REST-BASED INTERNET OF THINGS-SERVICES AND APPLICATIONS IN A SEMANTICS-AWARE ENTERPRISE CONTEXT

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Matthias Thoma

von Deutschland

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik

REST-BASED INTERNET OF THINGS-SERVICES AND APPLICATIONS IN A SEMANTICS-AWARE ENTERPRISE CONTEXT

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Matthias Thoma

von Deutschland

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, 21.12.2016

Der Dekan:
Prof. Dr. Gilberto Colangelo

Abstract

The Internet of Things (IoT) is the vision of a global infrastructure of networked physical objects. In order to use IoT in a semantics-aware enterprise there are some challenges: IoT-devices should integrate into enterprises as seamlessly as possible, both at a modeling level and at a technology level. At the modeling level, enterprise systems are often customized by non-software experts. In addition, more and more systems are applying machine-learning technologies. Both human modeling and machine reasoning need a precise semantic description of the entities they work with and their meaning. At the technology level, enterprise systems traditionally use a different protocol stack than IoT-devices. Neither these traditional enterprise protocols nor the IoT-protocols are semantics-aware. Furthermore, IoT-devices have properties that are typically unknown to enterprise information systems, such as their limited energy and consequently their maintenance needs (i.e. swapping batteries). Keeping this total cost of ownership low is one of the primary goals of IoT-operators.

This thesis contributes to enterprise integration and semantic-aware integration by developing and evaluating two different approaches: a top-down approach and a bottom-up approach. The top-down approach scales down the existing OData enterprise protocol to very constrained IoT-devices. The bottom-up approach semantically enriches existing protocols. It consists of a service description language called Linked USDL for IoT, which allows to semantically describe services from a very high abstraction level, down to their technical realization. In order to utilize those approaches in an enterprise, we propose an architecture and abstractions that enable the integration of BPMN tools, semantic services, and constrained IoT-devices. We also investigated some reasons for the reluctance of developers to apply semantics, a behavior coined *semaphobia* in previous research. We evaluated our approach with an architecture evaluation method and through several experiments. The experiments were done on an experimental platform called Mote Runner on the following two hardware platforms: MEMSIC IRIS and Waspmote Pro.

We propose an application-layer based framework for reducing the energy consumption by putting nodes to sleep (sleepy nodes). Sleepy Nodes were implemented and evaluated on the Mote Runner platform. We introduce a monitoring framework based upon high-level information that utilizes sleepy nodes to increase the network lifetime. As part of the monitoring framework, we present three different scheduling strategies: A simple first fit, an exhaustive strategy, and a strategy called dynamic partitioning. Dynamic partitioning is based on the observation that it is possible to combine measurements under certain circumstances. We were able to show that dynamic partitioning performs significantly better than first fit and only slightly worse than the exhaustive strategy.

Acknowledgments

This PhD thesis is based on work performed during my employment as a PhD-student at the Institute of Computer Science of the University of Bern, Switzerland and during my employment as a researcher at SAP (Switzerland) Inc.

The research conducted in this thesis was partially supported by the European Union under grant IoT-A and grant FI-WARE. Over the years, I was fortunate to meet and work with many intelligent people who contributed to my thesis – some maybe without even being aware of it through all the fruitful discussions we had.

I would wish to thank everybody who provided me with support, ideas, understanding, discussion, and encouragement during the course of my PhD thesis. First, I express my gratitude to Prof. Dr. Torsten Braun, head of the Communication and Distributed Systems group (CDS), who supervised and encouraged my work. He offered me the chance to become an external PhD-student working with his research group and he always tried to get me on the right track. I would also like to thank Prof. Dr. Stefan Fischer for accepting to read this work and Prof. Dr. Paolo Favaro, who was willing to be co-examiner. Furthermore, I thank the SAP research community for the challenging work environment.

Many thanks go to my colleagues at SAP for being a great team. In particular, I wish to thank (in alphabetical order): Alexandru-Florian Antonescu, Stephan Haller, Carsten Magerkurth, Sonja Meyer, Basil Hess and Klaus Sperner. I would also like to thank the team at the University of Bern for the time with them, their feedback and giving me a different perspective on my work. In particular Thomas Staub, Markus Anwander, Gerald Wagenknecht, Desislava Dimitrova, Carlos Anastasiades, Zan Li, Denis Rosario, Andre Gomes, Jonnahtan Saltarin, Imad Aad, Dima Mansour, Almerima Jamakovic-Kapic, Eryk Schiller, Andreea Hossmann and Eirina Bourtsoulatze. Special thanks go to Daniela Schroth for her support in the administrative tasks and reserving a parking spot for us.

Furthermore, I must explicitly thank two collaborators: Marcus Oestreicher from IBM Research and Stefan Meissner from the University of Surrey. Marcus provided me with all support around the Mote Runner platform one could wish for. Stefan introduced me to ideas in the semantic services community and helped me write several papers.

I would also like to thank all the students, who contributed to this thesis in one way or another. In particular, thanks go out to our interns at SAP – namely Triantafyllos Afouras, Michael Gede, Theofilos Kakantousis, Theano Mintsi, Georgious Tentes, and Martin Zabel.

Contents

List of Figures

List of Tables

Listings

List of Algorithms

Part I

Introduction and Related Work

This work is divided into five parts. In this first part we present an introduction into the problem areas covered in this thesis and related work. In Chapter 1 we motivate our work, introduce the context of the work and its contributions and briefly embed it into current research streams. Chapter 2 gives a comprehensive overview of the used technologies and discusses related work.

Chapter 1

Introduction

This chapter provides the motivation for this work. It briefly introduces the context and embeds it into current research streams in the Internet of Things (IoT) and semantic enterprise integration. As the term ”Internet of Things” is not precisely defined, we start with a short overview of different viewpoints. Next, we describe the benefits and challenges of semantics-aware IoT integration into enterprise systems, in a world where are enterprises move away from traditional, often SOAP-based, architectures towards REST-based architectures with semantic modeling and reasoning engines. We then limit the scope and introduce the considered IoT-stack. IoT-devices can either be very limited or have processing capabilities equal to modern PCs. This thesis considers only very constrained devices. We used a novel VM-based operating system to conduct our experiments that we expected to be easily adoptable by enterprise developers as it supports JAVA and C#. We then summarize our contributions. Finally, we give a brief overview of the remainder of this work.

1.1 Internet of Things

The ”Internet of Things” (IoT) is the vision of a global infrastructure of networked physical objects [214]. The main idea of the IoT is the pervasive presence of things in the world and their incorporation into information systems through technologies such as sensors, RFID (Radio Frequency IDentification), actuators, and mobiles phones.

The term Internet of Things was coined in 1999 by Kevin Ashton [14], who co-founded the Auto-ID Center [19] at the Massachusetts Institute of Technology. Many technologies and ideas from the Auto-ID center, especially those centered on RFID technology, are now widely used for tracking all kinds of objects [129]. The success of RFID technology inspired the development of the IoT, which aims for interoperability on a larger scale. The Internet of Things is anticipated to be a key technology that will enable various new applications in an industrial context in sectors such as logistics [401, 256], retail [195, 159], home and building automation (smart home) [101], industrial production (smart factory) [408, 386], automotive [189], electricity generation and distribution (smart grid) [402, 193], and health care [188, 65], among many others.

The work presented in this thesis has been conducted as part of several research projects, mainly the Internet of Things-Architecture (IoT-A) [1] [324] project and the FI-WARE Private Public Partnership (PPP) [2] [52]. The objective of IoT-A was to create an architectural reference model and to define an initial set of key building blocks, which is envisioned as the crucial foundation to grow a future Internet of Things organically [308].

Despite the work of the IoT-A project and its predecessors, there is still no common definition of the term ”Internet of Things”. Different authors give different focus to some aspects of the IoT. Therefore, instead of giving an exact definition, we will follow Atzori et al. [18] and describe the IoT paradigm as a combination of three main visions: the ”Things”-oriented vision, the Internet-oriented vision and the semantics-oriented vision. Figure 1.1 shows the main concepts, technologies and standards classified by the vision [18] to which they belong to.

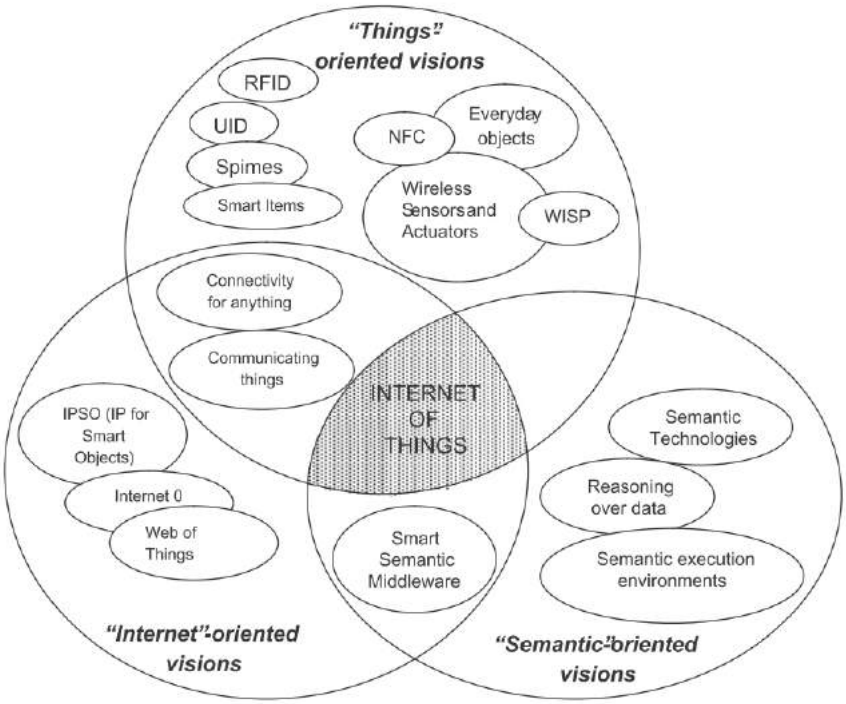


Figure 1.1: Converging visions leading to the Internet of Things [18]

The ”Internet-oriented” vision predicts the use of standardized Internet protocols, instead of custom protocols. The ”things-oriented” vision predicts interoperating (everyday) objects based on tags, sensor and actor technologies. Finally, the semantic-oriented vision predicts the use of semantic technology for all kinds of interoperability and information description and representation. Given the heterogeneity in IoT, not all

[1] http://www.iot-a.eu

[2] https://www.fi-ppp.eu

those visions have to be achieved to the same degree for them to be considered as a part of the Internet of Things. For example, RFID is often seen as part of the Internet of Things, even if the Internet-oriented vision and the semantic vision are not always completely fulfilled.

At this point, we are using all the terms in a rather intuitive way. In Chapter 3 we will formally define these terms, with a focus on services, as part of a service architecture.

1.2 Motivation and Problem Statement

The convergence of enterprise systems, machine-to-machine communication (M2M), and the sensing of and acting on physical objects has been recently in the focus of research and industry. They are anticipated to change the world as part of the so-called *Third Industrial Revolution* [320] or *Industry 4.0* [344]. Two trends can currently be observed in enterprise systems. One is that enterprises are moving away from traditional, often SOAP-based, architectures towards REST-based systems. Besides, more and more enterprises are basing their enterprise integration tools and libraries on semantic information. This semantic information allows easier service composition and orchestration, as well as enhanced M2M communication. Enabling semantics-aware enterprise integration allows different – even previously unknown – devices to connect to enterprise systems. Enterprise systems are foreseen to be able to fuse data from different sources, automatically convert data, and match incoming sensor streams to physical objects. They are supposed to perform automatic reasoning in order to react to complex situations through machine-learning algorithms. However, in order to enable such semantic-enhanced business processes in conjunction with very constrained Internet-of-Things four challenges arise:

1. The increased need of interaction with the physical world increases the complexity of enterprise IT systems. Therefore, IoT-devices should integrate into already existing software stacks as seamlessly as possible. The current shift away from traditional SOA architectures, as represented by BAPI and SOAP, towards REST-based enterprise architectures requires the use of REST-based protocols and architectures. The heterogeneity of IoT-devices needs to be abstracted away from the enterprise systems in order to reduce complexity.

2. Nowadays, enterprise business processes are often designed and modeled by people who are not software experts. In addition, more and more systems are applying machine-learning technologies. Both human modeling and machine reasoning need a precise description of the entities they work with and their meaning. The availability of a semantics-aware description of IoT services thus enables the usage of and reasoning on previously unknown devices and data, which can be integrated into an enterprise system without the need for a specialized software developer.

3. Very constrained devices make the use of typical enterprise protocols, such as, SOAP or BAPI cumbersome and sometimes even infeasible. Those traditional protocols are not semantics-aware either. Semantic additions to those existing protocols increases the amount of data to be processed and transmitted even further; therefore their applicability is limited even more. In order to be able to interact with a semantic-aware enterprise, the service interface of the IoT-devices should be based on lightweight protocols that are either semantics-aware by themselves without drastically increasing the data to be transmitted or can be described with a semantic service description.

4. The total cost of ownership of the solution should be low. Semantics add additional complexity to an IT system and IoT-devices come with their own limitations. A semantics-aware solution therefore needs to take the capabilities of the constrained devices into account. The benefits of using semantics should be balanced against the additional costs. A typical source of costs in wireless systems is maintenance and on-site work, and, thus, a decrease of these costs result in a lower total costs of ownership. Many industrial scenarios in which IoT devices are used operate in environments in which regular physical access to the sensor nodes and their batteries is not feasible. Especially, when manual (human) work is needed to replace the power source and/or network downtime is necessary for these maintenance tasks. Information available at an application-level, e.g. in semantic repositories, can be used to implement energy saving measures at application-layer.

The following sections discuss several aspects of four challenges, along with the kind of environment we use, in more detail: In Chapter 1.3 we describe the complexity problem and the anticipated semantic modeling approach. Next, in Chapter 1.4 we describe the kind of constrained devices used within this thesis. We then continue in Chapter 1.5 with the opportunities that application-layer energy saving can provide, reducing the total cost of ownership of an IoT solution.

1.3 Complexity of IoT-solutions and Semantics-aware Enterprise Integration

The overall complexity in enterprise IT systems is constantly increasing, as is the heterogeneity in such complex systems. This is not necessarily in the backend alone, where the trend is moving towards an even tighter standardization of components. Rather, the heterogeneity challenge that current and future enterprise IT systems have to face originates in the small mobile or embedded devices with which they have to interact. The integration of new services into enterprise IT systems nowadays is mainly done in a service-oriented way. Nonetheless, business systems and IoT devices, which are actually sensing or acting within the physical processes, use different protocol styles. Protocols for IoT-devices traditionally are tailored towards properties as small data

size and low computational complexity, while enterprise protocols typically were not designed with such constraints in mind. Currently, the integration of these devices is – from an enterprise point of view – still a cumbersome task that requires a lot of specialized knowledge, which a business process creator or enterprise developer might not have.

Semantics-aware services allow easier integration and interoperability on both a semantic and a modeling level. On the semantic level, the goal of semantic-aware services is to not only to provide machine-interpretable data, but to provide machine-understandable data in a service-oriented architecture. Exact interchange of semantically-enriched services allows the integration of previously unknown devices and services without human interaction. This allows an SOA-based integration of all kinds of services into reasoning [145], complex event processing [321] and machine learning systems [134]. It enables semantic querying [95, 67]. A semantic description of the technical layer also enables ERP system to access a technical interface on a device that is solely specified as part of the service description.

On the modeling level, semantic-service descriptions integrate well into recent modeling approaches. The goal of business IT is to have domain experts specifying the business process at a very high level, ignoring the technical details at the lower levels. As these new technologies emerge the competitive pressure for vendors of enterprise systems increases to deliver solutions that allow easy integration.

In future we foresee the use of Business Process Modeling (BPM) tools to model business processes that include IoT services. Many enterprises are working on extending the business process modeling notation (BPMN) [288, 87] to include also modeling elements for IoT-specific processes [267]. The BPMN standard includes a graphical and a machine-readable process representation. These process representations describe the planned process execution flow for the process execution engine of the enterprise system. To support both the design phase and the execution phase, business modelers need to know what they are modeling (e.g. what kind of things can be modeled, what services can be called and what they would get back from the services) [329, 265]. Modern modeling based IoT-integration are covered in detail in the works of Meyer et al. [265, 267], in Caracacs and Kramp et al. [74, 72, 70, 71] and Sungur et al. [366].

Our work is closely related to the modeling and enterprise integration work of Meyer et al. [265, 267, 268, 264] and Ruppen et al. [329, 330]. But its use in an enterprise modeling context is, of course, not limited to BPMN. Any other modeling language, for example CMMN [247], could be used as well. In the following, we use the BPMN model of Meyer et al. to illustrate the use of services and a description model. Their BPMN solution leverages on semantic descriptions.

Typically, the modeling expert is not a programming expert. Therefore, modeling experts are more interested to know what kinds of operations are available and on what kinds of entities they can operate, rather than on low-level programming related details. A solution to this are semantically-enriched service descriptions. An IoT process modeled in the graphical notation of BPMN is shown in Figure 1.2 [267].

The minimal business process, as modeled in Figure 1.2, consists of two lanes:

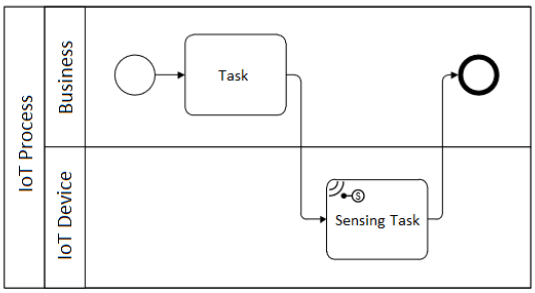


Figure 1.2: Graphical Modeling of a simple Internet of Things process (based on Meyer et al. [267])

one showing the actual business process and one showing the IoT-related tasks. The IoT-device shown in the IoT lane is represented as an IoT-sensing task.

The full approach for BPMN-based IoT integration is illustrated in Figure 1.3. As can be seen precise service definitions are necessary for such a system to work. Our work contributes to the technical aspects of IoT business process integration by introducing service descriptions and a semantics-aware underlying platform that can be used as a basis for a modeling framework. The business process binds to a RESTful API running on an IoT device. The RESTful API itself exposes a service running on an IoT device, for example, a temperature service running on a sensor node. The node itself is either attached to or related to a physical entity. The term physical entity refers to any physical object that is relevant from a user or application perspective. The service running on the IoT node and its technical interface are described in a description model that can be used by the modeler.

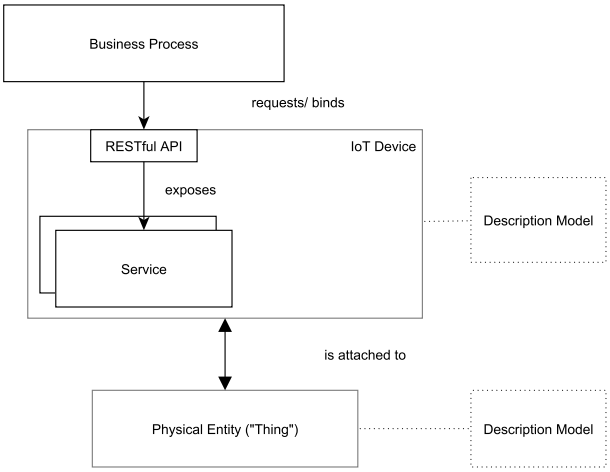


Figure 1.3: IoT-service BPMN integration (based on Ruppen et al. [329])

There are basically two different possibilities that could be used within such a business process tool-chain: a bottom-up and a top-down approach. Both approaches are visualized in Figure 1.4. Following a bottom-up approach means describing existing IoT-services and their technical interfaces by using an external service description language. This approach is separate from the actual service running on the IoT device and the protocol stack or services running on the sensor nodes do not need to be changed: The service description simply describes the offerings of that particular service. In contrast, a top-down approach would mean to using an existing enterprise service protocol and running it – maybe in a scaled-down or limited version – on the sensor node itself. We use the Open Data Protocol (OData) as the already existing enterprise protocol. In the bottom-up approach, we use a service description language named Linked USDL for IoT. We will introduce both in detail in the remainder of this thesis.

