Behailu Negash

Interoperating Networked Embedded Systems to Compose the Web of Things

Turku Centre *for* Computer Science

TUCS Dissertations
No 238, June 2019

Interoperating Networked Embedded Systems to Compose the Web of Things

Behailu Shiferaw Negash

To be presented, with the permission of the Faculty of Science and Engineering of the University of Turku, for public criticism in Auditorium XXI of Agora on June 4, 2019, at 12 noon.

University of Turku
Department of Future Technologies
20014 TURUN YLIOPISTO
FINLAND 2019

Supervisors

Adjunct Professor Tomi Westerlund
Department of Future Technologies
University of Turku
Vesilinnantie 5, 20500 Turku
Finland

Professor Hannu Tenhunen
Department of Electronics
KTH Royal Institute of Technology
Brinellvägen 8, 11428 Stockholm
Sweden

Reviewers

Professor Antti Ylä-Jääski
Department of Computer Science
Aalto University
Konemiehentie 2, Espoo
Finland

Associate Professor Cristina Seceleanu
Department of Networked and Embedded Systems
Mälardalen University
Högskoleplan 1, 72218 Västerås
Sweden

Opponent

Professor Sasu Tarkoma
Department of Computer Science
University of Helsinki
Pietari Kalmin Katu 5, Helsinki
Finland

ISBN 978-952-12-3828-4
ISSN 1239-1883

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin Originality Check service.

Abstract

Improvements in science and technology have enhanced our quality of life with better healthcare services, comfortable living and transportation among others. Human beings are now able to travel faster, communicate across the globe in fraction of seconds, understand nature better than ever before and generate and consume huge amount of information. The Internet played a central role in this development by providing a vast network of networks. Leveraging this global infrastructure, the World Wide Web is providing a shared information space for such unprecedented amount of knowledge that is mostly contributed and used by human beings. It has played such a critical role in the adoption of the Internet, it is common to find people referring specific web sites as Internet. This adoption coupled with advances in manufacturing of computing elements that led to the reduction in size and price has introduced a new wave of technology, called the Internet of Things.

A rudimentary description of the Internet of Things (IoT) is an Internet that connects, not only traditional computing devices (with higher capacity and provide user interface) but also everyday physical objects or 'Things' around us. These objects are augmented by small networked embedded computing elements that interact with the host via sensors and actuators. It is estimated that there will be Billions of such devices and Trillions of dollars of market value distributed in multiple aspects of our lives; such as healthcare, smart home, smart industries and smart cities. However, there are many challenges that are hindering the wide adoption of IoT. One of these challenges is heterogeneity of network interfaces, platforms, data formats and many standards that led to vertical islands of systems that are not interoperable at various levels.

To address the lack of interoperability, this thesis presents the author's contributions in three categories. The first part is a lightweight middleware called LISA that address variations in protocols and platforms. It is designed to work within the constrained resources of the networked embedded devices. The overhead of the middleware is evaluated and compared with other related frameworks. The second set of contributions focus on higher level of system integration and related challenges. It includes a domain specific IoT language (DoS-IL) and a server implementation to support the proposed

code on demand approach. The scripting language enables re-configuration of the behaviour of systems during integration or functional changes. The related server provides abstraction of the physical object and its embedded device to provide mobility services in addition to hosting the scripts. The last set of contributions are focused on either generalized architectural style design or a specific healthcare use case.

In summary, the overall thesis presents a highlevel architectural style that provides ease of understanding and communication of IoT systems, serves as a means for system level integration and provides the desired quality attributes for IoT systems. The other contributions fit in the architectural style to facilitate the adoption of the style or showcase specific instances of the architecture's use. The performance of the middleware, the scripting language and the server including their resource utilization and overhead have been analyzed and presented. In general, the combination of the contributions enable inter-operation of networked embedded systems that serve as building blocks for the Web of Things - a global system of IoT systems.

Tiivistelmä

Kehitykset tieteessä ja teknologiassa ovat parantaneet elämänlaatuamme muun muassa paremmilla terveydenhuollon palveluilla, viihtyisällä asumisella ja uusilla kuljetuspalveluilla. Ihmiset voivat nyt matkustaa nopeammin, kommunikoida ympäri maailmaa sekunnin murto-osassa, ymmärtää luontoa paremmin kuin koskaan ennen, ja tuottaa sekä kuluttaa valtavan määrän tietoa. Internetillä on ollut keskeinen rooli tässä kehityksessä tarjoamalla laajan verkostojen verkoston. World Wide Web (WWW) tarjoaa tämän maailmanlaajuisen infrastruktuurin avulla yhteistä informaatiotilaa ennennäkemättömälle määrälle tietämystä. WWW:llä on ollut niin kriittinen rooli internetin käyttöönotossa, että on yleistä nähdä ihmisten viittaavan tiettyihin verkkosivustoihin internettinä. Tämä muovautuminen, yhdistettynä edistysaskeliin tietokonekomponenttien valmistuksessa, joka on johtanut niiden koon ja hinnan pienenemiseen, on tuonut esiin uuden teknologian aallon, jota kutsutaan esineiden internetiksi.

Esineiden internetin (Internet of Things, IoT) alkeellinen kuvaus on Internet, joka yhdistää paitsi perinteiset tietokonelaitteet (tehokkaat laitteet käyttöliittymällä), myös arjen fyysiset esineet ympärillämme. Näitä laitteita on laajennettu pienillä, verkkoon kytkeytyneillä, sulautetuilla tietojenkäsittely-yksiköillä, jotka ovat vuorovaikutuksessa ympäristön kanssa antureiden ja toimilaitteiden välityksellä. On arvioitu, että tällaisia laitteita tulee olemaan miljardeja ja ne tuovat biljoonien dollarien markkina-arvon, joka jakautuu eri puolille elämäämme, kuten terveydenhuoltoon, älykkäisiin koteihin, älykkääseen teollisuuteen ja älykkäisiin kaupunkeihin. On kuitenkin monia haasteita, jotka haittaavat IoT:n laajaa käyttöönottoa. Yksi näistä haasteista on verkkoliitäntöjen, alustojen, tietomuotojen ja monien standardien heterogeenisyys. Heterogeenisyys on johtanut järjestelmien vertikaalisiin saariin, jotka eivät ole eri tasoilla yhteentoimivia.

Yhteentoimivuuden parantamiseksi tämä opinnäytetyö esittelee tekijän työpanosta kolmessa kategoriassa. Ensimmäisessä kategoriassa esitellään kevyt väliohjelmisto nimeltä LISA, joka mahdollistaa eri protokollien ja alustojen yhteiselon. Se on suunniteltu toimimaan verkkoon kytkeytyneiden sulautettujen laitteiden rajallisten resurssien puitteissa. LISA:n toteutuksen kustannukset laitteistotasolla arvioidaan ja niitä verrataan muihin vas-

taaviin väliohjelmistoihin. Seuraavaksi keskitytään järjestelmien integroinnin korkeammale tasolle ja siihen liittyviin haasteisiin. Tässä kategoriassa esitellään sovellusaluekohtainen IoT-kieli (DoS-IL) ja sen tarvitseman palvelimen toteutus. Skriptikieli mahdollistaa järjestelmien uudelleen ohjelmoinnin niiden integroinnin ja päivitysten aikana. Siihen liittyvä palvelin tarjoaa abstraktin version fyysisestä esineestä ja sen sulautetusta laitteesta liikkuvuuspalvelujen tarjoamiseksi sekä skriptien ylläpitämiseksi. Viimeisessä kategoriassa keskitytään sekä yleistettyyn arkkitehtuurityyliin että tiettyyn terveydenhuollon käyttötarkoitukseen.

Yhteenvetona voidaan todeta, että opinnäytetyössä esitellään korkeatasoinen arkkitehtoninen tyyli, joka helpottaa IoT-järjestelmien ymmärrystä ja niiden välistä kommunikaatiota, toimii järjestelmätason integraatiovälineenä ja tarjoaa halutut laatuominaisuudet IoT-järjestelmille. Muut opinnäytetyössä esitellyt asiat täydentävät arkkitehtonista tyyliä ja helpottavat sen käyttöönottoa sekä esittelemään tiettyjä sovelluksia. Väliohjelmiston, skriptikielen ja palvelimen suorituskyky, mukaan lukien resurssien käyttö ja yleiskäyttö, on analysoitu ja esitetty. Yleisesti ottaen näiden työpanosten yhdistäminen mahdollistaa verkkoon kytkeytyneiden sulautettujen järjestelmien yhteentoimivuuden, joka toimii Web of Thingsin, maailmanlaajuisen IoT-järjestelmien järjestelmän perustana.

Acknowledgements

Time is a strictly constrained and precious resource that we all have to spend it according to our free will. We invest our time and collect the dividend. I decided to come to Finland to spend the past few years on an adventure of exploring my passion for embedded systems. Lucky me, it was not just my time that is invested, but that of many people who are generous enough to spending their portion as well. I hope I was not a bad investment after all. First, for the one and only, who gave us the time and space - the Almighty God, thank you for blessing me with all the people who helped, challenged and comforted me. I want to start with a sincere thanks to my supervisors and mentors Adjunct Professor Tomi Westerlund and Professor Hannu Tenhunen, for your guidance, motivation and support throughout the journey. The time is worth it mainly because of your immense input that transformed me from a newbie to master. In addition, I am grateful for the assistance and advice of Professor Pasi Liljeberg and Adjunct Professor Amir Rahmanisani throughout the period of my project works at the University of Turku in IoT4Health research group. I appreciate the support of the Academy of Finland for funding the project I was involved in. I want to also say thank you to the reviewers, Professor Antti Ylä-Jääski and Associate Professor Cristina Seceleanu, for taking the time and shaping this thesis. I would like also to thank Professor Sasu Tarkoma for accepting to be an opponent. Furthermore, I am grateful of the support I received from my colleagues, both at the University of Turku and RDVelho Oy whenever I needed it. My stay in Finland would have been difficult if it was not for my friends that filled the social void and I want to say thank you. My parents and siblings - I am indebted to you, for I am a result of your care and advice from the early childhood. I am eternally grateful for my late mother Tezerash Abebe for all that I am. Besides, I want to take this opportunity to say thank you to my sisters for their selfless love and care. To my brothers who inspired me from early on and showed me the way, I want to extend my gratitude. Last but not least, my sincere gratitude goes to my dear wife and best friend, Mrs. Mahlet Tamirat, for your support, encouragement and being next to me in this adventure and the years to come. You know, I wouldn't do this if you were not with me - you are the best.

List of original publications

The work presented in this dissertation is based on the following publications:

1. Paper I
   Behailu Negash, Amir M. Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen, **LISA 2.0: Lightweight Internet of Things Service Bus Architecture Using Node Centric Networking**, Journal of Ambient Intelligence and Humanized Computing (JAIHC), 7(3):305-319, 2016.

2. Paper II
   Uzair A. Noman, **Behailu Negash**, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen **From Threads to Events: Adapting a Lightweight Middleware for Contiki OS**. The 14th Annual IEEE Consumer Communications and Networking Conference, 8-11 January 2017, Las Vegas, USA

3. Paper III
   Behailu Negash, Tomi Westerlund, Amir M. Rahmani, Pasi Liljeberg, Hannu Tenhunen, **DoS-IL: A Domain Specific Internet of Things Language for Resource Constrained Devices**, Procedia Computer Science, Volume 109, Pages 416 426, ISSN 1877-0509, https://doi.org/10. 1016/j.procs.2017.05.411, Elsevier International Conference on Ambient Systems, Networks and Technologies **(ANT17)**, 2017, Portugal.

4. Paper IV
   Behailu Negash, Tomi Westerlund, Pasi Liljeberg, Hannu Tenhunen, **Rethinking Things Fog Layer Interplay in IoT: a Mobile Code Approach**, In A Min Tjoa, Li-Rong Zheng, Zhuo Zou, Maria Raffai, Li Da Xu, and Niina Maarit Novak, editors, Research and Practical Issues of Enterprise Information Systems, pages 159-167, Cham, 2018. Springer International Publishing.

5. Paper V
   Behailu Negash, Tomi Westerlund, and Hannu Tenhunen, **Towards an interoperable Internet of Things through a web of virtual things at the Fog layer**, Future Generation Computer Systems(FGCS), 91:96-107, 2019.

6. Paper VI
   Amir M. Rahmani, Tuan Nguyen Gia, **Behailu Negash**, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, Pasi Liljeberg, and Hannu Tenhunen, **Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach**, Future Generation Computer Systems(FGCS), 78:641-658, 2018.

7. Paper VII
   Behailu Negash,Tomi Westerlund, and Hannu Tenhunen **A Pragmatic Architectural Style for Interoperable and Scalable Internet of Things Systems**. *Submitted*, 2019

Others not included in the thesis

8. **Behailu Negash**, Amir M. Rahmani, Pasi Liljeberg, Axel Jantsch, **Fog computing fundamentals in the Internet of Things** in **Fog Computing in the Internet of Things Intelligence at the Edge**, Springer, 2017.

9. **Behailu Negash**, Tuan Nguyen Gia, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, Tomi Westerlund, Amir M. Rahmani, Pasi Liljeberg, Hannu Tenhunen, **Leveraging Fog Computing for Healthcare IoT** in **Fog Computing in the Internet of Things Intelligence at the Edge**, Springer, 2017.

10. **Behailu Negash**, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen **Lisa: Lightweight internet of things service bus architecture.** Procedia Computer Science, 52:436 443, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

11. Amir-Mohammad Rahmani, Nanda Kumar Thanigaivelan, Tuan Nguyen Gia, Jose Granados, **Behailu Negash**, Pasi Liljeberg, and Hannu Tenhunen. **Smart e-health gateway: Bringing intelligence to Internet of Things based ubiquitous healthcare systems.** In Henry Holtzman, editor, IEEE Consumer Communications and Networking Conference, pages pp. 826834. IEEE Xplore Digital Library, 2015.

12. **Behailu Negash**, Amir M. Rahmani, Tomi Westerlund, Pasi Liljeberg,and Hannu Tenhunen, **Enabling Layered Interoperability for Internet of Things Through LISA.** Design, Automation and Test in Europe(DATE) Conference, DATE University Booth, 2016, Germany.

Contents

List of Figures

List of Tables

Part I

Research summary

Chapter 1

Introduction

The world has seen unprecedented amount of economic and technological growth in the past few decades. This is mainly the result of the advancements in information and communication technologies. A central element of the progress is the Internet and the fast adoption of the World Wide Web. It has shaped the way we communicate, work, and in general the way we live. The overall benefits can be summarized as the advancement of the quality of life. To keep the momentum and reach new heights, the Internet is continuously evolving. Few years after its inception, the connected entities to the Internet were mainly used by human beings to create content or access an existing shared information. However, the last two decades have introduced a range of embedded devices, which are built for specific applications, to the Internet. The earliest of these devices was the coke machine from the Carnegie Mellon University computer science department [98]. As manufacturing technologies and processes improve, the cost and size of computing devices dropped radically and increased the number of devices that are connected to the Internet. These small computing devices are mostly embedded in physical objects. In 1999, Kevin Ashton named the Internet that includes everyday objects as the Internet of Things [9]. In the past decade, the phrase Internet of Things (IoT) has dominated the industry and academia. This has resulted in a wide range of definitions and the emergence of many related terms that refer to the same concept. Regardless of the naming, IoT is expected to further enhance quality of life and change the way we interact with the physical world. It has already been adopted in a wide range of domains, such as smart healthcare, manufacturing industries, agriculture and city infrastructures. The advancement of manufacturing techniques to produce cheaper and smaller computing elements has also another positive impact on the processing of the huge amount of data generated by the Internet of Things. The parallel growth in Artificial Intelligence and machine learning algorithms added to the big data

collected from IoT creates an enormous opportunity to push the frontiers of technology; smart devices that understand the behaviour and intent of their owners self-organized to provide quality services, smart cities that are cleaner and more livable, smart transportation, and healthcare are only the beginning. However, the transition from where we are now to the envisioned utopia is not smooth. There are many challenges that hinder the progress in adoption of the Internet of Things. Some of these are security, privacy, heterogeneity of devices and networks, lack of standardization, and tight resource constraints. The main focus of this thesis is on the ways to overcome the challenges and maximize the advantages of IoT to make the quality of life better, more specifically the inter-operation of IoT systems. This chapter provides the foundation of the work by framing a working definition of IoT, its vision and challenges. It also gives an overview of the contributions of this thesis and its overall structure.

1.1 Definition, Characteristics and Challenges

The Internet of Things is a widely used terms that gets easily misinterpreted. Similarly, there are many terms that refer to the same concept. It is mandatory to clarify what the discussion is about, what it really is and its scope. This provides the boundaries where the challenges can be gathered, analyzed and addressed. It also helps to characterize the technology to extract key quality attributes of the systems it contains. Many standards organizations, industrial alliances, academic research groups and public projects define IoT in various ways. The IEEE IoT initiative has summarized definitions to clarify the confusion [59]. It defines IoT by listing its characteristics taking simple and complex use cases. A concise version of the definition presents IoT as the interconnection of physical objects, augmented by computing devices, that connect to the Internet to access information or provide service for others. These objects are uniquely identifiable and have sensors and (or) actuators and are capable of self-configuration. In addition, the objects have intelligence and they are accessible from anywhere at anytime. This description of IoT also makes its difference clear compared with other related terms, such as Wireless Sensor Networks and Cyber-physical Systems. Such an elaborate characterization raises the question of the purpose or vision that drives IoT development.

