to. However, as we have seen here, both transport layer protocols

clearly have both advantages and disadvantages and it really depends on the tasks the IoT device are going to do.

3.4 Network Layer: Internet Protocol (IP)

The network layer is responsible for moving network layer packets from one host to another. The Internet's network layer includes the IP.

3.4.1 Internet Protocol Overview

The main network layer protocol is IP and there are two versions of IP in use, IP version 4 (IPv4) and IPv6. IPv4 is most used, but Internet has grown bigger than first anticipated, causing a depletion of IPv4 addresses. An IPv4 address is 32-bit (or 4 bytes) long and on the form `a.b.c.d`. In 2011, IPv6 was launched providing 128-bit addresses, which should be sufficient to address every grain of sand in the world [KR10]. IPv6 addresses are written as eight groups of four hexadecimal digits separated by colons, for instance: `0000:1111:2222:3333:4444:5555:6666:7777`. In addition to addressing, IPv6 have fixed some of the problems of IPv4, including faster processing of the IP datagrams, a new definition of flow allowing video to be sent as a flow while for instance e-mail will not, hierarchical address allocation limits the expansion of routing tables; multicast addressing is expanded and simplified and thus provides additional optimization for the delivery of services. Also device mobility, security, and configuration aspects have been considered.

IPv6 addressed many problems in the Internet, but in the IoT, devices are getting smaller and having less computational power, thus making IPv6 too memory- and bandwidth-intensive for the IoT devices. Therefore IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) was created.

3.4.2 IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN)

Low-power Wireless Personal Area Networks (LoWPANs) comprise devices that conform to the Institute of Electrical and Electronics Engineers (IEEE) 802.15.4-2003 standard [802] by the IEEE (see Section 3.5.3). IEEE 802.15.4 devices are characterized by short range, low bit rate, low power, and low cost, similar to the devices in the IoT [KMS07].

6LoWPAN is an IPv6-based LoWPAN. The goal is to reduce packet overhead, bandwidth consumption, processing requirements, and power consumption. Since IPv6 requires support for packet sizes much larger than the largest IEEE 802.15.4 frame size (127 octets), an adaption layer between the link and network layers is

defined to enable efficient transmission of IPv6 datagrams over 802.15.4 links [HCC09]. This adaption layer provides three main services:

- Packet fragmentation and reassembly. Fragmentation is done when the payload size exceeds the Maximum Transmission Unit (MTU) size and cannot be carried within a single IEEE 802.15.4 frame. Breaking the link frame into multiple link fragments does this.
- Header compression. Header compression is done by assuming the use of common values on the network and link layer. This will avoid needless information duplication.
- The link layer uses link layer forwarding when multi-hop.

3.4.3 Internet Protocol in the IoT

This thesis assumes that everything is working over IP and this section present the advantages IoT devices can get from IP.

One big advantage with IP is that it already has proven to be long-lived and stable. The ability to evolve is important, but many IoT devices are designed to be long-lived, often up to ten years. Thus, it is important to know that the underlying protocols are stable and still available when the system is reaching the end of its life cycle. Because IP is the core of today's public Internet, it will continue to exist well into the future.

IP has also proven to be scalable and supports a range of applications, devices, and underlying technologies, something that is important in the IoT. The IoT is composed of a variety of link layer protocols and transmission mechanisms and because many devices have different properties, it is unlikely that they will share a single link layer protocol. By supporting a range of link layer technologies, IP is already providing interoperability between existing networks, applications and protocols [VD10].

The network layer does not care about the application layer applications. Thus, the network layer is just about sending a string of bits. As is stated in [VD10]:

> The network does not know if it is transporting a temperature reading from a temperature sensor, a piece of sound from a voice conversation, a control command, or a piece of a larger file. It only knows that it has been given a string of bits to transport from one end of the network to another. It is up to the applications running at the end points to make sense of the bits.

I will also mention that it is important with a small footprint as most IoT devices have low energy consumption, small physical size and low cost. IP is a heavyweight protocol and does not fit into these requirements, but by using lightweight implementations of IP such as 6LoWPAN this is not a problem.

3.5 Link Layer

The link layer is the layer that provides services to the network layer, what services it provides depends on the specific link layer protocol that is used. Examples of link layer protocols include Ethernet and Wireless Fidelity (Wi-Fi). As datagrams typically need to traverse several links to travel from source to destination, a datagram may be handled by different link layer protocols at different links along its route [KR10].

3.5.1 IEEE 802.3 (Ethernet)

Ethernet is the most common wired Local Area Network (LAN) technology. It is standardized in [IEE12a]. It can operate with speeds in a range from 1 Mega bit per second (Mb/s) to 100 Giga bit per second (Gb/s) and there is a different Ethernet version for each speed. Ethernet is using a common Media Access Control (MAC) specification and Management Information Base (MIB).

Ethernet is providing a connectionless and unreliable service to the network layer. Ethernet also offers a Cyclic Redundancy Check (CRC), thus bit errors in a transmission can be detected.

3.5.2 802.11 Standards (Wi-Fi)

IEEE 802.11 Wireless Local Area Network (WLAN) [IEE12b] is a group of standards developed by the IEEE and Wi-Fi is based on these standards. Wi-Fi is used in general as a synonym for WLAN since most WLANs are based on these standards and Wi-Fi is a very common wireless technology. As Ethernet, Wi-Fi is also a MAC specification. There are several 802.11 standards for WLAN technology, including 802.11a, 802.11b, and 802.11g. A number of dual-mode (802.11a/g) and tri-mode (802.11a/b/g) standards are also available and there is ongoing work on creating more [KR10]. Wi-Fi is a trademark of the Wi-Fi Alliance [wif] and the 'Wi-Fi CERTIFIED' trademark can only be used by products that successfully complete the Wi-Fi Alliance interoperability certification testing.

Wi-Fi uses a link-layer acknowledgement scheme in order to avoid failure. If the destination receives a frame that passes the CRC, it waits a short period and sends back an acknowledgement frame. Thus, Wi-Fi can offer a reliable service. Wi-Fi is

also using Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) in order to avoid collisions that may appear in the wireless network.

3.5.3 IEEE 802.15.4 (ZigBee, ISA100.11a, WirelessHart, MiWi)

IEEE 802.15.4 is a standard specified in [IEE11]. It specifies the physical layer and media access control for Low-Rate Wireless Personal Area Networks (LR-WPANs). Wireless Personal Area Networks (WPANs) are used to transfer information over relatively short distances. Unlike WLANs, connections done via WPANs involve little or no infrastructure. This feature allows small, power-efficient, inexpensive solutions to be implemented for a wide range of devices. The main objectives of an LR-WPAN such as described in [IEE11] are ease of installation, reliable data transfer, extremely low cost, and a reasonable battery life, while maintaining a simple and flexible protocol.

This standard is also the basis for specifications such as ZigBee, International Society of Automation (ISA) 100.11a, WirelessHART, and MiWi. All are wireless networking technologies designed for low data transmission rates and short distances.

3.5.4 IEEE 802.15.1 (Bluetooth)

IEEE 802.15.1 is standardized in [IEE05] and is more commonly known as Bluetooth. Bluetooth networks operate over a short range, at low power, and at low cost. 802.15.1 networks are sometimes referred to as WPANs. Bluetooth has with the latest version (4.0) data rates up to 24 Mb/s. 802.15.1 are ad hoc networks, i.e., no network infrastructure is needed to interconnect 802.15.1 devices.

3.5.5 Near Field Communication (NFC)

Near Field Communication (NFC) is a form of contactless communication between devices like smartphones or tablets. NFC is an offshoot of RFID with the exception that NFC is designed for use by devices within close proximity to each other. NFC maintains interoperability between different wireless communication methods like Bluetooth and other NFC standards [nfc]. NFC is standardized in International Organization for Standardization (ISO)/International Electrotechnical Commission (IEC) 18092 which can be found in [nfc13].

3.5.6 Z-Wave

The Z-Wave protocol is an interoperable, wireless, Radio Frequency (RF)-based communications technology designed specifically for control, monitoring, and status reading applications in residential (homes) and light commercial environments [zwa]. It is short range and is a MAC specification. It is standardized by ITU [zwa15].

	Wired/wireless	Physical layer	Data rates	Topology
Ethernet	Wired	Fiber, copper, etc.	1 Mbps - 100 Gbps	Bus or star
Wi-Fi	Wireless	RF	11 - 54 Mbps	Star
ZigBee	Wireless	RF	250 kbps	Star or mesh
Bluetooth	Wireless	RF	1 - 24 Mbps	Star
NFC	Wireless	RF	106 - 424 kbps	Point-to-point
Z-Wave	Wireless	RF	$<$ 100 kbps	Mesh
INSTEON	Wired / wireless	RF, power-line	13 - 38 kbps	Mesh
Cellular	Wireless	RF	$<$ 1 Gbps	Cellular

Table 3.3: Summary of the link layer protocols.

3.5.7 INSTEON

As Z-Wave, INSTEON also focuses its technology on home control. INSTEON is a cost-effective, dual-band network technology optimized for home management and control [ins]. INSTEON connects devices by using the power-line, RF, or both. All INSTEON devices are peers, meaning that any device can transmit, receive, or repeat other messages. Adding more devices makes an INSTEON network more robust.

3.5.8 Cellular

Today people connect to the Internet via their smart phones through Third Generation (3G) and Fourth Generation (4G) connections as well as older and slower connections such as Enhanced Data rates for GSM Evolution (EDGE) and General Packet Radio Service (GPRS). The technology behind these varies from generation to generation, but they have in common that they allow mobility of their users.

3.5.9 Summary

Table 3.3 shows a brief summary of the link layer protocols presented in this section.

Most of the protocols are wireless and using RF on the physical layer. This is a sign of well-proven technologies that will continue to exist into the future, which is an important thing for the IoT.

If we look at the example thermostat from Chapter 2, we can say that Ethernet is not suitable in this case. This is because it would require physical wiring through the wall to where the consumer wants his thermostat, or it would impose restrictions on where the consumer could place its thermostat since it requires a wire. Ethernet is offering very high data rates which is not needed for the thermostat that only will send a few bits with temperature measurements. However, Ethernet could be useful

in other areas of the IoT such as in industry environments where devices are made to last longer and it will not appear new devices without it being planned for.

Wi-Fi seems like a good solution for the thermostat. Most homes in the western world today already have Wi-Fi in the home thus should not require much effort to connect a thermostat to the network. However, this requires the thermostat to run on electrical power rather than batteries. The thermostat may be connected to the power-line, but for other home appliances such as remote controllers or alarm clocks, it might be a problem that Wi-Fi use too much power that will drain the batteries very quickly. Wi-Fi may also be a bit overkill to use as it offers very high data rates where that is not necessary. However, in time it may happen that we need higher data rates and the batteries will evolve and get better so Wi-Fi may be the future answer for the IoT.

ZigBee and Bluetooth would be good alternatives for the thermostat as they offer lower data rates and are specializing on low power networks such as in the IoT. However, a problem with both these protocols is that they are highly diversified. Both have several profiles and the developer have to know which profiles to use when developing the devices. This requires planning, and in order to have interoperability with other devices, the same profiles need to be used.