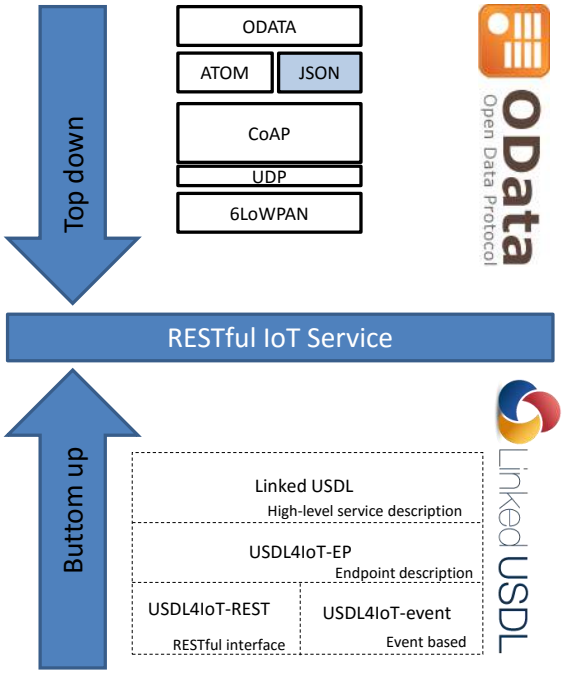


Figure 1.4: Bottom up vs. top-down approaches

1.4 Internet of Things Stack

The Internet of Things (IoT) is a broad and diverse field. The hardware alone can range from small sensing devices to smartphones that have a comparable processing power to desktop machines. In the following, we will briefly introduce the Internet of Things hardware and protocol stack we are considering in this thesis. It is important to note that there is no generic IoT stack. Therefore, whenever we refer to IoT stack or IoT

protocol stack, we refer to the stack as presented here. In situations where, for example, only mobile phones or tablets are involved, the traditional Internet stack could be used. Nonetheless, in our experiments we are targeting low-resource, very constrained devices running REST-based Internet technologies. All the concepts presented here can be scaled up and used in less constrained scenarios.

RFC 7228 [54] classifies constrained devices into three disjunctive classes. The three classes are shown in Table 1.1. We added the common processing capabilities to the respective classes, though those are not part of the official RFC 7228 classification.

Class	Data Size	Code Size	CPU
0	$<$ 10 KiB	$<$ 100 KiB	$<$ 15 MHz (often 8 bit)
1	10 KiB	100 KiB	25-30 MHz
2	50 KiB	250 KiB	$>$ 30 MHz (sometimes 32-bit)

Table 1.1: Constrained devices classification (based upon RFC7228 [54] with CPU capabilities added)

Class 0 devices are very restricted. Most sensor nodes are part of this category. Typically, Class 0 devices have very limited processing power, often less than 15 MHz [233], with 8bit architectures, and non-multithreaded and non-superscalar CPUs. Integrating such devices into an IP-based system is challenging. Often the help of network gateways or (transparent) proxies is necessary.

Class 1 devices are still somewhat limited, but they are already capable of running rather complex protocols. This class of devices can often be easily integrated into an IP-based network, without the dedicated help of a (transparent) proxy or gateway.

Class 2 devices are capable of nearly everything modern multiple-purpose systems can do. But they often need to fulfill certain energy constraints. This can be done, for instance, by using specialized protocols.

We are mainly interested in constrained devices and in the interoperability between such constrained devices and enterprise systems. While of our experiments used Class 0 devices, we are not limiting ourselves to such devices. Using devices at the lower end of the classification has the advantage of being able to work with any type of constrained device later on, as upsizing to a more powerful device is easier than downsizing to an even more constrained device than the one originally used.

Moore's Law [3] [276] is also valid in the embedded domain [291, 166]. Nonetheless, the developments in terms of processing power are at a much lower scale because cost and energy consumption are the major drivers in many applications, and not necessarily processing power. Therefore, the classification presented in Table 1.1 will be subject to

[3] In 1965 Gordon Moore, then with Fairchild Semiconductor, predicted that "the complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain constant for at least 10 years." – it is astonishing that he based his law on only three datapoints [338].

change in the upcoming years.

This work does rely on the traditional Internet stack, which is often used in current enterprise applications. Instead, we base our work on a stack on top of 6LoWPAN, UDP and the recent Constrained Application Layer Protocol (CoAP). The stack is shown in Figure 1.5. The basic ideas behind these technologies will be introduced in Chapter 2.

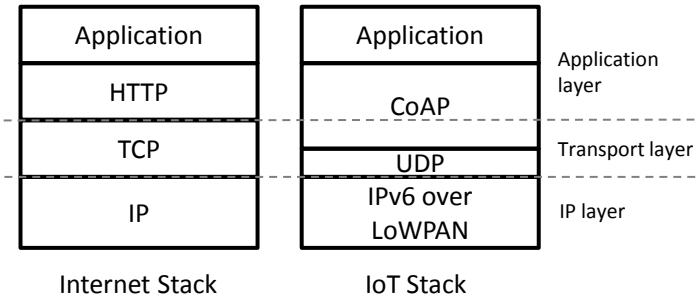


Figure 1.5: Assumed IoT protocol stack

Both protocols, HTTP and CoAP, are designed according to the RESTful paradigm. RESTful architectures are currently predominant on the Internet. In 2010, there were already five times more REST-based services available than WS*-based services. Between 2007 and 2010 alone, a 900% increase of publicly available REST services was observed, while the number of WS-* based services merely tripled [2]. Furthermore, also in the enterprise world, more and more systems are being moved to RESTful designs [279, 208]. For example, RESTful services for cloud applications and mobile applications are currently mainly written in RESTful way [89, 328], or the initiatives of SAP and Microsoft to make the RESTful OData protocol suite the RESTful successor of the WS-* family. Chapter 2.4 will provide a closer look at the details.

The used hardware and software platforms are visualized in Figure 1.6. The software was based on the Mote Runner IoT environment [182]. Mote Runner is an (prototype) operating system and integrated development environment published by IBM Research Zurich. Its development began in 2010. Mote Runner's main objective is to bring modern programming concepts to very constrained Internet of Things devices. It has a custom-written virtual machine and compilers for C# and JAVA. It also features an extensible and accurate simulation environment. We expect the Mote Runner platform to be well suited for enterprise application developers, as most of them are familiar with one of those languages and nearly no specialized knowledge is necessary. We explain the capabilities of the Mote Runner environment in Chapter 2.2.

The experiments were conducted on either the IRIS platform or the Waspmote Pro platform. Both platforms are Class 0 devices according to the classification in Table 1.1. They mainly differ in their capabilities to run in different hardware modes and different power consumption characteristics. The two platforms are introduced in Chapter 2.1.

We designed and implemented a measurement framework based on Mote Runner.

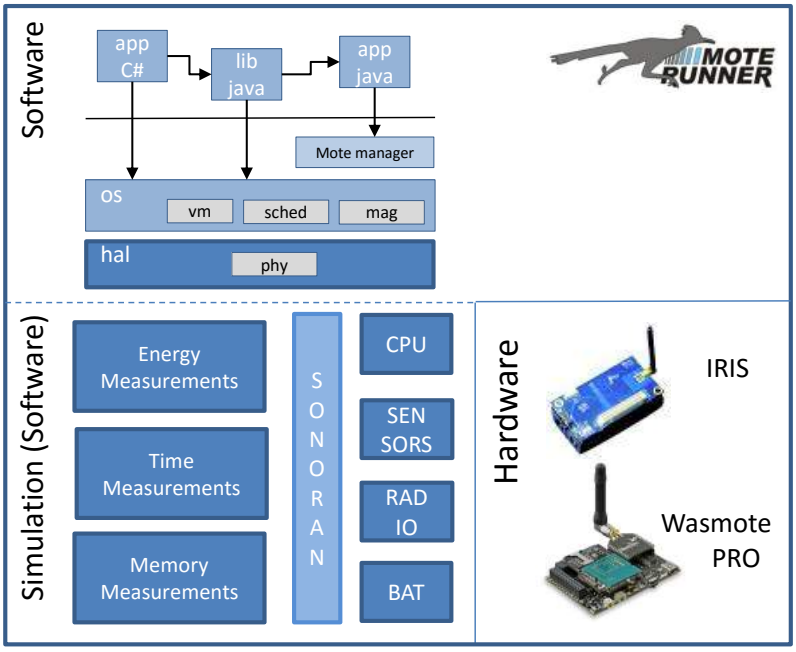


Figure 1.6: Hardware/Software stack

This framework enables us to measure power consumption, all kinds of time-based measurements as well as the memory consumption. Based on this, a comprehensive view of the system state and its hardware/software profile can be determined. We describe this measurement framework in Chapter 8.6.

1.5 Application-layer Energy Saving

A typical problem in wireless systems is the reduction of maintenance and on-site work, and, thus, a decrease in the total costs of ownership. Many industrial scenarios in which IoT devices are used – like in the supply chain or in automation – often operate in environments in which regular physical access to the sensor nodes and their batteries is not feasible. Especially, when manual (human) work is needed to replace the power source and/or network downtime is necessary for these maintenance tasks. A lot of research on energy saving has been done on the network, MAC layer and on the hardware. Semantics-aware enterprises allow utilizing the application layer as well. In remote monitoring scenarios, a close-to-zero on-site presence for maintenance and administration is desirable [201, 243, 172]. The problem of such a close to zero-maintenance has several management and operational aspects attached to it: Remote management of IoT devices allows configuring and reconfiguring nodes without on-site interactions. Frameworks such as MARWIS [393] or ADAM [364] allow the remote management of devices. This includes remote code updates [62, 215]. In the operations

network lifetime and energy-saving are the main drivers. The longer a node is running, the less costs-intensive the system typically is. In many industrial scenarios, regular physical access to the sensor nodes and their batteries is not a feasible option.

In terms of operations, several possible approaches address the IoT lifetime problem: Most are on the hardware-layer (CPU sleep modes, radio states) and they try to increase the radio-off time and decrease the CPU power consumption (e.g. frequency scaling or voltage scaling). Recently, the focus of research has shifted to some extend, towards the upper layers of a typical IoT-stack. While lower layer support for sleep states performs quite well, these functionalities are unaware of any timing or operational aspects of the application layer and therefore cannot leverage on that basis. A semantics-aware architecture makes this information available. It allows taking more decisions at the application layer. A concept called sleepy nodes [372, 311, 325, 172] assumes that many IoT-nodes will be used rather infrequently and, thus, they can sleep for longer periods of time. Nonetheless, for the user of an IoT system a node should never appear as unavailable. Despite the energy constraints, the user of an IoT-system should be served as best as possible taking known system behavior into account, anticipating or calculating the expected system usage. Even in case where the system did not anticipate the usage of a node and the node is sleeping, often there is value in information that is either cached or arriving late.

1.6 Contributions

During the development of the work and projects, forming this thesis, a number of contributions have been achieved. The main contributions can be summarized as follows:

1. **IoT-Services and Classification** We realized that the term ”Internet of Things-service” or ”IoT-service” is either used in a rather undefined or intuitive way and that it is only infrequently defined. We worked on a nomenclature of the key concepts of the Internet of Things domain and their relationships [380, 56, 35, 259], with a focus on service. Therefore, we surveyed [380, 259] the existing literature and compared existing uses of the term IoT-service. Based on our work on IoT concepts we introduced – to our knowledge – one of the first comprehensive definition and classification of IoT-service. Compared to related definitions, it does not only define service as a technical interface, but as more comprehensive concept that takes the physical nature of the Internet of Things into account.

2. **Enterprise embedded IoT-services** We present conceptual work on the architectural building blocks and design considerations for an Internet of Things service integration framework [381, 375, 382, 373, 374]. We present a novel IoT-enterprise integration framework based on a linked services approach [375, 373] to enable interoperability between a semantics-aware enterprise and IoT-devices. Its novelty is the combination of Linked Services and the distribution of services,

which combined, serve the needs of both enterprises and constrained IoT-devices. We argue that (distributed) Linked Services are especially well suited for IoT-applications, given their limited battery power, as well as storage and processing constraints. Thus, Linked Services do not only enable interoperability, but also suit the needs of constrained devices.

3. **Semantics-awareness and Linked USDL for IoT** We introduce an extension to Linked USDL [76], called Linked USDL for IoT [378, 259, 382, 187], to support the Internet of Things. We contribute four new vocabularies to Linked USDL to support Internet of Things applications. Each of these vocabularies targets a specific aspect of the Internet of Things. The covered aspects are events (usdl4iot-event), quality of information (usdl4iot-qoi), technical endpoints (usdl4iot-ep), and the REST paradigm (usdl4iot-rest). Furthermore, we embedded it into related ontologies. In the context of our semantics-aware architecture Linked USDL for IoT can be used to establish a bottom-up approach. In order to be able to evaluate Linked USDL and our architecture, we based our architecture evaluation on an architectural evaluation method, IoT-stakeholder workshops and a survey that allowed us to get insight into the current views of the enterprise and academia on the IoT community with respect to semantics [34, 377, 259]. The architecture evaluation showed that our architecture is capable of supporting distributed Linked Services. We also observed a trend and some anticipation towards RESTful designs, as well as an anticipation of an increased use of and interest in semantics. The field also seems to be more mature nowadays, moving towards standardized technology. A need for standardization of semantic vocabularies has been identified as well as a perceived lack of training.

4. **OData for very constrained IoT-devices** In conjunction with our architecture, a service description and a technical protocol stack is needed. Interoperability between enterprise systems and very constrained (e.g. Class 0) IoT devices is usually accomplished by having gateways that translate between the protocols used in the IoT domain and the protocols used in the enterprise domain. We investigate a top-down and a bottom-up approach. As the protocol for the top-down approach we chose OData [379]. OData is currently being discussed as the solution to interoperability issues within enterprise systems. According to our knowledge, we are the first who applied OData to sensor nodes. We showed the feasibility of this approach for sensor nodes [379] and evaluated the solution on a stack based on 6LowPAN and CoAP. We compared OData in its two representations JSON and ATOM with a minimal CoAP-only payload. The difference between this CoAP baseline and the OData representation can be considered "the price" to be paid for a standards-conform semantic interoperability.

5. **Sleepy Nodes Implementation and Monitoring Framework** Sleepy Nodes are an application-related research field [372, 311, 325, 172]. Sleepy nodes are sensor nodes that might be in an energy-saving mode for some time and, thus,

not available for any communication. We developed a complete prototype system [372] running on two hardware platforms: MEMSIC Iris and Waspmote Pro. Compared to energy saving measurements on protocol level or on hardware level (e.g. frequency or voltage scaling) we leverage on information that can be stored in semantic repositories based up on application layer knowledge of the application and the associated things to be monitored. We derived a hybrid energy model for both platforms. The energy model can be used to calculate beneficial sleep times. We were able to show the general benefit of application-layer sleepy nodes. We experimentally evaluated three different strategies: a first fit, an exhaustive approach and a heuristic named dynamic partitioning. We demonstrated that within a time-sliced (windowed) environment this heuristic based upon combinable subsequent measurements further reduces the energy consumption. It achieved only slightly worse network lifetimes than the exhaustive approach and generally performs better than first fit.

1.7 Outline

This thesis is divided into five parts. The remainder of this first part, Chapter 2, introduces the foundations and related work required to understand this thesis and introduce related work. We first describe the sensor network hardware and software platform used. Furthermore, we give a brief introduction to 6LoWPAN and the MAC layer protocol 802.15.4 on top of which the software platform is operating. In addition, a brief introduction into semantic modeling techniques is presented. We then present related research approaches, mainly in the areas of semantic modeling area and service integration.

The second part introduces our conceptual work on Internet of Things service architectures. Chapter 3 presents the building blocks of an Internet of Things architecture with a focus on services. We then proceed, in Chapter 4, with describing how IoT-services can be embedded into an enterprise environment.

The third part of this thesis introduces the service descriptions and protocols we developed. Chapter 5 presents an extension of the Linked USDL service description language and further extends it towards supporting the Internet of Things. We use a custom implementation of CoAP for a reactive VM-based OS, which we present in Chapter 6. In Chapter 6.3, based on the CoAP implementation, we describe how we used a downscaled OData implementation for small embedded IoT devices. Finally, Chapter 7 concludes this part by presenting a lightweight REST-style Sleepy Node protocol and a window-based integration into enterprise systems.

In the fourth part, which consists of Chapter 8, we present the evaluation results of the aforementioned contributions. We distinguish between an empirical evaluation and an experimental evaluation.

Finally, the fifth part concludes the thesis. Chapter 9 first summarizes the thesis and then gives further research directions that could develop out of this thesis.

Chapter 2

Foundations and Related Work

This work relates to multiple research and practice areas – ranging from networks and embedded systems, and enterprise systems and service science, to software engineering. This chapter introduces the necessary basic concepts and technologies that will assist in understanding the research presented in this thesis. Some attention is given to semantic modeling technologies, the embedded software, and the hardware stack. Furthermore, we give an overview of related work. We discuss previous approaches for describing services and semantically enrich them. We then continue with discussing enterprise integration based upon Web Services, REST services, and business process modeling.

2.1 Hardware

In this section, we first give a general overview of WSN/IoT sensor devices, with focus on the most well-known ones: MicaZ, TelosB, BTNode, and SUN Spot. We then introduce the hardware used in our experiments – IRIS and Waspmote Pro. IRIS is comparable to TelosB-class motes. Waspmote is based on a more recent platform and it features advanced capabilities like a very precise real-time clock (RTC).

2.1.1 Overview

The main components of a typical WSN/IoT sensor/actuator node are illustrated in Figure 2.1. The power supply unit (PSU) is often a battery, but it can also be the power grid directly. Battery powered devices can have as less as 2Ah of energy, and up to 85Ah and more [126]. Sometimes, IoT-devices are equipped with solar panels [10] or further energy harvesting technologies, such as temperature differences [342] or piezo electronic conversion [137] .

Sensors and actuators are either on-board (often temperature) or can be ”plugged in” on-demand. Most sensor platforms have GPIO, UART, and/or I2C interfaces to adopt to the actual needs of a specific application. Random access memory (RAM) is often a limiting factor. As shown in Table 1.1, in Class 1 and Class 2 IoT devices its not more than 10 KiB or 50 KiB, down to much less than 10 KiB in Class 0 nodes.

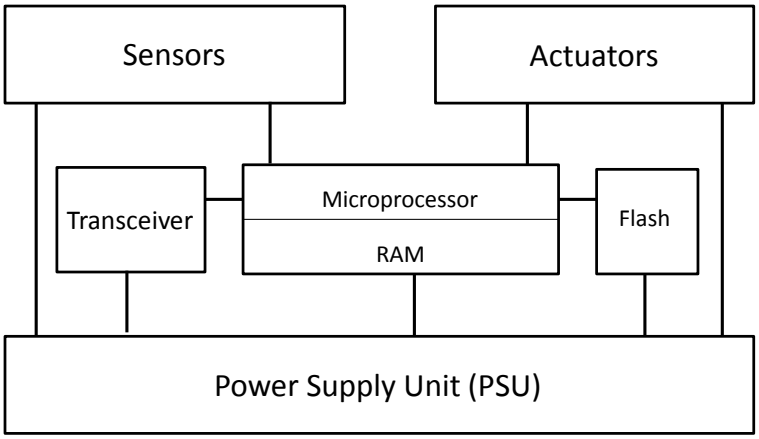
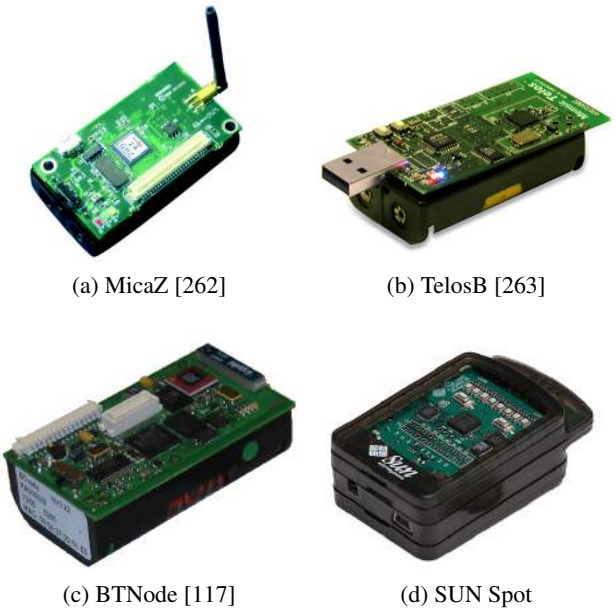


Figure 2.1: Main components of a typical sensor network platform



(a) MicaZ [262]

(b) TelosB [263]

(c) BTNode [117]

(d) SUN Spot

Figure 2.2: Sensor Nodes

One can observe two classes of sensor devices: The more ”classical” devices which have been developed in the late 90s and early 2000s, like the Berkeley family of Nodes [327]. The most prominent members of this family are WeC [258] (1999), Mica [167] (2001), Mica2 [100] (2002), MicaZ [260] (2004), Telos / Telos Rev. B [303] (2004). Mica2 and MicaZ are almost identical, but use different Radio hardware.