Atzori *et al.* [12] in their survey paper presented the vision of IoT as the intersection of three perspectives: Things vision, Internet Vision and Semantic Vision. The first view is focused on the physical objects that make up the leaf nodes in IoT. It includes the connectivity and intelligence of these devices. The Internet oriented vision mainly covers the global connectivity infrastructure that allows access to information from anywhere in the world

as extended by the introduction of these new types of devices. The last vision of IoT concentrates on the information provided by these new physical objects connected to the Internet and how it is structured and interpreted by the recipient without ambiguity. Each of these vision statements map to certain properties of IoT; for instance, the things oriented vision maps to the self-configuration and intelligence of IoT devices, and the Internet oriented vision encompasses the unique identification of the things on the Internet.

Based on the properties of IoT presented above and the vision set to be achieved, the challenges of IoT have been identified. Some of these challenges are interoperability, security, scalability and maintainability or re-configuration [104]. Al-Fuqaha *et al.* [1] have compiled a more comprehensive list of the challenges and possible applications of IoT systems. Considering the current state of IoT deployments, interoperability stands out as one of the most important challenges that hinders the adoption of IoT [32]. In 2015 McKinsey [47] reported that enabling IoT interoperability has the potential to unlock about 40% of the total economic value. In addition, the lack of interoperability also hinders the creation of novel cross domain solutions. This thesis is mainly oriented towards addressing interoperability of IoT at different levels, such as platform variations, the format of data and context of exchanged information. The main cause of this heterogeneity is the lack of dominant standards in different categories. Chapter two provides detailed analysis of the standards landscape the required interoperability.

1.2 Abstract representation

Working on domain specific areas of IoT and tackling the challenges of independent systems leads to more fragmentation as each type of system becomes more specialized. There is a clear functional distinction and non-functional requirements that seem to demand addressing the issues separately. However, generalization and abstraction of IoT systems provides a more scalable and futuristic design. To begin with the process of generalization, it is important to specify the distinction of the infrastructure or connectivity and the software systems. In most cases, IoT is discussed combining the devices, connectivity and the software components. However, in the context of this thesis, **IoT** refers to only the connectivity of physical objects or devices to the Internet where as **IoT systems** is used to specify the software elements. This naming convention is used analogous to the Internet and the World Wide Web as examples; IoT is for Internet as IoT systems is for the Internet scale web applications. This similarity also supports the argument that IoT as a connectivity layer should be agnostic to the applications that run over it and the abstract representation of IoT systems as a big interconnected system as a Web application. Explaining the design of

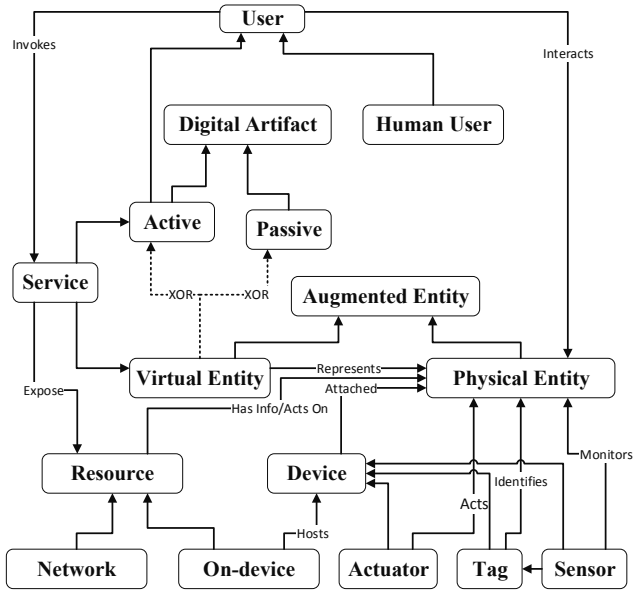

Figure 1.1: IoT domain model as presented by the IoT-A project [49]

the Internet, Vint Cerf, the father of the Internet, mentioned the openness as the main source of its success [22]. Using this analogy in reference to IoT, it will be considered as a connectivity layer with multiple alternatives for various bandwidth, range and coverage requirements by IoT systems. Similarly, generalizing IoT systems from variety of use cases facilitates in identifying common non-functional requirements. These common features are used for modeling purposes. One of the many EU projects that focuses on IoT systems is IoT-A. It is a project that was aimed at defining a reference architecture. It presents IoT as a generic system and develops a domain, functional and communication model [49]. Building on the domain model, the interoperability of IoT systems is explored.

In its simplest form, a typical computing device in IoT, which is integrated with a physical object, is composed of at least one of the three types of elements: a sensor, an actuator or a tag (such as RFID). This is presented by the IoT-A project in the domain model as shown in Figure 1.1. The model shows how the resources in this devices are exposed as services, how services are accessed by users and the interaction between the device and a physical object. It also shows how a digital representation is formed from the physical object and the computing device attached to it. Consid-

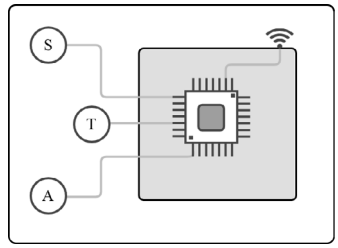

Figure 1.2: Abstraction of an integrated device with sensors (S), actuators (A) and a tag (T) in a IoT systems

ering the model from the computing devices perspective, a simplification of the model is presented in Figure 1.2. It presents an abstract computing device that is integrated in a physical object, with a network interface for external communication, sensors (shown as S), actuators (A) and a tag (T). This level of abstraction provides a simplified view of an IoT device and its components to help identify generic challenges in IoT systems and forms a virtual counter part of the device for application design. It also isolates the device from the details of how on-board resources are accessed so that the design process is not restricted to the form of communication shown in the IoT-A model. The next step is to abstract the interactions of the system components based on the generic device model. Considering distinct application domains that can cover almost all possible forms of inter-dependency between abstract devices, a generalization of the overall interaction among IoT system components will be identified. It also provides the system components and quality attributes that are used in architectural style design to enables interoperability at system level. Based on these findings, this thesis compiled a set of contributions in the design of a pragmatic architectural style and different utilities that assist in adopting the style. These contributions are highlighted in the section following the main research questions addressed by this work.

1.3 Research questions and methodology

In the preceding sections, the background of the research topic is presented while also highlighting the area of focus, lack of interoperability, in Section 1.1 as challenges of the Internet of Things. These challenges in building a secure and integrated IoT system that scales up to the level of the Internet are too broad to cover in one thesis. Therefore, this work concentrates on ways to provide interoperability among IoT systems while also touching related topics on the way. To prove the relevance of the research questions

and severity of the IoT challenge selected in the topic, the research starts with qualitatively analysing literature, generalize the various domain specific needs and evaluate previously proposed approaches. In addition, quantitative measures, such as performance, latency and resource utilization, are used in the papers to provide a comparative analysis of the individual solutions proposed to answer the research questions presented. The research begins with the hypothesis that"lack of interoperability is one of the critical challenges in IoT" and identify its research questions. This assumption is shown to be valid in the qualitative analysis of background materials provided in this and the next chapters and all the papers included in the thesis. Hence, the first research question is "Why is IoT still suffering from lack of interoperability with many standards and middleware out there, and what are the shortcomings of these approaches and possible solutions?". This research questions has led to the work in Paper I and Paper II, which aim at providing a lightweight middleware solution designed for IoT devices with severe resource constraints. However, the lightweight middleware is just one part of the solution to lack of interoperability. On the way of developing the middleware, more research questions have been raised that potentially hinder interoperability mainly due to the very large number of small IoT devices with distinct behaviours that get connected to the Internet. The follow-up question tries to find out optimal ways to organize these devices, their internal components and the interaction among them. The thesis tries to answer "Is there a better way to arrange the elements of IoT systems for a more interoperable IoT to realize the Web of Things?". This question has been the umbrella for subsequent papers that answer detailed questions, such as "How to facilitate inter-operation of IoT systems that have dynamic IoT devices that frequently change location with stationary ones?", "Is it possible to have a generic architecture style that facilitate interoperability across domain specific systems?". These research questions view the problem of lack of interoperability from different angles to provide a comprehensive solution. In summary, the thesis and the included papers address the following research questions:

- Is it feasible to build interoperable IoT systems that utilize multiple options of platforms, network protocol and resource constrained devices using existing middleware solution?
- Considering the large number of devices and global scale of IoT systems that form the Web of Things, is there a better way to organizing the elements of the system to enhance interoperability?
- Given the distinct nature of IoT devices, such as mobile and mostly staying in sleep state, is it possible to integrate such dynamic IoT systems?

- Is it possible to extract generalized IoT systems quality attributes to provide architectural solution for its challenges across a range of domain specific systems?

To address these research questions, this thesis has compiled seven publications that support each other and cover the over all topic envisioned. The following section provides these contributions and how they are organized.

1.4 Contribution and organization

The main focus of this dissertation is on ways of facilitating the interoperation of the networked embedded devices integrated to physical objects in IoT systems. These embedded devices form the building blocks needed to create a global system of IoT systems or Web of Things. To put the contributions in the context of IoT, the degrees of interoperability are discussed. There are different levels of interoperability of systems. The European Telecommunications Standards Institute (ETSI) [99] covers four different levels. The lower three levels are technical, syntactic and semantic interoperability. At the technical level, a message sent by one machine can be received by another one. This is related to the lowest level of the communication infrastructure, such as the frequency of a wireless network or number of wires. This is mostly related to the connectivity side of IoT and this level is not covered by these contributions. Devices with syntactic interoperability can parse the message exchanged in addition to simply exchanging; that is, the format of the message is known by the receiver. The third level of integration is when the devices understand the message context beyond parsing and exchanging. The fourth and highest level presented in [99] as organizational interoperability, which is equivalent to system level integration discussed in the context of this work.

The earliest contribution of this dissertation started by addressing syntactic interoperability through a lightweight middleware (LISA) that is designed for resource constrained devices, presented in Paper I. It has been further extended to port the middleware from the original targeted real-time operating system to another one and published in Paper II. In addition to addressing the syntactic level, this work extends to semantics level and system level interoperability. To provide this, the work contributed a pragmatic IoT systems architectural style (PI), Paper VII, and different components that assist in adopting the style. The first is a domain specific scripting language (DoS-IL), published in Paper III. The second component is a server component (WoVTS), which is presented in paper V, and designed for the Fog layer to enable interoperability as shown in Paper IV. As a typical case of the style and its components, a healthcare domain has been used in Paper VI. The contributions of the thesis are organized as shown in Figure

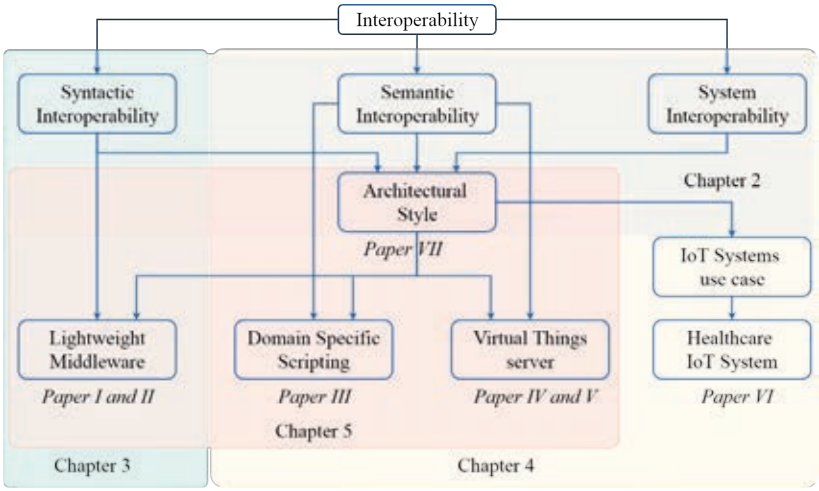

Figure 1.3: Contributions of the thesis organized by chapter and topics

1.3. The colored rectangles show the topics covered by the corresponding chapter and the related contributions are listed under the specific topic. Technical interoperability is not covered in the contribution of this thesis as it involves the low level specifications, such as signal level and frequency of communication, that is out of the scope of this work.

The thesis is organized in six main chapters that addresses the research questions raised earlier followed by a seventh chapter with an overview of the publications included in the thesis. The first chapter has given basics of the research concept, the problem domain and motivation at a higher level. The relation and topics of the next four chapters is shown in Figure 1.3. Chapter two is dedicated to providing the state of the art in IoT, such as standardization efforts, open source contributions, and architecture design. The first and second chapters provide answer to the first research question on the nature of IoT that distinguish it from existing technologies and its critical challenges, focusing on interoperability. Chapter three builds on the previous chapters to describe the middleware contribution of Paper I and II that partially answer the first and second research questions. Chapter four discusses architectural aspects of the thesis and the various components developed, such as a scripting language and a server, that are published in Papers III to VII. Chapter four addresses the last three research questions. Chapter five presents analysis and evaluation of the overall contributions made and how they fit in to the grand view of providing interoperability. The sixth Chapter provides the summary of the thesis with ongoing and future planed works related to the main topic of the thesis.

Chapter 2

State of the art

It is almost two decades since the Internet of Things was first introduced. There has been developments in different aspects since its inception. This chapter provides an overview on the current state of IoT systems to show the gaps and challenges that still exist, thereby highlighting the motivation of the thesis and the individual contributions. It is common to see Internet connected products in different domains such as healthcare and smart home systems. Most of these products come with a *smart* prefix to their names, making the domain also to be *smart*. Unfortunately, most of these devices use various hardware and software platforms, network protocols, architecture and data formats that form vertical system silos. In the following sections, an overview of standardization efforts for the IoT domain, middleware options, architectural proposals and sample implementation approaches are discussed.

2.1 Standards

The abstract device representation shown in Figure 1.2 is a standalone node that has integrated communication, sensing, actuating and identification capabilities. If it functions in isolation, the need for standardization of any part of the system would have been minimal. However, as part of a bigger global interconnected system it has to interact with other objects. Therefore, the devices should follow standards, for instance, at the level of the physical medium, the format of data and its context beside their internal composition. Without similar standards, these interacting objects cannot function with synergy unless other alternative solutions are introduced to bridge the differences. There are many levels of standardization efforts in IoT systems; for example identification, network protocol, architecture, application layer standards and semantic description of devices. Some of these standards are only used as general technical guidelines while others are used

as a means of achieving domain specific legal or certification requirements. For the sake of this thesis however, the focus is only on technical standards related to or affecting interoperability of the communicating devices. A wide range of standards organization and their contribution in different aspects of IoT are well organized by the Alliance for IoT Innovation (AIOTI) in the IoT standardization report [5]. The report catalogued these organizations in two ways. First based on the type of IoT market it address (Business to Business or Business to Consumer) and the layer the standard operates (connectivity, application and services). The second categorization is based on operating domain (horizontal or vertical domains) as shown in Figure 2.1. The existence of such a large number of (sometimes redundant) standards by itself is an indicator of heterogeneity in IoT. For example, in a single application domain there are at least a couple of standards that try to address similar challenge. In reference to the Internet, the existence of common open standards played the biggest role in allowing the integration of smaller networks forming such a global network [92]. From Figure 2.1, it is evident that such a dominant standard is missing in the IoT landscape.To get a closer look at the standards that are relevant for the main topic of this thesis, selected ones that are not specific to an application domain, are discussed in layers starting from device to system level. The layers considered relate to the five layered generic IoT framework discussed by Chou [23]. The framework is composed of Things, Connect, Collect, Learn and Do layers. A related classification method is used in [5] by mapping the standards into classes or knowledge areas. The standards considered in this chapter are not only those released as specification documents, but also specific implementations that serve as standards, such as open source frameworks, operating systems and hardware platforms.

2.1.1 Hardware resources and integration

The first class of standards apply at the lowest level of interoperability while connecting the basic elements of the abstract device shown in Figure 1.2. Some of these are standards that prescribe the computing platform and interfacing of sensors, actuators and unique identifiers. These standards operate in the 'Things' layer of the IoT framework created by Chou [23]. One of the standards organizations in this area is GS1, that promotes standards used in tags and identification using Barcodes and RFID [44]. Another prominent organization with many standards in IoT and other areas is the IEEE [10], working in areas ranging from low level networking specifications to interfacing specification for sensors and actuators. Some of the other types of standard platforms that related to this class are hardware platforms such as Arduino [8] and Particle [76] or real-time operating systems such as Contiki [31] and RIOT [13]. The use of any of these hardware or software

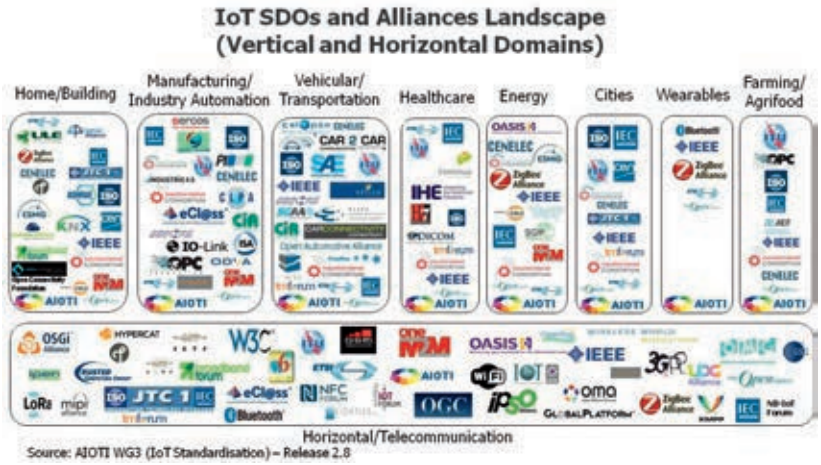

Figure 2.1: Standards organizations in vertical and horizontal domains

platforms restrict the development process in certain ways and serve as a standard for what additional components can be integrated above it. There are many such platforms and standards that constrain the available choices for integration but there is no single dominant standard.