NFC is not a good solution for the thermostat. It would not allow the consumer to contact it from outside the home and even inside the home, it requires standing very close in order to get contact. However, NFC supports low data rates and is not draining devices of battery. Even if NFC does not work very well in a home environment it can be useful other places in the IoT. NFC is already being used for payment and NFC tags are stored in posters so that consumers can scan the tag and get the information on their phones. This can be done for everything that can have additional information. It can for instance be in clothes that can communicate with the washing machine what program it should use.

Z-Wave and INSTEON are specializing in home environments and could be good choices for the thermostat. However, these technologies are not that old and adopted as many of the other technologies that have been presented. This may lead to problems with implementation because the developers do not know the technology well enough. However, in order for the technologies to be able to evolve, they need someone to use it. Since these technologies are specializing on smart homes and smaller environments it may be difficult to adopt them to other areas of the IoT.

Cellular networks are good for moving things such as cars, pets or health monitoring devices. Cellular networks are usually developed and already available, but with IoT there will be many more devices online and the networks need to be further developed. In San Francisco there is even talk about making a cellular network only

for things [Sim14]. It may not be the best alternative for the thermostat since that will be mounted on the wall, but for moving things and especially things that move outside the confinements of a home this is a good solution.

As can be seen, it is not easy to find a solution on which technology to use in the IoT, it depends on the requirements of the IoT device. For the thermostat it seems like Wi-Fi may be the best choice, but also Bluetooth, ZigBee, Z-Wave or INSTEON could be used. For other areas in the IoT this may not be suitable at all.

3.6 Pysical Layer

The job of the physical layer is to move each individual bit from the link layer from one node to the next. The protocols in this layer are dependent on the transmission medium of the link. Examples of transmission mediums on the physical layer are twisted-pair copper wire and single-mode fiber optics. Many of the link layer protocols have different protocols for different physical-layer protocols. In each case, a bit is moved across the link in a different way [KR10].

Chapter 4
Interoperability Initiatives

This chapter presents some initiatives that either aim to solve the interoperability issue in the IoT or that have interesting ideas that can be part of such solution. How a solution can be reached will be further discussed in Chapter 8.

All initiatives presented are free to use and most of them are open source. The initiatives rely on top of the application layer protocols presented in Section 3.2 and they focus on how data should be structured, communicated, and used by IoT devices. Some of the initiatives also include semantics in order to give the data meaning. How the IoT can incorporate semantics will be further discussed in Chapter 7.

4.1 Alliances and Consortia

The following section presents two initiatives that have resulted in frameworks for vendors to create interoperable products.

4.1.1 AllSeen Alliance and AllJoyn

AllSeen Alliance is an open, cross-industry consortium that aims to make an open, universal development framework to support the IoE. They believe that no single company can accomplish the level of interoperability required and thus a cross-industry effort is needed [alla].

AllJoyn The framework the AllSeen Alliance is working on is based on the AllJoyn open source project and is expanded with contributions from both the alliance's members as well a the open source community. AllJoyn is a framework that enables companies and enterprises to create interoperable products that can discover, connect and interact directly with other AllJoyn-enabled products [allb]. Products, applications and services created with this framework can communicate over various network layers, regardless of manufacturer or operating system. It does not require

Internet access. This enables creation of products that can easily communicate and interact.

The framework has an open source codebase that ensures interoperability and contains various modular services that enable fundamental activities. The initial capabilities planned for the codebase include: device discovery to exchange information and configurations; onboarding to join the user's network of connected devices; user notifications; a common control panel for creating rich user experiences; and audio streaming for simultaneous playback on multiple speakers [alla].

4.1.2 Open Interconnect Consortium (OIC) and IoTivity

The OIC is, as AllSeen Alliance, a cross-industry consortium. They aim to promote an open source implementation to improve interoperability between the devices that are making up the IoT [oic].

IoTivity The framework that the OIC makes is called IoTivity and is based on industry standard technologies in order to wirelessly connect and manage the information flow between devices regardless of form factor, operating system or service provider [oic]. The framework is open source and the goal is to be compliant with OIC specifications and pass the OIC certification testing.

The framework has four essential building blocks: discovery, data transmission, data management, and device management. Discovery supports multiple discovery mechanisms, data transmission is based on a messaging and streaming model and supports information exchange and control, data management supports collection, storage, and analysis of data from various resources and device management supports configuration, provisioning and diagnostics of devices [iotb].

4.2 Data Serialization Frameworks

This section look at two data serialization frameworks; one open source and one closed but free solution. Data serialization frameworks provide a data structure that enables developers to define data once. This data can then be serialized to different programming languages, enabling developers to use the data in their preferred programming language.

4.2.1 Apache Thrift

Thift is the name of an open source software project with a goal of making reliable, high performance communication and data serialization across languages as efficient and seamless as possible [thr]. Thrift offers code generation that is used to define and create services for several programming languages in a simple and approachable

way. It conforms the most common idioms of most programming languages and is thus transparent. Language-specific features are placed in extensions instead of in the core library, making it consistent. Thrift is easy to use and at the same time it offers high performance. Thrift was first developed at Facebook, but made open source in 2007 and part of Apache Incubator in 2008.

4.2.2 Google Protocol Buffers

Protocol Buffers is similar to Apache Thrift, both offer serialization of structured data. A developer defines a data structure once and use generated source code to read and write the structured data to and from a variety of data streams and using a variety of languages [Pro]. Protocol Buffers aims to be small, simple and fast as well as language- and platform neutral. Protocol Buffers is serialized to a binary format that is compact and enables updating the data structure while still being compatible with deployed programs that use the 'old' format.

4.3 Interface Definition and Code Generation

This section presents initiatives that focus on creating common interfaces to be used across vendors. These projects can be compared to the data serialization frameworks as they offer their own way of structuring data and code generation. However, the code generation in this case is more industry specified and does not only concern programming languages, but also Interface Definition Languages (IDLs), data formats and domain specific code.

4.3.1 Eclipse Vorto

Vorto is a project with a goal of enabling a global standardization by creating a repository of IoT device meta information models and helping distribute code by offering code generation and an IoT tool set [vorc]. Both the tool set and the meta information model are based on the Eclipse Modeling Framework (EMF). EMF is, as the name states, a modeling framework and a code generation facility for building tools and other applications based on a structured model [EMF]. Figure 4.1 shows an overview over the Vorto project.

The meta information model describes how information models are structured. Function blocks are describing the capabilities of a device, exposing properties, operations and events. These function blocks can be re-used for devices sharing the same functionalities.

The tool set provides the developer with options for creating information models based on the described meta-model. It can either be done by using a graphical environment with drag-and-drop mechanisms, or by using a Domain Specific Language

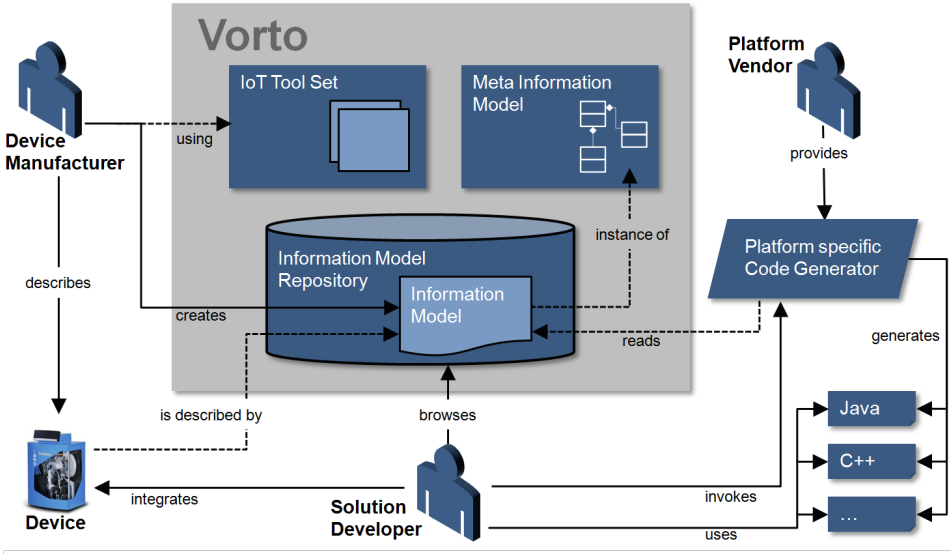

Figure 4.1: Overview over the Vorto project. Figure taken from the project's web site which can be found in [vora].

(DSL) that has been designed for creating the information models. This allows both developers with an Information Technology (IT) background as well as business users to use the tool set. The tool set also allows importing existing information models.

The information model repository is used as a centralized storage location for information models. During development the repository is allowing developers to access standardized information models and integrating them into applications.

The code generators allow developers to include described concepts into their applications. The tool set provides an extension point to include new code generators without effort. This allows for usage of the information models in various environments and device ecosystems. This part of Vorto is not intuitive to understand. In Figure 4.1 the code generators are not a part of Vorto and it is thus not clear what code generating Vorto does provide and how this can give more interoperability. Despite talking to representatives from Vorto I have not managed to get a proper understanding of what the generated code does and where it runs.

As in this thesis Vorto also concern consumer, developer and vendor. Vorto have the same goals for the different user groups as described in Chapter 2. The biggest

difference is that they differ between vendors of IoT devices and vendors of IoT platforms while these are treated as a single unit in this thesis. In Vorto's case, the consumer still wish to be able to buy devices from different vendors and flexibility and ease of use are important factors. The vendors of IoT platforms want to increase sales and number of ecosystems their devices can be integrated in. The vendors of IoT platforms also want to increase sales, but in addition they have something to gain by standardization and by being able to integrate as many devices as possible into their ecosystem. The application developers want to support a broad range of devices without a need to develop vendor specific code and thus increase sales and reduce complexity.

4.3.2 Eclipse Franca

Franca is not an IoT project, but it has interesting aspects that could be expanded to the IoT. Franca is a framework for definition and transformation of software interfaces and is underneath the modeling technology of Eclipse projects. It is intended for integrating software components from different suppliers, which are built based on various runtime frameworks, platforms and Inter-Process Communication (IPC) mechanisms [frab]. It is primarily aimed for the automotive and infotainment industries.

The problem Franca is trying to solve is mapping interfaces that use different IDLs as well as the lack of formality in modeling the dynamics of interfaces. Most IDLs only model static aspects of the interface. Franca wants to serve as a hub for IDL transformations and allow specification of dynamic behavior.

The Franca framework has five aspects: (1) An IDL and feature-rich editor, (2) transformations and generation, (3) specification of dynamic behavior, (4) flexible deployment models, and (5) rapid interface prototyping. These aspects make it possible for Franca to have a textual language for specification of Application Programming Interfaces (APIs), a framework for building transformations to/from other IDLs and model-based interface descriptions. Franca supports code generation with open-source code generators and specifies the dynamic behavior of client/server interactions using protocol state machines. Franca also makes it possible to extend interface specifications by platform- or target-specific information as well as having an executable test environment for an interface definition that can be generated instantly, allowing users to see the interface in action.

Franca can be useful for the IoT because it allows code to be translated and executed easily and if everything is using Franca or there is a Franca hub, everything can communicate even though the vendors use different languages. The biggest advantage of Franca is that the dynamic behavior can be specified. However, from

talking to projects contributors of Franca, currently it seems like, the interfaces creators have to talk together in order to get the interfaces to understand each other.