Similar platforms as the Berkeley line of nodes have been developed at various places. For example, BTNode [46, 45, 44] from the Federal Institute of Technology (ETH) Zurich, ScatterWeb [340, 339, 21] at the Free University of Berlin, IMote [206] and IMote2 [277] from Intel, and the Zolertia Z1 [407] motes. IMote, an exception at that time, was based on a Bluetooth communication stack. IMote2 introduced 802.15.4 support in 2008. One of the first platforms with VM-based high-level language support was the Sun Small Programmable Object Technology, better known as Sun SPOT [361]. The main difference, compared to the state of the art at that time, was the use of the squawk virtual machine [358, 359]. MicaZ, TelosB, BTNode and SunSPOT are shown in Figure 2.2. A comparison of their main features is given in table 2.1.

Although MicaZ and Telos were introduced as early as 2004, they are still used in many research papers even in 2014 (e.g. Basu et al. [31] or Zhu et al. [405]). More recent platforms are slowly being adopted by the research community. A recent boost in usage of more advanced hardware has been driven by projects like Arduino. This is mainly because it provides a modern platform and sensor/actor technology for a reasonable price.

	MicaZ	TelosB	BTNode	Sun SPOT
CPU	Atmega128L[16]	MSP430[369]	Atmega128L	ARM920T[15]
Radio	CC2420 [371]	CC2420	CC1000 [370]	CC2420
RAM	4 kb	10 kb	64 kb	512 kb
BUS	8 bit	16 bit	8 bit	32 bit
Clock	8 MHz	4-8 MHz	7.37 MHz	180 MHz
Flash	128 kB	48 kB	128 kB	4096 kB
Weight	16g	15g	30g	34g

Table 2.1: Sensor node comparison

In this work uses two (simulated) hardware platforms: IRIS [261] from MEMSIC which is comparable to the Berkeley families Mica2 and TelosB, and the state of the art Waspmote Pro [233] platform.

2.1.2 IRIS

The IRIS platform is the latest generation of Motes from MEMSIC, which also distributes the MICAz and TelosB nodes. As a typical embedded platform it has only a very limited processing capabilities (8 bit, 8 MHz, AVR RISC architecture) amount of memory (8 KByte), but plenty of flash: for program as well as for data. It as a Class 0

device according to the constrained devices classification [54] (see Chapter 1.1). The main features of the platform as used in our simulations are shown in Table 2.2. More fine-grained power specifications, specificaly for IRIS on Mote Runner, can be found in Caracacs et al. [69].

Figure 2.3: IRIS mote

IRIS		
CPU		
Type	ATmega 1281	
Architecture	RISC	
Instructions set	AVR, 135 instruction	
Memory		
Program flash	128 KByte	
Data flash	512 KByte	
RAM	8 KByte	
Configuration EEPROM	4 KByte	
RF Transceiver		
Frequency	2405-2480 MHz	
Data rate	250 kbps	
RF power	3 dBm (typ)	
Receive sensitivety	-101 dBm (typ)	
Power Specifcations		
Current draw (CPU)	8 mA	CPU, active
	8 μA	CPU, sleeping
Current draw	16mA	listening
	17 mA	TXing@+3 dBm

Table 2.2: IRIS mote (based on [261] and [17])

2.1.3 Waspmote Pro

The Waspmote Pro is a commercially available IoT sensor/actor node. Compared to most previous research (e.g. those based on MicaZ or TELOSB), it did not develop out of a research prototype. It uses state-of-the-art hardware and sensors, including advanced energy sleep modes and a precise temperature-compensated crystal clock with RTC capabilities [234]. It is a typical embedded platform (8 bit, 14 MHz, AVR RISC architecture). The power consumption is a little bit worse on the Waspmote Pro compared to the IRIS mote. Nonetheless, it features a hibernate mode where the mote consumes very little energy, but needs some time to wakeup. In simulation, we assumed the hibernate mode to use 0.7μA as per Waspmote datasheet [233] and confirmed by experimental results [360]. A (simulated) hibernate reset to took 8ms. Only, when an RTC alarm is activated, is the board powered again. Possible sleeping (hibernating) interval values go from seconds to minutes, hours, and even multiple days [235]. The features of the platform as used in our experiments are outlined in Table 2.3

Waspmote Pro		
CPU		
Type	ATmega 1281	
Architecture	RISC	
Instructions set	AVR, 135 instruction	
Memory		
Program flash Memory	128 KByte	
Serial flash	up to 2GB	
RAM	8 KByte	
SD-Card	2 GB	
Configuration EEPROM	4 KByte	
RF Transceiver		
Frequency	2405-2480 MHz	
Data rate	250 kbps	
RF power	3 dBm (typ)	
Receive sensitivety	-101 dBm (typ)	
Power Specifcations		
Current draw	15 mA	CPU, active
	55 μA	sleeping
	0.7 μA	hibernating
Current draw	12.3mA	listening
	14 mA	TXing@+3 dBm

Table 2.3: Simulated Waspmote (based on datasheets [235, 233, 234])

2.2 Mote Runner System

In the following we introduce the Mote Runner System. It is a VM-based operating system for embedded devices that we used for software development and evaluation of our protocols and algorithms.

2.2.1 Overview

IBM Mote Runner [73] is a run-time platform and development environment for the Internet of Things. Its operating system is suitable for a variety of CPUs (8,16, or 32 bit). Its minimal requirements are are just 8 KB RAM, 64 KB Flash and an 8 bit CPU; hence, it is especially suited for IoT applications. It follows a virtualization concept. Applications (written in C# or Java) are compiled into bytecode and executed by a virtual machine (VM). Figure 2.5 shows the main components of the Mote Runner. The Mote Runner system consists of a simulation environment, a stack or run-time environment that is running on the mote, and a JavaScript gateway called Sonoran to interact with both. Sonoran[1] is an off-mote, JavaScript based programming and management environment [181]. Furthermore, Mote Runner has a framework for developing web applications called Comote. It is mentioned only to ensure for completeness.

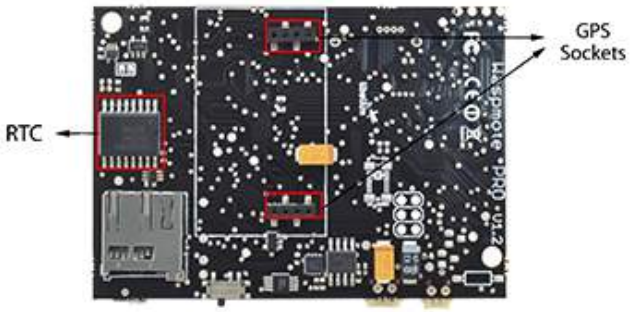
The actual environment as it is running on the mote is shown in Figure 2.6. All the applications are written in either C# or Java. Both languages can share libraries written in either C# or Java. The MRv6 protocol, which is explained in detail in Chapter 2.3.3, has been written in C# but is used as part of our Java code. The operating system itself (including the VM) and the hardware abstraction layer (HAL) are written in C.

As mentioned, the Mote Runner system uses Java and C# as high-level programming languages. This is supposed to decrease the development time of sensor network applications as developers can work with a familiar programming language. The reactive nature of the Mote Runner system is supposed to reduce concurrency-related programming errors. Nonetheless, there are some limitations compared to standard C# or Java. The main differences are as follows [181]:

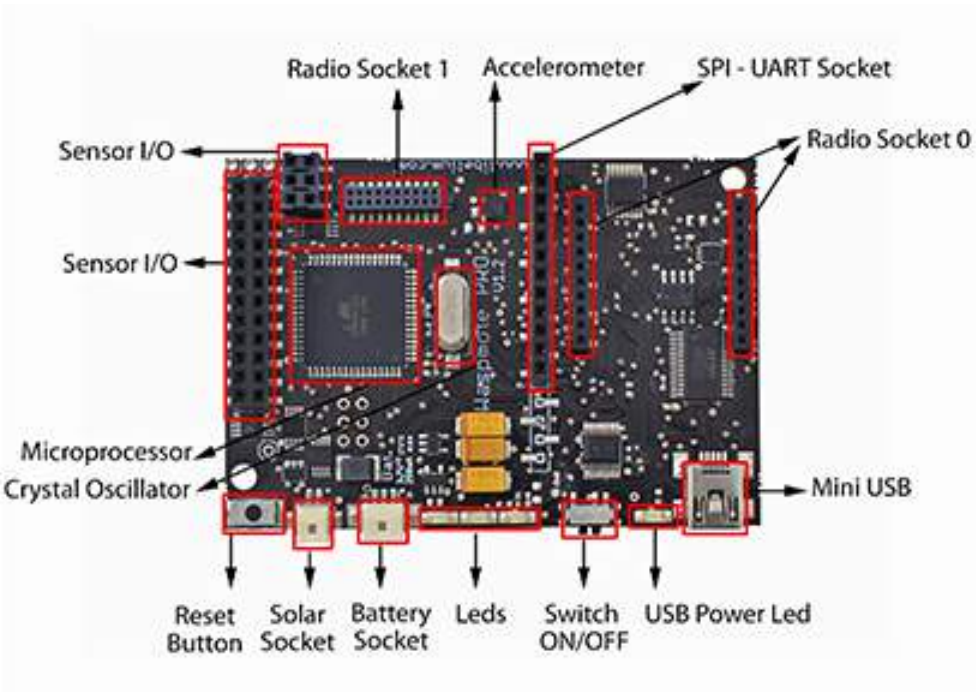
- No native float and double data types. Currently only fixed comma operations are possible
- Only 32-bit integer arithmetic. 64-bit or longer integer arithmetic is not supported
- Multi-dimensional arrays are not supported
- Run-time inspection, including getClass methods or Reflection is not supported
- Standard runtime APIs are not available.
- Enumerations are not supported

[1]The developers of Mote Runner give the following explanation for their naming scheme [181]: The Mote Runner and several of its components are themed around the Arizona desert. Mote Runner alludes to roadrunner, which is a bird in the cuckoo family. Saguaro is a large, tree-sized cactus specifies. Sonoran is the name of a desert south of Mojave and, finally, Comote stems from and is pronounced alike coyote, the American jackal or the prairie wolf.

(a) Node



(b) Waspmote Pro components (top)



(c) Waspmote Pro components (bottom)

Figure 2.4: Waspmote Pro

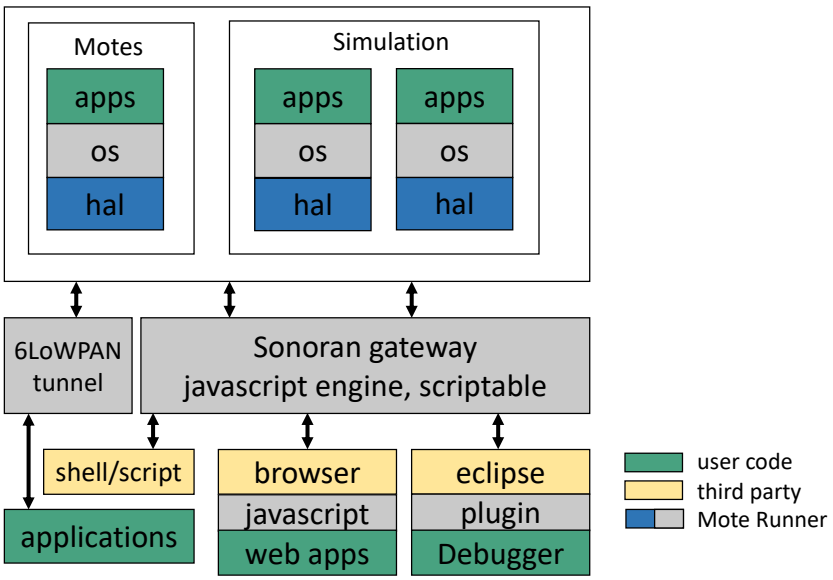


Figure 2.5: Mote Runner system overview (based on IBM [181])

- Templates are not supported
- Inner classes in C# are not supported (but are in Java)
- Multicast Delegates are not supported
- Threads/Multithreading are not supported (reactive programming model)
- Bool and Boolean arrays are not supported
- String type and String array are not supported
- Boxing is not supported
- Integer size is 16bit, not 32bit

The memory model of Mote Runner is similar to that of Java. It uses a Garbage Collector algorithm that is only executed when the VM is not active. However, as execution of the the garbage collection algorithm takes some time, it could interfere with event deadlines.

2.2.2 Toolchain

As the Mote Runner VM is based on an optimized bytecode and not the original, the system comes with its own tool chain – that supports C# and Java at the same time. Libraries written in one language can be used in the other language and vice versa. The toolchain is shown in Figure 2.7 [73]. The parts that have been specifically written for the Mote Runner system are in gray. The Java and C# compilers are the original compilers as delivered by Sun and the Mono project.

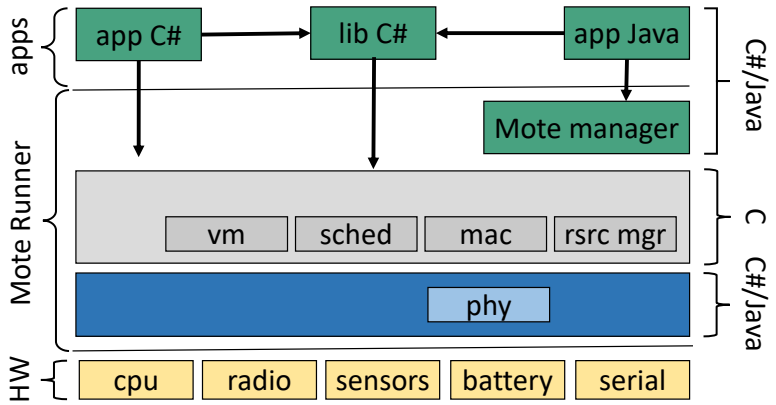


Figure 2.6: Mote Runner System overview – On Mote Runtime environment ([181])

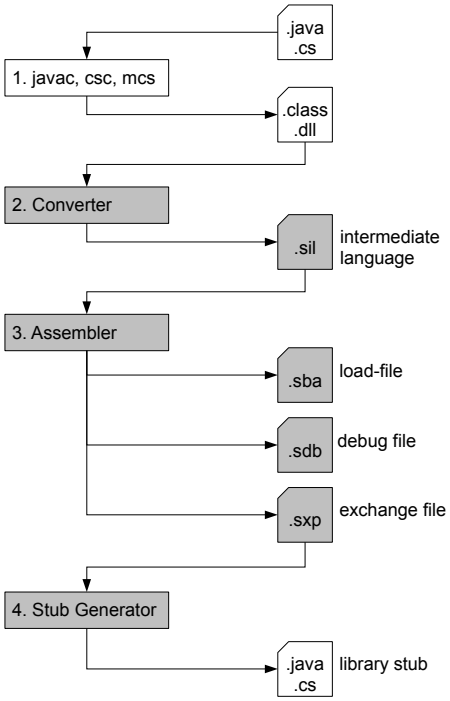


Figure 2.7: Mote Runner Tool Chain (Caracas et al. [73])

Mote Runner uses a purely reactive programming model [73]. No further support for concurrency (e.g. Threads) is provided. In a reactive programming model the application registers methods to be executed whenever a corresponding event res, for example a timer event. From an VM implementation point of view this requires one and only one virtual machine stack and no synchronization [73]. Only one event handler is active at any given point in time. This typically results in simple application code.

Dec 2014	Mote Runner Beta 17.1.8	Supports WiMOD880 developer kits as well as LoRaMote devices
Jun 2014	Mote Runner Beta 16	Supports IMST WiMOD880 developer kits as well as LoRa Blipper (v2) motes.
Dec 2013	Mote Runner Beta 14	Improved support for Waspmote Pro and IRIS.
Oct 2013	Mote Runner Beta 13	First version for Waspmote Pro. Shipped with an improved version of the 6LoWPAN implementation, called MRv6.
Mar 2013	Mote Runner Beta 11	This version was the first to ship a 6LoWPAN implementation.
Aug 2012	Mote Runner Beta 9.0	
Apr 2012	Mote Runner Beta 8.2	
Mar 2012	Mote Runner Beta 8.1	
Mar 2012	Mote Runner Beta 8	This was the first version we worked with. It did not support 6LoWPAN yet and was only usable for experiments on the MAC layer.
Aug 2011	Meta Runner Beta 5	
Jun 2011	Mote Runner Beta 4	
Dec 2010	Mote Runner Beta 3.0	
Jul 2010	Mote Runner SDK 2	

Table 2.4: Mote Runner timeline (based on information from [182])

Nonetheless, handling of events should not require lengthy processing. Such lengthy processing must be split up into smaller pieces which are driven by, for example, a priority task list or by some timer event [73].

2.2.3 Version History

The Mote Runner system was still a research prototype when we conducted this research. Several Mote Runner versions were used while this thesis was written. Mote Runner contributed to the EU-funded research project IoT-A, which also partly funded this research. We used publicly available snapshots of Mote Runner, as well as versions that were initially only made available to project partners. In Table 2.4 we give the version history of Mote Runner and a brief overview of what has been added.

2.2.4 Hardware

Mote Runner is a software platform that provides applications a vm-based (byte-code) abstraction of various underlying hardware platforms. However, not all versions of Mote Runner support all hardware platforms. Versions beta16 and beta17 do not support the 2.4GHz based devices.

AVR RAVEN Atmel RZRAVEN is a sensor mode consisting of an AT86RF230 radio transceiver and an AVR ATmega1284P microcontroller. AVR RAVEN has been supported since Mote Runner beta4.

AVR RZUSBSTICK AVR RZUSBSTICK is an USB stick based on the USB microcontroller AT90USB1287 and the AT86RF230 radio transceiver chip. AVR RZUSBSTICK has been supported since Mote Runner beta4.

WiMOD WiMOD is the name of a series of modules from IMST [2] for the 169 MHz, 433 MHz, 868 MHz and 2.4 GHz frequency bands. WiMOD has been supported since Mote Runner beta16.

Blipper LoRaMote A long range (LoRA) technology mode from IMST. Blipper LoRAMote is supported since beta16.

SmartMesh IP SmartMesh IP, developed by Linear[3], is a sensor network platform aiming for 6LoWPAN support. SmartMesh IP is supported since beta16.

IRIS The MEMSIC IRIS was one of the first types of devices supported by Mote Runner. It is one of the platforms that we use in our research. More details about the IRIS platform can be found in Chapter 2.1.2. IRIS has been (publicly) supported since Mote Runner beta3.

Waspmote Pro Waspmote Pro is a recent sensor network platform developed by Libellium. It is one of the platforms we are using in our research. More details about the Waspmote Pro platform can be found in Chapter 2.1.3. Waspmote Pro has been supported since Mote Runner beta13.1.

2.3 Protocols

The following presents two protocols that we used throughout this thesis. The first one is 802.15.4, the MAC layer protocol is used by the IRIS and Waspmote radios. The second one is 6LoWPAN [221]. 6LoWPAN is a networking layer protocol, that is often used on top of 802.15.4. Both protocols, 802.15.4 as well as 6LoWPAN, are very sophisticated. They fill several hundred pages of standardization documents. We give only a brief introduction into the main concepts needed to follow the remainder of

[2] www.wireless-solutions.de

[3] www.linear.com/products/smartmesh_ip

this thesis. Furthermore, we expect at least a basic understanding of typical networking protocol stacks.

2.3.1 802.15.4

802.15.4 is a MAC layer protocol. It controls access to the physical medium. The physical medium is shared by all nodes that are in the vicinity of each other. The MAC protocol, therefore, provides a means of determining when it is safe to send and when it is not. The 802.15.4 MAC layer, which is used throughout this work, provides channel access management, acknowledgment of frame reception and frame validation.

The 802.15.4 protocol is defined by the 802.15.4-2003 [183] standard. The original IEEE 802.15.4-2003 was revised in 2006 [184]. We will refer to this amendment as IEEE 802.15.4-2006. Another amendment was released in 2007 as IEEE 802.15.4a [185]. IEEE 802.15.4 defines Layer 1 (Physical Layer, PHY) and Layer 2 (Media Access Control Layer, MAC) of the ISO/OSI reference model. The main properties of an IEEE 802.15.4 network are:

- Packet size of a maximum of 127 bytes
- Tree, star or mesh topology
- Good energy vs. performance tradeoff
- Up to 65,536 nodes

Frequency (MHz)	Channels	Data rate (kBit/s)
868	1	20 (PSS: 250)
915	10 (2003), 30 (2006)	40 (PSS: 250)
2400	16	250

Table 2.5: 802.15.4 Frequencies, channels and data rates

The physical layer (PHY), according the original standard, can work in three different frequency bands – namely 868 MHz, 915 MHz and 2.4 GHz. 802.15.4-2006 introduced a technology called Parallel Spread Spectrum (PSS) that is able to increase the data rates in the 868 and 915 bands. Table 2.5 shows the possible 802.15.4 (not 802.15.4a) frequencies, channels and data rates. The 2007 standard 802.15.4a supports frequencies below 1 GHz, between 3 and 5 GHz and between 6 and 10 GHz. The main differences between 802.15.4 and 802.15.4.a are [192] the introduction of alternative PHYs – namely the Ultra Wide Band (UWB). Furthermore, another layer called CSS (Chirp Spread Spectrum) [192] has been introduced, which offers 14 channels in the 2450 MHz band providing data rates of 1MBit/s. Channel sharing is achieved by applying carrier sense multiple access (CSMA). Acknowledgments are provided for reliability.