2.1.2 Network interface and protocols

This class of standards concentrate on communication interfaces at different layers; physical, data link and network layers in the Open System Interconnect (OSI) model [103]. Referring the Iot framework [23] again, these standards operate at the 'Connect' layer. One of the earliest standards organizations working in the Internet domain, that also contributes to IoT is the Internet Engineering Task Force(IETF). Sheng *et al.* [89] summarized various open standards developed by IETF targeting the connectivity of IoT devices at various layers. Some of these are on the use of IPv6 over LoPWPAN (Low Power Wirelss Personal Area Network), and a routing algorithm for low power networks (Routing Over Low power and Lossy network -ROLL). Other types of wireless network standards for IoT are presented in [60], which also overlaps with the next class of standards (application layer) in Section 2.1.3. The paper highlights a wide selection of options such as Bluetooth Low Energy (BLE) [90], Zigbee [4], LoRa [2], Sigfox [91] and other cellular options like NB-IoT [45]. The selection of any of these standards for communication in a device relies on the standards followed in Section 2.1.1 for integration and the system requirements. For instance, selecting a specific model of an Arduino to address the requirement of monitoring

the level of a fluid container, guides the available options for the network interface standard. In general, the range of communication, the bit rate and the operating cost filter the available options.

2.1.3 Message and Data formats

Once a network is formed and communication started, the receiver has to understand the information contained within a message. These standards mostly vary with the network interface used. Using the Open System Interconnect (OSI) model [103] as a reference, these standards work at the application layer of the communication stack. There are many contributions in this category, of which majority of the protocols have been well summarized in [86] and [101]. Some of these protocols include CoAP [88], MQTT [73], XMPP [37] and DDS [43]. These standards specify the mode of communication, such as request-response or publish-subscribe, and the message format. Most of the examples given have been in use before IoT and have been customized to support the new requirements of IoT. In addition, the sample protocols above have similarity in the underlying network standard they require - that is IP. However, other set of standards in this category work with a different setting, without a clear separation from the previous standard. Some examples of these are BLE [90], Zigbee [4] and LoRa [2].

2.1.4 Horizontal integration

The standards classified into this category are targeted to address horizontal integration of variations at any of the previous three layers. There are many industrial alliances and research institutes that operate in this category. One of these organizations is the Open Connectivity Foundation (OCF) [74]. The specification has been implemented as a framework called IoTivity, targeting resource constrained devices. Similar concrete framework implementations are covered with more details in the following section discussing middleware. Focusing back on the OCF specification, it also covers the structured description of device resources. Similar efforts are also contributed by the World Wide Web Consortium (W3C) [28] and other contributors such as Mozilla [29]. Other standards in this category are from OneM2M [58], Zibee dotdot [3] and the OMA SpecWorks [94]. The IETF also has specification on the format of representing sensor information [52]. In addition to providing specification for methods of describing the physical objects, such as IPSO smart objects from OMA SpecWorks, device management and overall interoperability between different protocols is the focus of these standards. For instance, some of the specifications from OneM2M include different protocol bindings for integration similar to the one proposed by the W3C [27].

2.1.5 Cross-cutting and system level

The standards included in this category are those addressing common tasks shared by the previous layers. These include security and privacy, configuration, update and device management, and overall system architecture. Some of these standards are results of public projects and industrial alliances in addition to standards organizations. Many European Union projects have contributions in this area. An early effort is the Internet of Things Architecture (IoT-A) project [49]. A related contribution that is mentioned earlier is also the AIOTI [6] High Level Architecture. In addition, W3C also participates in the area of overall architecture of IoT [26]. Another alliance that participates in the standardization of Fog computing is the Open Fog Consortium. This reference architecture covers all the key components required for proper Fog operation in eight categories referred as pillars [75]. The IEEE standards association also covers architecture as part of its extensive list of specifications [11].

The standards discussed so far in this section are just a small part of a large pool of specifications that sometimes overlap and make the selection of a specific one very challenging. It is also obvious that companies having competing products that are based on different platforms try to enforce the ones they use as the industry standard. In addition, alliances and standards organizations that produce newer standards to bridge others also end up becoming just another entrant into the race. There are vendor specific Cloud based platforms [82] [102] that did not fit in the classification discussed in Section 2.1. These platforms also constrain the types of components that should be used at lower layers to communicate with it. However, this dissertation concentrates only on ways of generic integration methods of IoT systems components that operate below the Cloud tier. Therefore, platforms, standards, middleware and architecture proposals at the Cloud layer are not discussed. In summary, the lack of dominant standards in each layer added to the varying requirements of IoT systems has created a fragmented IoT landscape. To enable interoperability across these fragments, different alternatives are considered. The following sections provide the current state of these options.

2.2 Middleware

Components built using a common standard will work by design. However, the current state of IoT is that most standards are limited to only a certain range of application domains or set of components. This has resulted in heterogeneity of devices, platforms, protocols and formats. The need for a middleware arises in such circumstances where two distinct standards need to operate together. For instance, two devices using different application

layer protocol or network interface require a middleware to bridge the two sides. This has been a common approach for integrating traditional enterprise applications [46]. Similarly for IoT, various middleware options have been developed by industrial alliances, research institutes and open source organizations for different use cases. Part of these offerings are the implementation of the standards specified in Section 2.1.4. For instance, the specification of OCF [74] is implemented in IoTivitiy framework to provide interoperability. A long list of open source components and middleware from Eclipse IoT community [42] is organized and presented in three layers; device, gateway and platforms. A more generalized and extensive list of middleware options, classified based on the design approach, is compiled by Razzaque *et al.* [83]. It also provides the functional and non-functional requirements that are addressed by the middleware. There are many reasons driving the development of a middleware [14]. For brevity, the middleware discussion in this thesis is limited only to those addressing interoperability at different levels.

2.2.1 Basic integration

Integration of systems involves achieving multiple degrees of compatibility. The levels that can be attained through software components are covered in this and the upcoming sections. The middleware or frameworks discussed in here are those used for translation of message format as discussed in Section 2.1.3. These frameworks can be open sourced or as commercial offerings. About seventeen commercial frameworks have been organized based on the layer they operate in [30]. These include cloud based frameworks or those used in gateway or during embedded device firmware development process. In addition, most middleware in [83] are also addressing basic syntactic or technical interoperability. A typical example is presented in [34] where such a middleware is used to abstract a Zigbee and RFID message using protocol translation. One of the limiting factors in such middleware is the number of supported communication protocols. Each protocol requires a module that is used as adapter for applications to connects to it and most middleware support few protocols and exclude others. Moreover, the architecture used by the middleware determine certain application behaviours, such the mode of communication, scale and performance. These middleware also constrain the choice of underlying platforms as they are ported to only a subset of these platforms.

2.2.2 Semantic integration

More enhanced type of middleware, which are able to provide technical, syntactic, and semantic interoperability are also widely available. However,

the degree of support for semantic integration varies. One of the widely known frameworks in this category is the implementation of the OCF [74] specification, called IoTivity [50]. In contrast, the semantic interoperability framework developed by Song *et al.* [93] in Fujitsu Laboratories tries to bridge different standards at the application layer. Another EU project on IoT is the INTER-IoT project that targets platform integration [48]. A detailed analysis of the project on the support of semantic interoperability is done by Ganzha *et al.* [39]. Most middleware that support semantic interoperability tend to incline to a specific application domain as the ontology used by the systems is usually from a single domain, such as healthcare. However, there are also generic approaches as well. For instance, to represent an abstract device in the context of IoT systems, a simplification of the Semantic Sensor Network ontology [24], is proposed in [16]. In addition, methods to dynamically translate a thing description to a Resource Description Framework (RDF) have also been proposed [80]. These efforts enable cross-domain linking of devices descriptions belonging in different domains, leading to a scalable integration.

The list of middleware covered here are just a small subset of a vast pool of alternative solutions. These middleware also enforce certain specific approaches and design philosophy on the overall IoT system. In cases where the design approaches of the middleware used by two distinct systems is different, it leads to more fragmentation. Therefore, some initiatives use a more general approach of specifying generic system architecture in which the components can be organized to enable system integration in addition to other system qualities.

2.3 Architecture

The standardization efforts highlighted previously in Section 2.1.5 has touched on the topic of architecture related standards. This separate section is dedicated for architecture to serve as motivation for one of the main contributions of this thesis; the architectural style in Chapter 4. The first architecture surveyed is that of AIOTI. The alliance has developed a High Level Architecture (HLA) for the Internet of Things [6], which resembles the reference architecture of IoT-A [49], providing domain and functional models. These kind of generalized architecture proposals are designed to serve as a guide to identify the main components of the system and proceed with a concrete architecture design [49]. Some alliances, such as the Industrial Internet Consortium (IIC) have also proposed domain specific reference architecture [25]. These and other architectural contributions from Europe are summarized in [56]. Other specific types of architectures are also proposed for the Internet of Things. Some of these proposals are simple layered styles with three,

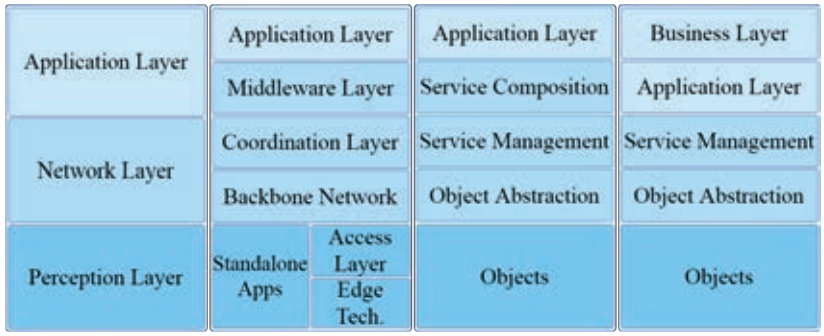

Figure 2.2: Comparison of layered IoT architecture proposals in [1]

five or six layers [1], [100], [54] as shown in Figure 2.2. In comparison to reference architecture proposals, such recommendations are very limited in representing generic attributes of IoT and do not provide much information during detailed architecture design. Therefore, more generalized proposals with more detailed specification are favored in this thesis. Another more generalized architecture proposed by IETF [97] provides an overview of the interactions modes of different smart object components over a network. These interaction patterns are considered in the evolving architecture proposal of the W3C [26]. In addition, the Open Fog Consortium also provides a reference architecture for the organization of the major elements of the Fog computing layer [75]. Even though there are good architectural references that serve as a guide for IoT systems implementation, they are either too open (abstract) leading to divergent systems or too closed to address various domains. In addition, most of them are not simple enough for understanding and communication among developers. Therefore, IoT system developers use different architecture styles and patterns that are inherited from platforms, middleware or back-end system they rely on.

2.4 Implementation approach

The previous sections presented the current state of the Internet of Things landscape. This section is a more practical and typical scenario, mostly the author's perspective and experience, of the implementation procedure of IoT systems. It includes an overview of practical challenges of an IoT system. Understanding the important properties of the overall system, especially the perception layer (interaction with the physical object), provides the criteria needed for selecting the hardware, software and communication components. To extract these properties in application domain agnostic manner, a simple

Table 2.1: IoT systems classification approach

Criteria	**Categories**
1. Proximity to hub	Short range communication
	Long range communication
2. Network interface	Wired network
	Wireless network
3. Data generation	Continuous (streaming) data
	Intermittent (regular interval)
4. Size of data and speed of transfer	Low data rate requirement
	High speed requirement
5. Source of power	Battery operated device
	Plugged to outlet device
6. Ownership	Private (consumer device)
	Industrial application
	Public (open data) usage
7. Portability	Mobile device
	Stationary device
8. Operating mode	Collection (monitoring) only
	Control (dual)
9. Integration mode	Embedded inside
	Wearable device
	Attached on object
10. Real-time needs	Hard (strict) requirement
	Soft (relaxed) requirement

set of criteria are listed in Table 2.1 with their corresponding possible alternatives. Deciding the device's categories for each criterion clarify the decision points to be made later, during specification phase [69].

As a sample use case, the following scenario is assumed: a machine and its operating environment are monitored to predict the next maintenance time. Figure 2.3 shows the example scenario. The machine has an external device with a magnetometer sensor and the room has a separate device that monitors the room temperature and humidity of the operating environment. In this scenario, it is assumed that the machine of interest and the environment sensing device are located few meters apart. Using Table 2.1 to identify the properties, the following attributes can describe the system: short range between machine and room monitoring device (acting as hub), wireless network (typical case), regular data interval, low data rate, battery operated, industrial application, stationary device, for monitoring purpose, attached to machine and has relaxed time constraints. These attributes constrain the possible options suitable for the system implementation; for

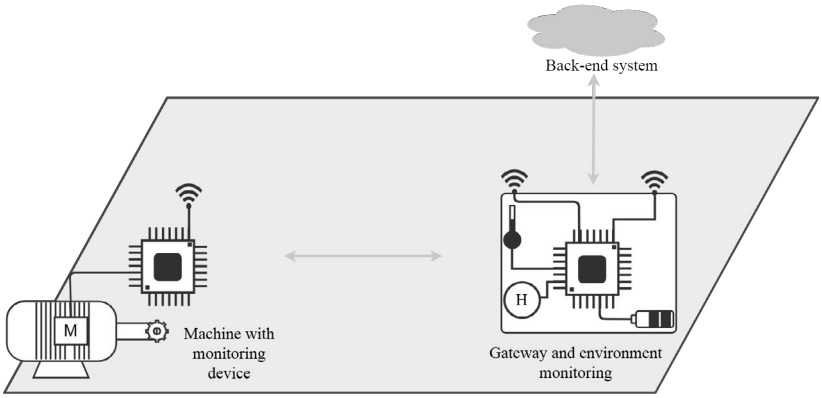

Figure 2.3: Sample IoT scenario to monitor a machine in a room

instance, the communication between the device attached to the machine and the hub can be achieved over Bluetooth low energy (short range and wireless). Using the functional specification of the system and the standards in Section 2.1.1 and 2.1.2, the selection of hardware and software components proceeds. The overall system architecture design is also a critical step in identifying other architectural elements and their interaction.

There are a wide range of hardware and software components that require early design decision to continue with the implementation of the two main embedded systems components. The first is the device that has a magnetometer, a Bluetooth low energy (BLE) module and a small micro-controller. A developer has to decide on the type and model of the processor based on the available interfaces for the peripherals, while also considering the required performance, energy consumption and available software tools and platforms for development and integration. In addition, one has to decide if there is a need for a real-time operating system or work on bare-metal and understand the sensors and communication protocol to collect the data. Understanding the communication protocol stack is also mandatory for proper networking of the embedded systems. Once the integration is done, formatting the message in accordance with the specification of the receiving end is mandatory. The second embedded system that collects the data sent over BLE is the one acting as a hub or gateway. It has the responsibility of temporarily storing the data, analyze the data and finally send it to the cloud. Similarly, the gateway also requires careful consideration of the operating system, database used for storage, analysis techniques, network interfaces and drivers. The remaining parts of the system, the back-end to support the system and the user interface component, are not discussed here for brevity, besides the fact that these concerns are outside the scope the thesis. After

the completion of the software development to achieve the above functional and non-functional requirements and the devices are installed, updating a simple feature in these embedded systems is a challenge - specially in the case of the device attached to the machine that is under monitoring. As the number of these devices increase, management becomes a nightmare. In addition, extending the feature of the system with additional devices involves re-imagining the overall system. In general, the finalized system ultimately becomes a vertically isolated system that is difficult to integrate with other Systems. This is a typical approach that resulted in fragmented, heterogeneous IoT systems.

2.5 Summary

This chapter provided the current state of IoT systems from standardization, middleware, architecture and implementation perspectives. In general, considering the wide range of variability in selected hardware and software platforms, network protocols, data format, overall system architecture, and the lack of dominant standards in each category, the current state of IoT systems requires a systematic approach to addressing the lack of interoperability. The contributions of this thesis aim to address some of the challenges in building an interoperable IoT system. In summary, the aim of this chapter was not to provide a complete list of contributions and process of implementing an IoT system, but it is dedicated to showcase the distinct features in implementation of IoT systems in contrast to an Internet application, such as e-commerce. The same is true for the dissertation; it concentrates on the networked embedded systems side and their inter-operation, where the main differences exist compared to traditional Web based applications.

Chapter 3

Interoperating: Middleware approach

A middleware is a component that resides between distinct components to provide certain services that facilitate interaction. A classical approach towards addressing systems integration in enterprise applications domain is the use of middleware. Considering the interoperability needs of IoT systems, there are different layers where a middleware can be used. A subset of these alternatives for the Internet of Things is discussed in Chapter 2. The focus of this chapter is specifically on one of the contributions of the thesis in using a middleware to provide interoperability for IoT systems components that reside in resource constrained embedded systems. In the following sections, an overview of the resource constraints of IoT devices will be presented along with the unique challenges the constraints bring to the design of a middleware. It continues further highlighting the contributions made in Paper I and Paper II to enable integration of IoT systems using a middleware approach.