4.4 Standardization

This section concerns standardization initiatives in the IoT that includes guidelines and standards approved by the EU as well as a data management protocol standardized by the Open Mobile Alliance (OMA).

4.4.1 Internet of Things - Architecture (IoT-A)

IoT-A is a project initiated by the EU and is a proposed architectural reference model and a definition of an initial set of key building blocks. The idea is that together, they are the foundations for fostering the emerging of IoT. They have created an Architectural Reference Model (ARM) to use as the common ground for the IoT. The IoT ARM should provide a common structure and guidelines for dealing with core aspects of developing, using and analyzing IoT systems [BBF+13].

The IoT-A allows developers to make choices that make the architecture fit the device they are developing and are at the same time providing them with guidelines to follow. The IoT ARM does not guarantee interoperability between any two concrete architectures, but it is a tool in helping to achieve interoperability between IoT systems.

4.4.2 European Telecommunications Standards Institute (ETSI) Machine to Machine (M2M) Standards

ETSI M2M is a technical committee that is developing standards for M2M Communications [etsb]. They create standards on all layers and aim to provide an end-to-end view of M2M standardization. They publish technical reports and technical specifications that include studies and test specifications.

Especially two documents are relevant for this thesis. They are the technical specification on functional architecture and the technical report that studies semantic support for M2M data. The functional architecture is end-to-end and includes descriptions of the functional entities, the related reference points and the service capabilities, information model, security, and management, charging and implementation guidance. This is helpful for developers wanting to make their devices interoperable [ets13b]. The study on semantic support for M2M data is motivated by the fact that semantics need to be available for M2M data that is transferred within a M2M system [ets13a]. Through semantics applications can discover M2M data that they previously did not know about. They find the ability of applications

to discover, interpret and use M2M data from different sources essential for creating high-level M2M services and to develop open markets for M2M data.

4.4.3 Lightweight Machine to Machine (LWM2M)

Lightweight M2M (LWM2M) is a device management[1] protocol specialized to fit the requirements of the IoT. It considers factors as low bandwidth and lossy networks. LWM2M is not restricted to device management; it should also be able transfer service/application data. The protocol implements the interface between M2M device and M2M Server [lwmb]. LWM2M is standardized by OMA and it is based on CoAP and Datagram Transport Layer Security (DTLS) and is bound up to UDP and Short Message Service (SMS). LWM2M defines a simple object model by using the CoAP REST API, a device management architecture using REST objects, and system features including control of asynchronous notifications. There is also a project working on making LWM2M work over MQTT [LWMa].

4.5 Other Initiatives

This section presents other initiatives that focus on interoperability.

4.5.1 Eclipse Ponte

Ponte is a project that aims to make a bridge between CoAP, MQTT and HTTP. Ponte is defining a simple REST API to expose the machines' need through REST [pon]. Ponte should also be able to convert between different data formats such as JSON, MsgPack, Byson, Binary JSON (BSON) and Extensible Markup Language (XML) as well as offering security and authentication, but this is yet to be supported.

Ponte is replacing the broker in MQTT and making devices using either of the protocols MQTT, CoAP, or HTTP able to communicate with each other. Since most IoT devices are using CoAP or MQTT, Ponte is a step in the right direction in terms of intercommunication. Ponte provide the possibility to chose the best protocol for the device, while allowing communication with other devices over other protocols if desired. Another advantage is that web applications that already use HTTP can be used in the IoT.

If Ponte is combined with translation between different data formats, Ponte could be a good solution for the IoT. The developers would not only be able to select the most appropriate protocol, but also which data format that would be best. However, this requires Ponte to be able to do the translations efficiently so that the consumer do not notice any delay.

[1] Device management includes provisioning, configuration, software upgrades as well as fault management.

4.5.2 HyperCat

HyperCat is an open interoperability layer for the IoT that allows applications to explore what data and resources are available. All devices use the JSON data format and HyperCat help to find the right URIs for resources. HyperCat requires semantics in order to understand what resources are being requested and what the data represent. HyperCat is running over HTTP.

The idea is that all devices interact with a hub and a hub can communicate with other hubs. Each hub has a catalog that allows the other hubs to access, search, and understand the data it can provide. A catalog is a specific type of resource representing an unordered collection of resource items. Each item in a catalog refers to a single resource by its URI [HCs].

4.5.3 Machine-to-Machine Measurement (M3)

M3 is a framework to semantically annotate and interpret IoT data. M3 enables designing interoperable domain-specific or cross-domain SWoT applications. The M3 framework is helping IoT developers accomplish four tasks: (1) design SWoT applications, (2) semantically annotate IoT data, (3) interpret IoT data, and (4) secure IoT applications. The goal is to enable developers to create SWoT applications without needing to learn semantic web technologies as M3 automatically generates the code for these four tasks [Gyr15].

For each of the four tasks the framework have a tool for helping accomplishing the task. (1) A SWoT generator that enables designing SWoT applications to interpret IoT data, (2) a M3 converter that is used to semantically annotate IoT data, (3) Sensor-based Linked Open Rules (S-LOR) which is interpreting IoT data, and (4) Security Toolbox: Attack & Countermeasure (STAC) which is assisting developers in designing secure applications and architectures.

The aim of the M3 framework is to use ideas from the semantic web to create a semantic web of things and a interoperable IoT. The framework is compliant with ETSI M2M standards.

4.5.4 Google Weave

In the end of May 2015, Google announced that they are introducing a common language for IoT devices called Weave. Weave is created to achieve a shared understanding and make devices, humans, and smartphones talk to each other. It is based on standardized schemas that are used to describe devices' properties. Google is going to offer Weave certification in order to ensure quality and that all devices are interacting seamlessly. Weave is also cross-platform, meaning that a smartphone

running Brillo[2] can turn on an oven on another platform, like Samsung's SmartThings. Weave can be used both on existing software stacks as well as over Android stacks. Weave will not become available before the fourth quarter of 2015 [YTW, Bri].

4.6 Summary

This chapter has presented 13 different initiatives that can contribute to solving the interoperability issue in the IoT. Frameworks made by powerful members of consortia and alliances, data serialization frameworks, standards, projects focusing on interface definition and code generation have been mentioned along with projects and frameworks concerning protocols, data, and semantics.

The initiatives offer the IoT a way of structuring data. IoTivity and AllJoyn both offer APIs while the data serialization frameworks provide their own data formats. These frameworks also offer code generation in order to make the data formats used compatible with several programming languages. The interface definition and code generation projects offer their own data format and structured data can be used to create domain specific code or usual data formats such as XML or JSON. LWM2M uses XML to structure data while HyperCat is using JSON. M3 is using Sensor Markup Language (SenML) while Ponte is aiming to be able to translate between the most used data formats. There is not much information about Weave, but because Google makes it, we can guess they are using Protocol Buffers as their data format.

By structuring data in the IoT we are one step closer to interoperability. Teaching devices to work with structured data is easy, but we still need to decide on a common data format so that all devices are able to work with the same data. In addition, some semantics are required in order to understand the data. Vorto is solving this by using metamodels stored in a public repository while M3 is using ontologies to do the same. Weave offers semantics in form of standardized schemas. How the initiatives can be used to gain interoperability in the IoT will be further discussed in the following chapters.

[2]Brillo is an operating system based on the lower levels of Android made to suit the requirements of the IoT. Brillo is going to be released in the third quarter of 2015.

Chapter 5
Experiments

This chapter describes experiments done with four of the initiatives mentioned in the previous chapter. The goal of the experiments is to better understand the initiatives, and their capabilities.

The experiments have been trying to create the thermostat presented in Chapter 2. However, because of limited documentation and initiatives that are under development, the requirements of the thermostat were limited to being able to show the current temperature, humidity, if people are present in the room, and to set and display a target temperature.

5.1 LWM2M Experiment

The experiment was done with the LWM2M Object Editor Ver 1.2.01 [lwmc], which is a tool for creating, editing and viewing OMA LWM2M objects. An object can have several resources and can be downloaded and used to request Open Mobile Naming Authority (OMNA) to register the Object Identifier (ID) and Resource IDs. The IDs can be approved and listed in the OMNA website [OMA]. The IDs on this site can be re-used by other developers.

The tool offers public LWM2M objects that can be loaded into the tool and used as a template for creating an object. In this experiment the 'Device' object was chosen as a starting point for creating the thermostat object. The 'Device' already had many resources such as manufacturer, model number, serial number, etc. Three new resources were defined in this experiment: Temperature, Humidity, and People present.

Figure 5.1 shows the Object Viewer of the LWM2M Object Editor Ver 1.2.01 after making an ExampleThermostat. In order to make it fit into a single page some of the pre-defined resources that came with the public XML file were deleted and the descriptions were shortened. If one wish to register the resource IDs, one would

ExampleThermostat

Description

Self learning, energy saving thermostat.

Object definition

Name	Object ID	Instances	Mandatory	Object URN
ExampleThermostat	3	Single	Mandatory	TBD

Resource definitions

ID	Name	Operations	Instances	Mandatory	Type	Range or Enumeration	Units	Description
0	Manufacturer	R	Single	Optional	String			Human readable manufacturer name
1	Model Number	R	Single	Optional	String			A model identifier (manufacturer specified string)
2	Serial Number	R	Single	Optional	String			Serial Number
3	Firmware Version	R	Single	Optional	String			Current firmware version
4	Reboot	E	Single	Mandatory				Reboot the LWM2M Device to restore the Device from unexpected firmware failure.
5	Factory Reset	E	Single	Optional	String			
7	Power Source Voltage	R	Multiple	Optional	String		mV	Present voltage for each Available Power Sources Resource Instance.
8	Power Source Current	R	Multiple	Optional	Integer		mA	Present current for each Available Power Source
10	Memory Free	R	Single	Optional	String		KB	Estimated current available amount of storage.
13	Current Time	RW	Single	Optional	String			Current UNIX time of the LWM2M Client.
14	UTC Offset	RW	Single	Optional	String			Indicates the UTC offset.
15	Timezone	RW	Single	Optional	String			Indicates in which time zone the LWM2M Device is located.
17	Temperature	R	Single	Mandatory	Float		Celcius	Temperature in degrees Celcius.
18	Humidity	R	Single	Mandatory	Float			Indicates the humidity in the room.
19	People Present	R	Single	Mandatory	Boolean			Indicate if there is people present in the room.

Figure 5.1: Screenshot of the Object Viewer of the LWM2M Object Editor Ver 1.2.01 after making an ExampleThermostat.

need to pre-register which resource IDs one want, and then receive pre-allocated IDs from OMNA. The object with these new IDs would be admitted if OMNA did not find any issues and it would be displayed in the OMNA web page [OMA].

The documentation gives the impression that objects should be smaller than a thermostat. There exist objects for temperature-, humidity-, and presence sensors which, if put together, make up a thermostat's sensors. By providing objects as small as sensors, actuators, and tags, it allows vendors to build their products as a combination of these objects, instead of providing objects for complete products. It is easier for OMNA to create objects for sensor, actuators, and tags as these are the smallest components used and they do not need to consider all possible combinations of these.