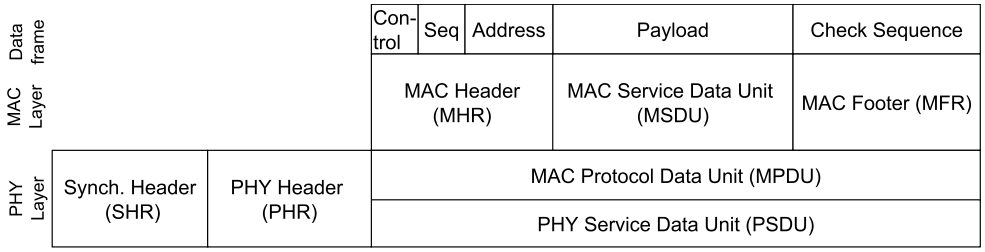


Figure 2.8: 802.15.4 frame

The packet format of 802.15.4 consists of three parts: a header, a data chunk, and a footer. The header contains all the necessary control data (for example addresses or sequence numbers) for transmitting a frame. The layout of an IEEE 802.15.4 frame is shown in Figure 2.8. It consists of control data, a sequence number, an address, some extra security headers, the actual data to be transmitted, and a frame check sequence. The control data specifies how a receiver should interpret the frame and if it is to be acknowledged or not. The IEEE 802.15.4 MAC layer defines four different frame types:

Data frames: Date frames are used for the transport of actual data. Most protocols building on the 802.15.4 MAC layer just use data frames. The 6LoWPAN implementation used in this thesis (MRv6, see Chapter 2.3.3), for example, packages all its information (including the superframe) into MAC layer data frames.

Acknowledgment frames: These are used to acknowledge the receipt of a frame.

Command frames: Command frames allow sending low-level commands from one node to another.

Beacon frames: Beacon frames can be used as coordinators to structure the communication with nodes.

The sequence number, following the control data, is used to match acknowledgments to frames. The address section contains the address of the sender of the frame and the address of the receiver.

2.3.2 6LoWPAN

In the following, a very brief introduction to the core ideas and concepts behind 6LoWPAN is given. The complete specifications of IPv6 and 6LoWPAN consist of several hundred pages; therefore, for a more thorough introduction, the specifications [221, 176] or specialized books like the one by Shelby [351] are recommended.

Notably, 6LoWPAN is considered one of the key enablers of the IoT [113], as it allows direct integration of motes into an IPv6 network [80]. 6LoWPAN is an acronym for *IPv6 over Low-power Wireless Personal Area Network*. Nonetheless, some authors

argue (in particular, Shelby and Bormann [351]) that the term ”personal” is still only present for historical reasons and they advocate 6LoWPAN as *IPv6 over low-power wireless area networks* only.

The main goal of the 6LoWPAN working group was to combine traditional TCP/IP technology and 802.15.4. The working group had to overcome several technical hurdles due to the very different nature of TCP/IP on the one hand and 802.15.4 on the other. A major technical difficulty was the different payload sizes. The maximum payload size in 802.15.4 is 127 bytes, from which 25 bytes are already being used by the MAC header. A regular IPv6 would need 40 bytes and the UDP header needs 8 bytes, thus leading to a remaining payload of not more than 54bytes (roughly 43%). In cases where encryption like AES-CCM-128 is used, the situation is even worse [143, 337]. AES-CCM-128 needs another 21 bytes, which leads to only 33 bytes (26%) being left for data [333]. This has been addressed by introducing header compression schemes [221, 176]. Typically, the IPv6 and UDP headers can be compressed to as little as 7 bytes; thus, the payload can be increased to up to 95 bytes. The general idea behind header compression is to remove all information from the header that either is superfluous or can be determined otherwise. For example, IPv6 contains a header field called Version, which always has the value 6. This field can be omitted. Some parts of the address information can be deduced from the MAC header. Meanwhile, for some configurable settings in the IPv6 protocol standard values are assumed.

Another major issue within the working group was the maximum MTU of 1280 bytes, which IPv6 requires. This lead to the need of fragmentation and defragmentation algorithms within the 6LoWPAN layer. As fragmentation and defragmentation are rather complex tasks, and due the lack of reliability and increased energy consumption [351], the general recommendation is to avoid it whenever possible.

In the beginning, the 6LoWPAN standardization was aimed at the IEEE 802.15.4 standard only and it assumed 802.15.4-specific features such as a beacon-enabled mode and association mechanisms. More recently, 6LoWPAN standardization work has been generalized to work with a larger range of link layers and to avoid the assumption of IEEE 802.15.4-specific features [351].

2.3.3 MRv6

In the following, we present MRv6 as currently implemented in the Mote Runner system. The information given in this section is a precondition for understanding the implementation of the sleepy node protocol in Chapter 7.5.1. MRv6 is a specific 6LoWPAN (see Chapter 2.3.2) implementation for the Mote Runner system supporting UDP over 6LoWPAN and header compression according to RFC4944 and RFC6282. Its implementation is based on TDMA (Time Division, Multiple Access) medium access and supports multi-hop network communication. This enables direct IPv6-based communication between Internet hosts and motes in the wireless network. MRv6 is implemented in C#, and it is treated like any other Mote Runner package. The code is freely available and can be modified.

The specification of 6LoWPAN defines a design space that allows different actual implementations. In the following, MRv6 is briefly described. For a more detailed explanation of MRv6 and its specialties please refer to the protocol documentation [180].

2.3.3.1 Network Management

The default network stack that is shipped with the Mote Runner system is called WLIP [181]. Any mote first checks the availability of a WLIP gateway within its range. If no WLIP gateway is found, then the MRv6 library will take ownership of the radio device. From that point on it will try to join an existing network. In the current implementation, the edge mote (the root node) is responsible for network management. It manages the tree, associations between motes, and addressing and routing. Furthermore, it implements the protocol to communicate with the gateway (tunnel).

2.3.3.2 Communication API

The API provided by the 6LoWPAN implementation abstracts the actual communication away from the user application. A user application just needs to use a socket like interface for accessing the network. The code fragment in Listing 2.1 shows the Socket API for receiving a 6LoWPAN packet.

Listing 2.1: Mote Runner OnPacket socket interface

```
public class MirrorDemo extends UDPSocket {
  internal static uint LOCAL_PORT = 4711;
  internal static MirrorDemo socket = new MirrorDemo();

   public MirrorDemo() {
     this.bind(LOCAL_PORT);
   }

   public int onPacket(Packet p) {
     uint len = packet.payloadLen;
     byte[] buf = (byte[]) Util.alloca((byte) len, Util.BYTE_ARRAY
         );
     Util.copyData(p.payloadBuf, pa.payloadOff, buf, 0, len);
     buf[0]=(byte) 'x';

      try {
        packet.swap(len);
      } catch {
        return 0;
      }
      Util.copyData(buf, 0, p.payloadBuf, p.payloadOff, len);
      this.send(packet);
      return 0;
    }
  }
```

```
}
```

2.3.3.3 Protocol details

As described, MRv6 implements a beaconing-based, multi-hop TDMA protocol. Communication slots between parent and children are globally assigned. The communication consists of a series of superframes. Figure 2.9 illustrates such a superframe. The communication slots between parent and children are globally assigned by the edge mote. The maximum number of nodes in the network is predefined but configurable.

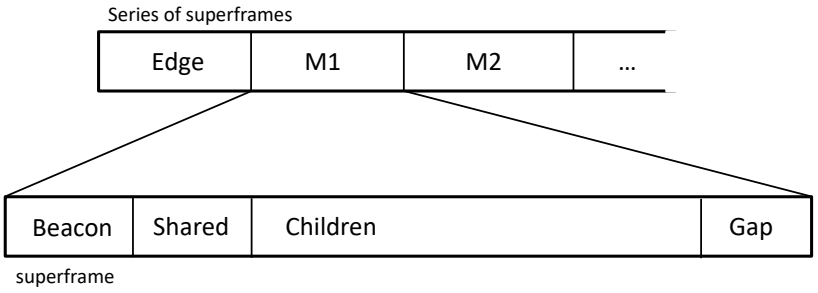


Figure 2.9: MRv6 superframe

At the beginning of each communication period, the parent motes use superframes to send out beacons that announce the state and schedule for the communication slots and to synchronize their clocks. Furthermore, each superframe supports a multiple-purpose shared slot. This slot can be used for association requests or responses and broadcast messages.

Length of a slot, the time for listen operations and the gaps between slots are constants and can be configure. The timings for a slot are specified by the (configurable) constants [180] such as:

- RECV_SAFETY_MILLIS: Number of milliseconds the radio is switched on before the slot is scheduled. This number is used to cope with clock-drift in a standard scenario, without sleeping nodes.
- SLOT_RECV_MILLIS: Number of milliseconds the radio is switched on in a slot.
- SLOT_GAP_MILLIS: Number of milliseconds after the radio is switched on and before the next slot.

Every mote, that is not the edge mote, is constantly switching between parent and child mode. Figure 2.10 shows the (simplified) state diagram of the MRv6 protocol. A mote that is not part of a communication structure constantly scans for beacons

(CHILD_SCAN_BEACON) and once received (CHILD_RECEIVE_BEACON), it attempts to join the network. Whenever a mote joins the network, it attaches to a single parent. Later on, those motes can have children of their own. If one or more possible parents are detected, the mote evaluates them to find the closest parent that has free slots, a sufficient good link quality, and signal strength. If a suitable parent mote has been identified and multiple beacons from that parent mote have been received, the mote attempts to join the network. The mote sends an association request (in the shared slot) (CHILD_TRANSMIT_ASSOC) and waits for an answer (CHILD_RECEIVE_ASSOC). If no answer is received within a specified period the association attempt is repeated. The same thing happens when a mote is not receiving beacons from its parent for some intervals. In such a case, it resets its state and starts looking for a new parent again.

After the initial association to a parent, the communication between parent and its children work as follows: The beacon contains information about whether the parent wishes to send a packet in the shared or in the respective child slot. If either is the case, the child switches to STATE_CHILD_HANDLE_SLOT and schedules a radio rx at an appropriate timestamp that is determined by adding a constant offset (depending on the slot type) to the last received beacon's one. If the parent does not wish to send a packet, then the mote's final action as a child is to check if there is a packet for its parent. If yes, then the mote needs to transmit it.

After the child role actions are completed, the mote switches to the parent role. The parent is initially in the state STATE_PARENT_SEND_BEACON. It transmits its beacon and then any data that are to be sent through the shared slot. Next, it proceeds with iterating over all slots of its children. Whenever there is data to be sent through a slot, it transmits it, otherwise it switches to STATE_PARENT_HANDLE_SLOT and listens for any data from the respective child. After the last slot, the motes' participation in the current superframe is over. It switches to the child role again and waits for the next beacon.

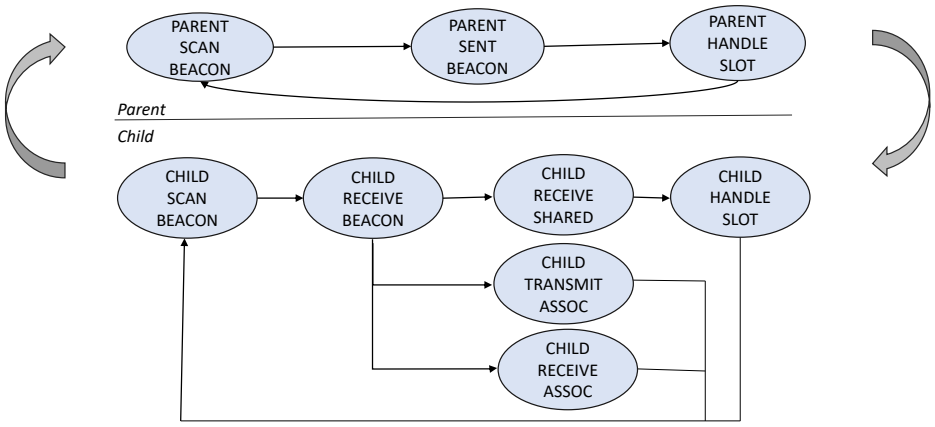


Figure 2.10: MRv6 States

Network manages is done by so-called ”System messages”. The MRv6 protocol [180] specifies the following system messages:

Ping: A ping is specified as follows [180]: A parent sends ”ping” packets to children that have been inactive for a configurable amount of time. If the child does not acknowledge the packet, it is removed from the internal list of children and, therefore, from future beacon announcements. A disassociation message is sent to the edge to notify it about the node loss. If the child is still active despite not having reacted to the ping, it notices its removal when inspecting the next beacon message, resets its state and reassociates.

Synchronization Messages: Synchronization messages are specified as follows [180]: The edge mote sends out periodic synchronization messages to all wireless nodes to synchronize their state with his own.

Information Messages: Information messages are specified as follows [180]: Nodes periodically send information messages to the edge delivering statistics about transmission rates, signal strengths, communication reliability and more

Disassociation Notifications: A disassociation notification is sent by a parent mote that has lost one of its children to the edge.

Communication between motes is done in the designated time slots. A parent that would like to send a packet to a child has to announce a pending transmit in its beacon. This will cause the child to switch its radio receiver on in the specified time slot. When the parent has no pending transmissions it will listen for messages from the child node instead. Whenever a mote receives a packet from its parent it checks if it is the final receiver of the packet. If this be the case, the package is forwarded to the receiving socket, as shown in Listing 2.1. If it is not the final destination of the package, it forwards the packet as required to one of its children.

Addressing works as follows [180]: The 802.15.4 header consists of the short addresses of the destination and source motes as allocated by the edge mote, the PAN ID as configured for the network, an incremental sequence number managed for the current parent and child communication slot and an 802.15.4 header byte labeling the packets as data packets expecting acknowledgments.

MRv6 supports only a subset of the 6lowpan specification: Only short or full IPv6 addresses are supported and used in the compressed header. Port compression is not supported.

2.4 Representational State Transfer

The term Representational State Transfer (REST) is closely connected with Roy Fielding, who first described the concept [128] as a generalization of the HTTP object model. It is an architecture-paradigm for distributed systems closely connected with

the development of HTTP and the World Wide Web. It abstracts the basic principles of the web architecture and its components. In practice, the terms REST and RESTful are used interchangeably, although REST usually refers to the architectural style and RESTful to a service or architecture that complies with this architectural style. The term RESTful is an adjective that was introduced later, the dissertation of Fielding does not use the term.

The following provides a brief introduction to the main concepts of a RESTful architecture. The characteristics of a RESTful architecture are given below:

1. Stateless
2. Client-server communication
3. Cacheable
4. Layered system
5. Uniform interface
   (a) Resources and representations
   (b) Operations for manipulation of resources through representations
   (c) Self-descriptive messages
   (d) Hypermedia as the engine of application state

and, optionally:

6. Code on demand (optional)

Communication in a client-server based REST-system is always stateless. This means that the client cannot expect any context to be valid. Instead, the client always has to deliver all the information that a server needs to fulfill the request. This is one of the properties that make REST interesting for small embedded devices as they are not required to use their scarce resources for state handling. Stateless communication is also a main differentiator between the REST-paradigm and enterprise solutions such as CORBA [357].

The concept of a REST resource is abstract. It is independent of its representation. A resource can have many representations, for example, in XML and JSON. Resources and representations are defined as:

> A *resource* is any information that can be named. This includes images, collections of other resources, or even real objects. A resource is a conceptual mapping to a set of entities, not the entity that corresponds to the mapping at any particular point in time. – Fielding [128].

> A *representation* consists of data, metadata describing the data, and, on occasion, metadata to describe the metadata (usually for verifying message integrity). Metadata is in the form of name-value pairs, where the name corresponds to a standard that defines the values structure and semantics. Response messages may include both representation metadata and resource metadata: information about the resource that is not specific to the supplied representation. – Fielding [128]

A resource is commonly uniquely identified by a URL/URI. A typical URI identifying a resource in a 6LoWPAN network might look as follows:

$$\underbrace{[\text{ffff:ffff:ffff:fc03:0005}]}_{\text{service}}\underbrace{/\text{temperature}}_{\text{resource}}$$

The operations, as a part of the uniform interface, are usually implemented using verbs, or to be more precise, HTTP-like verbs. HTTP 1.1 defines eight verbs. The five most important ones are:

Operation	Meaning
GET	Requests the representation of a resource
PUT	Creates a new resource or replaces an existing one
POST	Transmits data for processing
DELETE	Removes a resource
HEAD	Is identical to GET, except that the server does not return a body in the response, but only the meta-information contained in the HTTP headers. These should be identical to the information sent in response to a GET request

Get, Put, Head and Delete operations are supposed to be idempotent. This means operations can be repeated with the same data on the same resources and the result (effect) shall always the same. Making multiple requests has the same effect as making only a single request.

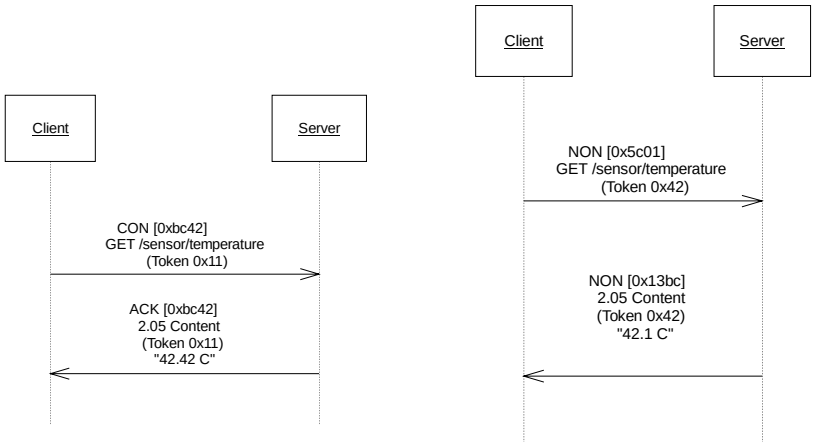
Finally, the uniform interface does not only consist of resources. It also needs to follow the *hypermedia as the engine of application state* (HATEOAS) principle. It means that the use of hyperlinks should be the only way of navigating an applications state machine. HATEOS is considered as one of the least understood constraints of the REST architecture [317].

Code on demand is the only optional feature defined by Fielding [128]. It allows functionality on the client to be extended by downloading and executing code in the form of scripts. It needs to be taken into consideration that this feature was defined when the web was largely static and JavaScript-based applications not common.

2.5 Constrained Application Protocol

The Constrained Application Protocol (CoAP) is a cross-layer protocol defined by the IETF Constraint RESTful environments (CoRE) working group. It defines a RESTful protocol specifically tailored towards resource-constrained devices. CoAP is specified in RFC 7252 [353]. CoAP was mainly developed with UDP-based networks in mind. Nonetheless, it is possible to use CoAP with other transport layer protocols such as TCP [53], or even SMS [36]. It has also been demonstrated that CoAP has significantly lower energy consumption compared to HTTP[93], as well as lower response time and less protocol overhead than HTTP [94].

Even though CoAP makes uses many HTTP-like functionalities, it was not designed to be a scaled-down implementation of it. Its main advantages are the low processing complexity and the small communication overhead. It utilizes a non-reliable transport protocol (UDP), which means that it does have to deal with congestion control, unlike HTTP with TCP. However, CoAP provides lightweight reliable transmission and de-duplication of messages. It achieves that by making use of a stop-and-wait retransmission mechanism and by detecting duplicate messages. These messages are detected based on their unique message id so duplicate ones are discarded. Messages are also divided in two categories, Confirmable (CON) and Non-Confirmable (NON). CON messages require an acknowledgment by the recipient whereas NON messages must not be acknowledged. This provides a more lightweight communication mechanism. Confirmable messages are resent after a timeout when no acknowledgment message was received. On CoAP level, there is no way to detect a lost unconfirmable message.