3.1 Resource constrained devices

The majority of IoT devices that are getting connected to the Internet differ from traditional computing elements mainly in the limitation of the available resources. The IETF [21] categorizes those devices with severe resource constraints into three classes based on the size of the available RAM and Flash (storage). The best of these classes have a maximum of only 50KB of RAM and 250KB of flash storage. Compared to a typical computing device that is traditionally connected to the Internet, these devices require state of the art methods to enable them get connected to the Internet. Identifying these constraints provides a baseline in the possible approaches to integrate them with others, in addition to providing the required functionality. For in-

stance, the processing speed of a typical embedded device (less than 50MHz) to read a sensor as compared to the processor speed of a traditional Internet connected device (more than 1 GHZ) is very low. In addition to the processing power, the amount of available storage (for instance a flash size of 250KB), and the data rate of the network interfaces attached to the embedded device is small. Moreover, most of these devices reside far from a power line and operate with a battery that has to last years. The variations in network protocols, platforms and data formats make the integration of these resource constrained devices even more difficult. Most of these variations are the results of efforts to address the resource constraints of IoT devices and networks. For instance, specialized application protocols, such as the Constrained Application Protocol (CoAP) [88] and MQTT-SN [95], have been designed to address such constraints. The following sections present the design and implementation of a lightweight middleware that is targeted to address interoperability of IoT systems within the above resource limitations of its components. The details of the middleware and its extended features are published in Paper I and Paper II.

3.2 A lightweight middleware

Enabling resource constrained embedded devices to inter-operate across systems, platform, protocol and standards boundary requires the introduction of a middleware to the design. One of the earliest contributions of this thesis is the introduction of a lightweight interoperability middleware, Lightweight IoT Service bus Architecture (LISA) [65]. LISA is designed to facilitate the inter-operation of IoT devices across network protocol and platform boundaries. It is inspired by an open framework designed by Nokia and an open source reference implementation of the framework called Network on Terminal Architecture (NoTA) [18]. It follows a Service Oriented Architecture (SOA) to organize the different components of the system into application nodes and service nodes. The original target of the specification was to facilitate the integration of mobile device components during new product development. This is accomplished by making any mobile device component, such as camera, to act like an application (client) or service node, and other elements to consume the service exposed. The interaction of application and service nodes is handled through a custom developed stack called Device Interconnect Protocol (DIP) [18]. A closely related specification has been adopted in the development of LISA; it follows the SOA approach in NoTA and organizes the interacting devices as application or service nodes. In addition, it also assigns a manager node that serves a gateway or central node to application and service nodes. Sub-networks of manager, application and service nodes are hierarchically organized in

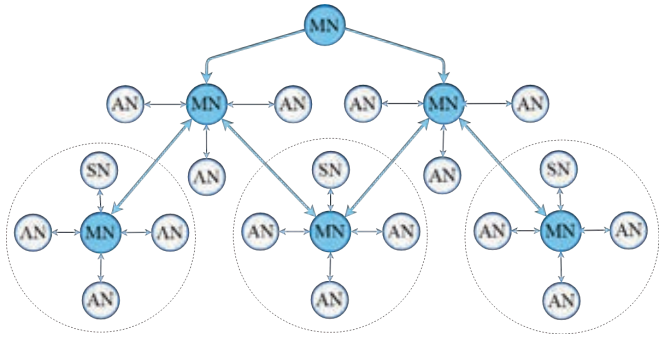

Figure 3.1: Federated organization of nodes in LISA.

a federated architecture as shown in Figure 3.1. The manager node is a device with relaxed resource constraints than the other two types of nodes that interact with physical objects. The assignment of nodes thus far, as application or service node, is a simplification. However, in reality a device can host both types of nodes. For instance a device with one sensor and an actuator - the sensor can behave as service node where as the actuator component is an application node. In addition to the assignment of nodes, LISA also implements a simplified protocol of NoTA. Service nodes first send registration request to the manager node, which keeps a registry of all available services with detailed information. An application node interested to access a specific service first sends a service discovery request to the manager specifying the name of the service. If the service is already registered in the manager, the service information is sent to the application node as a discovery response. If a service information is delivered to an application node, it gets authenticated by the service and the two nodes communicate, as depicted in Figure 3.2. This process is possible only if the two nodes have similar network interfaces. However, if an application node and a service node are using different protocols, all communications pass through the manager node [65]. In general, the interaction of the application, service and manager node use a broker pattern where the manager acts as a broker. In addition to providing interoperability between application and service nodes over different network protocols, the middleware simplifies the development process by abstracting the details of the underlying network interface stack.

3.3 Topology and architecture

The federation of nodes as shown in Figure 3.1 is based on two factors: the location of the manager node (MN) relative to other leaf nodes (Ap-

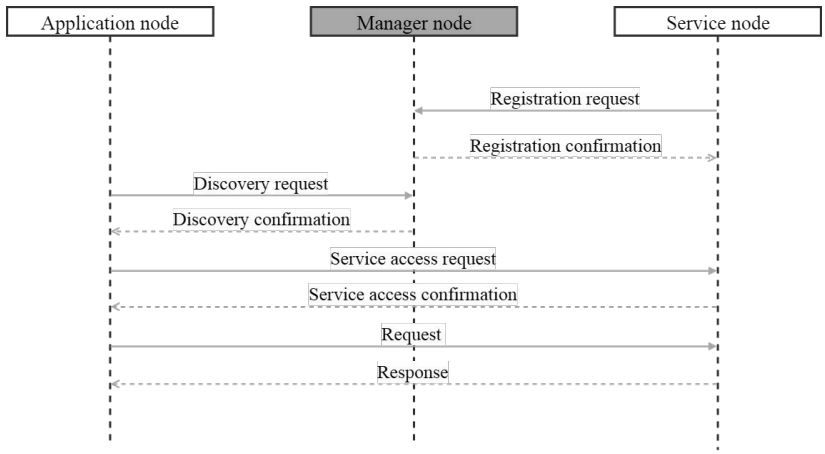

Figure 3.2: Interaction between nodes and the LISA protocol

plication nodes - shown as AN and Service nodes - shown as SN) and the available resources of the manager node. The dotted circles around a group of application and service nodes attached to one manager node in Figure 3.1 indicates the locality of the nodes, for instance in a smart home as hub and smart appliances. As briefly mentioned previously, two or more nodes can also reside in a single device having multiple sensors and actuators. The nodes form a star topology with the manager node at the center. In addition, the interaction between manager nodes is also arranged in a star topology where the center node is a manager node at a higher layer in the hierarchy. The overall topology formed becomes star of stars with a manager node in the cloud acts as the central node (the top most node in Figure 3.1). Top level manager nodes connect to the cloud node forming the inner most star, which is extended by next level stars formed around the top level manager nodes. This topology evolves in the next chapter to form a mesh network of the manager nodes that are represented as gateways in the Fog computing layer [20].

The first contribution (Paper I) on LISA [65] also presents an overall architecture of IoT systems based on the node oriented organization of the resources available in physical objects. The node centric networking (NCN) idea of LISA was inspired by content centric networking [51], a type of information centric networking [77], [19]. The basic principle behind it advocates the treatment of the main content transferred between devices to be at the center of the design. In contrast to existing methods where the computing devices (hosts) are the central elements, content centric networking uses addressed contents to be routed to nodes with interest of that specific content.

Similarly, LISA uses nodes as main components of the communication network by giving unique addresses for each node. Addressing of nodes follows a specific pattern as:

$$NodeAddress : DomainID|HomeManagerNode|NodeType|NodeID$$

A typical example of an address constructed using this pattern is concatenation of 0x01 (Smart home domain) followed by 0x00(no manager node for the device), 0x01(type of manager node) and finally the unique sequential number (0x01). Therefore the node address will be 0x01000101. Other nodes will register to this manger node and inherit its node Id as their home manager node [65]. These addresses are generated by manager nodes for their child nodes during service registration or service discovery process (see Figure 3.2). The federated arrangement of nodes and the hierarchical addresses help during the discovery of services that are not available in the home manager of the requesting application. For example, consider an application node with address 0x01010301 (home manager address is 0x01) requesting to access a service node with 0x01020201 (home manager address is 0x02). The manager node with node id 0x01, containing the application node, contacts the other manager with address 0x02 to pass the discovery request for the service. Detailed description of the routing algorithm through the manager nodes is given in [65].

3.4 Portability of LISA

The main purpose of the middleware is to enable interoperability of IoT systems across platform, protocol and system boundaries on devices with resource constraints. To allow this, the middleware is designed in such a way that the core functionality and protocol is built as a generic layer over an abstract interface that interacts with the underlying network protocol stack and embedded real-time operating system. As a proof of concept, it was originally developed on RIOT-Os [13] and a 6LoWPAN based network stack. To extend and evaluate the portability of the middleware, LISA was also adopted to Contiki-os [31]. A representation of two platforms with different network interfaces, with LISA middleware used for inter-operation is shown in Figure 3.3. The figure also indicates the internal structure of the middleware to accommodate different platforms and protocols. Application and service nodes interact with a uniform interface exposed from the core module of LISA. The core module initializes the required network stack and passes requests to system level functions in the specific real-time operating system implementation underneath it. The overall structure of the middleware is organized in a strict layered style where upper layer components can communicate only to a layer immediately below it. In Figure

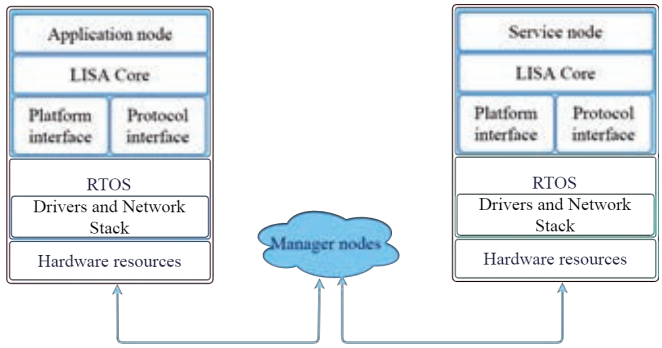

Figure 3.3: Interacting nodes in different platforms using different protocols

3.3 the manager node is represented as a service in the cloud, to simplify the hierarchical structure of manager nodes passing requests made across heterogeneous standards. It is mandatory for manager nodes to have support of multiple network interfaces to be able to function as bridges between distinct protocols.

3.5 Scientific contribution of LISA

The overall features and advantages of the lightweight IoT middleware has been covered in this chapter. This section covers the main contributions of the papers related to the LISA. The original paper [66], not included in this thesis, was initially published to show the concept of LISA and the result of simulation utilizing the middleware. This has been further extended to include the idea of a Node Centric Networking (NCN) with LISA and the routing of messages accross multiple manager nodes [65]. Due to the potential of the middleware, a project was proposed to extend the middleware further at the University of Turku and Åbo Akademi University with two years funding from the Academy of Finland. During the first phase of the project, LISA was ported to Contiki OS and the results of the work was published in [72]. During the project time line, there has been changes in the state of IoT middleware landscape both in research and industrial communities; for instance, the introduction of Iotivity [50] and merging of Allseen alliance and Open Connectivity Foundation (OCF) [74]. This has opened new opportunities for the project to pivot to exploring novel approaches that utilize these types of middleware, while addressing the same IoT systems challenge - interoperbility. These approaches are explored in the coming chapter in detail.

Chapter 4

Interoperating: Architectural approach

Architecture plays a significant role in the overall characteristics of a system. A detailed architecture specifies the individual components that form a system, the development patterns, programming abstractions, and interactions among other things. It is a means by which the functional and non-functional requirements of a system are brought to a usable system. In contrast, architectural styles are a set of constraints that are enforced on components of a system and the possible interactions among them to achieve certain quality attributes. Styles allow understanding of complex systems and facilitate communication among developers and stakeholders. In addition, architectural styles provide interoperability of systems by design. In the following sections, architectural concepts are discussed, architectural style for IoT systems is derived, and different enabler components contributed in this thesis are presented.

4.1 Architectural concept

The significance of architecture design becomes obvious as the system complexity increases. This discussion of architecture is simplified by taking examples from building construction. For instance, simpler application can be developed with sequential flow of activities without the need for architecture design as it is so for simple constructions, such as a dog house. Similarly, bigger buildings require architectural design of the structural, electrical, sanitary, the interior and exterior of the building among others. The same level of detailed architectural design is required by systems of bigger sizes. IoT systems, being one of these types of complex systems, require careful design of the architecture. This section provides strict architecture terminology that will be used in subsequent sections. Most of these terminologies are

borrowed from Perry and Wolf [78] to be consistent with most architecture related publications. Beginning with architecture, it is the identification of the system components and their composition to provide the desired functional and non-functional system requirements. Perry *et al.* [78] proposed a model of software architecture as a collection of system components or elements and their relation or form guided by certain requirements or rationale. It also identifies three main types of elements: processing, connecting and data elements. In contrast, an architectural style is more abstract and generalization of the relations in various architectures [78]. Fielding [36] describes architectural style as a collection of constraints that apply to the components of a system and their interaction. Revisiting the buildings analogy, while architecture specify detailed guides on the structural, electrical and look and feel of the interior and exterior of a building, architectural style describes the building architecture as Gothic, modern or classical which provides ease of understanding at a higher level of abstraction. The last terminology used to describe architecture in this thesis is a 'view'. An architectural view describes the design of a system from a specific perspective, which represents a stakeholder of a system. As a system has multiple stakeholders, it is described using multiple architectural views. For instance, four views are presented by Kruchten [55] in the 4+1 view model for architecture discussion, each addressing the corresponding concerns of the system's stakeholders. In the following sections, the architectural style derived for IoT systems is discussed using the terminologies presented thus far.

4.2 Pragmatic Architectural style

IoT systems are one of the most complex systems that require a wide range of development tools, standards, techniques and professionals as described in Chapter 2. In the simplest case, it requires embedded systems developers to program the devices that are integrated to a physical object, front-end developers to design user interface for the application and back-end developers for the storage and analysis of the data collected. In general, it is inherently distributed over different types of networks and wide range of devices. This complex system can be simplified and easily described if there is a widely used set of architectural styles. This thesis contributes in this regard towards the derivation of a pragmatic architectural style for IoT systems. The process starts assuming the overall system as a standalone application to provide a generic function that represent IoT systems. Fielding [36] refers this model of the system, where no style is applied, as Null style. The components of this initial abstract system will be identified and in trying to fulfill the definition of an IoT system and address its challenges, architec-

Table 4.1: Software architectural styles

Usage scenario	**Architectural styles**
Distributed systems	1. Client-server
	2. Space based
	3. Peer-to-peer
	4. Broker
Structure	5. Pipes and filters
	6. Layered
	7. Monolithic application
Messaging	8. Event-driven
	9. Publish-subscribe
Shared memory (repository)	10. Blackboard
	11. Rule-based
Mobile code	12. Remote execution
	13. Code on demand
	14. Mobile agents

tural styles will be applied to it in steps. The result of the final stage will be the list of components (processing, connecting and data elements) and a set of architectural styles that are given a collective name for simplicity and to uniquely identify it from others. Garlan and Shaw [40] present two ways of combining architectural styles to form derived heterogeneous ones. The first approach is to follow from higher abstraction to lower, applying appropriate styles for outer and each internal component. This approach is used here in the beginning stages to identify the core elements of IoT systems and their interaction. The second method is to use combination of styles in a single component, identified in earlier stages, to achieve desired system properties. To apply styles or combine them, it is necessary to provide an overview of existing common styles, shown in Table 4.1, that are usable in subsequent sections.

4.2.1 Origin: monolithic system

The initial stage in the derivation of an architectural style is to assume the whole IoT system as a single program on a device, without any layers internally. This is equivalent to an embedded systems software without any style that reads sensors and based on the value of the reading instructs an actuator on the same device. Figure 1.2 is an abstraction of such a system on a single device. However, this is a hypothetical system that is not inline with the definition of IoT systems; it is not even distributed over a local network. Moreover, the majority of devices are resource constrained

Figure 4.1: Applying client-server style to the origin

to store the collected data and process it. In addition, the model above is generalization of at least two main purposes of such devices; monitoring and control. With the monolithic model, it is not possible to achieve monitoring function unless there is another component of the system closer to the user to present the collected data. In the next step, this drawbacks of the model will be addressed by applying a client server style to the original system model.

4.2.2 Stage one: Client-server

Given the drawbacks of the system in Section 4.2.1, different architectural styles will be applied on it to achieve the requirements. To provide access to the user for monitoring and control of the device over a network, the functions of the system are split in two concerns. Therefore, two processing components will be identified in this step: first is the interaction with the physical object and the second is the interaction with a user. This separation of system concerns is achieved by applying Client-server architectural style from Distributed systems usage scenario of Table 4.1. In a client-server style, clients consume services provided by the server component using remote procedure calls [40]. In this model of an abstract IoT system that is partitioned into client and server, the server is the embedded systems device interacting with the physical object, as shown in Figure 4.1. There are some architecture proposals for IoT systems, which resemble this style in their simplified forms, given the interaction of the client and server is done through the Internet. For instance, the three layered architecture summarized in Figure 2.2 or the basic IoT system presented by Khan *et al.* [53] can be simplified to two layers excluding the infrastructure. This simplification makes the architectures similar, except the partitioning of the functions of the two layers. Even though applying the style enhanced the system in allowing the remote access of the device and the system looks like an IoT system, the resource constraints still remain unaddressed. In the next stage, the constraints will be considered by applying more architectural styles.