From the experiment we can conclude that the LWM2M Object Editor allows a vendor or developer to:

- Create an OMA LWM2M object.
  - Use pre-defined IDs for objects and/or resources.
  - Define their own object and/or resource IDs.
- Download the created LWM2M object as an XML file.
- Request OMNA to register the used IDs.

5.2 Eclipse Vorto Experiment

The experiment was done with the initial code contribution of Vorto, which can be found in [Vorb]. This version allows the user to create function blocks that are used to describe the functionalities of a device. These function blocks can also be used to generate a web application.

Listing 5.1 shows how the `.fbmodel` file that makes up the Thermostat function block. Figure 5.2 shows how the Hypertext Markup Language (HTML) file Vorto generated from this function block looked. All values are written directly in the HTML file and the buttons do not work.

With the available version of Vorto, vendors and developers can:

- Create a function block that allows them to:
  - Define configuration details.
  - Define status details.
  - Define fault details.
  - Define operations details.
- Auto generate code for a Web Device Application

Notice that status and fault details are not included in the available documentation and is the reason why the thermostat from the experiment is lacking such details. This is also far from all the functionalities Vorto are planning to offer. Having a public repository for these function blocks and being able to re-use them would be a step towards a solution to the interoperability issue in the IoT.

```
1  functionblock Thermostat {
2    displayname "Thermostat"
3    description "Thermostats␣keeps␣the␣home␣varm"
4    vendor www.bosch.com
5    category demo
6    version 1.0.0
7
8    configuration{
9      mandatory temperature as double "temperature␣in␣the␣room"
10     mandatory targetTemp as double "target␣temperature␣in␣the␣room"
11     mandatory humidity as double "humidity␣in␣the␣room"
12     mandatory peoplePresent as boolean "true␣if␣there␣is␣people␣
           present␣in␣the␣room"
13   }
14
15   status{ }
16
17   fault{ }
18
19   operations{
20     setTemperature(targetTemp as int)
21     getTemperature() returns int
22     getHumidity() returns int
23     areTherePeopleInTheRoom() returns boolean
24   }
25 }
```

Listing 5.1: `.fbmodel` file defining a function block for the thermostat.

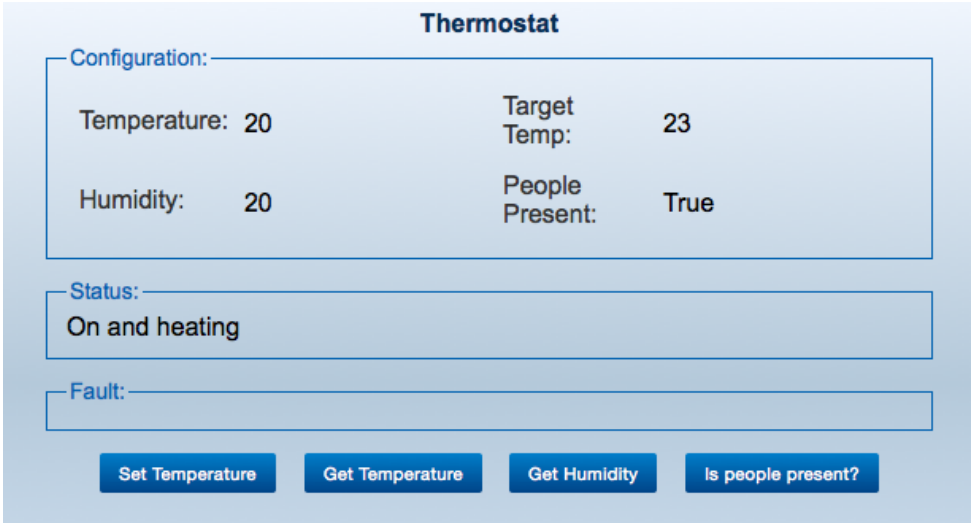

Figure 5.2: Screenshot of the web application HTML file after making the Thermostat and generating the web application.

5.3 Eclipse Franca Experiment

The experiment was done with Franca version 0.9.1, which can be found in [fraa]. This version allows definition of interfaces consisting of attributes, methods, and broadcasts in addition to the possibility to define the dynamic behavior of the interface. It should also be possible to build generators for creating code and documentation from the interfaces.

The interface in the experiment was defined in a `.fidl` file which can be seen in Listing 5.2. The contract part is what is defining the dynamic behavior of the thermostat by using a Protocol State Machine (PSM). The idea here is to create a working and an idle state where the working state means that the thermostat is heating or cooling while in the idle state, the `targetTemperature` and the `currentTemperature` is the same.

Franca allows vendors and developers to:

- Define an interface. This involves the capability to:
  - Define attributes that are properties on the provider side defined with type and name.
  - Define methods that can be called by one of the clients using the interface; the server will send the response.
  - Define broadcasts that are sent by the server and will be received by the clients using the interface.
  - Define contracts that describe the dynamic behavior of the interface.
- Build generators for creating code and documentation from the interfaces.

Notice that building generators for creating code and documentation from the interfaces has not been tested in this experiment.

5.4 Google Protocol Buffers Experiment

The experiment was done with Google Protocol Buffers version 2.6.1, which can be found in [GPB]. With this version it is possible to define a data structure, generate code from it and then read and write data to and from a variety of data streams using a variety of languages. Java, C++, and Python can be used.

The data format for the thermostat was defined by making a `.proto` file that can be seen in Listing 5.3. After having defined a data structure the protocol buffer compiler for Python was run and it generated a data access class. Then this class

```
interface Thermostat {
  version {
    major 0
    minor 1
  }
  attribute Double currentTemperature readonly
  attribute Double targetTemperature
  attribute Double humidity readonly
  attribute Boolean peoplePresent

  method getCurrentTemperature {
    out {
      Double currentTemperature
    }
  }
  method setTargetTemperature {
    in {
      Double targetTemperature
    }
  }
  method getHumidity {
    out{
      Double humidity
    }
  }
  method checkPeoplePresence {
    out {
      Boolean peoplePresent
    }
  }
  broadcast targetTemperatureReached {
    out {
      Double currentTemperature
    }
  }

  contract {
    PSM {
      initial idle
      state idle {
        on call setTargetTemperature -> working
      }
      state working {
        on signal targetTemperatureReached -> idle
      }
    }
  }
}
```

Listing 5.2: `.fidl` file defining an interface for the thermostat.

```
message Thermostat {
  enum Degree {
    CELCIUS = 0;
    FARENHEIT = 1;
    KELVIN = 2;
  }

  message Temperature {
    required double temp = 1;
    required Degree deg = 2 [default = CELCIUS];
  }

  required Temperature currentTemperature = 1;
  optional Temperature targetTemperature = 2;
  required double humidity = 3;
  required bool peoplePresent = 4;
}
```

Listing 5.3: `.proto` file made to represent the thermostat.

was used as the data format when creating a small program that made it possible to change a target temperature. The program was very basic, but showed that it is possible to use the `.proto` file to represent data.

From this we can see that Protocol Buffers offers vendors and developers the possibility to:

- Create a data structure that allows hierarchically structuring.
  - Create own Protocol Buffers message types.
- Generate data access classes for Java, C++, and Python.

Notice that in this experiment it was only made one data access class for Python.

5.5 Summary

What can be seen from these experiments is that it is easy to define data. All the tested initiatives allow defining a data structure for the thermostat and it looks similar in three of them. The object created with the LWM2M is a bit more complex than the rest due to use of XML and being more focused on creating objects than creating a readable interface. However, the web interface the editor uses is easy to understand and make it more usable for non-programmers. Franca allows its users to define dynamic behavior, which is unique for Franca. If we could agree on one way of defining interfaces, everyone could make devices that use compatible interfaces.

Chapter 6
Strategies

This chapter describes three strategies that can be used to solve the interoperability challenges in the IoT. These strategies are used in some of the initiatives presented in Chapter 4.

6.1 Creating a Common Standard

ETSI [Std] provides the following definition of a standard:

> In simple terms, a standard is a document that provides rules or guidelines to achieve order in a given context.

A new standard that fit the IoT and makes it more interoperable would thus require a document that can provide rules and guidelines on how to make IoT devices. It does exist standards for the IoT and M2M. We saw in Section 4.4.2 that ETSI have made several M2M standards and the EU has made a book about the IoT-A and guidelines on how to implement an architecture to apply in the IoT [BBF+13]. Still, this has not made the IoT more interoperable, as it lacks a common standard that everyone agrees to use. The existing standards tend to be too wide or too narrow and not suitable to what the vendors are trying to make.

A common standard would be favorable, as it would provide the IoT with order and certainty. Some argue that this can be achieved by creating an open standard where everyone can contribute. However, one of the problems that often appear with open source work is that they are poorly documented and often favors the developers instead of the consumers. Thus, a standardization organization would be the best suited to create a standard. These organizations have experience and are trusted. Trust is important for a standard as few would use a standard from a unknown organization where there are uncertainties regarding testing and documentation.

A problem that will appear if a standard strategy is adapted in the IoT is that it should be forwards-compatible, meaning that for devices and technologies that appear in near and distant future, it should be easy to add them to the standard: after they have been added, the standard should be backwards-compatible. This is a problem with any strategy, but in this specific case there are no dynamic updates as changes are all documented in a standards document and as the standard is updated, devices are using different versions of the standard.

In an ideal world, we would be able to create a standard that covers every aspect of the IoT and is trusted and used by everyone. However, it is more likely that there will be several smaller standards that can be combined in order to create desired IoT devices. This introduces the problem of making these smaller standards interoperable.

The standards strategy can help solve the interoperability issue only if everyone agree on which standards to use and where to use them.

6.2 Code Generation

What is meant by code generation is a program that enables developers or vendors to structure data in one way and then is able to compile it to different (programming) languages. By doing this, the developers do not need to agree on one language to use. The vendor can specify the structure of the data and then developers can use generated source code (the code generator) to easily read and write structured data to and from a variety of data streams and using a variety of languages.

Code generation can be done with software frameworks such as Apache Thrift or Google Protocol Buffers, as well as Eclipse Vorto and Eclipse Franca. They allow developers to specify a schema for their data using a specification language. By doing this, developers only have to specify semantics for objects once, this will eliminate the problem of inconsistency that may happen if things need to be specified several times in different formats.

One advantage of using this strategy is that if the data structure need to be updated it can be changed once, and then recompiled for all applications where it is used. The creators of Protocol Buffers also argue that their solution is forwards- and backwards-compatible. Another advantage is that even though applications may be written in different languages, they have the same data structure and are able to understand each other.

The problems that arise when using code generators are that the code may not be as efficient as it could have been if it was written natively or that compiling requires more power and thus devices live for a shorter time.

In order for the code generation to help solve the interoperability issue the data structures that are used need to be available so that everyone can use them. If the data structures are not available, it will not solve any problems, as only 'insiders' will be able to understand the data. However, if they are publicly available, different vendors can build devices that are able to understand the same data formats. i.e. become interoperable.