(a) Confirmable (b) Non confirmable

Figure 2.11: CoAP basic message exchange

Figure 2.11, depicts the two basic message exchanges between a client and a server. In Figure 2.11a, a client (on the left-hand side) requests the representation of a resource /sensor/temperature. It is a confirmable (CON) request. The server responds with an ACK message and a textual representation of the resource. This is called a piggybacked response. The ACK message mirrors the message ID (0xbc42) of the original CON message. A non-confirmable message (short NON) exchange is shown in 2.11b. The response to the original message is sent back as a NON message as well.

If the request cannot be answered immediately, it is possible to send an ACK first, with an empty payload, and later on send the response when it is ready. This is called a separate response. The message exchange is shown in Figure 2.12. The separate response in that case is sent by the server as a confirmable message and it is acknowledged by the client.

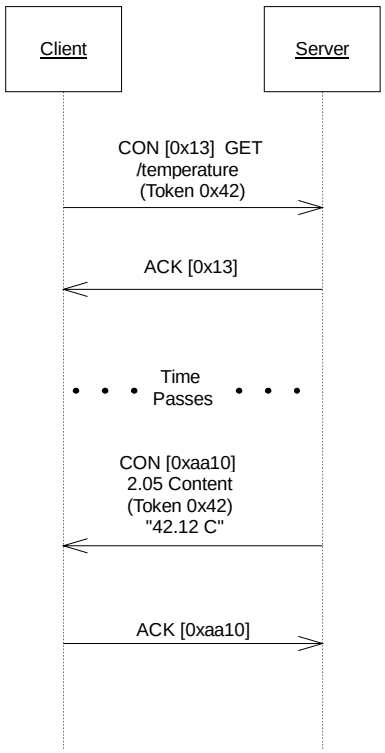


Figure 2.12: CoAP message exchange with delayed (separate) response

CoAP also implements mechanisms that allow the clients to consume services in a more complex way than just exchanging request-response pairs. Two of the most important mechanisms that we also experimented with are observe and block-wise transfer extensions.

Resources can be addressed and identified by following the CoAP URI scheme convention. When a node receives a CoAP message, it extracts its options from the URI path and acts on the request accordingly. These options provide a means of adding request-specific information to the CoAP message. The IETF Core Working Group aimed to minimize CoAP message overhead and fragmentation. The message comprises of a 4-byte header followed by variable-length token and options sections, while the rest is occupied by the optional payload. The CoAP message can fill up the UDP datagram data section, which is roughly 80 bytes in the case of 6LoWPAN.

The format of a CoAP message is shown in Figure 2.13. The mandatory CoAP header consists of the following components:

Ver: Protocol version. Currently only 0b01 is supported.

T: Message type. This can be either confirmable (0b00), non-confirmable (0b01), acknowledgment (0b10) or reset (0b11).

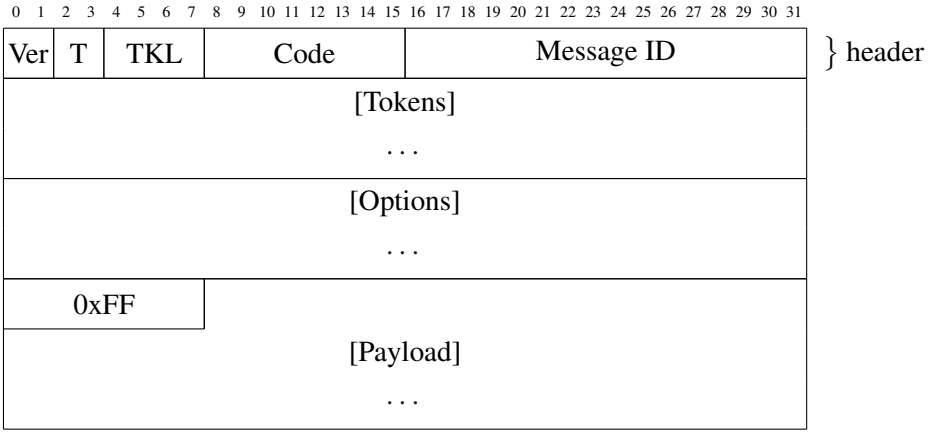


Figure 2.13: CoAP message format [353]

Code: The message code specifies the type or result codes of a given message. It is an 8-bit unsigned integer. The 3 MSB define the class and the 5 LSB detail the class. This leads to a class/detail scheme in the form of "c.dd" where "c" is a digit from 0 to 7 for the class and "dd" are two digits from 00 to 31 for the 5-bit detail. The class indicates whether the message represents a request (0), a success response (2), a client error response (4), or a server error response (5). A list of possible message codes is detailed out in Table 2.6.

Message ID: The message id identifies the message. It is unique per communication partner, within the message ID lifespan. The message ID is used for managing acknowledgment and for message deduplication (the process of detecting and removing duplicates).

Confirmable, non-confirmable and acknowledgment messages have been explained earlier. A reset message indicates that a message has been received, but cannot be processed properly. Most often, this is due to some context missing to process it. A reset message can also be used as a CoAP-level liveliness check (called "CoAP ping"). An empty confirmable message will be responded with a reset message. An empty message is a message that consist only of the 4-byte header. It has a Code of 0.00. It is neither a request nor a response.

Tokens can be used by applications to match a response with a request. The token length can be up to 8 bytes. Every request has a token, generated by the client, which the server echoes in any resulting response. A token is mainly intended for use as a client-local identifier in order to differentiate between concurrent requests or to map observe callbacks to corresponding requests.

Class	Detail	Meaning
0	00	Empty Message
	01	GET
	02	POST
	03	PUT
	04	DELETE
2	01	Created
	02	Deleted
	03	Valid
	04	Changed
	05	Content
3	xx	Reserved for further use
4	00	Bad Request
	01	Unautorized
	02	Bad Option
	03	Forbidden
	04	Not Found
	05	Method Not Allowed
	06	Not Acceptable
	12	Precondition failed
	13	Request Entity Too Large
	15	Unsupported Content-Format
5	00	Internal Server Error
	01	Not Implemented
	02	Bad Gateway
	03	Service Unavailable
	04	Gateway Timeout
	05	Proxying not supported

Table 2.6: CoAP message codes as specified by [353]

Options

In earlier CoAP drafts, options were just a list of pairs (optionNo, value). This has been changed to a delta encoding scheme that we will briefly discuss. Each option consists of an unsigned 16-bit option number and a value. The length of a value in bytes is called option length (OL). The options Op_0 to Op_{n-1} are calculated as follows:

$$\begin{aligned} \delta(Op_0) &= Op_0 \\ \delta(Op_i) &= Op_i - Op_{i-1} \;\; \forall i \geq 0 \end{aligned} \tag{2.1}$$

Storing options in a CoAP message is now as follows: Each entry starts with a 4-bit option delta, followed by a 4-bit option length. We call this the option header. As mentioned, the option number can be an arbitrary unsigned 16-bit integer value. As $\delta(Op_i)$ between two options OP_i and OP_{i-1} can be larger than $2^4 - 1$ the special bitsets 0d13 and 0d14 are reserved. If the 4-bit option delta is specified as 0d13 then another byte OPE_0 is added succeeding the option header. $\delta(Op_i)$ is then calculated as $13 + OPE_0$. In such cases, were $\delta(Op_i)$ is larger than 13+255 then the 4-bit option delta is set to 0d14 and two bytes OPE_0 and OPE_1 are following the 1-byte header. Now, $\delta(Op_i)$ is calculated as OPE_0 and OPE_1 interpreted as one integer in network byte order - 269. The use of 0d15 in the 1-byte header is prohibited and is considered a protocol error.

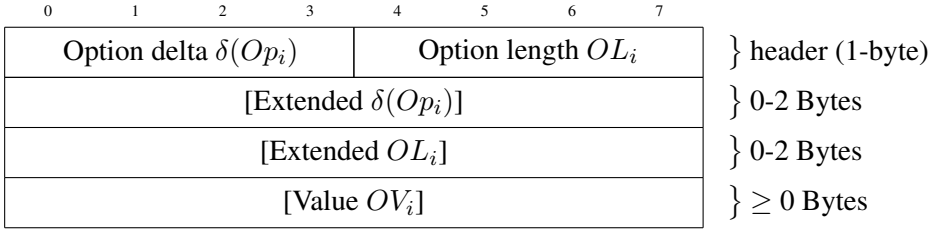


Figure 2.14: CoAP option format [353]

A list of currently defined options is presented in Table 2.7. The option range 0-255 is defined as being used after IETF review or IESG approval only. For further specifications the range 256-2047 is reserved, whereas 2048-64999 should only be used after expert review. The range 65000-65535 is revered for experimental use and should not be used operationally.

Number	Name	Number	Name
0	Reserved	15	URI-Query
1	If-Match	17	Accept
3	URI-host	20	Location-Query
4	ETAG	35	Proxy URI
5	If-None-Match	39	Proxy Scheme
7	URI-port	60	Size1
8	Location-path	128	Reserved
11	URI-Path	132	Reserved
12	Content-Format	136	Reserved
14	Max-Age	140	Reserved

Table 2.7: Currently defined CoAP options [353]

Observe Option

In a client-server environment such as CoAP, a client that wants to monitor changes in the state of a resource has no other option than to poll regularly for its state. This is rather inefficient and it causes unnecessary load in a constrained environment. Therefore, CoAP was extended to support the observer design pattern [133]. *Observers*, being interested in changes of specific resources, register at providers called *subjects* [162]. The subject subsequently notifies the observers whenever it changes its state. The protocol is based on registrations from the client and notifications from the server. A registration is initiated by the client. It sends a GET message to a resource with a special ”Observe” option. The server then adds the client to the list of observers. Now, whenever the state of the observed resource changes, the server sends a notification to the client.

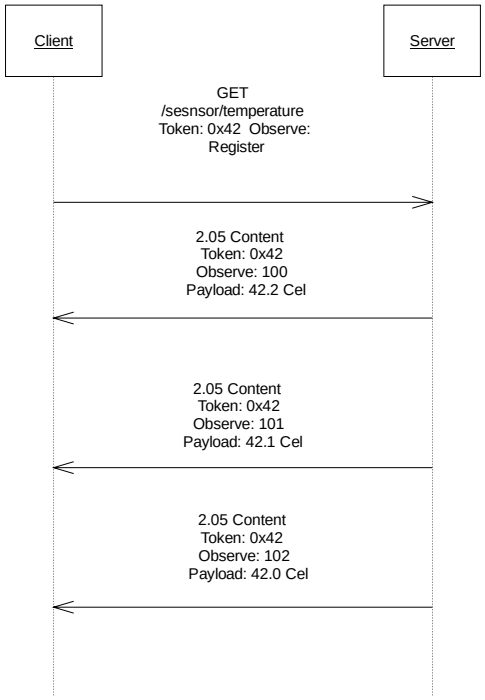


Figure 2.15: CoAP: Observe Option

The message flow of an observe in CoAP is shown in Figure 2.15. The client issues a request to observe the resource */sensor/temperature*. The token is set to 0x42. The observe option is set to 0, which means register. The server now adds this particular client to the list of observers and sends notifications whenever the resource changes its state. The server always duplicates the token from the original requests. The observe option, when sent from the server, is a number in ascending order. A client can use this number to identify and distinguish one observation from the other. A client can

No.	Name	Format	Length	Default
6	Observe	uint	0-3 Byte	none

Table 2.8: Observe option format [162]

deregister by issuing a new GET request with the same token as the token of the observation and an option value of 1. It can also reject a notification. Furthermore, if the transmission of a notification times out regularly, the client is also removed from the list of observers. The option itself is shown in Figure 2.8. The notifications can be confirmable or non-confirmable. The decision if a confirmable or non-confirmable message is sent is up to the server.

Block Option

The CoAP block extension [352] is used for fragmentation and defragmentation of large data blocks at application layer. As CoAP is based on datagram transports (e.g. UDP) the size of resource representation is limited by the maximum size of a UDP packet. Even if the maximum size is not used, small amounts of data cannot be transferred without creating IP fragmentation or adaptation layer fragmentation. Fragmentation at the adaptation layer or IP layer burdens the lower layers with conversation state that is, according to the CoRE working group, better managed in the application layer [352]. The CoAP block-wise transfer extension now enables an application-layer blockwise transfer of resource representations with minimal overhead and minimal communication state handling. The design goals of the CoAP block extensions are [352]:

1. Transfers larger than what can be accommodated in constrained- network link-layer packets can be performed in smaller blocks.

2. No hard-to-manage conversation state is created at the adaptation layer or IP layer for fragmentation.

3. The transfer of each block is acknowledged, enabling individual retransmission if required.

4. Both sides have a say in the block size that actually will be used.

5. The resulting exchanges are easy to understand using packet analyzer tools and, thus, quite accessible to debugging.

6. If needed, the Block options can also be used (without changes) to provide random access to power-of-two sized blocks within a resource representation.

The CoAP block option specifies two different block options, depending on whether it is used as part of the request (block1) or response (block2). The two options are shown

No.	Name	Format	Length	Default
23	Block2	uint	0-3 Byte	none
27	Block1	uint	0-3 Byte	none

Table 2.9: Block option format [352]

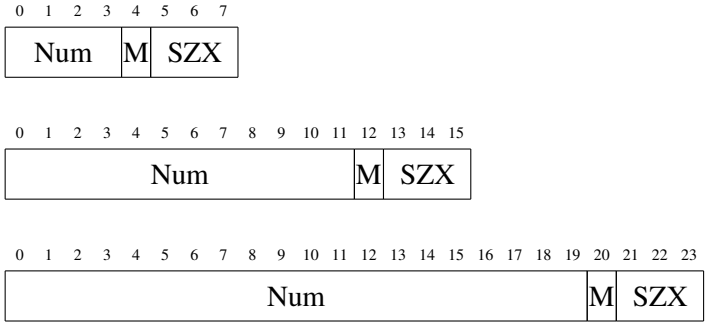


Figure 2.16: CoAP block option format [352]

in Table 2.9. Here, block1 is used by the client to request a resource representation in block-wise fashion; block2 is used to send data back as blocks.

The content (option value) of the two different option values is shown in Figure 2.16. A block option consists of three pieces of information [352]:

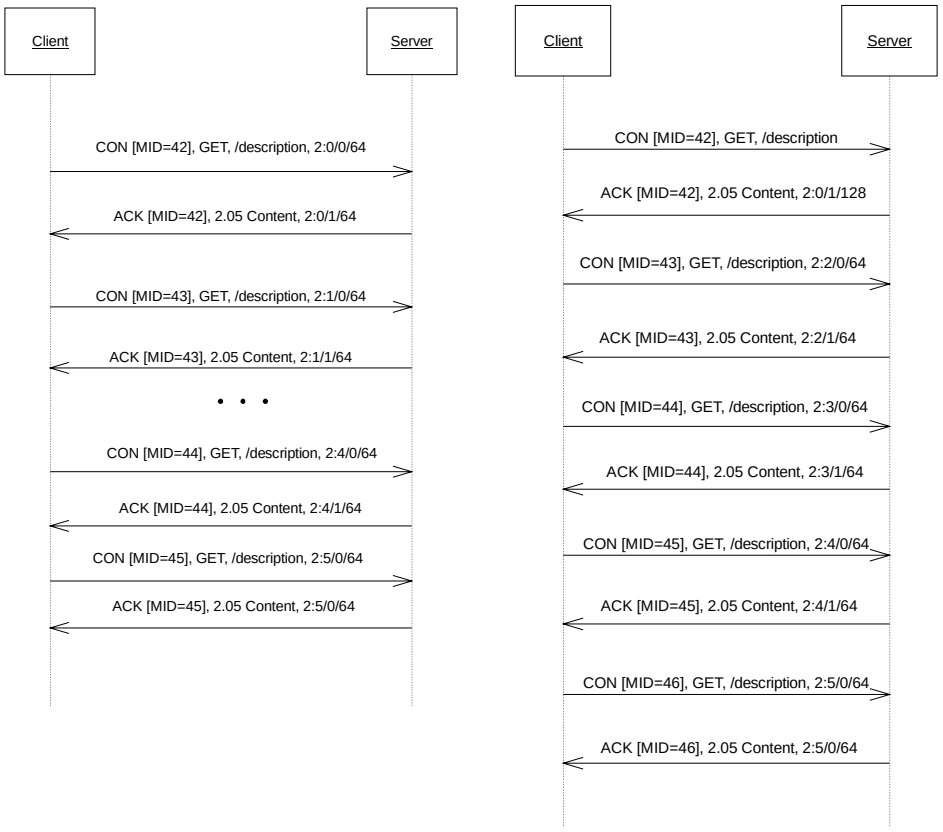
1. The block size (SZX). The block size is represented as three-bit uint. The block size in bytes is four plus the size of a block to the power of two. The block size in bytes is thus calculated as Size In Bytes = 2^{4+szx}. Therefore, possible block sizes are therefore 16,32,64...1024.

2. An indicator if more blocks are following (M). If the more flag if unset, the payload in this message is the last. No further blocks can be requested.

3. The relative number of the block (NUM) within a sequence of blocks with the given size. It specifies the block number being requested (block1) or provided (block). The first block is number 0.

The basic interaction paradigms are shown in Figure 2.17a and Figure 2.17b. The first case is called early negation. The second case is called late negotiation. In early negotiation, the client already knows or anticipates the need for a blockwise transfer. It sends a block size proposal (block 1). The server agrees with this block size and, from now on, all return messages carry 64 bytes of payload. The last return message may carry between 1 byte and 64 bytes.

In case of late negotiation the client does not anticipate any block-wise transfer and, therefore, does not send a blockwise request to the server. The server, however, requests a blockwise transfer and sends a proposal of a blocksize of 128 bytes. But the client cannot handle such big blocks and requests a blocksize of 64 bytes instead.



(a) Early message negotiation (b) Late message negotiation

Figure 2.17: CoAP basic message exchange

2.6 OData

In the following, we will briefly present the Open Data Protocol (OData) [274] protocol. OData is a recent data access protocol based on widely-used technologies (HTTP, AtomPub and JSON). OData, compared to the formerly predominant SOAP services, follows a REST-based approach, aims for semantic interoperability, and follows a more lightweight approach than traditional XML-based web services.

2.6.1 Overview

OData is a pure data access protocol on top of already existing application-layer protocols. It is based on widely-used technologies – namely HTTP, AtomPub and JSON. OData consists of the following four main parts:

- **OData Protocol:** OData specifies a protocol defining how clients can query and manipulate data sources. It supports CRUD operations and different serialization

formats, such as Atom Syndication Format and JSON. OData defines a query language as an extension of the URI. This query language provides a set of query options that allows clients to specify the data in which they are interested.

- **OData data model**: The structure of the data is defined by an abstract data model called Entity Data Model (EDM). It can be seen as a realization of the well-known entity relationship model, where data is modeled as entities and associations among those entities. An OData service provides a Service Metadata Document that defines the EDM-based model of the service in the XML-based Conceptual Schema Definition Language (CSDL).

- **OData service:** An OData service exposes a callable endpoint that allows accessing data or calling functions. It implements the OData protocol and uses the OData data model.

- **OData client:** An OData client accesses an OData service through the OData protocol and the known OData data model.

Chapter 2.6.3 will first describe the main concepts of OData services, as well as their relationships with resources and the OData data model. Thereafter, we will discuss possible data representations (Chapter 2.6.4).

2.6.2 Version History

OData is a relatively young development. The development started in 2007. A first OData version was released to the public in 2010. OData has been constantly and significantly enhanced since. The first release of OData was followed by OData version 2.0 [269] in 2011. The first version that gained widespread popularity was OData version 3.0 [274], which was released in 2012. OData subsequently was standardized by OASIS with the latest release – OData version 4.0 [287]– in 2014. OASIS submitted OData to ISO in order to establish an international standard. OData version 3.0 was the first that introduced comprehensive modeling capabilities that were further extended in OData version 4.0. More information on the differences between OData version 3 and OData version 2, respectively between OData version 4.0 and OData version 3.0, can be found in the official OData specifications [269, 274, 287] and technical documentation [368].

2.6.3 Services, Resources and Data Model

OData uses URIs to reference resources and to specify queries. An URI as used in OData can consist of three different parts: A service root URI, the resource path and a query. The service root URI identifies the root of an OData service. The resource path identifies the resource the service consumer wants to interact with (for example a specific temperature sensor, or some actor). Such a resource commonly addresses

Op	Description	Op	Description
Eq	Equal	Not	Logical Negation
Ne	Not equal	Add	Arithmetic Addition
Gt	Greater than	Sub	Arithmetic Subtraction
Ge	Greater than or equal	Mul	Arithmetic Multiplication
And	Logical and	Div	Arithmetic Division
Or	Logical or	Mod	Arithmetic Modulo

Table 2.10: OData operators (excerpt)

a collection of entities (e.g. several sensors), or a single entity (e.g. one specific temperature sensor).