4.2.3 Stage two: Client-server

One of the distinct features of most IoT devices is the limitation of the resources compared to traditional Internet connected device, such as personal

Figure 4.2: Applying client-server style again on the embedded systems side

computers and mobile phones. Some of these limited resources are the battery life, processing power and storage capacity of the devices, which are also presented in detail in Chapter 3. Considering the model in Figure 4.1, there are many limitations of the systems. First, the overall scaling of the system is limited as the embedded system cannot handle lots of concurrent user requests. The second restriction is in the amount of data stored and processed in the embedded system. Another obvious problem is, for the embedded device to provide a reliable and always available service, it has to always be in listening mode. This results in consuming too much power, resulting in shorter battery life-time for those devices running on batteries. In addition, the embedded devices use a wide range of network protocols, making it difficult for the client device to support all of these protocols. Therefore, separating the processing, storage and handling of user requests from the function of interacting with the physical world forms two separate layers; a client and server. The role of the embedded system component in this model is as a client, accessing the services provided by the back-end system as shown in Figure 4.2. The model in this stage has better scalability to handle as many users and large number of embedded systems. It also takes the burden of storage and processing from the resource constrained devices. However, this model is not generic enough to represent systems that have low latency requirements and mobile devices, referring the Table 2.1. In the next stage, these types of time critical IoT systems will be addressed.

4.2.4 Stage three: Layered Client-server

Bonomi *et al.* [20] proposed Fog computing to provide network, computing and storage closer to the devices interacting with physical objects. This novel approach enables low latency communication for time sensitive applications in Networked Embedded Systems (NES). In addition, Fog computing layer provides services that are relevant to address the challenges in the networked embedded system. For instance, to support various network interfaces used by the NES and provide mobility services [81]. The introduction of this computing tier forms a Layered Client-Server style where the NES is a client of the Fog Computing System (FCS), which is in turn a client for the Back-End System (BES). The architecture style at this stage is complete enough to address most quality attributes required by IoT sys-

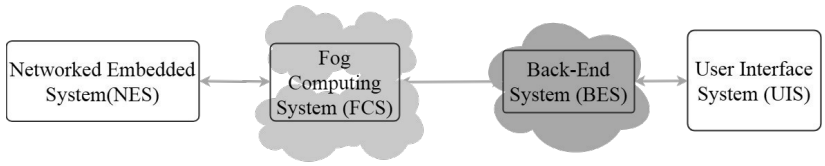

Figure 4.3: Leveraging Fog computing in IoT systems

tems. At least four processing components have also been identified during the previous stages; a networked embedded system component (NES), Fog component (FCS), Back-end component (BES) and finally the User Interface Component (UIS) as shown in Figure 4.3. However, the connector components have not been clearly characterized to assist in addressing the requirements of IoT systems. For instance, the above model does not explain how the resource constrained devices can be made visible over the Internet and also the mode of interaction of two or more NES components. In the following section, the model will be further elaborated adding constraints on the connectors of the processing components.

4.2.5 Stage four: NES internals

The preceding stages enable IoT systems to operate with the resource constraints of devices containing the processing component labelled as NES by migrating all those functions that can be executed remotely on advanced hardware. One of the advantages of this separation is to enable the NES component to operate only when necessary and remain in sleep state otherwise for longer battery life. However, this solution brings the challenge that the state of these devices will be unknown or inaccessible while they are in sleep state. In addition, mobile devices can change location and tracking last known position is necessary. A suitable component to retain this state information is in the FCS, which is operating as a server to the NES. In addition, the client-server communication between NES and FCS is over heterogeneous protocols. As a solution to the syntactic interoperability challenge of IoT systems, a uniform interface is proposed as part of this thesis work in [67]. This is a connector component that introduces a constraint on the interaction of NES and FCS. It effectively hides the details of the underlying network protocol by utilizing a middleware, such as the one contributed in this theis [66].

The implementation of a typical IoT system has been highlighted in Chapter two, while presenting the state of the art in different aspects. It presented the issues to be taken into consideration while developing the NES component that is responsible for the interaction with a physical object.

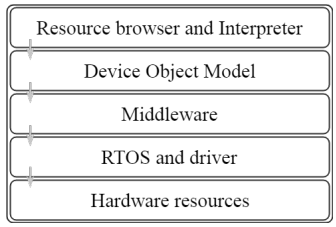

Figure 4.4: Internal layered organization of a networked embedded systems

Considering the possible deployment environment of the NES component, which might not be easily accessible, updates to the component have to be done over the air. However, in comparison to the size of a typical NES component with the bandwidth of most network interfaces used, it is not feasible to make frequent updates. One of the contributions of this thesis is to use a code on demand style to simplify reconfiguration and maintainability of the NES component [68]. The details of the scripting language designed are discussed in Section 4.3. Concentrating on the architectural style, a code on demand approach (from a mobile code type in Table 4.1) enables scripts stored in the server to be transported to the client where it can be interpreted and executed. However, mobile code approach introduces a security risk. To overcome this vulnerability and get advantage of code on demand style, the internal components of NES are organized in strictly layered architectural style to restrict access to critical layers [69]. An Application Programming Interface (API) is exposed to the Interpreter sub-component, hiding the underlying details. The layers under the middleware can optionally be organized in other alternative ways as necessary. Figure 4.4 shows the layered organization of the internals of NES component with typical layers. The use of a middleware and uniform interface enable the NES to inter-operate with other NES components using different network interface via the FCS [81] [70]. In addition, the use of 'thing descriptions' as part of the IoT model, which is shown as Device Object Model layer in Figure 4.4, enables the NES to understand basic semantic information when exchanging information with other devices. More information on this topic is presented in Section 4.4.

4.2.6 Stage five: FCS internals

The overall deployment tiers of the components of IoT systems have been outlined and the details of the NES component and its connector with the FCS have been completed. This stage analyzes the FCS internal structure to provide the required services for its clients. A logical extension to the previous stage is the uniform interface on the connector with NES. Devices

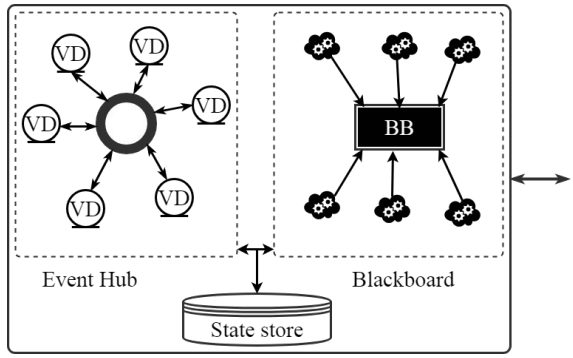

Figure 4.5: Combination of styles in FCS internal organization. Virtual devices (VD) in an event hub and data analytic services interacting over a Blackboard (BB) style

in the Fog layer are usually composed of multiple network interfaces to handle requests from their clients in the perception layer (interacting with the physical world). The use of a uniform interface makes the variations in the network interfaces fade out allowing a developer to focus on the main function. Another required feature introduced during the design of the NES is the retention of the state of its resources (introduced in Section 4.2.5). One of the contributions of this thesis represent the state information of the devices with a virtual counterpart in the FCS memory [70]. In addition to retaining the state and providing interoperability over a uniform interface, other benefits of using Fog computing layer include localized processing of the data, support mobility and faster notification to events of interest [81]. Inside the FCS, the interactions among the virtual representation of the states of devices, are made using an event-driven approach (from messaging type in Table 4.1) in [70]. Moreover, to handle the data processing needs in the FCS, a blackboard approach from the shared memory category in Table 4.1 is optionally used. Depending on the data processing requirements of the application domain, other alternatives can be considered as well (such as rule based or pipes and filters) [40]. Combining the above architectural styles to address various challenges, the internal organization of the FCS is shown in Figure 4.5 [69]. One of the missing function of the FCS is the support of mobility of devices running the NES. To support that, replication of the state information to the neighbours of the gateway is necessary. The federated organization of the gateways in the Fog layer [75], as also used in the federated case of LISA [65], helps to determine the possible number of gateways to replicate the state information. This communication with other gateways can be accomplished optionally using a peer-to-peer style.

In one of the contributions of this paper [70], the replication of the state information is done using REST style. At this stage, the inter-operation of Networked Embedded Systems is possible and the challenges of most IoT systems can be mitigated. In the final stage, the remaining parts of the complete architectural style is discussed and combined with the results of the previous steps.

4.2.7 Final stage: combining styles

The final stage of the derivation is to analyze the interaction of the BES with FCS and BES with UIS. These interactions occur over existing Internet infrastructure and mostly through the Web. Therefore, using an existing style is preferred over trying to redesign a working system. The Representational State Transfer (REST) [36] architectural style is a dominant style used for the World Wide Web. After the first introduction of the style and its wide use in the Web for almost two decades, Fielding *et al.* [35] have shown the impact REST has, its shortcomings (for instance in handling push notifications) and other related styles inspired by it. Some of these styles are Computational REST (CREST) [33] and Computational State Transfer (COAST) [41] that address specific requirements. Similarly, this design of an architectural style is also partly motivated by REST, to address specific requirements of IoT systems. Using the REST style to complement the combination of the styles identified in previous stages provides the full picture of a more practical IoT systems architectural style. It is the combination of REST with the previous styles that is referred in this thesis as Pragmatic IoT systems architectural style or PI [69]. In general, it identifies six processing components, four connectors and five data elements of the style as presented in [69], excluding those elements covered by REST styles. The constraints, topology and quality attributes of the overall style are covered in Chapter 5. The final view of a generic IoT system that utilizes the PI architectural style is shown in Figure 4.6 [69]. The figure shows the all the simple styles picked on the way, such as the internal layered style in NES, uniform connector between the NES and FCS, the combination of styles used in the FCS, and the REST style on the right most side. In the remaining sections of this chapter, the contributions of the thesis that facilitate the adoption of the PI style are covered.

4.3 Domain specific scripting

To address the unique challenges of IoT systems and find a working architectural style, the Networked Embedded Systems (NES) layer has been at the center of this thesis. One of the architectural styles proposed is the use of code on demand approach to enable the reconfiguration and enhance

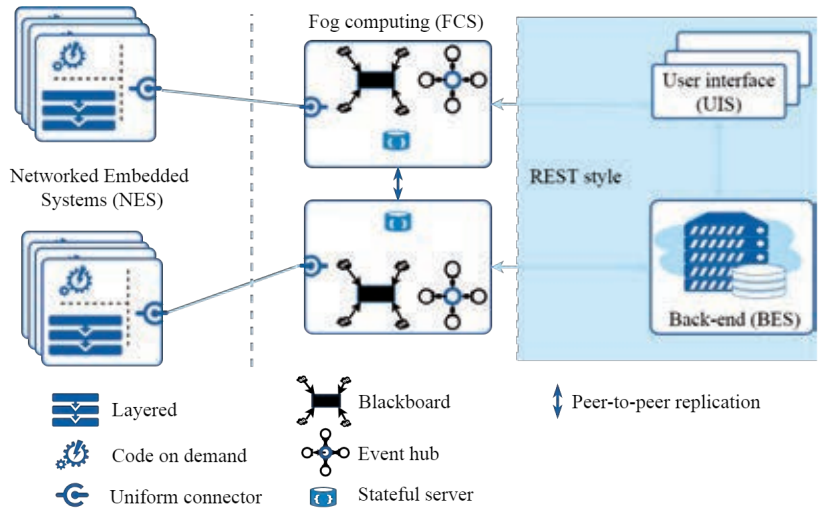

Figure 4.6: Overall PI architectural style for IoT systems

maintainability of the NES. A code on demand approach is a style used with a client-server approach where an executable code is sent to the client from the server [38]. This approach was first proposed for IoT systems in one of the contributions of this thesis [68]. At the center of this style is the description of the resources available for manipulation by the code sent from the server. This description of the device (or the physical object interacting with the device) is referred as Thing Description by the W3C [28]. A related description, extracted from IoT domain model of the IoT-A project [49], called Device Object Model (DOM) is used in [68]. A simplified representation of the resources in a device and the corresponding DOM model is shown in Figure 4.7.

The DOM exposes the values of hardware resources (such as sensors and actuators), events and metadata as API, to be manipulated in the interpreter by the code sent from server (Figure 4.4). Traversing the tree from the root to the leaf nodes, the device resources in the DOM are organized based on the access level; private resources of the DOM have read-only API functions while public resources can have read and write functions. In addition, the DOM interface also exposes functions that are used for sending and receiving messages to or from FCS layer. The implementation of the API exposed by the DOM are assumed to be shipped with the device by the manufacturer. Updates to the DOM and the underlying implementation of the network protocol or real-time operating system specifics can be done over the air. However, the main argument in this approach is that updates to these layers

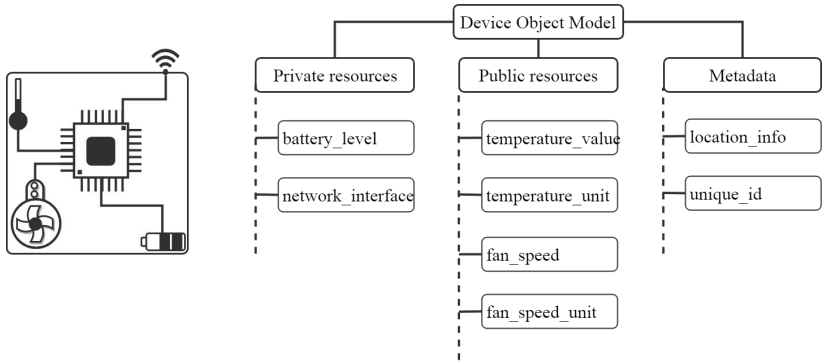

Figure 4.7: A simple IoT device (left) and the corresponding Device Object Model (right)

are not frequent compared to small functional specifications such as change in the formatting of the data or unit of measurement. The scripting language is designed to help in re-configuring the device behaviour using simple code stored in the Fog layer, similar to web pages stored in a web server [67]. The DOM tree or the thing description files are stored in a shared server to simplify writing scripts that consume resources exposed by other devices to enable interoperability of devices from different vendors [69].

```
typedef struct{
    int initialized;
    status_t (*sync)();
    status_t (*every)(uint16_t time);
    status_t (*read)(char *property, void *buffer);
    status_t (*write)(char *property, void* value);
    status_t (*send)(void *buffer);
    status_t (*receive)(void *buffer);
}dom_t;
```

Listing 4.1: Simple DOM api interface

```
version 1 ! version of the script
! Optional comment
thermo : include "thermostat.td"
dom : include "aircondition.td"
! Initialization block
init
{
    dom.write(unit, "rpm");
```

```
    dom.write(fanspeed, 0);
}
dom.every(12);
run{
    temp = thermo.read(temperature);
    dom.write(temperature, temp);
    unt = thermo.read(unit);
    if(unt == "c") ! Degree celecius
    {
        if(temp > 25)
        {
            dom.write(fanspeed, 10);
            ! Update the fanspeed value
        }
        ...
}
!end of script
```

Listing 4.2: Sample DOS-IL script fragment

The instructions interpreted in the NES are written using a domain specific language that is developed from the ground for IoT systems, called Domain Specific IoT Language (DoS-IL) [68]. The design motivation of the language is to simplify writing a set of commands that manipulate the function of devices that interact with the physical world. In comparison to other alternatives available, such as Node-RED [71] and DSL-4-IoT [85] that provide graphical programming environment to generate code, DoS-IL gives ease of re-configuration using code on demand style (Table 4.1). It has simple construct to write conditional instructions and declaration of the DOM of the device and others it interact with (see sample script in Listing 4.2). To elaborate on the relation of the scripting language and the DOM API exposed (Listing 4.1), the model represented in Figure 4.7 is used as an example. A simplified script portion that calls this API functions to achieve the system features is shown in Listing 4.2 with the corresponding simple DOM API. The sample script shown is written for an air conditioner device and it is interacting with a thermostat to get the temperature value. It also sets the unit of the speed of the local fan, which can be shared with other devices. The original scripting language and monolithic interpreter has been updated using a formal grammar and parser generator tools [69] to optimize the memory and processing requirement. The upgrade also includes the simplification of the interpreter required in NES by preprocessing the script in the FCS to generate a list of DOM calls. The source code of the scripting language is also made open source [61] to further enhance it and increase the visibility.