6.3 Extend the Web

A third potential strategy is to extend the existing web. Since the IoT devices will also be connected to the Internet, why do there need to be a difference between the web and the IoT? This is the idea behind the WoT. The web has been developed over many years and offers a range of standards, protocols, and solutions to the IoT. However, the web is also wide in the sense that it includes many complex solutions that may not be suitable for the IoT. In [BBF+13] they say that:

> ... It was noted that most people use the same concepts when discussing IoT as when discussing the Internet in general. There is a significant difference, however. IoT involves objects talking to each other without user consent, with possibly un-envisaged functionalities. Cameras, for example, might take on functions that are different from their overt primary functions. These possibilities, once perceived, may cause user anxieties to rise. Moreover, what is the role of user consent if objects may be able to talk to each other spontaneously? It will be very difficult to backtrack after the deployment of million of chips employing a passive approach to connectivity. ...

In other words: on the web, people are giving orders and giving consent before things happen, in the IoT devices are talking to each other and sometimes this happens without the consumer's consent. Thus, it is not necessarily a good idea to extend the web to include the IoT.

The advantage of extending the web to include the IoT is that many devices in the IoT will also have a web interface. By extending the web this will be trivial to realize, but other applications in the IoT may be more difficult to make. The web could be extended to include IoT devices, but most likely that would also lead to increased complexity and make implementations harder for developers.

The main argument against using web technologies in the IoT is that they have too much overhead to fit the requirements of the constrained devices in the IoT. However, it is likely that devices in the future will be capable of handle these complexities. The

	Pros	Cons
New standard	Done before Available for everyone Tested and documented	Requires trust
Code Generation	Write once, share, and re-use Many choices Backwards-compatible	Less efficient (?) Use more power (?) Proprietary data structure
Extend the Web	Done before Easy for developers	IoT and the web are different More complexity Less efficient (?)

Table 6.1: Pros and cons of the different IoT interoperability strategies.

technologies are moving forward and even though the batteries as of today cannot handle web technologies, batteries in the future will be smaller and last longer and thus feasible. This is not helping solving the problem today, but we should be aware that the IoT may adapt to technologies as complex as web technologies in the future.

In order for the web strategy to help solve the interoperability problem, the web need to include more lightweight standards and protocols that are interoperable with existing standards and protocols.

6.4 Summary

Table 6.1 provides a brief summary of the three strategies presented in this chapter, and Table 6.2 shows which strategies are used by the initiatives presented in Chapter 4. We can see that many of the initiatives do not use any of these strategies and that code generation and standards are the most popular of the ones presented here. This table is interesting as it shows that all the presented strategies are used, both alone and in combination. It can also be noticed that there are only two initiatives that are not using any of the three, presented strategies.

Since all strategies are used in initiatives in the IoT we cannot conclude that one is better than the other. It can even turn out to be a combination of all that is the best solution in the end.

The Internet and the web is built on standards, from standards for the link layer protocols to standards for web design. Thus, one could think that extending the web by adding additional standards would be sufficient for the IoT. In this way we can use two strategies: both making a new standard and extending the web. However, adding more standards to the web may only make it more complex.

Initiative	Standard	Code generation	WoT	Other
AllJoyn				x
IoTivity	x			x
Apache Thrift		x		
Google Protocol Buffers		x		
Eclipse Vorto		x		x
Eclipse Franca		x		x
IoT-A	x			
ETSI M2M	x			
LWM2M	x			
Eclipse Ponte		x	x	x
HyperCat			x	
M3	x		x	
Weave				x

Table 6.2: What strategies the different IoT initiatives are using. IoT-A is not providing a standard, but is guidelines from the EU.

Both standards and code generation are popular among the initiatives, but never used together. This indicates that there are no standards in the code generation area. This again is an indication that everyone working on code generation do it their own way. Maybe the next step should be making standards that could be used by projects that include code generation. One of the problems with the code generation is that all of them have their own way of structuring data. This makes it difficult to achieve interoperability between different code generators. This is something Franca is working on, to provide their own IDL, while at the same time being able to build transformations between other IDLs. This could be a good solution to the code generation problem if different vendors want to use different code generators. By combining standards and code generators we could be looking at the best strategy for the IoT.

If we look at the thermostat from Chapter 2 it is no obvious strategy that is the best one. The thermostat needs a web interface to show overview of energy use, temperature, etc. This could lead us to think that extending the web to be able to talk to the thermostat is the best strategy. However, since HTTP was not the best protocol for it, heavier applications on top of that will not make it better. Using a standard is usually a safe choice if it is well tested and documented, but this requires that it have been used for a while. Thus, the developer would likely prefer to use code generation if it were to be implemented today. With code generation the

data only need to be defined once and the developer can chose desired language to implement the applications with. This allows fast development, which is cheaper for the vendor and in the end also cheaper for the consumer. If we also look at future costs we can see that code generation makes it easy to make compatible devices that understand the same data model, also in the future. If the data format is developed for the thermostat, this can be re-used by other devices that want to communicate with it. A standard may not make the future development cost lower as the standard not necessarily enable re-use. However, even if the code generation strategy is the best here, the standards strategy may be just as good if the standard is tested and documented.

Regardless which of these strategies get adopted, they all need some semantics in order to solve the interoperability issue. How this can be achieved will be further discussed in Chapter 7.

Chapter 7
Interoperability Approaches

This chapter presents four approaches on how to achieve interoperability in the IoT. These should be considered in addition to the strategies discussed in the previous chapter.

The approaches concentrate on semantics needed in the IoT. As mentioned before, semantics are necessary in order to understand the meaning of data. According to Euzenat [ES07], semantics "provides the rules for interpreting the syntax which do not provide the meaning directly but constrains the possible interpretations of what is declared." In other words, semantics gives the machines rules so that they can understand the data in a given data format.

7.1 Everything JSON

This approach aims to find out how the IoT will look if everything uses the data format JSON. The reason for looking at JSON is that it is more lightweight than for instance XML. It is also easy for humans to understand and thus understandable for the developers unlike for instance BSON or Efficient XML Interchange (EXI) which are binary data formats. The advantage of binary formats is that they are even more lightweight than JSON, but because they are difficult for humans to understand they are also difficult to debug.

JSON is a text format for serialization of structured data. It is derived from the object literals of JavaScript and information about it can be found in [Cro06]. JSON can represent four primitive types; strings, numbers, Booleans, and null, and two structured types; objects and arrays. A string is a sequence of zero or more Unicode characters. An object is an unordered collection of zero or more name/value pairs, where a name is a string and a value is a string, number, Boolean, null, object, or array. An array is an ordered sequence of zero or more values. The terms 'object' and 'array' come from the conventions of JavaScript. JSON's design goals were for it to be minimal, portable, textual, and a subset of JavaScript.

One of JSON's limitations is that it only has four primitive types and two structured types. This is a help to create simple data formats, but it can be a problem in the IoT if machines are going to be to be self learning and understand the data format. Objects and arrays can be used to make more complex types, but without common, agreed upon data types it is likely that everyone using JSON does it differently. Another limitation JSON has is that it has to conform the same naming conventions regarding use of language specific keywords and other rules as JavaScript. JavaScript is used in the web, but is not necessarily the best for use in IoT devices. However, by having the same conventions it is easier to translate between the web and the IoT.

One advantage of using JSON is that there are no versions and no need for validation. By having no versioning there is no backwards-compatibility problems and there should not be any problem with updating devices. Regarding validation it can be a problem as different vendors are using different conventions, but if there exists common guidelines this should not be a problem. Regarding simplicity it is good that JSON is not extensible. This means that no organization can come in and make own definitions. Everyone needs to use the four primitive types and the two structured types. This is also an advantage regarding backwards-compatibility as if there is added extra information the ones that do not understand it can safely ignore it. Another advantage with JSON is, as mentioned earlier, that it is easy to understand both for humans and for machines. This is helpful for developers as well as for the machines that need to read it.

One possible solution for IoT that include JSON is HyperCat. In order to use JSON the developer need to read and understand the documentation and then implement different things for all applications and services. This is difficult to make work between different vendors. HyperCat has a solution where an application asks another application what services it offers and then it can use the ones it understands. For instance can the application understand temperature, but cannot know what the URI in the other application is to get temperature. Therefore it asks for what it understands and gets back the URI so it can fetch the temperatures. This is a beginning for a good solution for the IoT where everything uses JSON, but it is still necessary with some common agreement about semantics.

If everything uses JSON, we still need a common way of making the devices understand data. Data may have a format of JSON, but that does not mean that the devices understand what data is. Thus, JSON as a common way of data structuring, is not sufficient to solve the interoperability issue, we still need more.

7.2 Web Services

W3C [W3Wb] defines a web service as:

> A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language (WSDL)). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

The definition says that web services are designed to support interoperable M2M interaction, which makes this highly relevant for this thesis. However, web services are designed for the web and not the IoT which, include a level of complexity and protocol overhead that can create problems in the IoT.

In order for message exchange to work, the two devices that want to communicate need to agree both on the semantics and on the mechanics of the message exchange. The mechanics are documented in a Web Services Description (WSD), which is a machine-processable specification of the web service's interface, written in WSDL. This WSD defines the message formats, data types, transport protocols, and transport serialization formats that should be used between the communicating devices. It can also contain information on expected message exchange patterns. While the service description represents a contract that concerns the mechanics of interacting with a particular service, the semantics represents a contract governing the meaning and purpose of that interaction. The dividing line between these two is not necessarily definite. As more semantically rich languages are used to describe the mechanics of the interaction, more of the essential information may migrate from the informal semantics to the service description. As this transformation occurs, more of the work required to achieve successful interaction can be automated [W3Wa].

From this is seems like web services are solving the interoperability problem by making devices agree on the data they are sending and on message exchange. However, there is one limitation. Web services are using SOAP, which is a heavy and complex protocol, using XML, which, as previously mentioned, is not suited for the IoT. However, the web services can possibly be adapted to fit the requirements of the IoT in the future. By using CoAP instead of HTTP and JSON instead of XML some of the immediate problems could be solved.

Another limitation is that someone need define the WSD and the semantics so that the devices can agree on common data formats and other mechanics. If all

vendors only implement a single data format, it has to be the same for all vendors in order for all devices to be able to communicate. Hence, someone has to create rules for these mechanics and semantics.

Even though web services in the traditional sense is too complex for the IoT, the idea of using a WSD for the devices to negotiate and agree on the communication can be applied in the IoT.

7.3 Metamodel

A metamodel is a model that explains a set of related models and is usually a simplified version of the original models. In this case the metamodel would be used as part of a language to express resources in a device. Metamodels are related to ontologies (see Section 7.4) as both are often used to describe and analyze the relations between concepts. Metamodels can be seen as a strict rule set while ontologies are vocabularies.

From the initiatives outlined Chapter 4, it can be seen that Eclipse Vorto and Eclipse Franca mention using metamodels in order to achieve interoperability.

Eclipse also has a group working on an M2M application metamodel [EMe] and they emphasize that such a model need to be able to model configuration capabilities as well as communication capabilities. This includes description of the variables and events, and the actions that can be invoked on the system. There should also be descriptions of the different communication channels a given element of a system exposes to others, or depends on for communication.

Also the IoT-A project mentions applying metamodel-based architectural styles to achieve interoperability on the architectural level.