A typical OData URI looks as follows:

$$\underbrace{\text{http://services.sap.com/service.svc}}_{\text{service root URI}}\underbrace{\text{/sensor/temp}}_{\text{resource path}}\underbrace{\text{?\$filter=temperature gt 20}}_{\text{query}}$$

The query can consist of one or more pre-defined options (called *system query options*), user-defined *custom query options*, or *service operation parameters.* Service operations are functions exposed by an OData service in a RESTful style. These operations might require zero or more parameters, which are passed as part of the query string. This work will mainly concentrate on the built-in system query options. We briefly introduce the most important system query options:

- *orderby* allows clients to request resources in a particular order. This is comparable to a SQL orderby clause.
- *top* allows to retrieve only the first n-results of a result set.
- *expand* allows clients to request related resources when a resource that satisfies a particular request is retrieved.
- *select* (projection) is used to select certain properties only.
- *filter* identifies a subset of the entries from the collection of entries identified by the resource path. The subset is determined by filtering out the Entries that satisfy the expression specified by the filter query option. Some of the operators are listed in Table 2.10 and Table 2.11. For a complete list please refer to [274].
- *format* is used to identify the data format requested by the client.

Discovering the capabilities of an OData service is possible through the *service document* and the *$metadata* information. The service document allows to discover the locations of the available collections of resources. It is returned when doing a get request on the service URI. This is a *must-have* feature according to the OData protocol

Function	Description
bool startswith(string p0, string p1)	Checks if string p0, starts with the string p1
int length(string p0)	Length of string
string trim(string p0)	Removes whitespaces at beginning and end
string toupper(string p0)	Transforms to upper case
string tolower(string p0)	Transforms to lower case
double round(double p0)	Arithmetic rounding
double floor(double p0)	next lowest integer value by rounding down

Table 2.11: OData functions (excerpt)

specification. In addition, every service should present information about the structure and organization of all the resources. This is done by appending a $metadata path segment to the path. The result is in Common Schema Definition Language (CSDL) [271] format.

2.6.4 Data Representation

The OData protocol was not designed to work with constrained devices. It was originally intended to work with more heavy-weight clients, like in communicating enterprise systems or communication between an enterprise system and a mobile phone. The OData standard defines two ways for data representation (sometimes called serialization), of which one is rather heavyweight: ATOM/AtomPub [282] and JSON [98]. The two formats are used to represent the result of a service call. They are also used to model the service itself using the common schema definition language as described earlier.

ATOM is an XML based format that defines XML elements and their meaning. It is defined in RFC 4287 [282] and its roots are in the structured description of entry based data feeds. Each feed can contain an arbitrary number of entries. Each entry holds some content. AtomPub (Atom Publishing Protocol) defines the notion of a service. A service consists of one or more collections. AtomPub also defines a set of RESTful interactions for accessing a service.

The relation between ATOM/AtomPub and EDM is shown in Figure 2.18. Each AtomPub service corresponds to an entity container in EDM. Collections and their feeds can be directly mapped onto entity sets. An Atom entry corresponds directly to an EDM entity. In the following we will refer to ATOM/ATOMPub simply as ATOM, as it always goes together.

JSON is a standard format originally designed for the serialization of JavaScript objects. It defines a text format for serializing (structured) data. All data is serialized as an unordered collection of name/value pairs. JSON, other than JSON-LD, does not

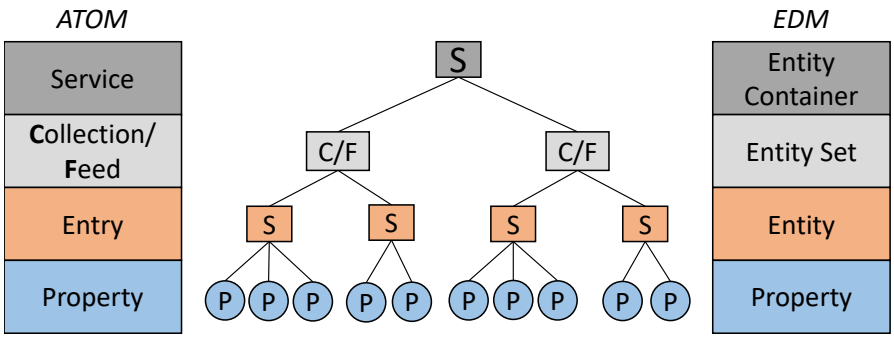


Figure 2.18: OData: Relationship between ATOM and the entity data model [82]

define any further semantics. OData extends the JSON format by defining conventions for the name-value pairs that annotate a JSON object, property, or array. For this purpose, OData defines a set of a set of canonical annotations for control information, such as IDs, types, and links, and custom annotations that may be used to add domain-specific information.

The OData protocol, its two representations, and the CDSL/EDM come with their own comprehensive specifications that go beyond what can be described here. More detailed information can be found in the OData specifications. OData v3 consists of seven standardization documents, each of hundreds of pages. Most important are the OData Version 3.0 Core Protocol [274], OData Version 3.0 Common Schema Definition Language (CSDL) [271], the Url Conventions [273] and the two data representation: Atom Format [270] and JSON Verbose Format [272].

2.7 Modelling of Vocabularies and Ontologies

This section will give a brief overview on how to model Linked Services with Semantic Web technologies to an extent that makes this work self-contained and comprehensible without deep prior understanding of semantic web technologies like RDF, OWL or SPARQL. An introduction to these technologies is provided by Segaran et al. [343], Allemang and Hendler [11] or Hitzler et al. [168]. The W3C standards[4] are also a good source of information.

2.7.1 Resource Description Framework

The Resource Description Framework (RDF) is a set of W3C specifications used for the conceptual description or modeling of information of any kind with the aim of data interchange. The main idea behind RDF is to express statements about resources. These statements are expressed as (subject, predicate, object) triples and constitute a

[4] http://www.w3c.org/RDF

directed graph. It is important to distinguish between RDF and its schema language RDF(s). RDF itself has only a very limited vocabulary [5]. RDF(s) is used to model RDF vocabularies, describing the elements of a domain. The relationship between RDF and RDF(s) is the same as between XML and XML-Schema. In the following tables, the origin of a concept is determined by its prefix, which is either rdf: or rdfs:. The main concepts of RDF/RDFS are described in Table 2.12.

Concept	Identifier	Description
Resource	rdfs:Resource	All things described by RDF are called resources, and are instances of the class rdfs:Resource. This is the class of everything. All other classes are subclasses of this class. rdfs:Resource is an instance of rdfs:Class.
Class	rdfs:Class	This is the class of resources that are RDF classes. rdfs:Class is an instance of rdfs:Class.
Literal	rdfs:Literal	The class rdfs:Literal is the class of literal values such as strings and integers. Property values such as textual strings are examples of RDF literals. Literals may be plain or typed. A typed literal is an instance of a datatype class. This specification does not define the class of plain literals. rdfs:Literal is an instance of rdfs:Class. rdfs:Literal is a subclass of rdfs:Resource.
Datatype	rdfs:Datatype	rdfs:Datatype is the class of datatypes. rdfs:Datatype is both an instance of and a subclass of rdfs:Class. Each instance of rdfs:Datatype is a subclass of rdfs:Literal.
Property	rdf:Property	rdf:Property is the class of RDF properties. rdf:Property is an instance of rdfs:Class.

Table 2.12: Main concepts of RDF/RDFS (Source: Brickley et al. [61])

The property concept of RDF is describing a relationship between two resources (subject and object resource). Properties can be further defined, for example, they can have a destination (range) and a source (domain). The main means of extending and further defining a RDFS:property is described in Table 2.13.

Concept	Identifier	Description
Range	rdfs:range	dfs:range is an instance of rdf:Property that is used to state that the values of a property are instances of one or more classes.
Domain	rdfs:domain	rdfs:domain is an instance of rdf:Property that is used to state that any resource that has a given property is an instance of one or more classes.

[5] also often called ontology. The terms ontology and vocabulary are often used interchangeable in RDF-related literature.

Type	rdf:type	rdf:type is an instance of rdf:Property that is used to state that a resource is an instance of a class.
Sub class	rdfs:subClassOf	The property rdfs:subClassOf is an instance of rdf:Property that is used to state that all the instances of one class are instances of another
Sub property	rdfs:subPropertyOf	The property rdfs:subPropertyOf is an instance of rdf:Property that is used to state that all resources related by one property are also related by another.
Label	rdfs:label	rdfs:label is an instance of rdf:Property that may be used to provide a human-readable version of a resource's name.
Comment	rdfs:Comment	rdfs:comment is an instance of rdf:Property that may be used to provide a human-readable description of a resource.

Table 2.13: Concepts further defining properties. (Source: Brickley et al. [61])

RDF predefines some properties, as shown in Table 2.14. Each property is applicable to a given domain (subject) and a range (object).

Name	Description	Domain	Range
rdf:type	The subject is an instance of a class	rdfs:Resource	rdfs:Class
rdfs:subClassOf	The subject is a subclass of a class.	rdfs:Class	rdfs:Class
rdfs:subPropertyOf	The subject is a subproperty of a property.	rdf:Property	rdf:Property
rdfs:domain	A domain of the subject property.	rdf:Property	rdfs:Class
rdfs:range	A range of the subject property.	rdf:Property	rdfs:Class
rdfs:label	A human-readable name for the subject.	rdfs:Resource	rdfs:Literal
rdfs:comment	A description of the subject resource.	rdfs:Resource	rdfs:Literal
rdfs:seeAlso	Further information about the subject resource.	rdfs:Resource	rdfs:Resource
rdfs:isDefinedBy	The definition of the subject resource.	rdfs:Resource	rdfs:Resource
rdf:value	Idiomatic property used for structured values	rdfs:Resource	rdfs:Resource
rdf:subject	The subject of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:predicate	The predicate of the subject RDF statement.	rdf:Statement	rdfs:Resource
rdf:object	The object of the subject RDF statement.	rdf:Statement	rdfs:Resource

Table 2.14: RDF properties (Source: Brickley et al. [61], Section 6.2 – shortend excerpt)

2.7.2 Turtle Notation

The Terse RDF Triple Language (short: Turtle) is a format developed by Dave Beckett and Tim Bernes-Lee [38] to express data in RDF. It is one of the many possible concrete syntaxes for RDF. Others include RDF/XML [37], JSON [102], and Notation 3 (N3) [43]. Turtle is a compatible subset of N3 and currently the most used notation language, recognized for its compactness and human readability. We will shortly introduce the

main concepts of Turtle by example. For a more thorough introduction to Turtle, please refer to Hitzler et al. [168] or the Turtle specification [38].

The following fragment shows the description of a class *service*. It has a label and is a subclass of some other class (gr:ProductOrService).

Listing 2.2: Modelling of classes with RDF

```
example:Service a rdfs:Class;
    rdfs:label "Service"@en ;
    rdfs:subClassOf gr:ProductOrService .
```

Properties are defined in the same manner: We introduce a property examples:includes which is a rdf:Propoerty. As mentioned before properties can be hierarchical as well. Example:includes is a subproperty of gr:includes, and it has a domain and a range. So, the includes property can be applied to any example:ServiceOffering and connect it to a example:Service. In other words, every *example:ServiceOffering* can *include* an *example:Service*.

Listing 2.3: Modelling of properties with RDF

```
example:includes a rdf:Property;
    rdfs:label "includes"@en ;
    rdfs:subPropertyOf gr:includes;
    rdfs:domain example:ServiceOffering;
    rdfs:range example:Service .
```

Concrete instances can be created with the same syntax. The following code fragment models a service MyService, a service offering MyServiceOffering and lets MyServiceOffering include MyService.

Listing 2.4: Concrete instance of a class modeled with RDF

```
:MyService a example:Service;
    rdfs:label "MyService is a concrete instance of a class"@en .

:MyServiceOffering a example:ServiceOffering;
    example:includes :MyService .
```

Within this thesis, we use Turtle notation for writing RDF.

2.7.3 Web Ontology Language

OWL, the Web Ontology Language, is a knowledge representation language for modeling ontologies and knowledge bases. Like RDF, its main concern is to define the terminology for a given domain. Currently, two versions are available. Work on the first OWL specification [257] started in 2002 and it was first published in 2004. The later OWL 2 specification [392] was published as a W3C recommendation in late 2009. OWL is based on description logic [20], which is a decidable subset of first-order

predicate logic. The OWL standard defines three different flavors or variants of OWL with increasing expressivity: OWL Lite, OWL DL and OWL full, with OWL Lite $\subset$ OWL DL $\subset$ OWL full $\subset$ First Order Logic.

OWL Lite is decidable and has complexity of ExpTime: The class of decision problems solvable by a *deterministic* turing machine in $O(2^{p(n)})$ time. OWL DL is also decidable and has a complexity of NExpTime. NExpTime are decision problems that can be solved by a *non-deterministic* Turing machine using $O(2^{p(n)})$ time. OWL Full includes all features of RDFS. It is undecidable.

Instances

Instances, often called individuals, are modeling elements in OWL. They can be seen as objects or instances of classes. They often represent real-world objects. An instance/individual can be part of a class (Chapter 2.7.3). The class membership of an OWL individual can either be direct (implicitly) defined or deduced through its properties (Chapter 2.7.4). In OWL, there are no unique identifiers for instances. Two different identifiers can refer to the same objects.

Classes

A class is a set of instances. It is used to express that members of a class share certain properties. Every instance is a member of *owl:Thing*. It is the most general class possible. The empty class, which does not contain any instances, is called *owl:Nothing*.

Concept	Identifier	Description
Intersection	owl:intersectionOf	intersection of two classes
Union	owl:unionOf	constructs the union of two given classes
Complement	owl:complementOf	complement of two classes
Enumerated class	owl:oneOf	one of the listed classes

Table 2.15: OWL class constructors. Source: [257]

Classes can also be constructed through class constructors. Some class constructors are listed in the Table 2.15. The union constructor, for example, takes two classes A and B, and constructs the class consisting of all instances of class A and all instances of class B. The following code fragment shows the use of a union. The meaning of the code fragment is explained in Chapter 5. Important at this point is the *domain* of the property *hasInteractionPoint*. It is defined as an owl:Class, and more specifically, as the union of services (usdl:Service) and service models (usdl:ServiceModel):

Listing 2.5: OWL class example

```
usdl:hasInteractionPoint a rdf:Property;
```

```
rdfs:domain [a owl:Class; owl:unionOf (usdl:Service usdl:
    ServiceModel)];
rdfs:range usdl:InteractionPoint .
```

2.7.4 Properties

Properties are used to express relationships between two objects (instances, classes, or datatypes). Properties, similar to classes, can be hierarchical. In OWL only binary relationships are possible, between a domain and a range, similar to the concept in RDF. OWL distinguishes two types of properties: (i) Datatype properties and (ii) Object properties. A datatype property relates instances to RDF literals or datatypes. OWL uses the owl:DatatypeProperty type for modeling this relationship. Object properties, formulated using owl:ObjectProperty, relate two instances of two classes.

Concept	Identifier
Min. cardinality	owl:minQualifiedCardinality
Max. cardinality	owl:maxQualifiedCardinality
Local reflexitivity	owl:hasSelf
Key	owl:hasKey
Symmetric	owl:SymetricProperty
Asymmetric	owl:AsymmetricProperty
Reflexive	owl:ReflexiveProperty
Irreflexive	owl:IrreflexiveProperty
Transitivity	owl:TransitiveProperty

Source: [257]

2.7.5 SPARQL

SPARQL (SPARQL Protocol and RDF Query Language) is a query language for RDF. It is able to retrieve and manipulate data stored in RDF format, for example, in a triple store. Since RDF can be interpreted as directed graphs, SPARQL works with graph patterns. The results of SPARQL queries can be either results sets or RDF graphs. SPARQL can be seen as the SQL for the semantic web.

2.8 Data Reduction

In our experiments, we included a compressed version of both XML and JSON. Previous research at the CDS working group [110] and similar research, e.g. by Marcelloni et al. [246] and Sadler[331], has shown that standard compression algorithms have a quite decent compression performance, also compared to specialized algorithms. In the following, we briefly summarize these results and then compare to the possible alternatives.

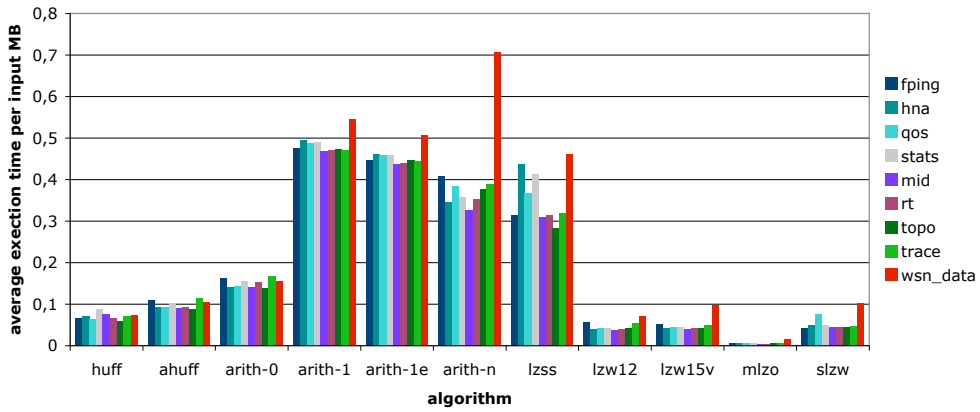


Figure 2.19: Execution timer per input MB for different algorithms [110]

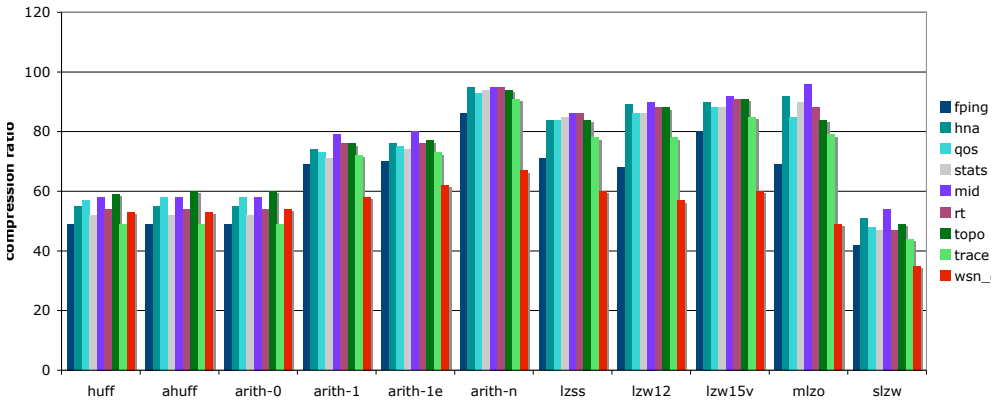


Figure 2.20: Compression ratio [110]

A comparison of different compression schemes was conducted by Dolfus and Braun at the CDS group. The comparison included two Huffman coding schemes: static [175] (huff) and adaptive [207] (ahuff) huffman coding. Four versions of arithmetic coding [398]: simple arithmetic coding (arith-0), arithmetic coding order 1 (arith-1), order 1 arithmetic coding (arith-1e) considering escape characters as well, Order n arithmetic coding (arith-n). Five versions of dictionary / LZ(W) based compressions schemes [278]: LZW with 12-bit symbols, LZW with variable-sized symbols up to 15-bit, S-LZW [331], LZ77 and MLZO [27].

The average execution time is shown in Figure 2.19. The compression time of the arithmetic coding was significantly higher than the times of all the others. The dictionary based algorithms, and standard huffman coding had the least compression times. The compression ratios are shown in Figure 2.19.