4.4 Virtual things server

A major part of the solutions proposed to address IoT systems challenges is provided by the components of the Fog computing layer [64]. The PI architectural style discussion in earlier sections has identified the processing component running at the Fog layer and its internal composition using multiple styles. This thesis also contributes an implementation of a Web of Virtual Things Server (WoVTS) [70] that supports the PI architectural style. One of the main purposes of the WoVTS is in creating a virtual representation of the physical objects in memory at the Fog layer. This virtual counter-part of the physical object contains all the state information required by applications and the thing description or DOM is used as a blueprint for the creation of this abstraction. As an additional level of interoperability, the server supports multiple formats or standards of thing descriptions to create the in memory objects [70]. These virtual things interact with each other to exchange event and data representing the change in state of the physical object. For instance, a virtual representation of the DOM shown in Figure 4.7 has properties representing the temperature and its units, and events such as arrival of new temperature reading, or exceeding a maximum temperature value. Subscriptions to events are managed by a central event hub. The virtual objects are stored in a central registry that serves as a state repository that can be replicated to neighboring gateways in the Fog layer for reliability and facilitate mobility of devices. In addition, the server also implements a uniform connector to interact with the physical objects, hiding the details of the network protocols used. This uniform connector is similar to the one used by the NES and it resembles the uniform interface available in REST. Using this interface helps abstract the underlying network interfaces and simplify the communication to four basic verbs: GET (request a resource), POST (update a resource or create new one), DELETE (remove existing resource) and NOTIFY (subscribe to event or state information change). Moreover, it was mentioned in Section 4.3 that the server also hosts scripts written in DoS-IL to manipulate the DOM of the physical object. The storage and serving of these scripts is also one of the contributions made in this thesis [67]. The server also adopted IoT-lite ontology [16] to annotate the client device and uses the device name as a unique identifier for the script written for a specific device. The missing component out of the FCS implementation in WoVTS is the blackboard style that is aimed to address the data processing needs of IoT systems. However, the data processing needed by IoT systems is mostly domain specific. Therefore, the PI style derivation for the data processing sub-component is taken from a healthcare case study presented in one of the contribution of this paper [81]. The implementation of the server is done in two versions that are both made avaiable as open source. The first version is to support script hosting and

uniform interface [62] and it is enhanced in the second version implementing virtual object creation and their interactions [63].

4.5 Summary

In summary, this chapter provided the overall IoT systems architectural style derivation process starting from a system with no style to an elaborate four tier design. The PI architectural style is formed by detailed analysis of generic IoT systems requirements focusing on networked embedded systems and combining it with REST to provide the full picture. The chapter also highlighted the contributions of the thesis in different papers containing tools that allow the adoption of the architectural style. Some of these tools include a domain specific scripting language designed for IoT systems, an embedded interpreter that execute the script, a server designed for the Fog layer that hosts the scripts and creates virtual images of the physical objects.

Chapter 5

Analysis and Evaluation

The main elements of the proposed solutions to overcome lack of interoperability has been covered in previous chapters. In this chapter, qualitative and quantitative results of the measure of effectiveness and efficiency of the proposals is presented. The overall contributions of the thesis are analyzed from the perspective of enabling interoperability to IoT systems, with a focus on networked embedded systems. It also considers any additional benefits that can be achieved or system qualities compromised due to the introduction of either the middleware or the PI architectural style. The discussion considers the resource utilization and effectiveness of the components introduced by the individual contributions.

5.1 Analysis of contributions

The core target of the thesis is to address the lack of interoperability in IoT systems at different levels of abstraction. This challenge is aggravated by the resource limitations in most of the devices used at the perception layer (interaction with the physical world). The analysis of the contributions of the thesis mainly focuses on the resource efficiency of the solutions. Some of the quantitatively studied attributes are the memory required for storage, processing and the performance (time constraint) for LISA and WoVT server. The architectural style analysis however targets to identify details about the style on how well it address the desired system quality attributes needed by IoT systems and the constraints it enforces in a qualitative manner.

5.1.1 LISA middleware

The fundamental design target of the interopreability middleware is to make it lightweight, as the name indicates. To meet this target, the original NoTA framework has been studied to identify the crucial components that are

needed for IoT systems. This step is followed by identifying what optimization can be done to fit within the target environment. In the first step, it was identified that the protocol (such as registration and discovery) and modular organization of the network protocol specific implementations are critical to provide interoperability. In addition, moving features that can be relocated to the manager nodes provides relief to the resource constraints of devices running the NES component. The LISA middleware has two types of messages: setup and user messages. Setup messages are used during build-up stage, which includes service registration, deregistration, acknowledgement, discovery or access request. In contrast, user messages carry the actual payload exchanged between an application and service node. Setup messages have a header size of 8 bytes and user messages have a header size of 9 bytes [65]. These headers contain the addresses given to nodes according to the generation format described in Chapter three, Section 3.3, to uniquely identify the device, its domain and the manager connected to it.

Another important measure of the middleware performance analyzed in this work includes the time taken by nodes to handle each type of setup message. This provides an indicator of the total time required to start the actual user message exchange. The simulation is carried out using the version of LISA working on RIOT [13] operating system by creating tap interfaces representing each node. The complete setup is done on a virtual machine running Ubuntu 14.04 that has 4GB of RAM and a speed of 3.16 GHz. The time taken to process discovery request by manager node and discovery response by application node, registration request by manager node and registration response by service node are shown in [65]. It was observed that a service node requires an average of 17.5 µs to get ready to handle requests. Similarly, an application node requires an average of less than 1ms to complete discovery and authentication. This delay vary depending on the number of registered services. The performance of the middleware was also analyzed for message routing from one sub-network controlled by one manager node to a different sub-network. From the perspective of the client discovering a service, the time required is comparable to the previous discovery process as the manager node responds with its own address in the discovery response. The details of the algorithm used to route messages across a network of manager nodes is also presented in [65]. The routing follows the hierarchy of manager nodes and the performance also varies based on the number of service nodes in each sub-network.

A crucial attribute considered in the analysis is the memory requirement of the middleware. In a typical NES system, shown in Figure 4.4, identifying the size of the middleware separately from the other layers is challenging as the final binary is almost always combined in one file. To find the size of the middleware, the size of the object files generated from the source code is summed up. In contrast to the size of the NoTA, which was 260KB, the size

of LISA was only 22KB [65]. In the sample binary, LISA took less than 20% of the total file (130KB). The overall resource utilization, in comparison to the benefits brought by LISA, was taken as a design compromise for interoperability, mobility and ease of programming. All of the analyses done so far are based on the initial implementation on RIOT operating system. The final analysis section of LISA was done after porting the middleware to Contiki RTOS [31]. In addition to the above resource utilization analyses, the new version includes the power consumption measurements for each setup and user messages. One of the notable optimization made in this Contiki based version is the decrease in memory footprint, which is inline with the initial motivation of integrating resource constrained IoT devices. It only used less than 15KB of memory, which is 7KB less than the RIOT based implementation [72]. In addition, the power consumption and processor time taken by LISA were compared with a reference UDP server implementation included with the Contiki RTOS. In summary, the analyses of LISA has shown the overall overhead introduced by the middleware in comparison to the benefits shown in Chapter 3. Based on the analyses, it is demonstrated that LISA is useful for resource constrained nodes with limited memory, bandwidth and battery life. Referring the classes of constrained devices listed in [21], LISA can also be used in Class 0 (having flash size of less than 100KB) devices. However, it needs to be ported to multiple RTOS and support multiple protocols with a possibility of running on a bare-metal to enable interoperability of IoT systems.

5.1.2 DoS-IL script

To analyze the scripting language, it is mandatory to first identify the design goals. The motivation for creating DoS-IL was to provide interoperability for IoT systems using similar techniques used in the Web. The inventor of the World Wide Web, Sir Timothy Berners-Lee, in his paper [17] covering the history and future of the Web, summarizes the purpose and motivation of the Web at the time. According to the paper, the Web provided an interoperability layer for various networks. Following this analogy, the various network protocols used by devices in the perception layer can be integrated using a client-server configuration with a code on demand approach as described in the derivation of the PI style. To elaborate on the advantage of the code on demand approach, consider a smart home system that contains a smart bulb and smart lock components, as shown in Figure 5.1. The current state of the art approach, discussed in Chapter 2, is to write the complete code of the smart devices and program them to work according to the initial specification; for instance, when the smart lock is opened turn on the light. Assume the system needs integrating additional smart devices in separate steps, first extend the system to consider the room tempera-

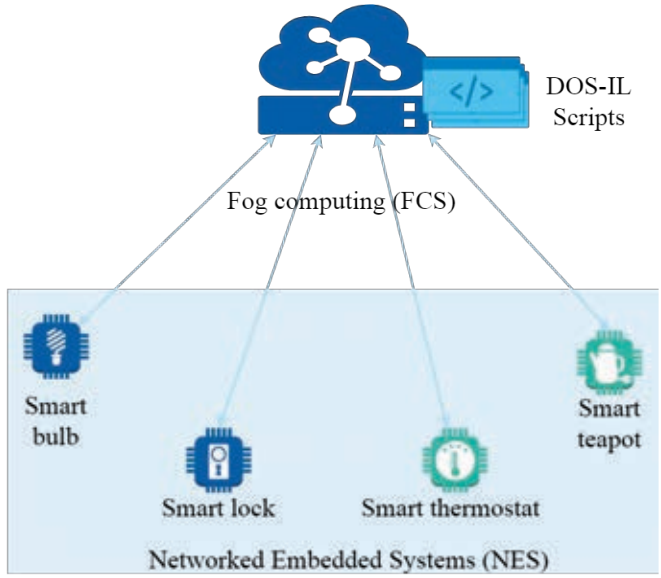

Figure 5.1: A hypothetical system with two smart devices (left side shown in blue) is extended to support two more smart devices (right side in green).

ture from the thermostat and next turn on the teapot. In each step, the additional functional requirement introduced to the initial specification has to be implemented and the update has to be sent to the devices already installed. If the vendor of the devices is not the same, the options of integrating these distinct devices are limited or may not exist. In contrast, code on demand approach allows the smart devices to be re-configured according to the new specification without the need for update of the core firmware by simply modifying the scripts. This approach facilitates system level integration. However, there is no such a language that can be used to address re-configurability of the feature of IoT systems. Hence, DoS-IL is contributed as part of this thesis. The new domain specific language should consider the resource limitations of the devices and bandwidth of the network interface. These attributes determine the efficiency and conciseness of the scripting language.

The first version of the scripting language and the interpreter published in [68] presented the size of sample scripts and the interpreter. The scripts considered are both less than 600 bytes and the size of the interpreter was only 76KB [68]. The analyses of the script from the servers perspective are given in [67]. The total time required from requesting the script to complete transfer was shown in comparison with a two way communication with no

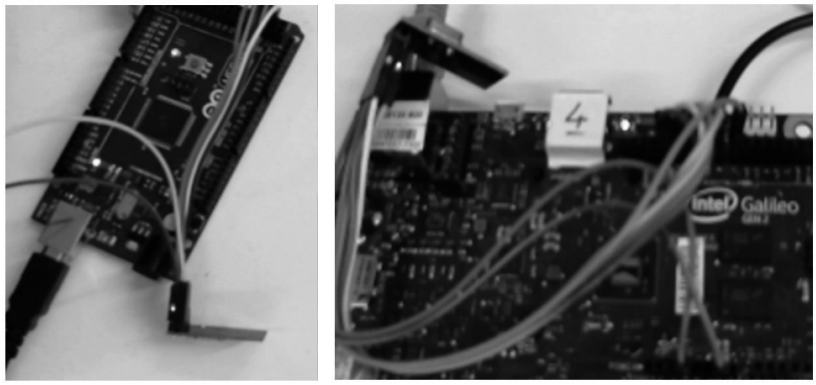

Figure 5.2: Experimental client (left) and server (or gateway in the right side) setup for analysis.

payload. Both scripts were completed in approximately 400ms, around four times the one required by the reference two way communication. One of the main time consuming part was the integration of an IoT-lite [16] ontology to query the path of the script based on the name of the requesting device. The experimental setup used for these measurements was built using an nRF24L01 [87] radio based network. To fit within the maximum payload limitations of the network, the script is fragmented into parts following predefined protocol. Three bytes of header define the status code, remaining packets and the checksum of the message [70]. The setup used for this analysis is shown in Figure 5.2 [67]

The second version of the script was enhanced to include thing descriptions as *include* files in the script to help during references to properties and actions of devices. The interpreter is also built using standard tools, Bison and Flex [57], that optimized the size of the binary. In addition, alternate ways to the transported instructions have been defined. In contrast to sending the complete script for the client device, DOM call instructions generated from the abstract syntax tree of the script parser can be sent. These are very simple instructions that can be interpreted with smaller sized interpreter [69] more efficiently.

5.1.3 WoVT Server

A small part of the Web of Virtual Things (WoVT) Server has been covered as part of the script analysis. This section gives details of timing and memory usage of the server in handling other types of requests. The location of this server provides a wide range of opportunities to address the challenges of IoT

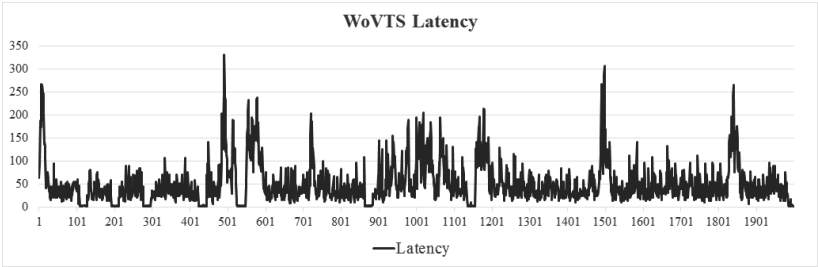

Figure 5.3: WoVT Server latency graph over the simulation period.

systems. As mentioned in Chapter 4, one of these advantages is the storage of state information of devices in the perception layer. The size of a state of a device varies depending on the attributes of the thing description of the object. However, the overall memory consumption of the server over 120s of simulation time while creating 129 virtual objects has been maximum of 45MB [70]. Considering a typical gateway device, such as Raspberry Pi [79], with gigabytes of RAM and the number of devices simulated, the memory consumption of WoVT server is bearable. This communication is carried out over a uniform connector that forms a layer of interoperability over any preferred middleware. It also provides a uniform message format for requests and responses. A request has a four bytes header that indicate attributes such as the version, type of request, payload length and resource url. Similarly, a response header contains three bytes of response code, remaining packets and checksum of the payload [70]. This provides the server to receive or send messages that are understandable in the perception layer of an IoT system.

The second important measure of the performance of the server is the latency of processing a request or the time needed for the server to reply a message. Throughput, a related term, on the other hand measures the total number of requests handled per unit time. This analysis simulates three network interfaces each sending an average of 4 requests per second over a total period of 100 seconds. Based on the gathered data, the server had a maximum throughput of 51 requests per second or an average of 19.6ms processing time per request [70]. The simulated requests represented devices in the perception layer. The other communication that requires analysis is the REST service for replicating state information with other gateways or for cloud interaction. For this test, a test tool called Apache JMeter [96] was used. Similarly, a total of 500 nodes each sending 4 requests over a period of 100 seconds is simulated. The average response time of the server was reported to be 54.6ms, which is also shown as a throughput of 1201 requests per minute [70]. The latency of the server over the whole simulation period

is shown in Figure 5.3. Even though there are observations that had a value of more than 300ms at times, more than 60% of the requests were handled in less than the average time.

The last part of the analysis of the server targets the event handling capacity. The implementation takes advantage of an open source database that stores a key value pair in memory, called Redis [84], for device state and event management. Using a built-in command line tool to extract its performance measures, the two mostly used functions resulted in throughput of 90,000 for Set method and 86,500 requests per second for Get method [70]. Redis also has a feature for replication of the data to neighboring gateways, which can be used for mobility services in the gateway. The throughput values are also sufficient enough to handle the requests of devices in the perception layer and as event hub for in-memory virtual things for sub-networks of about 129 devices (virtual devices created). Replication of the data using Redis is an optional way of sharing state information and the alternative way is using the REST service discussed earlier.

5.1.4 PI style

Analyzing architectural styles qualitatively provides an understanding of the behaviours of systems that follow the style [15]. There are structured ways to do the analysis for comparison with other related styles and ease of understanding the overall features of the style. One of these frameworks is presented in [40]. It identifies the components, connectors and data elements of the style, the topology of their interaction, including the constraints enforced and the benefits or drawbacks of the style. Some of the elements of the PI style have been discussed inline with the derivation process, focusing on the lower end of the overall style (left of the REST style in Figure 4.6). Similarly, the analysis focuses on the NES and FCS components, their connectors and the data flowing through. The major elements of the style categorized based on their function are presented in Table 5.1 [69]. There are processing components that are listed as 'other' to indicate the possibility of adding components depending on the application domain specific requirements. For instance, other NES components include the RTOS, middleware and drivers used by the firmware. In addition, there are elements that can be categorized as both processing component and connector element; for instance, resource browser and state manager (store).

Identification of the elements of the style simplifies the process of defining the topology. These components are also shown in the general view of the architectural style shown in Figure 4.6. Following the derivation process in Chapter 4, the topology is discussed hierarchically starting from arrangement of larger components that have multiple internal components, such as NES and FCS, followed by their internal sub-components. In general,

Table 5.1: Main architectural elements of the PI style.

Processing Components	**Description**
Interpreter (NES layer)	For script execution
Event hub (FCS internal)	Event manager for virtual things
Virtual things manager (FCS)	Handling state of devices
Shared data store (FCS internal)	Central data store for processing
Other NES components	NES firmware layers
Other Fog components	Local storage and processing
BES and UI (included in REST)	Processing, storage and access
Connectors	**Description**
Resource browser layer in NES	Communication handler with FC
Uniform interface	Uniform message interface
State and script store in FCS	Communicate to NES
Replication interface	FCS point-to-point connector
Data	**Description**
Thing description	Description of device resources
Device state	State of device between FC
Scripts	Instruction to device
Device command	Command (feedback) to device
Identifier to thing description	URL or unique ID

multiple NES components consume services hosted in one FCS component forming a star topology (Figure 4.6). This has the drawback that it has a single point of failure. However, the replication of state to other FCS components provides redundancy during failure of any FCS element. This leads to the hierarchical mesh type arrangement of the gateways hosting FCS to provide the necessary system quality. One of the driving reasons for this arrangement is the need for peer-to-peer communication of FCS gateways to exchange state information of mobile devices. Moreover, sub-network of gateway devices can be managed by a single higher layer gateway as recommended by the OpenFog reference architecture [75]. Any layer that access the back-end system (BES) acts as a client in a REST style as shown in Figure 4.6. In this topology analysis, the part of the style left for REST is considered as a black box. The overall topology of the architectural style is shown in Figure 5.4. The main processing elements are labeled and the different connectors are indicated by changing color and arrow type. For example, thin blue arrows show uniform interface connector and green thicker arrows show the state store connectors in a peer-to-peer fashion.