In order for the concept of metamodeling to work for the IoT someone need to create the models and thus decide what is important to include. Vorto is solving this by letting all vendors create their own model for each device. These models are then stored in a repository. The problem with this is that all vendors are making their own model. This leads to hundreds of different models for the same devices and developers have to implement applications specifically for each device. However, with a metamodel for each device type instead of all vendors having their own, this could work. With one metamodel for each device type, vendors could still add additional functionalities offered by their devices and at the same time being able to communicate with other devices for the common functionalities. However, this would require someone to make such metamodels for different device types.

Metamodels can help solve the interoperability issue, but as with all other initiatives someone need to organize it.

7.4 Ontology

The last approach explored is to use ontologies in the IoT in order to agree on semantics. In 1992, Tom Gruber defined an ontology as [Gru92]:

> An ontology is a formal specification of a shared conceptualization.

An ontology consist of four components: classes, relations, attributes and individuals. Classes are describing concepts; they can have one or more children, which are used to define more specific concepts. They have attributes and these can represent the classes' properties and characteristics. Individuals are instances of classes or their properties while relations are the edges that connect all the components.

Web Ontology Language (OWL) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things and relations between things. The language is a computational logic-based language such that the knowledge expressed with OWL can be understood and used by computer programs. OWL documents are what is known as ontologies [OWL]. One such ontology is the Semantic Sensor Network (SSN) ontology [SSN]. This ontology is relevant for the IoT as it describes sensors and sensor observations, and related concepts. It does not cover every thinkable aspect of the IoT, but what it does not cover can be included from other ontologies via OWL imports. However, it is not necessarily intuitive to know which ontologies to include.

The advantage of ontologies is that they can be expanded to include new devices and made backwards-compatible. Ontologies would not only allow devices from different vendors to communicate, but would also make it easier for third party developers to develop applications for IoT devices.

One of the problems the semantic web has been facing is that there are too many ontologies that contains synonymous classes. This makes it necessary to have a mapping between the ontologies. This is likely to be a problem in the IoT as well. One could argue that making a single ontology that covers everything would be ideal. The problem with making this big ontology is that it may be too abstract and hard to understand, as it is difficult to define everything. Some argue that one should rather make several, simple ontologies that cover very small areas. This could be a solution, but then remains the problem of finding the most suitable ontologies and make everyone use the same ontologies for the same kind of devices. If everyone use

different ontologies for the same types of devices, it will not help on interoperability since the devices will not necessarily know that they are dealing with equal data. Thus we need some mapping service or a standardization organ that can create guidelines for the ontologies.

The M3 framework use ontologies to describe data in the IoT and they have looked at over 270 ontology based projects relevant for the IoT and used them to create a dataset. They emphasize the need for publishing domain knowledge online so that everyone can find it, have good documentation and labels in English to enable easy re-use. The ontologies should also follow semantic web best practices and preferably be in the Linked Open Vocabularies (LOV) catalogue that references more than 400 well-designed ontologies [Gyr15]. These are definitely considerations that need to be taken when designing ontologies. Since there already exists hundreds of ontologies it is unlikely that the most feasible solution is a single big ontology.

To make the IoT work with ontologies it would be necessary with some organ to be responsible for the ontologies. Both in order to organize them and create guidelines on which ones to use where, but also to create mappings between them in order to know which ones covers the same.

7.5 Summary

Four approaches have been presented and of them, the first one was not sufficient to solve our problem. The following three can be used, however, it is not necessarily clear what the difference between them are. It has been mentioned that metamodels and ontologies are related; Metamodels can be seen as a strict rule set while an ontology is a vocabulary. Both these solutions need to be used together with other approaches in order to work. The web services are the most complex solution. This covers the whole application layer including protocols, data, and semantics. SOAP runs over HTTP and uses XML and a WSD defines the semantics using a WSDL. By using web services the devices can have different mechanics and negotiate which ones to use. That is not possible with ontologies and metamodels. In ontologies and metamodels there are a fixed language or a model that should be used. Web services are designed for the web and need some alterations in order to suit the IoT while metamodels and ontologies already exist for the IoT. A valid metamodel is an ontology, but not all ontologies are modeled as metamodels. Ontologies usually use Resource Description Framework (RDF) schemas; this is similar to XML and may be too extensive to the IoT.

What is common for all three solutions is that they need the semantics to be stored or presented somewhere so that everyone can understand it and re-use it. For instance a repository as Vorto is planning on or a website with explanations such as

can be found in [SSN] for the SSN ontology.

Chapter 8
Discussion

In this chapter I discuss initiatives, strategies, and approaches presented in previous chapters.

8.1 Dissecting the Interoperability in IoT at the Application Layer

Figure 8.1 gives an overview of what have been presented in this thesis and how an application layer could be built from these parts. The application layer can be divided into three separate layers: a protocol layer, a data layer, and a semantics layer. The protocol layer is responsible for transporting data from one device to another. In the IoT it is natural that not all devices will support the same protocol, thus translation between the protocols are required and represented by Ponte in the figure. The data layer concerns the structuring of messages that are being transferred between devices. The uppermost layer is the semantics layer, which enables devices to understand data that is sent and received.

Interoperability can be achieved by a global agreement on one combination of a semantics approach and a data approach, built upon interchangeable protocols. Combining the two topmost layers results in a *language* for the IoT, by agreeing on a single language, the selected combination can serve as a *common* language for the IoT.

8.2 Data Transfer Protocols

The thesis has previously presented three application layer protocols; HTTP, CoAP, and MQTT, but in Figure 8.1 also SOAP is included because it is used with WSD in web services. SOAP lies on top of HTTP, but reside among the protocols in the figure because its specifications have more similarities with the other protocols in the figure than of the data structures on the layer above.

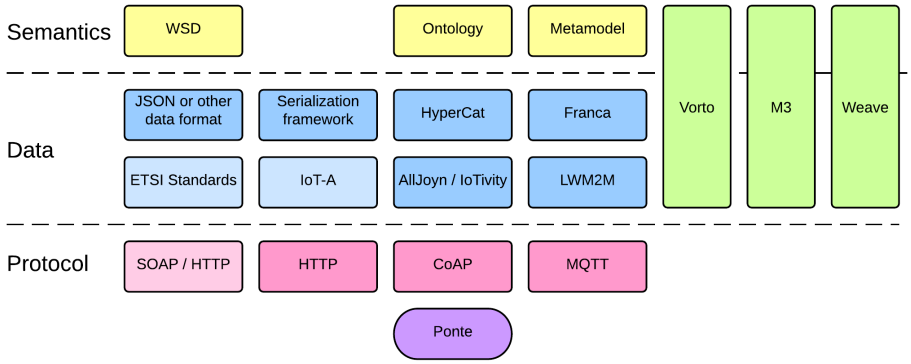

Figure 8.1: The application layer.

HTTP suits devices that send and receive considerable amounts of data and are connected to the power-line. However, HTTP is not appropriate for small, battery powered devices because it has more overhead than CoAP and MQTT which suggests that it will drain the device for battery in a shorter amount of time. MQTT has a publish/subscribe pattern unlike HTTP and CoAP and is thus more suitable in environments with many subscribing devices. HTTP and MQTT lie on top of TCP, which is not necessary for all devices as TCP have more protocol overhead than UDP, which is used by CoAP. HTTP can also be useful for devices that require a web interface, since HTTP is the most common web protocol. If the devices are constrained, CoAP can easily be mapped to HTTP.

There is not one single suitable application layer protocol for the IoT. Thus a mapping between them is needed. Ponte provides a solution to this by replacing the broker in MQTT and acting as both client and server in CoAP and HTTP. Desai et al. [DSA14] have a similar solution, but they perform the mapping between CoAP and MQTT by adding an additional broker. The IoT needs to support several application layer protocols and Ponte is an available solution that makes this possible.

8.3 Data Representation

The middle layer in Figure 8.1 concerns data representation. How data is structured is important in order to make machines understand it. There exist many data formats, and JSON and XML have been mentioned previously in this thesis. XML is a widely adopted format and is used in several of the discussed initiatives. The problem with XML is that it introduces big overhead, which is a problem in the

IoT where devices are constrained and often only need to transfer small amounts of data. JSON provides less overhead than XML, but is still more extensive than binary formats such as BSON, EXI, or Google Protocol Buffers. The advantages of JSON and XML is that they are easy to understand for humans and widely adopted in the web, thus developers already know how to use them. However, JSON is not extensible which can be a problem since it only offers six data types. By relying on JSON as the data format, the IoT language will require more semantics in order to meet future requirements.

The 'data serialization framework' in the figure refers to Google Protocol Buffers, Apache Thrift, and other such frameworks. They offer binary data formats and are able to serialize the formats to different programming languages. The data formats are binary and thus fast on the wire and to process, which is an advantage in the IoT. The serialization to different programming languages gives the developers more choices regarding implementation as they can choose what they are more familiar with. Also, not all devices in the IoT may support higher level programming languages; hence multiple programming languages have to be supported.

What data representation to use is not trivial since all the alternatives have their advantages and disadvantages. For the IoT language not to be too extensive it is important that the representation does not introduce unnecessary overhead. The data representation should also be extensible; as the IoT will grow, so will the language, introducing the need for new data types.

8.4 Semantics

Being able to transfer data with a common format is not enough to achieve interoperability in the IoT, as an understanding of the data is necessary. This is where the semantics come in. This thesis has previously presented three possible solutions for this layer: ontologies, metamodels, or WSDs and web services.

Web services concern the whole application layer and include protocol and data representation as well as semantics. They use protocols that already are widely adopted in the Internet. However, as web services are created for the web, they have overhead which is not appreciated in the IoT. Thus, web services need some adaption in order to comply with the requirements of the IoT. A version of SOAP that use JSON or a binary format with CoAP as the underlying protocol have to be developed if web services are to meet the needs of the IoT. The idea of using a definition of mechanics and letting devices agree on what data formats and semantics to use is good. This allows developers more choices during implementation, which may result in better solutions. If the web services are adapted, a common agreement on how to make WSDs is required.

Metamodels and ontologies are already made for the IoT, but in order for them to be part of a common language, someone needs to develop, maintain, and store them in a public place where developers and vendors can brows for suitable models meeting their needs. There should also be rules or guidelines in order to know where different classes or models should be used.

A problem with ontologies used in the semantic web is that they use long URIs that are represented by RDF serialization formats similar to XML. This can be a problem for the IoT especially if the data format underneath only has standard data types, as this would require the URI describing field types to be contained in the conveyed data. Thus ontologies coupled together with JSON could provide more overhead than desired.

The semantic model used has to be stored on devices. A device may ask a server or another device when its knowledge of the semantic model does not cover messages observed by the device. This way the device can extend its semantic model to be able to expand its understanding of the language. It is unlikely that the whole language will be saved on individual devices as they can be constrained and should contain as little as possible. Devices can be self-learning and only have to ask one time for each unknown concept. This can create a slow start for the device as it has to send many requests when it is newly installed, but this is only a installation problem.

Today, many existing ontologies, metamodels and WSDs cover the same concepts, and it would be more ideal to combine the ones that covers the same. Regardless if the solution is to create many small semantic models, or a single model covering everything, it should be well documented so that developers and vendors are able to use it correctly.