Similar research has been done by Sadler et al. [331]. They describe the aforementioned embedded version of LZW called S-LZW, which is similar to a standard LZW

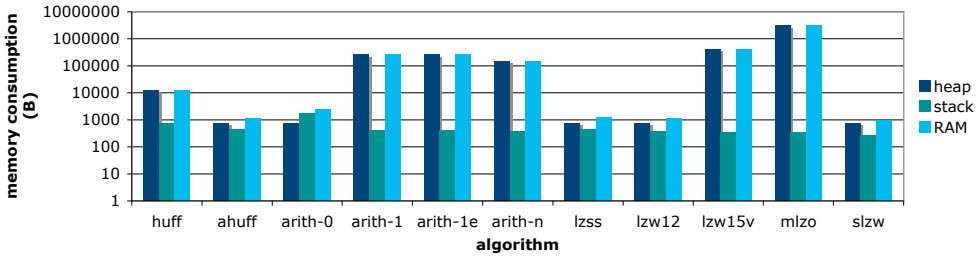


Figure 2.21: Memory consumption of different compression algorithms [110]

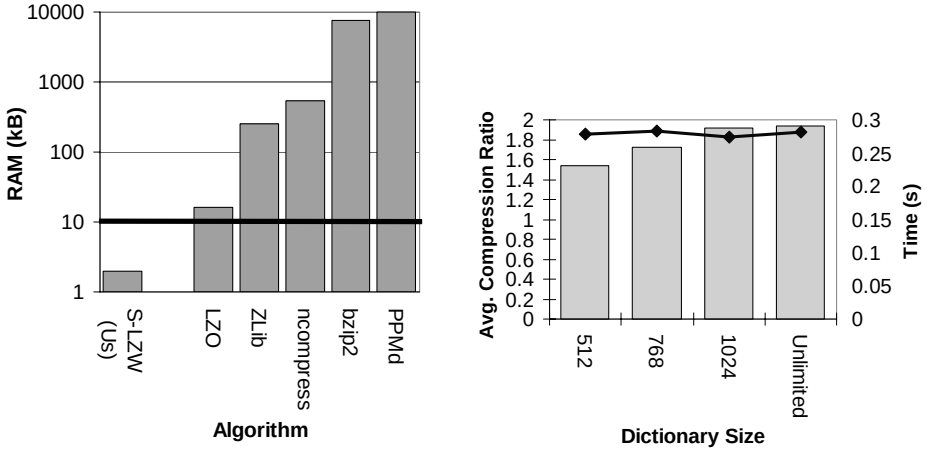


Figure 2.22: Compression ratio of S-LZW compared to standard algorithms and its average compression ratio and time based on example data [331]

implementation, but more optimized towards very constrained platforms. The hardware platform considered by them is similar to ours. They have slightly more RAM (10kb vs 8kb) available, but way less ROM. They have a total flash space of 48kb. The IRIS mote is equipped with 128kb of program flash and an additional 512KB of serial flash. Furthermore, the used TI MSP430x1611 is less advanced than the ATMEL chipset and runs at only 4MHz. They also showed that more sophisticated algorithms cannot run on such really constrained devices (Figure 2.22) for memory reasons.

Many alternatives to standard compression formats have been proposed and are discussed in literature. The specialized compression schemes try to leverage on specific known properties of the compressed data or format. For XML, several approaches exist. As XML itself has been designed as a human-readable text-based format one of the most obvious measure to decrease the size of XML-encoded data was to move to a binary format. In 2014, the W3C started the XML Binary characterization working group [92] to research the requirements to an future binary XML format. The successor

of the W3C XML Binary Characterization was the Efficient XML Interchange Working group. It started in late 2005 and released the first draft of the efficient XML interchange (EXI) format two years later in December 2007 [191]. Sakr [332] gives an overview of existing XML compression techniques.

For RDF, similar techniques as for XML can be applied. Fernandez et al. [124] present a binary RDF representation based on a header, a dictionary and the actual triples. The header describes the RDF dataset. The dictionary provides a catalog of the RDF terms (URIs, blank nodes, literals), while based on this dictionary the RDF triples are stored.

The results of Dolfus and Braun and the fact that more sophisticated algorithms will not run on the mote encouraged us to also use standard compression schemes for compressing the service descriptions and the data representation of the service call results. We opted for a general-purpose algorithm, as it is easy to implement with a small memory and computing footprint. Furthermore, such algorithms are suitable for both JSON and ATOM. So we didn't have to support two compression formats. In addition, the feasibility of EXI for such small platforms as our experimentation platform remains unclear. Reported memory consumptions for an embedded EXI implementation [78] are higher than for LZW-based formats. To our knowledge, only one implementation on an 8kb mote exists. Caputo et al. [68] describe such a platform for Contiki and 8kb platforms, nonetheless without the support of EXI schema encoding or any decoding. Furthermore, they give only one data point, no information on further compression restrictions nor about the energy or memory consumption.

Compared to Marcelloni [246] and Sadler [331] an optimized implementation was not in the focus of our research. We used a standard implementation that uses flash memory for paging circumventing the problem of limited RAM, and hashing for faster memory access. In cases where nothing, for example in a service description, or only parts of the compressed data changes the compressed file can be precomputed before uploading to the mote as the dictionary (LZW) can be precomputed and stored on the mote. The mote itself then just needs to resume compression starting from the point where it injects data, saving some running time and thus energy.

2.9 Sleepy Nodes

Sleepy Nodes are currently discussed in the CoRE community on an application-level as in Rahman et al. [311, 312, 313]. In [311], a sleepy node infrastructure based on a resource directory (RFC 6990) [350] is discussed. The focus is on how the sleep state can be tracked centrally. The nodes themselves announce their sleep state to the resource directory. The way in which these nodes determine when and how to enter the sleep state remains open. In contrast, our work does not prescribe any resource directory, but introduces an application-layer protocol for setting nodes to sleep. We then present and analyze the usage of a system utilizing such sleepy nodes. The techniques outlined in [311] can be easily integrated in our system, although we aim to

use semantic technologies to describe the sensor network, as well as its services and resources.

Allocating tasks to computing nodes is a lively research field. The main stream is centralized approaches versus self-organizing. Kuorilehto et al.[220] provide a comprehensive overview of both. The windowing algorithm that we use later on to evaluate the sleepy nodes implementation is part of the centralized family.

Another research stream focuses on minimizing energy consumption while maintaining partial [112] or full coverage [164, 383] of a specific area, often with a given degree of redundancy. These approaches differ from ours as we are not just monitoring an area. Moreover, they raise an alarm when specific conditions are met but run a business logic that is often only interested in the state of a given monitored entity at a given point in time. Similar ideas as in our approach have been implemented in the dynamic sleep scheduling protocol (DSSP) [66]. DSSP focuses on dense networks and maintaining coverage. In our approach, we assume that more information about upcoming events is available as part of a centralized business logic or enterprise resource planning (ERP) system and, therefore, that it outweighs the benefits of a non-centralized approach.

2.10 Services

Service-oriented systems are the glue that keeps nowadays enterprise systems and the Internet running. We will have a closer look into services in general and services in the Internet of Things in particular in Chapter 3 and Chapter 4. When talking about the ”Enterprise”-world and the ”Internet”-world, interestingly, both worlds are getting closer and closer. In practice, three major classes of services can be distinguished: legacy enterprise services, SOAP-based services and RESTful services. Traditionally, the enterprise domain had its own service architectures based on technologies such as BAPI and CORBA. They are still in use today, but are considered a legacy. Later on, XML-based web service descriptions and protocols became predominant in the enterprise sector. Meanwhile, on the Internet, the REST paradigm is becoming increasingly popular. More recently, there seems to be a convergence towards REST services even in the enterprise. REST services started to supersede SOAP-based services and traditional BAPI/CORBA installations. A comparison between SOAP-services and RESTful-services has been performed by Pautasso et al. [295]. In the following, we will briefly discuss the service categories.

2.10.1 Legacy Enterprise Services

With the widespread adoption of enterprise IT-systems, especially with the advent of Enterprise Resource Planning (ERP) systems, the integration and interoperability between systems became one of the major issues for enterprise IT vendors. A separation of concerns by using something like services was considered early on as one of the

most promising ways of enabling interoperability, even when it might not have been called services at the time. The term ”service-oriented architecture” first came up in 1996, but the concept itself is much older. It can be traced back to the early ideas of structural programming and separation of concerns [294, 293]. Preceding service-oriented architecture was Enterprise Application Integration (EAI). EAI and service-oriented architectures share the same goals, but EAI did not lay stress on the term ”service”; it was more focused on brokers and an enterprise bus that connected different systems with adapters. Service-oriented architecture tried to eliminate the need for those adapters with the goal of standardizing the data exchange and message interfaces between different systems, without the need of specialized adapters.

SAP ERP systems introduced the concept of BAPI (Business Application Programming Interface) [336] in the mid-1990s. They were defined to represent the boundaries of (business-related) components. Nowadays, they would be called a service, instead of a programming interface. BAPIs are used to decouple components and define a common way to access them at an application level. They do not specify the means of transportation for the message themselves. The message transport was delegated to other protocols, commonly RFC (SAP Remote Function Call) or later via CORBA. RFC is a SAP-proprietary protocol for communication between distributed systems that was originally based on Common Programming Interface for Communications (CPI-C). RFC was later on extended to work also with, for example, TCP/IP.

With the rise of SOAP services all the enterprise vendors incorporated web service interfaces. In more recent times, the industry is moving towards REST-based services.

2.10.2 SOAP-based Services

SOAP, originally named Simple Object Access Protocol is an application layer protocol. It does not specify the transport of data. Most often it is used on top of HTTP and HTTP(s). The purpose of SOAP is to exchange data between a service consumer and a service provider over networks. SOAP specifies the encapsulation and encoding of XML data, and it is defining the rules for transmitting and receiving that data.

In the following, we will briefly present the structure and content of a SOAP message. It needs to be in XML. It needs to contain a SOAP-envelope and a SOAP-body. It can have a SOAP-header and a SOAP-Attachment as well. The general structure of a SOAP message is visualized in Figure 2.23.

A SOAP-envelope identifies the message as a SOAP message and describes its basic properties – for example the SOAP version. The SOAP headers contain information that is independent from the transported data or that is application-specific. A typical example of SOAP header information is a unique message ID. Every SOAP-message needs to have one SOAP-body. The body represents the actual information to be transmitted. In case of a remote function call, this could be, for example, the method name and its parameters. Finally, a SOAP-attachment can be added to a SOAP message. There are no restrictions on the content or representation of the SOAP attachment.

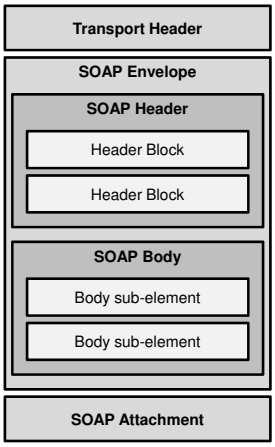


Figure 2.23: SOAP message

The SOAP specification [147] is rather large. An overview is given, for example, by Mitra et al. [275] or Weerawarana et al. [396].

2.10.3 REST WEB APIs

REST-based services and RESTful services are dominating the Internet nowadays. According to the ProgrammableWeb index [307] four times more publicly available APIs are using REST than SOAP. The degree of *RESTfulness* varies though. For example, quite some services ignore the hypermedia aspect of the RESTful paradigm. This caused a controversy in the community, leading to proposals such as introducing a maturity model for RESTful APIs; or of calling those services that do not 100% comply with the RESTful-paradigm as just Web APIs or REST-like APIs. The maturity model of Leonard Richardson [318] consists of four maturity levels:

0.
   - Uses XML-based RPC or even SOAP
   - HTTP, often only one verb
   - Uses only one address to access the service
1.
   - Uses different URIs
   - Uses different Resources
   - HTTP, often only one verb
2.
   - Uses verbs according to their HTTP meaning
   - Uses HTTP result codes
   - Uses different Resources
   - Uses different URIs
3.
   - Uses HATEOS for navigation
   - Uses verbs according to their HTTP meaning
   - Uses HTTP result codes
   - Uses different Resources

- Uses different URIs

Strictly speaking, the term RESTful can only be applied to Level 3. Many existing Web APIs that call themselves RESTful are on Level 2 or Level 3, or somewhere in-between. Level 0 and Level 1 were mainly important in the beginning of REST era where many RESTful APIs were just RPC using (one) HTTP verb.

2.11 Service Descriptions

In the following, we will discuss related approaches for semantically describing and annotating services. As Chapter 3 will discuss in more detail, "service" is a heavily overloaded term. For the following discussion it is important to know that our definition of service is more throughout and does not only include what we call the technical interface. A service is not only what is (technically) accessed and how, but also who interacts with whom from a business point of view, what Quality of Service attributes are attached to it, and so on. Most of the approaches discussed here are only concerned with the technical aspect of accessing a service and describing its inputs and outputs; they are less concerned with the underlying high-level concepts.

2.11.1 Semantic Annotations for WSDL and XML Schema

The Semantic Annotations for WSDL and XML Schema (SAWSDL) [212] are a W3C recommendation [120] for semantical annotation of XML-schemas in general and WSDL 2.0 in particular. WSDL describes a Web service on a purely syntactic level: WSDL species the messages format and what operations can be performed. It does not specify any meanings of the messages or what the operations mean.

SAWSDL introduces several attributes and seeks to extend WSDL. Those extensions are used to annotate interfaces, operations, and parameters. SAWSDL is very generic in its use. It defines three attributes: modelReference, liftingSchemaMapping, and loweringSchemaMapping [212]. They can be seen as reference annotations and transformation annotations [391].

The modelReference attribute may, by specification, be used with every element in WSDL. Nonetheless, it specifies only its use as the semantic annotation of interfaces (wsdl:interface), operations (wsdl:operations), and faults (wsdl:fault). As well as of the more general elements: xs:element, xs:complexType, xs:simpleType and xs:attribute.

The modelReference references the semantic concept behind those elements. It has the very generic purpose of referring from a WSDL element to a semantic concept. SAWDSL does not use or recommend any ontology of its own. While the modelReference identifies the concept to which the XML data can be lifted, the liftingSchema defines how the lifting can be done. The loweringSchema, on the contrary, can be used to lower the data again from an ontological level to its XML representation.

SAWSDL is one of the early works in semantic modeling of services and it has influenced all future approaches. Compared to Linked USDL, it does not describe

services in particular, but mainly technical interfaces in terms of ”how to access the service”. It does not provide a basic vocabulary for services and it is solely based upon WSDL. SAWSDL follows a conservative approach: It is not the service describing the technical interface, but the technical interface describing the service. The modelReference attribute was probably the most influential. It has been reused several times – for example, for iService [298], a linked services publishing platform. It definitely influenced Linked USDL with the idea of referencing from one element to the other. OWL-S [249], a more advanced approach that we will discuss in Chapter 2.11.2, supports a modelReference based grounding [252]. ModelReference strongly influenced OWL-S and all its grounding schemes, which in the end, are not more than specializations of the modelReference concept. ReLL [9] knows a reference tag and SA-REST [355, 229] is largely based on the SAWDSL idea.

2.11.2 OWL-S

Web Ontology Language for Web Services (OWL-S) is an OWL based ontology for describing web-services. OWL-S can be divided into three main parts: a service profile, a service model and a service grounding. We will briefly discuss these three parts.

Service Profile: The service profile is mainly used to allow service advertising and discovery. The service profile contains a human-readable description of the service. Furthermore, it can specify functional aspects of a service as input/output/precondition/effect (IOPE) specification.

Service Model: The service model describes a service as a process model. It describes a service operation in terms of what a service consumer can do and when. Furthermore, it can describe the composition of one or more services.

Service Grounding: The service grounding describes the necessary means to call a service – for example, regarding the message format and its protocols. The focus on the service grounding is on describing WSDL services.

OWL-S is comparable in terms of its goals with Linked USDL. Nonetheless, it lacks the business part of Linked USDL and also has not the focus of linking ontologies and lightweight REST-based services as we will use throughout this work. Rather, it is concerned with heavyweight XML/WSDL-based services.

2.11.3 Web Service Modeling Ontology

The Web Service Modeling Ontology (WSMO) is a framework for the semantical description of web services. It is based upon the Web Service Modeling Framework. WSMO, as a framework, consists of the following four elements [111]: ontologies that define the formal domain knowledge; Web Service descriptions that define the objectives of the service consumer; goals that define the objective of the service consumer; and

mediators that enable the interoperability and handling of heterogeneity between data structures and processes.

2.11.4 Web Application Description Language and the Semantic Bridge for Web Services

The the Web Application Description Language [154], commonly known as WADL, was developed by Sun Microsystems as a RESTful counterpart of the SOAP-based WSDL. The goal of WADL is to technically describe a ReST-service in the same way as WSDL describes a web service. WADL models the resources provided by a service and the relations between them. In WADL each resource is described as a request based on an HTTP method, its required inputs and the provided responses. WADL only specifies the technical interface and the data encoding, not its meaning. It was not designed for semantic interoperability, and it does not provide placeholders for semantic references of the operations, parameters, and results. The Semantic Bridge for Web Services (SBWS) [32] tries to fill this gap and allows adding semantics to the input and output of a REST method similarly to SAWDSL with modelReference-like properties. It wraps service operations described by a WADL document to create a SPARQL endpoint describing those services.

WADL differs from our approach first because it is not a semantic approach. It mereley describes a technical interface. Furthermore, it is bound to HTTP while we are aiming at any RESTful technology, including the Constrained Application Protocol as introduced in detail in Chapter 6. WADL is typically perceived as rather complex and it has not gained widespread use.

2.11.5 Semantic Annotations for REST

Semantic Annotations for REST (SA-REST) [355, 229] is based on SAWDSL. Nonetheless, its approach in the use of annotations is different. Most of the semantic annotations are used to annotate an already existing technical description of a service, for example WSDL and – on-top of that – to add semantic information. SA-REST annotations are added to the services described in (human-readable) HTML pages. SA-REST uses RDFa [3] to make this information machine-readable.

2.11.6 HTML for RESTful Services

HTML for RESTful Services (hRESTS) [211] follows a similar approach as SA-REST. Instead of using RDFa, however, it uses microformats [49]. The purpose of the hRESTS is to provide a machine-readable representation of common web services and API descriptions embedded into human-readable HTML. Microformats, in general, take advantage of existing (X)HTML facilities (e.g. the class and rel properties) to annotate the interesting parts of a human-readable API description. hRESTS defines seven

classes which can be used to annotate a web page. Namely: service, operation, address, method, input, output and label.

2.11.7 MicroWSMO

MicroWSMO [210, 209, 130] is a SAWDSL-like extension used to semantically annotate RESTful services based upon hRESTS. MicroWSMO supports model (same as modelReference), lowering and lifting. Those can be attached to any hRESTSs service property to specify the mapping between semantic data and its technological realization.

2.11.8 Resource Linking Language

Resource Linking Language (RELL) [7, 9] introduces a metamodel for REST services and a service description based on a meta model. The meta model formally defines a service as a set of resources. Every resource is associated with different properties, such as a unique identifier, a name and a URI pattern. ReLL descriptions are XML documents created according to the ReLL schema. ReLL is not specifically designed for the Internet of Things, however; it is more focused on XML and the use of XML-related technologies, such as XPATH and XSLT. Based on these technologies the authors presented a crawler [8] that can access the resources described in ReLL and transform them into RDF.

ReLL shares some similarities with Linked USDL and Linked USDL for IoT, mainly its lightweight design principles and our endpoint descriptions, as will be shown in Chapter 5, in the sense that we also build upon URI patterns.

2.11.9 RESTdesc

A recent approach to semantically describe RESTful service is RESTdesc [389, 388]. It is an RDF based notation of a service, written in N3. Thus, it is similar to Linked USDL and OWL-S. RESTdesc, other than Linked USDL, is closer to the actual RESTful interface. It was designed to perform reasoning on the interface. It models technical service interfaces not in terms of resources and operations, but as functional descriptions. The functionality is expressed as preconditions that when fulfilled will yield some postconditions. A RESTdesc service consumer can define a certain goal and RESTdesc in combination with a reasoner can deduce the series of REST operations that will result in the specified goal. RESTdesc has been applied in the Internet of Things domain [217].

2.11.10 Semantic RESTful Data Services

Semantic RESTful Data Services (SEREDASj), introduced by Lanthaler and Gtel [227, 225, 224], is a service description language for describing the syntactical and

semantical aspects of RESTful services. SEREDASj was specifically designed with JSON as data representation. A SEREDASj service description consists of two sections: a metadata section and an element section. The metadata section describes prefixes and relationships between resources. The element section describes the structure of the data representation. Both sections can contain references to other ontologies to semantically describe the elements.

2.12 Integration: Internet of Things and Web of Things

In the following we will discuss three different streams of integration of IoT or WSN services into enterprise systems or into the web at general. The first stream is XML/SOAP-based integration techniques. The second one, which we will summarize as Web of Things Integration, contains RESTful approaches. At a higher abstraction level we will also briefly discuss integration based upon business process modeling.

2.12.1 Web-Service based Enterprise Integration of WSNs

An approach that shares similar goals as our research has been performed by Glombitza et al. [140, 139]. This was based on the observation that most enterprise IT systems at that time used SOAP-based webservices as the main means to implement service-oriented architectures. To allow a seamless integration of sensor networks into the Enterprise world, Glombitza et al. investigated how webservices can be made available on single sensor nodes. However, the solutions that have already were available could not be directly downscaled to wireless sensor nodes due to resource constraints. This problem was solved by suggesting a novel transport protocol [141] as well as a new SOAP-transport-binding [140]. To support seamless integration and to enable communication between systems that do not both are able to communicate via this specialized protocol and binding an automatic webservice binding conversion [139] has been developed. Glombitza et al. also did some work business process integration, which we will discuss in Chapter 2.12.3

A second approach was put forward by Spiess et al. [363]. They suggested an architecture for integration Internet of Things devices into enterprise systems. Similarly to Glombitza et al. they suggest to use SOAP and, furthermore, the full stack of WS-* protocols. The focus of their work was more on device and service management as a whole and not that much on the actual integration of the devices. They assume that IoT-devices are either capable of running a SOAP protocol or not. In the latter case, they suggest the use of a middleware that translates SOAP messages to whatever protocol the device supports and vice versa.