Internal structure of each processing element is composed of multiple sub-components that have different arrangements. A strict layering is rec-

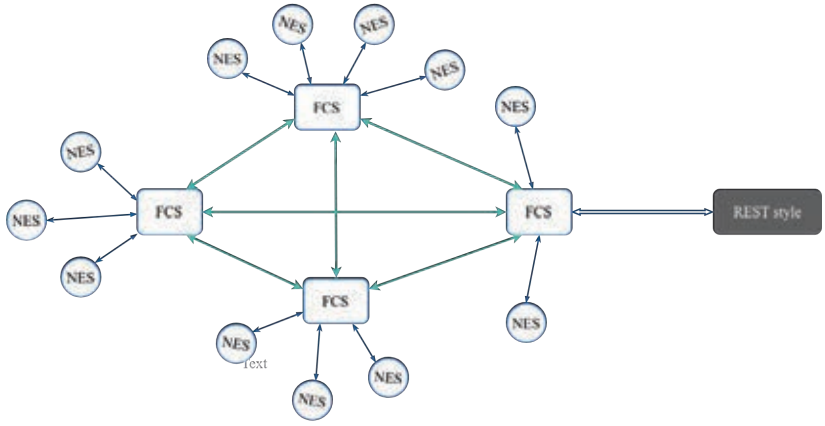

Figure 5.4: Topology of the PI architectural style

ommended (top-down or linear topology) in the internal organization of the NES component to limit the side effect of the code on demand approach selected for re-configuration. In strictly layered organization, upper layers can only access the layer just underneath. Therefore, the layer containing the interpreter shall only access the API exposed by the DOM layer only. To provide flexibility for implementation, the other layers can be optionally made non-strict. The second major component is the FCS, which has three main types of styles. The first type is the use of replicated repository that makes the FCS server stateful (store state of its clients). This is part of the connector that is located at the center of the star arrangement shown in Figure 5.4. Inside the FCS, it is shared by the other two sub-components arranged in blackboard and event hub styles (see Figure 4.5). The event handler component of the FCS that uses the event oriented style forms a star topology; the event hub sits at the center and virtual devices connecting to it. Event publishers contact the hub, which manage and notify subscriptions. If an event results in change of state, this will be pushed to the state repository. The last part of the FCS is the data analysis [81] section that use the blackboard style. This style inherently leads to a star topology where the shared repository (the blackboard) is located at the center [40].

The final part of the architectural style analysis is to provide the advantages and drawbacks of the style. An obvious advantage of using this style is that it provides interoperability at different levels. First, systems developed using this style can be integrated with minimal effort as both use similar organization. Second, the use of thing descriptions enable the use of domain specific ontologies for semantic interoperability at higher layers (Back-end). Moreover, the use of middleware to abstract the underlying network proto-

col and the uniform connectors provide syntactic interoperability. Another advantage of using this style is ease of re-configuration of the perception layer devices using domain specific scripts. Furthermore, storing the state information of the devices and the virtual representation of a physical object enables support of mobile nodes. In general, the overall advantages of using this style are summarized in Table 5.2 [69].

Table 5.2: Advantages of PI architectural style for IoT systems

Advantages	**Style to address**
Interoperability	The PI style designed to address this
Re-configurability	Layered, code on demand and client-server
Mobility	Replicated state repository
Scalability	Code on demand and replication
Performance	Hierarchy of FCS gateways
Portability	Code on demand and layered approach
Reliability	Replicated repository in FCS
Simplicity	Functional separation and uniform interface
Security	Strict Layered style in NES

The advantages listed above come at the cost of limitations introduced by the style. One of these drawbacks is the restriction of peer-to-peer connection of the NES components. There are circumstances where a point-to-point communication of the perception layer nodes is needed. In such scenarios, the PI style can only be achieved by allowing the sync node to act as NES component representing all the other mesh nodes. In addition, PI restricts the direct access of the NES component from the user interface component [69]. As a final point, the PI architecture style is an initial attempt to generalize IoT system qualities and derive a common style. This generalization can easily fail to address specific types of IoT systems and newer approaches. Hence, subsequent revisions are necessary.

5.2 Security analysis

In most of the discussions of the middleware and architectural style, security and privacy have not been discussed. It is one of the main challenges that constrain the adoption of IoT systems. Given the sensitivity of the information gathered by IoT devices, ensuring the security and privacy of the data and the overall integrity of the system is mandatory. Special care is required when integrating systems of different domains. In the classification method given in Table 2.1, security consideration is highlighted with 'Ownership' criteria in three categories. This is introduced to indicate the need for identifying the different levels of security required by each IoT system. In most

of the contributions of this thesis, security has been considered as a constraint. For instance, LISA protocol has an access request from the service identified during discovery request (see Figure 3.2). The access token generated by the service node is used during subsequent interactions. Similarly, the code on demand approach introduced has considered the security risks associated with the style and strict layering is enforced to protect underlying components [69]. Moreover, the FCS implementation of the state store and event hub in [70] also has security considerations included. However, detailed analysis of the security and privacy perspectives of the contributions, such as in [7], are outside the scope of this thesis.

5.3 A Complete view

The contributions of the thesis have been analyzed for resource utilization and suitability for the desired target. This section provides a holistic view of the contributions in a typical IoT system. The first category of contributions (Paper I and Paper II) [65] [72] provide a middleware layer to abstract the underlying network interface and platform differences. These provide syntax level inter-operation of Networked Embedded Systems. The middleware also provides ease of programming. The second category of contributions (Paper III, Paper IV, Paper V and Paper VII) are targeted to provide semantic and system level integration. These are enabling code on demand style in NES by scripting and an interpreter layer, an FCS component containing state store, virtual representation of physical things and their interaction via events. This category also contains a contribution discussing the architectural style. The last category provides a healthcare domain as a use case for the above components (Paper VI). The contributions of the thesis mapped to architectural elements of an IoT system identified in PI are shown in Table 5.3.

The preceding chapters provided the background of the thesis, the current state and the main contributions of the dissertation including the analysis and evaluation of the efficiency. In the upcoming last chapter, concluding remarks are given including recommendations, future plans and ongoing works to realize the web of things.

Table 5.3: Contributions of the thesis relative to an IoT system

Architectural element	**Contribution**
NES middleware layer	Paper I [65] and Paper II [72]
NES interpreter layer	Paper III [68]
NES DOM API layer	Paper III [68]
NES connector	Paper III [68] and Paper IV [67]
FCS uniform connector	Paper IV and Paper V [70]
FCS State repository	Paper V [70]
FCS Event hub	Paper V [70]
NES - FCS code on demand	Paper III, Paper IV and Paper VII
Overall style	Paper VII [69]
Healthcare use case	Paper VI [81]

Chapter 6

Conclusions and recommendations

The absence of interoperability is one of the critical challenges of IoT systems that prohibit the adoption and emergence of novel solutions that span multiple application domains. It constrains the creative add-ons that can potentially introduce new dimensions by forming mash-ups as in the Web today. To address this challenge, the thesis has contributions in three main levels of interoperability: syntactic (in Paper I and II), semantic (in Papers III, IV and V) and system levels (in Paper VII). The contributions are organized in two categories. The first approach is using a middleware that is discussed in Chapter 3 followed by an architectural style and enabling components. These contributions and findings are summarized in the following sections followed by the research works that will be continued in the future.

6.1 Summary

The first set of contributions address variations in protocol, platform and data formats in IoT systems. In Paper I, LISA was proposed together with a node oriented architecture. It has a core module that defines a custom protocol to help organize the features of the system into application and service nodes. These nodes are managed with a central manager node that registers services and provide service information for application nodes. Paper II extended LISA to port the middleware to a different operating system that follows a different internal architecture. These contributions provide a mechanism by which the underlying variations of platform and protocol are abstracted and a uniform interface is exposed for applications. The main motivation of the middleware was to integrate resource constrained nodes. In this regard, the memory footprint, the latency and message header sizes are analyzed to show the efficiency of LISA. The size of the middleware was

22KB and 15KB in the first and second implementation respectively. In addition, a message header of less than 10 bytes is used in both setup and user messages. Considering the classification of resource constrained devices in [21], it was shown that LISA can fit even in devices in the lower limits. In addition, the node centered approach allowed LISA to uniquely identify services and applications as the central elements of communication in IoT sub-networks. It follows a unique addressing technique that is tailored for mobility support and routing of messages across protocol boundaries through a federated organization of nodes.

Another major contribution of the thesis is the introduction of a pragmatic IoT (PI) architectural style derived from a standalone cyber-physical system. This allows the derivation of an architectural style that is specifically targeted for IoT systems. The derivation process focused on the Networked embedded systems side and it is combined with REST style. The Networked Embedded System side contains two of the major processing elements that are closer to the physical world: the embedded component running in the devices that interact with the object and another component in the Fog layer. The internal organization of these components and specialized sub-components that enable the PI style are presented. One of these sub-component of networked embedded system is an interpreter for a domain specific scripting language. This forms a code on demand style where an executable code is sent from the Fog layer to re-configure the behaviour of the perception layer devices. Paper III introduced the scripting language and the organization of the resources of devices that are manipulated by the script as Device Object Model. The concise syntax of the script and the memory footprint of the interpreter are some of the evaluation points used to analyze the contributions. The sample scripts presented had sizes of less than 600 bytes and compared to a full firmware binary update, which is typically few hundreds of KB, the script can be sent efficiently to client devices.

Leveraging the proximity of the Fog computing layer, which is introduced to address the challenges of resource constrained devices, the PI architectural style addresses generic challenges of IoT systems. Some of the design constraints in PI include a stateful Fog server that also replicates the state information to neighboring gateways. Papers IV and V contribute in the implementation of a server that follows the principles of PI. The server stores the state information as virtual representation of a device and the physical object it interacts with. The events of change of state of a virtual thing is published to a central event hub that notify all subscribers as recommended by the PI style. The performance of the server and its resource utilization have been presented in Paper V. The use of ontology names to identify the thing descriptions and scripts stored in this server forms the foundation for semantic interoperability that can be extended to specific application do-

mains. In addition, the uniform interfaces (connectors) used to interact with its clients allow syntactic integration of multiple network protocols. Using simulated requests, the server had a response time of 19.6ms and used 45MB of memory to handle 129 virtual objects. Moreover, it had a REST service for interacting with other gateways or the cloud. Similar analysis has shown that it has an average response time of 54.6ms. Comparing the resources of typical gateway devices, with hundreds of MB or even GB of memory, and the size of typical sub-networks connected to a gateway, the server implementation can efficiently provide the desired services. The replication of the state information facilitates services for mobile devices and increase the reliability of the overall system.

The analysis of the architectural style has followed a standard framework to provide the major elements of the style categorized as processing elements, connectors and data. It also provided the detailed topology and constraints that restrict the arrangement of the components and their internal structure. In addition, it presented the advantages and drawbacks of the PI style including the quality attributes promoted by it. The main target of the overall architectural style derivation is to provide interoperability among systems that use the style. It also enables ease of communication and deployment of IoT systems.

Lack of interoperability in IoT systems is prohibiting the next wave of technology from unleashing its full potential. Standardization based efforts for integration of IoT systems has so far led to creating additional silos. This in turn led to the introduction of more standards that act like a bridge to the different sides. In this regard, middleware or reference architecture contributions also directly or indirectly set a standard that should be followed by developers. In IoT systems, these variations range from the hardware architecture and sensor or actuator interfacing standards to programming languages and patterns. It also extends to the physical mode of communication, voltage level and frequency, data formats and the context of the information exchanged. It is clear that the current state of IoT is shaping to silos or intranet of domain specific vertical applications. The Web of Things promises to provide the highest level of interoperability similar to the Web. The contribution of the thesis can be summarized as forming interoperable networked embedded systems that can be used to build the Web of Things. It visualizes the full picture of the software system starting from those running in resource constrained embedded devices all the way to the cloud, but focuses on the non-traditional Internet side which is more heterogeneous.

6.2 Future works and recommendations

Most of the contributions in the thesis have been made available as open source. This creates better chance of visibility by other interested researchers and developers that can contribute to the repository. These open source repositories include the middleware (LISA), the scripting language (DoS-IL) and the interpreter, and the virtual things server (WoVTS). The development of LISA has not been active for a while now. This is mainly due to the start of similar middleware solutions (such as Iotivity) that are developed by big industrial alliances. However, the scripting language and the virtual things server are still under improvement and will also continue in the future to enhance both solutions. In addition, there are many efforts by different organizations and research groups that work towards the realization of the Web of Things. In this regard, the author has been working with individuals having similar interest, contributing open source code. One of these areas is a different approach on the use of things description and the interaction model of devices. This extends the work done in the WoVT server and enhance the reach of the contributions made during the dissertation work.

As a final note, the involvement of big industry players in standards organizations and also the contributions made by industrial alliances as compared to open standards tend to incline to the interest of the involved companies. Some of these contributed standards start from a clean slate while others are more biased towards the products and services delivered by specific companies. For IoT to deliver its promises, unbiased open standards are crucial for ease of adoption. This leads to a more integrated IoT and a global Web of Things.

Chapter 7

Overview of Included Publications

This chapter highlights the included original publications and the contribution of the author in each of these publications.

7.1 Paper I: LISA 2.0: Lightweight Internet of Things Service Bus Architecture Using Node Centric Networking

This publication introduces a lightweight middleware (LISA) that was developed to address the lack of interoperability in resource constrained IoT devices. It also introduces a node-centric networking approach that works with the middleware. This journal article tries to address the syntactic interoperability challenge that exist when using multiple network protocols and platforms in a single system. The middleware acts as a layer where service and application nodes in a service oriented architecture (SOA) can interact.

Author's contribution: The author is the main person behind this publication, from the initial design and development to writing the majority of the text. This publication is based on the previous work the author did as a proof of concept to build a lightweight middleware. A proposal based on this publication has been awarded an Academy of Finland project for two years, through which some of the next publication have been done.

7.2 Paper II: From Threads to Events: Adapting a Lightweight Middleware for Contiki OS

This publication tries to address the syntactic interoperability challenge in IoT focused on platform variations. Compared to the original version of the middleware idea, this version works in Contiki operating system and a different network stack. In addition, due to the availability of different evaluation tools and simulators in Contiki platform, extensive evaluation of implementation the specification is done in this work.

Author's contribution: The author assisted and supervised the implementation of the work that extended his work in publication one. This has been done as a masters thesis for the first author of the publication. In addition, the author contributed in the overall development and vision of the work, writing the publication and reviewing the results.

7.3 Paper III: DoS-IL: A Domain Specific Internet of Things Language for Resource Constrained Devices

This publication introduces an IoT domain specific scripting language that can be used to enhance the interoperability of IoT systems. The scripting language derives from an abstract representation of an IoT device that leads to providing the basis for semantic interoperability of IoT where the ontology in the domain can be reflected in the scripting language. The publication introduces a lightweight interpreter that is embedded in resource constrained devices that can execute the scripts that these devices fetch from the Fog layer (gateway).

Author's contribution: This publication is also the brain child of the author from the concept, to implementation and writing. The author had also the chance to present his work in the conference where it was published. This publication has also served as a means to another funded proposal by Academy of Finland and National Science Foundation.

7.4 Paper IV: Rethinking Things Fog Layer Interplay in IoT: a Mobile Code Approach

In this publication, the idea of providing semantic interoperability in a generic way is further expanded from that of the third publication. The focus is on how the perception layer (things) interact with the Fog layer. In traditional approach, all the application logic is stored in the 'Things' or devices at the perception layer and this publication shows the roll of the

Fog layer in the process of enabling a mobile code approach; that is to show how the IoT domain scripts are stored and access at the Fog layer by the devices in perception layer.

Author's contribution: Similar to the previous publications, the author is the main responsible for the concept, implementation and writing of the publication. Another main component of the overall vision of the thesis is included in this publication. It ties the previous work and subsequent publications as a glue to form the bigger picture.

7.5 Paper V: Towards an interoperable Internet of Things through a web of virtual things at the Fog layer

In the process of providing a compelete means to provide interoperability for IoT, this publication introduces a comprehensive server for the Fog layer that creates and stores virtual things (a digital abstract counter part of devices in the perception layer). It acts as a space where these virtual objects interact by subscribing and publishing events of interest to the application. As in the previous cases, this contribution is also generic in that it provides the platform without any application domain specific focus (such as healthcare or smart home).

Author's contribution: The responsibility of the author in this publication is also from defining the concept and scope of the paper, to implementation and writing of the contents. It also defines the basis for the last publication of the thesis.