8.5 Vendors' Perspective

Despite the IoT being relatively young, many vendors have already tried to enter the market with their products. However, existing IoT devices do not provide interoperability with other vendors and usually require special skills to set up. However, from a vendor's perspective, interoperability with other vendors' devices may not be a goal. A vendor is likely to look for a standard, as standards with good reputations are usually well tested. However, many of the current standards organizations are still working on their application layer standards and vendors do not like to wait for formal standards. Thus, if a vendor is big enough, they could create proprietary solutions. This has the side effect of creating lock-in effects that are undesirable to customers while good for the vendor. If a vendor's goal is to make their customers satisfied by providing interoperability they may join an alliance, such as the AllSeen Alliance, or a consortium, such as the OIC. These organizations help

to motivate cooperation among its members, thus motivating interoperability.

For startups and smaller companies it may not be feasible to be part of such an organization. At the same time, these are the companies that have the most to gain from interoperability. However, for interoperability to be possible, the smaller vendors require knowledge of how other devices are working. Some organizations offer their frameworks for free while others charge a fee.

Vendors that find suitable standards may execute the 'embrace, extend and extinguish' strategy [EEE] in order to exclude competitors. This is a strategy Microsoft has executed in the past; they found standards and developed software that was compatible with the standards and competing products. Then they extended the standards by adding features that were not supported by competing products and thus created interoperability problems. Since Microsoft is a big company, their products gained a big market share and their extensions to the standards thus became the new standards. This lead to winning market shares from the competitors that did not support these extensions. This is an example of a undesired move by vendors and is something that should be avoided when creating standards in the IoT. By being aware of this, it is easier to avoid. However, vendors should still be able to offer additional functionalities to their costumers that other vendors do not offer to ensure competition and innovation.

8.6 Standards

Vendors like standards, but many standardization efforts are in progress and others are already completed. In this thesis we have seen standards from IEEE, ETSI, and OMA; in addition, ITU are making standards to be used in the application layer of the IoT. These standardization efforts are summarized in Table 8.1.

All these standards are open and royalty-free, allowing all vendors to use these standards in their devices. Many standards developed by industry consortia and individual companies charge royalty-fees. This is undesirable as it may exclude small companies and potentially cause more expensive devices.

8.7 Centralization

It is not necessarily a good idea to centralize the IoT in a single repository. Someone would need to manage the information without bias and create a feeling of 'insiders' and 'outsiders'. AllJoyn is open source, and everyone is allowed to contribute, but those who are not members of the AllSeen Alliance will likely feel like outsiders. The voice of the outsiders may be ignored favoring vendors who pay a fee to be part of

Organization	Standards
IEEE	Provides standards for health informatics, smart transducers, smart grid interoperability as well as many of the standards describing network layer protocols.
ETSI	Provides a functional architecture specification for M2M communication as well as test specifications, use cases, service requirements and studies.
OMA	Provides the standard for the LWM2M protocol as well as work in progress on device management, device classification, firmware update management, and an open connection manager API.
ITU	Provides a recommendation that explains IoT and its requirements. They have many working groups in the IoT field, but currently few standards and recommendations.

Table 8.1: Standardization efforts in the IoT.

the alliance. Thus, the outsiders may be inclined to look for an open source solution where everyone is equal.

A premise for interoperability in the IoT is that it has to be easy to find information about the technologies used by others, and thus a centralized repository or web site hosting this information would be a good solution. The centralized information hub may be in the form of an organization collecting information provided by others and thus not excluding any innovation from outsiders. The M3 framework takes this approach for ontologies and provides information on which ontologies are suitable and relevant for their devices.

8.8 A Language for the IoT

Say the world is able to agree on semantics and data representation and create a common IoT language. Such language would need the ability to convey a device's status, for instance if a thermostat is heating or cooling. The language should be able to describe the functionalities provided by a given device, as well as properties of the device. It would also be desirable if the language were able to describe the communication patterns and dynamic behaviors of a device. Table 8.2 provides descriptions of these language requirements and Table 8.3 shows an overview of which of the requirements are fulfilled by the different initiatives.

As can be seen in Table 8.3 some of the data initiatives support status, function and properties as well as message and dynamic behavior and one could think that they offer some semantics. However, these initiatives do not have a common way of

Requirement	Description
Status	The language should be able to describe the status of a device, tell what a device is doing at a given time as well as being able to tell if a device is ready to receive requests or if it is working and a request has to wait.
Function	The language should be able to describe the functionalities of a device and describe what a device is able to do.
Properties	The language should be able to describe the properties of a device included, but not limited to: serial number, model number, manufacturer, etc.
Messages	The language should be able to describe what messages a device can send and receive and describe the message formats as well as what messages a device is can understand.
Dynamic behavior	The language should be able to describe in what order messages can be sent and received and describe the dynamic behavior of a device and allowed sequences of events.

Table 8.2: Description of the different language requirements.

Initiative	Status	Function	Properties	Messages	Dynamic behavior
WSD	x	x	x	x	
Ontology	x	x	x		
Metamodel	x	x	x		
AllJoyn / IoTivity	x	x	x	x	
Data serialization framework	x		x		
JSON or other data format	x		x		
Eclipse Vorto	x	x	x	x	
Eclipse Franca	x	x	x	x	x
LWM2M	x		x		
HyperCat	x		x	x	
M3	x	x	x	x	
Weave	x	x	x	?	?

Table 8.3: Overview over which language requirements the IoT approaches fulfill.

expressing these requirements. Semantics require a set of rules or guidelines so that everyone can understand and use data the same way.

We can see that Franca is the only initiative that fulfills all the requirements, including being able to describe dynamic behavior. From Franca's Eclipse proposal [frac] we can read that:

> ... many severe bugs in complex software systems are caused by mismatches of the dynamic aspects of interfaces. These bugs will occur especially in late project phases (during system integration or even after customer delivery), are hard to identify and expensive to fix. Thus, it is necessary that the dynamic aspects should be part of the original interface definition, allowing extensive formal validation of the interface's implementations and usage.

Thus being able to describe dynamic behavior is a desired requirement for a IoT language. However, because there are no other initiatives pursuing this, one could argue that this is not necessary for creating an interoperable IoT. Notice that there is not enough information available regarding Google's Weave, and they may support description of dynamic behavior.

If dynamic behavior is a requirement of the language, Franca may be a natural part of a solution. A proper language could be constructed by Franca coupled with ontologies or with a metamodel. Franca coupled with a WSD would not work as WSDs are made for a specific data format and these data formats are not supported by Franca. It is also noteworthy that Franca is not made for the IoT suggesting that it will introduce too much overhead to fit the requirements of IoT. Thus, in addition to being coupled with a semantic solution, Franca would need to be adapted to the IoT in order to become a language.

If it is agreed that being able to describe dynamic behavior is not a language requirement, there are more possible solutions available. Vorto, M3, and possibly Weave could become languages under these constraints. AllJoyn, IoTivity or HyperCat coupled with ontologies or metamodels could be sufficient, and web services also potentially fill the requirements of a language. Data serialization frameworks, traditional data frameworks and LWM2M lack support for message description in order to meet all requirements on their own. However, these are often used as parts of the other initiatives.

It has already been established that web services as it is today cannot be a proper language for the IoT as they are too heavyweight. The same can be said for HyperCat, even though it is made for the IoT, it uses HTTP as the data transport protocol and

JSON as the data format. The M3 framework is working on extending the semantic web to include concepts from the IoT, but existing semantic web technologies can be too extensive to fit the IoT requirements. With these observations it seems like Vorto, AllJoyn, IoTivity or Weave are the only actual candidates to becoming a language.

Vorto is a highly relevant project that is supported by Bosch, a big actor in the IoT. Vorto is open source and can attract many contributors, but for this to happen it needs to attract the attention of vendors and developers. As of today, Vorto is not well documented. The framework should be further developed before drawing conclusions as to if it is suitable for the purpose of being the language for the IoT.

AllJoyn and IoTivity are frameworks supported by several powerful industry members. This makes it likely that members will use these frameworks in the future. However, it is not certain that the member companies will use these frameworks; Cisco is a member of both AllSeen Alliance and OIC, giving them the advantage of choosing which framework to use, or decide on using their own.

We do not know much yet about Google's Weave project, but Google is a powerful actor in the IT industry. Since we have limited information about Weave it is difficult to say if it will be a good solution for interoperability in the IoT, but because Google is behind it, it is likely to attract the attention of developers. The information released on Weave suggests that it is a highly relevant project, but we have to wait and see how it turns out before we know how useful it will be.

Google is not producing devices for the IoT.[1] Thus, their Weave solution can be appreciated because they are not focusing on making the language work with a given set of devices. This suggests a strategy where the language is made first, then the products that go with it. With such an approach it may be easier to make an extensible language that is forward-compatible. Many vendors are members of alliances, and may be strictly motivated by making their own product catalog interoperable. However, by starting with a language that provides interoperability, all future devices using the language can be interoperable, not only devices from a specific vendor. Hence an approach where the focus is making a language providing interoperability rather than focus on making interoperable devices is the desired approach to this problem.

If one of the initiatives providing an alternative language for the IoT, and this language attracts a lot of vendors and developers, value is added to the language. Hence before we end up with an agreement on a single language, many languages may compete for popularity. Such language is required to be long lived, thus it should be easy to add new functionalities and devices to the language.

[1] Google owns the IoT device vendor Nest.

All the discussed initiatives are free to use, but many of them are in development and more lack documentation. Thus, as of today, it is not feasible to use the current state of the initiatives as a common language. However, the ideas they represent can make the IoT interoperable in the future. What is required to achieve interoperability is to agree on semantics and data as well as encourage re-use and sharing.

Chapter 9
Concluding Remarks

This thesis has presented several different strategies and solutions that can be used in order to solve the issue of interoperability in the IoT. We have looked at 13 more or less different initiatives that all aim to improve the IoT, or have ideas that can be used in the IoT. We have also looked at three strategies used by the initiatives, and four approaches to fill the gaps in terms of achieving interoperability.

During this thesis I have mainly focused on the home environment and smart homes. However, the IoT is much bigger, and all its different areas have different requirements. Thus, the solution that is suitable in one area may not necessarily the best for another area. This is something to consider when developing a common language.

Regardless of area, there need to be a common agreement on how the application layer is built. This thesis divides the application layer into three sub-layers: one for data transfer protocols, one for data representation, and one for semantics and understanding of data. For the lowest layer, there already exist solutions that enable IoT vendors to choose the protocols that best suit their devices. Thus, it is the approaches on the two uppermost layers that need a common agreement in order to achieve an interoperable IoT. Not all studied data representations fulfill the requirements of the IoT. We can conclude that if a single data format was to be chosen for this purpose, it should be a binary format that is fast to process and transfer. For the semantics layer we saw that web services were not suitable for the IoT, while metamodels and ontologies would be appropriate.

A language consisting of initiatives from the data and semantics layers needs to be long lived. New functionalities and concepts will appear in both near and distant future and should be easy to add. The language should encourage re-use and sharing as well as being able to represent status, functionalities, properties, and messages.

It is not unthinkable that the solution will come from a powerful company such

as Google or an organization such as AllSeen Alliance or OIC. This is because these powerful organizations are able to attract many vendors and developers and thus also consumers. A language requires many users in order for it to have value, and a language created by someone with a large market share is more likely to succeed.