7.6 Paper VI: Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach

This paper is a healthcare specific impelementation of an IoT system which utilizes a smart gateway at the Fog layer. It implements services that address the requirements of a healthcare system to monitor patients. There are many services that process the data collected and it also showcase the need for interoperable IoT system. In addition to the need for interoperability, it also serves to highlight the significance of architecture in building IoT systems that are cable of interoperating with other IoT systems across application domain boundary.

Author's contribution: This journal article is an extension of the work done previously by the author and other contributors during summer of 2014. The main contribution of the author is specifically on the topics related to

interoperability in the Smart gateways and the overall architecture of the system. This publication serves only as a use case for the main topic covered in the thesis.

7.7 Paper VII: A Pragmatic Architectural Style for Interoperable and Scalable Internet of Things Systems

This publication introduces an architectural style that is aggregated as best practice from previous publication to build IoT systems thereby allowing these IoT systems to finally collaborate in a standard way. It starts from identifying the challenges in IoT systems that constrain the quality attributes that must be satisfied via architecture. It then builds up starting from a simple embedded system step by step to a global scale web of things system. In doing so, it identifies and lists recommended architectral styles for each of the components identified.

Author's contribution: This last publication is like an umbrella to the overall contributions made by the author in the thesis. It presents a novel architectural style that helps in facilitating interoperability for IoT and the author has been the main person responsible for the concept, implementation and writing.

Bibliography

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, Fourthquarter 2015.

[2] LoRa Alliance. What is the LoRaWAN specification? `https://lora-alliance.org/about-lorawan`.

[3] Zibee Alliance. dotdot by zibee alliance. `https://www.speakdotdot.com/dotdotstory`.

[4] Zigbee Alliance. Low-power, low-cost, low-complexity networking for the internet of things. `https://www.zigbee.org/zigbee-for-developers/network-specifications/`.

[5] Alliance for Internet of Things Innovation. Iot lsp standard framework concepts - release 2.8 AIOTI WG03 Iot standardisation. Technical report, ALLIANCE FOR INTERNET OF THINGS INNOVATION, 2017.

[6] Alliance for Internet of Things Innovation. High level architecture (hla) - release 4.0 AIOTI WG03 Iot standardisation. Technical report, ALLIANCE FOR INTERNET OF THINGS INNOVATION, June 2018.

[7] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. Internet of things: A survey on the security of iot frameworks. *Journal of Information Security and Applications*, 38:8–27, 02 2018.

[8] Arduino. Getting started with arduino and genuino products. `https://www.arduino.cc/en/Guide/HomePage`.

[9] Kevin Ashton. That 'Internet of Things' thing. `https://www.rfidjournal.com/articles/view?4986`.

[10] IEEE Standards Association. Internet of things: Ieee standards enabling products with real-world applications. `https://standards.ieee.org/initiatives/iot/stds.html`.

[11] IEEE Standards Association. P2413 - standard for an architectural framework for the internet of things (iot). `https://standards.ieee.org/project/2413.html`.

[12] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787 – 2805, 2010.

[13] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt. Riot os: Towards an os for the internet of things. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 79–80, April 2013.

[14] Soma Bandyopadhyay, Munmun Sengupta, Souvik Maiti, and Subhajit Dutta. Role of middleware for internet of things: A study. *International Journal of Computer Science Engineering Survey*, 2, 08 2011.

[15] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 3rd edition, 2012.

[16] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor. Iot-lite: A lightweight semantic model for the internet of things. In *2016 Intl. IEEE Conferences on UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld*, pages 90–97, July 2016.

[17] T. Berners-Lee. Www: past, present, and future. *Computer*, 29(10):69–77, Oct 1996.

[18] Dirk-Jan C. Binnema. *NoTA programming guide*. Nokia Research Centre, Finland, 2009.

[19] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the internet: The end-to-end arguments vs. the brave new world. In *Communications Policy in Transition*, pages 91–139. MIT Press, Cambridge, MA, USA, 2001.

[20] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[21] C. Bormann, M. Ersue, and A. Keranen. Terminology for constrained-node networks. RFC 7228, RFC Editor, May 2014. `http://www.rfc-editor.org/rfc/rfc7228.txt`.

[22] Vint Cerf. The open internet: what it is, and why it matters. *Telecommunications Journal of Australia*, 59(2):pp. 18.1 to 18.10., 2009.

[23] T. Chou. *Precision: Principles, Practices and Solutions for the Internet of Things*. CrowdStory Publishing, 2016.

[24] Michael Compton and et al. The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web*, 17:25 – 32, 2012.

[25] Industrial Internet Consortium. The industrial internet of things volume g1: Reference architecture. Technical report, Industrial Internet Consortium, 2017.

[26] World Wide Web Consortium. Web of things (wot) architecture - editor's draft. `https://w3c.github.io/wot-architecture/`.

[27] World Wide Web Consortium. Web of things (wot) protocol binding templates - editor's draft. `https://w3c.github.io/wot-binding-templates/`.

[28] World Wide Web Consortium. Web of things (wot) thing description - editor's draft. `https://w3c.github.io/wot-thing-description/`.

[29] Mozilla Corporation. Web thing api unofficial draft. `https://iot.mozilla.org/wot/`.

[30] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. A survey of commercial frameworks for the internet of things. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–8, Sept 2015.

[31] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Nov 2004.

[32] Eclipse IoT Working Group. IoT developer suvey. Technical report, IEEE, IoT Eclipse, Agile, 2016.

[33] Justin Ryan Erenkrantz. *Computational Rest: A New Model for Decentralized, Internet-scale Applications*. PhD thesis, UCI Institute for Software Research, Long Beach, CA, USA, 2009. AAI3372349.

[34] Chunxiao Fan, Zhigang Wen, Fan Wang, and Yuexin Wu. A middleware of internet of things(iot) based on zigbee and rfid. In *IET International Conference on Communication Technology and Application (ICCTA 2011)*, pages 732–736, Oct 2011.

[35] Roy T. Fielding, Richard N. Taylor, Justin R. Erenkrantz, Michael M. Gorlick, Jim Whitehead, Rohit Khare, and Peyman Oreizy. Reflections on the rest architectural style and "principled design of the modern web architecture" (impact paper award). In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, pages 4–14, New York, NY, USA, 2017. ACM.

[36] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[37] XMPP Standards Foundation. An overview of XMPP. `https://xmpp.org/about/technology-overview.html`.

[38] A. Fuggetta, G. P. Picco, and G. Vigna. Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[39] Maria Ganzha, Marcin Paprzycki, Wiesaw Pawowski, Pawe Szmeja, and Katarzyna Wasielewska. Semantic interoperability in the internet of things: An overview from the inter-iot perspective. *Journal of Network and Computer Applications*, 81:111 – 124, 2017.

[40] David Garlan and Mary Shaw. An introduction to software architecture. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[41] Michael M. Gorlick. *Computational State Transfer: An Architectural Style for Decentralized Systems*. Technical, UCI Institute for Software Research, August 2016.

[42] Eclipse IoT Working Group. The three software stacks required for iot architectures. Technical report, Eclipse Foundation, September 2016.

[43] Object Management Group. Data distribution service (DDS). `https://www.omg.org/omg-dds-portal/`.

[44] GS1 AISBL. EPC tag data standard: defines the electronic product code and specifies the memory contents of gen 2 rfid tags. Technical report, GS1, September 2017.

[45] GSMA. Narrowband internet of things (NB-IoT). `https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/`.

[46] Gregor Hohpe and Bobby Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[47] MCKINSEY Global Institute. The internet of things: Mapping the value beyond the hype. Technical report, McKinsey, 2015.

[48] interiot Project. Inter-iot project. `http://www.inter-iot-project.eu/`.

[49] IoT-A Project. Internet of things - architecture, IoT-A, deliverable d1.5 - final architecture reference model for the IoT v3.0. Technical report, EU-FP7, 2013.

[50] IoTvity. Iotivity 1.3.1 - project documentation. `https://iotivity.org/downloads/iotivity-1.3.1`.

[51] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking Named Content. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.

[52] C. Jennings, Z. Shelby, J. Arkko, A. Keranen, and C. Bormann. Sensor measurement lists (senml). RFC 8428, RFC Editor, August 2018. `https://tools.ietf.org/html/rfc8428`.

[53] R. Khan, S. U. Khan, R. Zaheer, and S. Khan. Future internet: The internet of things architecture, possible applications and key challenges. In *Frontiers of Information Technology (FIT), 2012 10th International Conference on*, pages 257–260, Dec 2012.

[54] Denis Kozlov, Jari Veijalainen, and Yasir Ali. Security and privacy threats in iot architectures. In *Proceedings of the 7th International Conference on Body Area Networks*, BodyNets '12, pages 256–262, ICST, Brussels, Belgium, Belgium, 2012. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[55] Philippe Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12:45–50, 11 1995.

[56] S. Kro, B. Pokri, and F. Carrez. Designing iot architecture(s): A european perspective. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 79–84, March 2014.

[57] John Levine and Levine John. *Flex & Bison*. O'Reilly Media, Inc., 1st edition, 2009.

[58] One M2M. Standards for m2m and the internet of things. `http://www.onem2m.org/technical/published-drafts`.

[59] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the internet of things (IoT). Technical report, IEEE, 2015.

[60] Katsuhiro Naito. A survey on the internet-of-things: Standards, challenges and future prospects. *JIP*, 25:23–31, 2017.

[61] Behailu Negash. DoS-IL implementation. `https://github.com/behailus/DoS-IL.git`.

[62] Behailu Negash. ISSS implementation. https://github.com/behailus/ISSS.

[63] Behailu Negash. WoVTS python implementation source code. https://github.com/behailus/WOVT.

[64] Behailu Negash, Amir M. Rahmani, Pasi Liljeberg, and Axel Jantsch. *Fog Computing Fundamentals in the Internet-of-Things*, pages 3–13. Springer International Publishing, Cham, 2018.

[65] Behailu Negash, Amir M. Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. LISA 2.0: lightweight internet of things service bus architecture using node centric networking. *Journal of Ambient Intelligence and Humanized Computing*, 7(3):305–319, 2016.

[66] Behailu Negash, Amir-Mohammad Rahmani, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Lisa: Lightweight internet of things service bus architecture. *Procedia Computer Science*, 52:436 – 443, 2015. The 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).

[67] Behailu Negash, Tomi Westerlund, Pasi Liljeberg, and Hannu Tenhunen. Rethinking 'things' - fog layer interplay in iot: A mobile code approach. In A Min Tjoa, Li-Rong Zheng, Zhuo Zou, Maria Raffai, Li Da Xu, and Niina Maarit Novak, editors, *Research and Practical Issues of Enterprise Information Systems*, pages 159–167, Cham, 2018. Springer International Publishing.

[68] Behailu Negash, Tomi Westerlund, Amir M. Rahmani, Pasi Liljeberg, and Hannu Tenhunen. Dos-il: A domain specific internet of things language for resource constrained devices. *Procedia Computer Science*, 109:416 – 423, 2017. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal.

[69] Behailu Negash, Tomi Westerlund, and Hannu Tenhunen. A pragmatic architectural style for interoperable and scalable internet of things systems. *Submitted*, 2019.

[70] Behailu Negash, Tomi Westerlund, and Hannu Tenhunen. Towards an interoperable internet of things through a web of virtual things at the fog layer. *Future Generation Computer Systems*, 91:96 – 107, 2019.

[71] Node-RED. A visual tool for wiring the internet of things. `http://nodered.org/docs/`.

[72] Uzair A. Noman, Behailu Negash, Amir M. Rahmani, Pasi Liljeberg, and Hannu Tenhunen. From threads to events: Adapting a lightweight middleware for contiki os. *2017 14th IEEE Annual Consumer Communications Networking Conference (CCNC)*, pages 486–491, 2017.

[73] OASIS. MQTT version 3.1.1. Technical report, OASIS Open, October 2014.

[74] Open Connectivity Foundation. OIC Core Candidate Specification v1.1.0. Technical report, OCF, 2018.

[75] OpenFog Consortium. OpenFog Reference Architecture for Fog Computing. Technical report, OpenFog Consortium, 2017.

[76] Particle. Meet the only all-in-one iot platform on the market. `https://www.particle.io/what-is-particle`.

[77] K. Pentikousis, B. Ohlman, D. Corujo, G. Boggia, G. Tyson, E. Davies, A. Molinaro, and S. Eum. Information-Centric Networking: Baseline Scenarios. RFC 7476, RFC Editor, March 2015.

[78] Dewayne E. Perry and Alexander L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, October 1992.

[79] Raspberry Pi. Raspberry pi 3 model b+. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/`.

[80] Dave Raggett. Wot thing description to rdf test page. `https://www.w3.org/WoT/demos/td2ttl/`.

[81] Amir M. Rahmani, Tuan Nguyen Gia, Behailu Negash, Arman Anzanpour, Iman Azimi, Mingzhe Jiang, and Pasi Liljeberg. Exploiting smart e-health gateways at the edge of healthcare internet-of-things: A fog computing approach. *Future Generation Computer Systems*, 78:641 – 658, 2018.

[82] Partha Pratim Ray. A survey of iot cloud platforms. *Future Computing and Informatics Journal*, 1(1):35 – 46, 2016.

[83] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke. Middleware for internet of things: A survey. *IEEE Internet of Things Journal*, 3:70–95, 2016.

[84] RedisLabs. Why redis. https://redislabs.com/why-redis/.

[85] Adnan Salihbegovic, Teo Eterovic, Enio Kaljic, and Samir Ribic. Design of a domain specific language and IDE for internet of things applications. In *38th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2015, Opatija, Croatia, May 25-29, 2015*, pages 996–1001, 2015.

[86] Tara Salman and Raj Jain. A survey of protocols and standards for internet of things. *Advanced Computing and Communications*, 1, 04 2017.

[87] NORDIC Semiconductor. nrf24l01 ultra low power 2.4ghz rf transceiver ic. `https://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01`.

[88] Z. Shelby, K. Hartke, and C. Bormann. The constrained application protocol (coap). RFC 7252, RFC Editor, June 2014. `http://www.rfc-editor.org/rfc/rfc7252.txt`.

[89] Z. Sheng, Shusen Yang, Yifan Yu, Athanasios Vasilakos, Julie A. McCann, and Kin K. Leung. A survey on the ietf protocol suite for the internet of things: standards, challenges, and opportunities. *IEEE Wireless Communications*, 20:91–98, 2013.

[90] Bluetooth SIG. Bluetooth 5. `https://www.bluetooth.com/bluetooth-technology/bluetooth5`.

[91] Sigfox. Sigfox, the worlds leading iot services provider. `https://www.sigfox.com/en`.

[92] Internet Society. Open internet standards. `https://www.internetsociety.org/issues/open-internet-standards/`.

[93] Z. Song, A. A. Cardenas, and R. Masuoka. Semantic middleware for the internet of things. In *2010 Internet of Things (IOT)*, pages 1–8, Nov 2010.

[94] OMA SpecWorks. Lightweight m2m (LWM2M). `https://www.omaspecworks.org/what-is-oma-specworks/iot/lightweight-m2m-lwm2m/`.

[95] A. Stanford-Clark and H. L. Truong. Mqtt for sensor networks (MQTT-SN) protocol specification. Technical report, International Business Machines Corporation (IBM), 2013.

[96] The Apache Software Foundation. Apache JMeter. http://jmeter.apache.org/.

[97] H. Tschofenig, J. Arkko, D. Thaler, and D. McPherson. Architectural considerations in smart object networking. RFC 7452, RFC Editor, March 2015.

[98] Carnegie Mellon University. The "Only" coke machine on the internet. `https://www.cs.cmu.edu/~coke/history_long.txt`.

[99] H. van der Veer and A. Wiles. Achieving Technical Interoperability - the ETSI Approach. Technical Report 3, European Telecommunications Standards Institute, 2008.

[100] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on the architecture of internet of things. In *2010 3rd International Conference on Advanced Computer Theory and Engineering(ICACTE)*, volume 5, pages V5–484–V5–487, Aug 2010.

[101] M. B. Yassein, M. Q. Shatnawi, and D. Al-zoubi. Application layer protocols for the internet of things: A survey. In *2016 International Conference on Engineering MIS (ICEMIS)*, pages 1–4, Sept 2016.

[102] Milan Zdravković, Miroslav Trajanović, Joo Sarraipa, Ricardo Jardim-Gonçalves, Mario Lezoche, Alexis Aubry, and Hervé Panetto. Survey of Internet-of-Things platforms. In *6th International Conference on Information Society and Techology, ICIST 2016*, volume 1, pages 216–220, Kopaonik, Serbia, February 2016. ISBN: 978-86-85525-18-6.

[103] H. Zimmermann. Osi reference model - the ISO model of architecture for open systems interconnection. In C. Partridge, editor, *Innovations in Internetworking*, pages 2–9. Artech House, Inc., Norwood, MA, USA, 1988.

[104] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi. From today's intranet of things to a future internet of things: a wireless- and mobility-related view. *IEEE Wireless Communications*, 17(6):44–51, December 2010.

Turku Centre *for* Computer Science

http://www. tucs.fi

tucs@abo.fi

University of Turku
Faculty of Science and Engineering
- Department of Future Technologies
- Department of Mathematics and Statistics

Turku School of Economics
- Institute of Information Systems Science

Åbo Akademi University
Faculty of Science and Engineering
- Computer Engineering
- Computer Science

Faculty of Social Sciences, Business and Economics
- Information Systems

ISBN 978-952-12-3828-4
ISSN 1239-1883