In the end, the result will likely be several language standards, possibly two or three. This has been seen in other technology areas before. There are several popular operating systems for personal computers and there are two main operating systems for smartphones. We see that many companies are joining alliances and consortia in the hope of being able to achieve interoperability. Also Google have showed that they want to offer a solution to the interoperability issue by introducing Weave. Several big companies are working for interoperability in the IoT, but there have not yet been presented a solution that the IoT community has agreed upon.

The thesis studies many initiatives, but there still exist several initiatives and approaches that were not discussed due to lack of information and to retain a manageable scope of this thesis. These alternatives should also be considered when wanting to build an interoperable IoT.

Notice that security was an issue out of scope of this thesis. Security is a big issue in the IoT and should also be an important factor in a language. This makes finding a solution to the interoperability issue even harder.

References

[802] Part 15.4: Wireless medium access control (mac) and physical layer (phy) specifications for low-rate wireless personal area networks (lr-wpans). http://user.engineering.uiowa.edu/~mcover/lab4/802.15.4-2003.pdf. Accessed: 2015-03-03.

[alla] About. https://allseenalliance.org/about. Accessed: 2015-04-20.

[allb] Allseen faqs. https://allseenalliance.org/about/faqs. Accessed: 2015-04-20.

[BBF+13] Alessandro Bassi, Martin Bauer, Martin Fiedler, Thorsten Kramp, Rob van Kranenburg, Sebastian Lange, and Stefan Meissner. *Enabling Things to Talk*. Springer, first edition, 2013.

[Bri] Project brillo. https://developers.google.com/brillo/. Accessed: 2015-06-01.

[coa] Coap. http://coap.technology/. Accessed: 2015-02-16.

[cok] The "only" coke machine on the internet. https://www.cs.cmu.edu/~coke/history_long.txt. Accessed: 2015-04-20.

[Cou08] US National Intelligence Council. Disruptive civil technologies. April 2008. Accessed: 2015-04-20.

[Cro06] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627, Internet Engineering Task Force, July 2006.

[Dod03] Sean Dodson. The internet of things. *The Guardian*, October 2003. Accessed: 2015-04-20.

[DSA14] Pratikkumar Desai, Amit Sheth, and Pramod Anantharam. Semantic gateway as a service architecture for iot interoperability. October 2014.

[EEE] Embrace, extend, extinguish (it vendor strategies). http://www.hr.com/SITEFORUM?&t=/Default/gateway&i=1116423256281&application=story&active=no&ParentID=1119278077613&StoryID=1119649742078&xref=https%3A//www.google.no/. Accessed: 2015-05-31.

[EMe] Iot/m2miwg/m2m meta-model. https://wiki.eclipse.org/IoT/M2MIWG/M2M_meta-model. Accessed: 2015-06-03.

[EMF] Eclipse modeling framework (emf). http://www.eclipse.org/modeling/emf/. Accessed: 2015-05-11.

[ES07] Jerome Euzenat and Pavel Shvaiko. *Ontology matching*. Springer, 2007.

[etsa] About etsi. http://www.etsi.org/about. Accessed: 2015-04-20.

[etsb] Machine to machine communications. http://www.etsi.org/technologies-clusters/technologies/m2m. Accessed: 2015-04-20.

[ets13a] Etsi ts 101 584 v2.1.1 (2013-12) machine-to-mahine communications (m2m); study on semantic support for m2m data. http://www.etsi.org/deliver/etsi_tr/101500_101599/101584/02.01.01_60/tr_101584v020101p.pdf, 2013. Accessed: 2015-05-12.

[ets13b] Etsi ts 102 690 v2.2.1 (2013-10) machine-to-mahine communications (m2m); functional architecture. http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/02.01.01_60/ts_102690v020101p.pdf, 2013. Accessed: 2015-05-12.

[eui] Internet of things 2008. http://www.the-internet-of-things.org/iot2008/. Accessed: 2015-04-20.

[Eva11] Dave Evans. The internet of things. April 2011. Accessed: 2015-04-20.

[FGM+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Internet Engineering Task Force, June 1999.

[fraa] 0.9. https://a29eed10a342ed44b9242976382a23bbd1389fd3.googledrive.com/host/0B7JseVbR6jvhazEtRDVsSk9mX1k/Releases/0.9/. Accessed: 2015-05-20.

[frab] Franca. https://www.eclipse.org/proposals/modeling.franca/. Accessed: 2015-05-11.

[frac] Franca. http://eclipse.org/proposals/modeling.franca/. Accessed: 2015-04-19.

[GKC] Neil Gershenfeld, Raffi Krikorian, and Danny Cohen. The internet of things.

[GPB] Download protocol buffers. https://developers.google.com/protocol-buffers/docs/downloads. Accessed: 2015-05-21.

[Gru92] Tom Gruber. What is an ontology? http://www-ksl.stanford.edu/kst/what-is-an-ontology.html, 1992. Accessed: 2015-05-18.

[Gyr15] Amèlie Gyrard. PhD thesis, TELECOM ParisTech, 2015.

[HCC09] Jonathan Hui, David Culle, and Samita Chakrabarti. 6lowpan: Incorporating ieee 802.15.4 into the ip architecture. Technical report, IPSO Alliance, January 2009.

[HCs] Hyper/cat. https://drive.google.com/file/d/0B5JKRgbs8OyldnVMbTFOc0xpZDg/edit. Accessed: 2015-05-12.

[hypa] Gartner hype cycle. http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp. Accessed: 2015-04-20.

[hypb] Gartner's 2014 hype cycle for emerging technologies maps the journey to digital business. http://www.gartner.com/newsroom/id/2819918. Accessed: 2015-04-20.

[hypc] Hypercat. http://www.hypercat.io/. Accessed: 2015-04-19.

[IEE05] Ieee standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirements. - part 15.1: Wireless medium access control (mac) and physical layer (phy) specifications for wireless personal area networks (wpans). *IEEE Std 802.15.1-2005 (Revision of IEEE Std 802.15.1-2002)*, pages 1–580, 2005.

[IEE11] Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans). *IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006)*, pages 1–314, Sept 2011.

[IEE12a] Ieee standard for ethernet - section 1. *IEEE Std 802.3-2012 (Revision to IEEE Std 802.3-2008)*, pages 1–634, Dec 2012.

[IEE12b] Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, March 2012.

[ins] Insteon details. http://cache.insteon.com/documentation/insteon_details.pdf. Accessed: 2015-03-12.

[iota] About. https://www.iotivity.org/about. Accessed: 2015-04-19.

[iotb] Architecture overview. https://www.iotivity.org/documentation/architecture-overview. Accessed: 2015-05-06.

[iotc] Internet of things. http://www.oxforddictionaries.com/definition/american_english/Internet-of-things. Accessed: 2015-04-18.

[iotd] That 'internet of things' thing. http://www.rfidjournal.com/articles/view?4986. Accessed: 2015-04-20.

[ips] About ipso. http://www.ipso-alliance.org/about/mission. Accessed: 2015-04-19.

[ITU05] ITU. The internet of things. November 2005. Accessed: 2015-04-20.

[KMS07] N. Kushalnagar, G. Montenegro, and C. Schumacher. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919, Internet Engineering Task Force, August 2007.

[KR10] James F. Kurose and Keith W. Ross. *Computer Networking A Top-down Approach*. Pearson Education, Boston, MA 02116, fifth edition, 2010.

[lgf] Lg internet refrigerator is at the heart of the digital home network. http://www.prnewswire.com/news-releases/lg-internet-refrigerator-is-at-the-heart-of-the-digital-home-network-75420177.html. Accessed: 2015-04-20.

[LWMa] Lwm2m over mqtt. http://openiotchallenge.tumblr.com/post/110888019710/lwm2m-over-mqtt. Accessed: 2015-06-05.

[lwmb] Oma lightweightm2m v1.0. http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0. Accessed: 2015-05-11.

[lwmc] Oma lwm2m managemnet object editor ver: 1.2.01. http://dev01.alnas.com/OMA/Default. Accessed: 2015-05-19.

[mqt] Mqtt version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf. Accessed: 2015-02-18.

[nfc] About near field communication. http://www.nearfieldcommunication.org/about-nfc.html. Accessed: 2015-03-12.

[nfc13] Information technology - telecommunications and information exchange between systems - near field communication - interface and protocol (nfcip-1). https://www.iso.org/obp/ui/#iso:std:iso-iec:18092:ed-2:v1:en, 2013. Accessed: 2015-04-21.

[oic] Open interconnect consortium. http://openinterconnect.org/. Accessed: 2015-04-19.

[OMA] Omna lightweight m2m (lwm2m) object & resource registry. http://technical.openmobilealliance.org/Technical/technical-information/omna/lightweight-m2m-lwm2m-object-registry. Accessed: 2015-05-25.

[OWL] Owl. http://www.w3.org/2001/sw/wiki/OWL. Accessed: 2015-05-23.

[pon] Ponte - m2m bridge framework for rest developers. http://eclipse.org/proposals/technology.ponte/. Accessed: 2015-05-11.

[Pro] Protocol buffers. https://developers.google.com/protocol-buffers/. Accessed: 2015-05-13.

[SHB14] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7252, Internet Engineering Task Force, June 2014.

[Sim14] Tom Simonite. Silicon valley to get a cellular network, just for things. *MIT Technology Review*, May 2014. Accessed: 2015-04-29.

[SSN] Semantic sensor network ontology. http://www.w3.org/2005/Incubator/ssn/ssnx/ssn. Accessed: 2015-06-03.

[Std] What are standards? http://www.etsi.org/standards/what-are-standards. Accessed: 2015-06-02.

[thr] Apache thrift. https://thrift.apache.org/about. Accessed: 2015-05-07.

[TW10] Lu Tan and Neng Wang. Future internet: The internet of things. In *Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on*, volume 5, pages V5–376–V5–380, Aug 2010.

[VD10] Jean-Philippe Vasseur and Adam Dunkels. *Interconnecting Smart Objects with IP*. Morgan Kaufmann, 30 Corporate Drive, Suite 400, Burlington, MA 01803, USA, 2010.

[vora] About vorto. https://eclipse.org/vorto/about.html. Accessed: 2015-06-05.

[Vorb] index : org.eclipse.vorto.git. http://git.eclipse.org/c/vorto/org.eclipse.vorto.git/. Accessed: 2015-05-21.

[vorc] Vorto. https://projects.eclipse.org/proposals/vorto. Accessed: 2015-05-11.

[W3Wa] Web services architecture. http://www.w3.org/TR/ws-arch/. Accessed: 2015-05-21.

[W3Wb] Web services glossaries. http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/. Accessed: 2015-05-19.

[Wei04] Robert Weisman. The internet of things. *Boston Globe*, October 2004. Accessed: 2015-04-20.

[wif] Certification. http://www.wi-fi.org/certification. Accessed: 2015-04-21.

[YTW] Google i/o 2015 - keynote. https://www.youtube.com/watch?v=7V-fIGMDsmE&feature=youtu.be&t=2265. Accessed: 2015-06-01.

[zwa] About z-wave technology. http://z-wavealliance.org/about_z-wave_technology/. Accessed: 2015-03-12.

[zwa15] Short range narrow-band digital radiocommunication transceivers - phy and mac layer specifications. http://www.itu.int/rec/T-REC-G.9959-201501-P/en, 2015. Accessed: 2015-04-21.