We share the goals of Glombitza and Spiess. We can leverage on their ideas, but we are taking a different approach to achieve the integration. We see and foresee a trend towards RESTful systems, also in the enterprise world. The current generation, and even more the next generation, of business modeling tools and enterprise tools

in general will support RESTful systems. This new trend is best illustrated with the standardization efforts of the RESTful OData protocol that is being driven by all the main enterprise vendors.

A second main differentiator is that the web-service based approaches did not had semantic descriptionability in the focus of their work. Both are based on traditional WSDL-based webservices. Both works could, of course, be appended by using one of the WSDL/XML based techniques discussed in Chapter 2.11.

2.12.2 Web of Things Integration

Similar to the Web-Service based approaches discussed in Chapter 2.12.1 there has been work done on (enterprise) integration based on RESTful architectures. Commonly, those are referred to as the Web of Things [150], because the main goal of this stream of research was to make things accessible through the web. The web of things is advocated as a refinement of the Internet of Things by integrating smart things not only into the Internet (the network), but into the Web (the application layer) [150, 151, 153, 148]. The Web of Things advocates the use of RESTful web services. Regardless of the kind of device, to be a member of the Web of Things it has to offer its API over a RESTful web service. Therefore, the web of things can be defined as [330]:

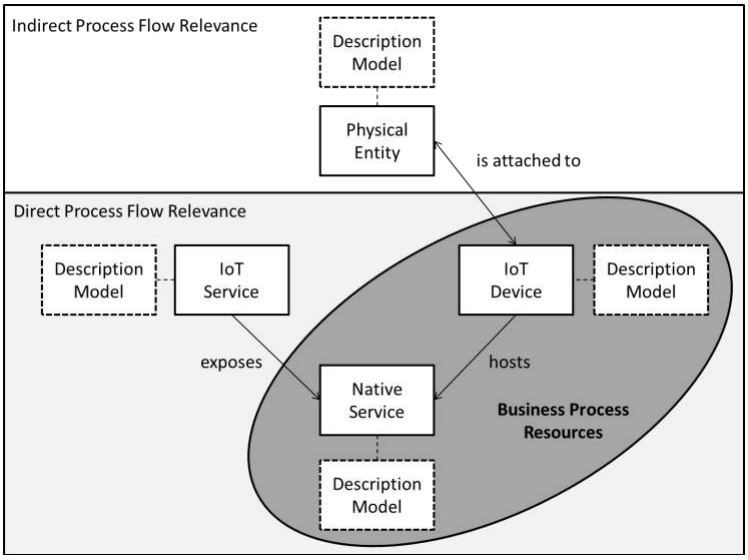
> The WoT enhances physical objects with a virtual counterpart representing the latter in the virtual world. In contrast with the IoT, the WoT mandates the strict application of RESTful principles to its APIs.
> – Ruppen, 2015 [330]

From a technical point of view we are considering constrained devices way more than the WoT movement does. The latter is largely concerned with being able to use the full ”web-stack”. We share with them the vision of REST-based access to IoT devices.

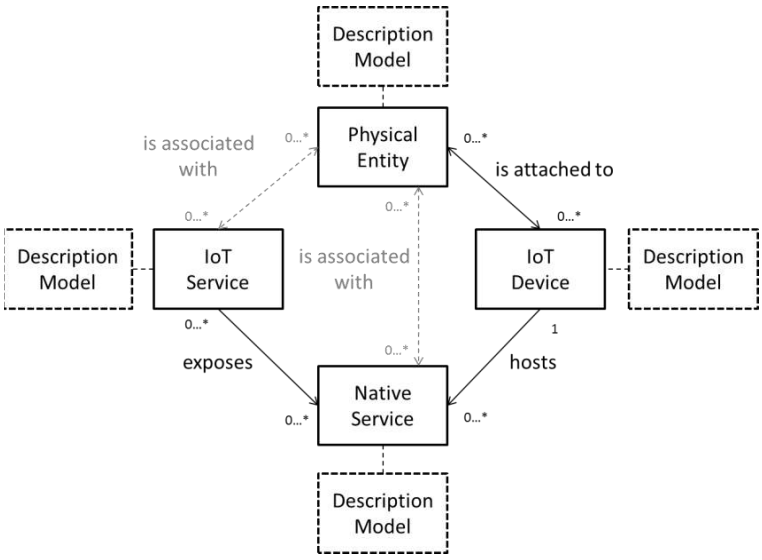
2.12.3 Process-based Integration

Integration into workflow and business process management is important when talking about enterprise integration from a business point of view. Graphical modeling tools such as BPEL [285] and BPMN [397] allow business specialists, who are not reassuringly software developers, to create workflows that can run to fulfill a business goal. Our work is focused on the service aspects of enterprise integration and leveraging on semantically enriched services. It has to be integrated with a business process approach as presented by Meyer et al. [264]. The IoT-BPMN integration presented by Meyer et al. was developed in the same research project as the work presented here and they can complement each other. We will first discuss the approach of Meyer et al. and also discuss related work from Glombitza et al. [138] based upon web-services, as well as Caracas et al. [71]. The three works all try to solve the same problem, but with very different means. Meyer et al. introduce IoT modeling concepts and base their

integration upon service descriptions, Glombitza et al. use the traditional WS-* stack and Caracas et al. do not alter the BPMN model at all, but generate code for smart devices out of it.



(a) BPMN Process-flow and IoT services [267]



(b) Description based integration into BPMN processes [267]

Figure 2.24: Description-based integration of IoT-devices into Business Process Modeling [267]

Meyer et al. [266, 267] take a service description oriented view on BPMN integra-

tion. As shown in Figure 2.24, they identified services as one of the main components that can be used as process resources and, thus, modeled by the business specialist. They propose, to make BPMN IoT-aware to not only use the facilities already included in BPMN, but a specification of the IoT device and its native software components as an extension to the BPMN 2.0 notation [267].

Glombitza et al. developed, a workflow management system [138, 142] based on their work on web services. The system allows transparent execution, monitoring and optimization of business processes. It aims for being used directly by the business expert. In [138], they propose a language called Graphical Workflow Execution Language for Sensor Networks (GWELS). Its objective is the creation of a link between the world of WSNs and enterprise IT systems. GWELS issues a series of service calls, based upon a workflow graph. Furthermore, they introduced a state-machine approach and domain-specific language for model driven development [142]. The goal of the state machine approach was, similar to the GWELS approach, to model application logic graphically.

A different approach has been suggested by Caracacs et al. [71]. They argue that neither the syntax nor the semantics of BPMN needs to be changed. WSN interactions are modeled solely using BPMN events and BPMN sequence flows. They use BPMN sub-tasks that can map to an API call on the target device – for instance, the wireless sensor node. Instead of calling services that already run on the sensor device they then compile out of the model the necessary code for the sensor node and upload a new assembly.

2.13 Summary

In this chapter we briefly introduced the main technologies and concepts used in this work. We are using two hardware platforms (IRIS and Waspmote Pro, see Chapter 2.1), both based on a 2.4 GHz radio using the IEEE 802.15.4 MAC layer protocol. We will later on leverage on two differences between Waspmote Pro and IRIS: Waspmote Pro is the more advanced platform. It features a more precise real-time clock and an additional very-low energy sleep mode. Our software stack mainly consists of the Mote Runner environment, as introduced in Chapter 2.2. Mote Runner is a virtual machine based software platform for running IoT applications. It is purely reactive and thus not have any concurrency mechanisms (e.g. threads). We also introduce the essential parts of MRv6 (Chapter 2.3.3) – Mote Runners implementation of 6LoWPAN (Chapter 2.3.2).

On the architectural side a basic understanding of RESTful architectures is necessary (see Chapter 2.4). We introduced RDF and OWL as modeling languages. Furthermore, we introduced related work in the area of semantic services and enterprise integration. We showed how a service description language integrates into process models. We mainly refer to the work of Meyer et al. [267], which was designed with semantic service description based enterprise integration in mind.

Part II

Service Architecture for Embedding IoT-services into Enterprise Environments

In the second part of this work we present conceptual and architectural work on semantics-aware enterprise IoT-integration. In Chapter 3 we discuss the concept of an IoT-service. We present a literature survey on previous definitions and the use of the term service in the IoT in general. We contribute our own definition and classification of IoT service. Chapter 4 discusses the special needs of enterprises with regard to IoT-services. We present a novel architecture based on distributed Linked Services. We will argue that distributed Linked Services integrates well into both enterprises and IoT-systems. Furthermore, we will present a concept for easing modeling of business processes in a semantics-aware enterprise.

Chapter 3

Services and the Internet of Things

In enterprises, the terms ”service” and ”service-oriented architecture” are key concepts [380, 56, 35, 259]. In this chapter, we argue, based on a literature survey [381, 259], that no common definition of IoT service exists. We contribute to an overall understanding of the IoT-domain by deriving a definition and classification of IoT-services. We embed services in a comprehensive IoT domain model — the IoT-A model — and describe its relationship to other modeling elements of this domain model. Our understanding of the term ”service” and the differences with other concepts, like a technical interface, will be used in all subsequent chapters.

3.1 Introduction

As part of our work on actuator and sensor integration into enterprise IT systems, we identified services as one of the main building blocks of the future Internet. Surprisingly, there is still no common nomenclature and the term ”IoT-Service” has different meanings in different works [381]. As most enterprise software solutions nowadays are written following a service-oriented approach a convergence of the Internet of Services (IoS) and the enterprise IT world is anticipated for describing and provisioning IoT-services. Therefore, it is a necessity to have a good understanding of what an IoT-Service is, what its relationships to existing concepts are and how these concepts can be brought into the IoT world.

This chapter will first describe services in general and as a main building block of the Internet of Things. Furthermore, we will survey existing definitions of services in an IoT-context and later present our own coherent classification [380, 259]. In addition, we will show how this abstraction can help a business domain expert in working with IoT-services.

3.2 Services

Services are a suitable abstraction for building complex software systems. They are the fundament of most of today's enterprise systems [399]. In the same way that they play a crucial role in nowadays IT-systems, it is suggested that they will also play a crucial role in the Internet-of-Things [152]. Services are a central part of many domain and reference models, including the IoT-A architecture. Furthermore, currently a lot of research is being carried out in the areas of IoT and services [18].

Nonetheless, "service" is a somewhat overloaded term that can have many meanings. For example, one wide-spread use of the term service is to use it as a synonym for what we call a *technical interface* that is software functionality provided by a defined service interface (e.g. Web-Services). Barros and Oberle [29] define a service as:

> a commercial transaction where one party grants temporary access to the resources of another party (...)
> — Barros and Oberle, 2012 [29]

In service science and in Internet of Services (IoS) research there were and are many efforts to establish a single definition of the term "service". A comprehensive discussion of several ways to define service is given by Ferrario [125]. However, no definition has been accepted by the wider community yet [190, 29, 125].

3.3 Survey on the term "Service" in IoT

To get a clearer picture of what is currently considered an IoT-Service, we surveyed several publicly funded projects (e.g. SIRENA [51], SOCRADES [152], SENSEI [173], RUNES [97], and OASiS [284]) and did a comprehensive search through the ACM and IEEE literature databases for service concepts in the realm of the Internet of Things, Web of Things (WoT), cyber-physical systems (CPS), and related terms.

Only publicly available material was used. For example, no internal reports were taken into consideration. The search returned more than 620 documents that had a high probability of being of interest. These documents were automatically downloaded, a full-text layer added if needed (tessaract and hocr2pdf), analyzed with a full-text search engine (pdfgrep) and ranked according to their likelihood of discussing services and service-oriented concepts using a weighted mean algorithm based on selected keywords. The process is depicted in Figure 3.1. Out of all papers taken into consideration most papers were in the realm of the Internet of Things ($\approx$52%), followed by Cyber-Physical Systems ($\approx$40%).

A vast majority of the selected papers used or mentioned service-oriented principles ($>$90%). This is not a surprise since we were searching for service-related papers. What is surprising, however, is that only very few papers and projects dealing with IoT-services defined precisely what they consider an IoT-service ($<$10%), how it differs from traditional services, and how to combine IoT and non-IoT services. This is

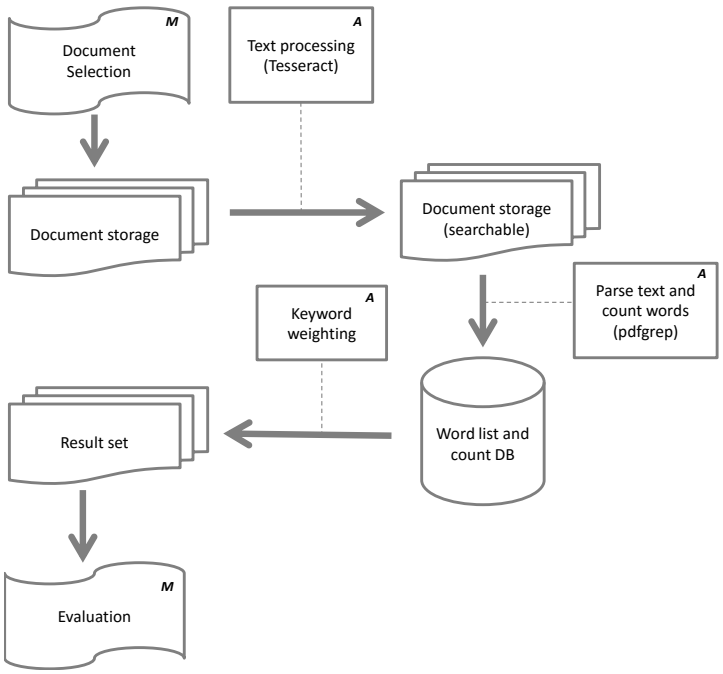


Figure 3.1: Survey: Process

even more astonishing since there are many IoT-middleware and IoT service oriented frameworks that claim to bridge the gap between the Enterprise SOA world and the physical world. Mainly papers from the realm of IoT dealt with SOA or services. This can be explained by a slight bias towards the term ”Internet of Things”, as its mainly used by European projects. Furthermore, CPS is typically more directed towards real-time systems [230] than IoT is; therefore CPS is less often used in a service context. Most papers ($\approx$ 80%) had only an implicit definition of IoT-service, or gave just a general definition of service.

In the following, we will focus on summarizing the key findings with regard to services. In a technical sense, there are differences between traditional services and IoT-services, like special QoS-parameters and the fact that the devices running these services are often resource-constrained with respect to computing, storage, communication, and energy capabilities. Many works focus on dealing with the special characteristics of such systems. Furthermore, IoT services might only occasionally be connected to a network [156, 194]. The findings generally are inline with a comprehensive survey on the IoT in general, carried out by Atzori et al. [18]. Atzori et al. also noticed a tendency towards service-orientation. Nonetheless, IoT services are not explicitly covered by them.

From all the mentions of the term service, only a few definitions go beyond a technical interface. In the end, we were only able to identify three specific definitions that particularly concern what we will later call an IoT-service: the terms ”real-world

SOA" used by Karnouskos et al. [194], "real-world service" by De et al. [104], and the term "web presence" by Debaty et al. [106]. In the following, we briefly summarize those definitions:

Karnouskos et al. [194] define the term "Real-world SOA" as:

> Real-world SOA, is a SOA where each device offers its functionality in a service-oriented way; is able to discover other devices and their hosted services dynamically at runtime; can invoke actions of the discovered services dynamically; and is able to publish and subscribe to typed, asynchronous events
> — Karnouskos et al. [194]

However, a specific discussion of what constitutes a real world (or IoT) service is missing though.

The similar term "real-world service" is used by De et al. [104]

> The Internet of Things envisions a multitude of heterogeneous objects and interactions with the physical environment. The functionalities provided by these objects can be termed as "real-world services" as they provide a near real-time state of the physical world.
> — De et. al. [104]

Nonetheless, the term "real-world service" is ambiguous as it is also used for explicitly non-IT services provided in the real world (e.g. the transportation of goods) [5]. This is why we use the term "IoT-service", which has no predefined meaning in other domains.

A similar definition is given by Debaty et. al. [106]:

> To computer applications, the incarnation of a Web presence is a set of Web services to learn and interact with the physical entity. *where a web presence is* the virtual representation of a physical entity as an integrant part of the Web.
> — Debaty et al. [106]

These are not the only definitions available. For example, some research projects (e.g. [316, 173]) differentiate between sensing and actuation services. Nonetheless, the definitions of these two kinds of services, which are essential for the Internet of Things, are only implicitly given. For example, when the term IoT-service is used for describing interfaces to devices that perform the actual sensing or actuation task.

3.4 Services and the Internet of Things-Architecture

This section shows how we integrated services in the Internet of Things-Architecture. We continue using an intuitive definition of the term "service". In Chapter 3.5, we will

introduce our definition and a classification of service. We now introduce the Internet-of-Things Architecture (IoT-A), with a strong focus on how we utilized services in this architecture. The IoT-A project started with the perception that even after many years of heavy discussion in the community, there was still no clear and common definition of many concepts. The range of the IoT domain and IoT solutions is already large. The objective of the project was not to develop or consider a ”one-size-fits-all” protocol or architecture, nor to define interoperability layers between all sorts of devices. The main driver of the project was the observation that there was a need for a common ground at more abstract levels. The results of the IoT-A project are based on a cooperation of more than 20 organizations from industry and academia that have been working for a period of more than three years. The overall results of the project can be found in Bassi et al. [34]. In this chapter, we only describe the service-related part that we contributed and how we integrated it into the IoT-A architecture.

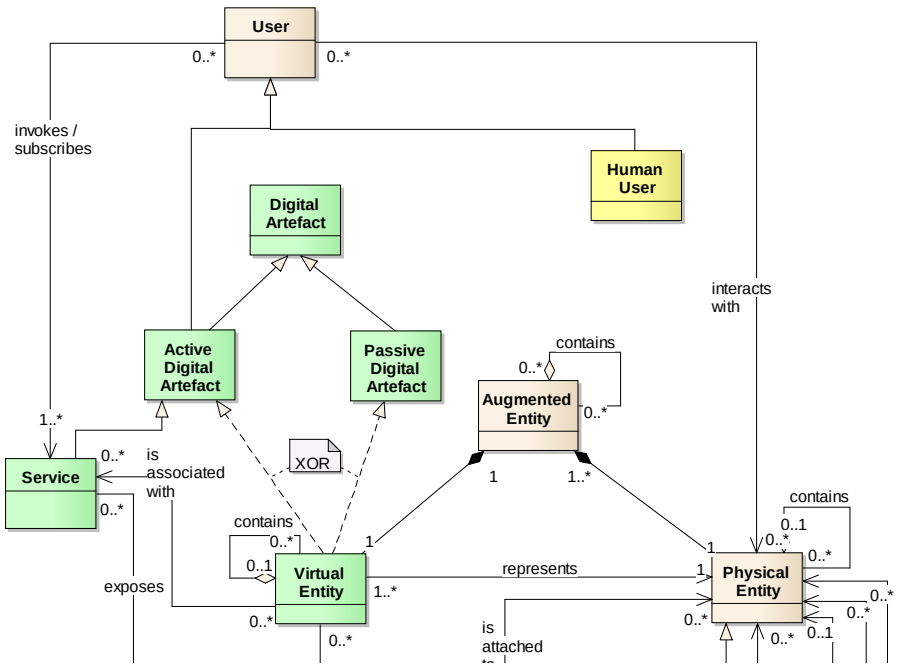


Figure 3.2: Relationship between a user and a physical entity [33]

As a part of the Internet of Things-Architecture project, six core concepts of the Internet of Things have been identified [160], namely: user, physical entity, virtual entity, augmented entity, device, resource, and service. The main driver behind any IoT-system is the user who has the requirement to interact with a so-called real-world object. The term **user** is used in a generic way and does not need to be human. Any technical device or technical process can be the user of a system. The real-world objects can be passively monitored, or actively manipulated by the user. A *physical*

entity (sometimes just called *Entity* or *Thing*) is a real-world object in which a user is interested.

The term ”physical entity” was mainly used to avoid misunderstandings between a real-world entity and entities that might exist otherwise, for example, as part of an information system. Other works sometimes use the terms real-world object [103, 323], or Entity of Interest [394, 155]. The relationship between a user and a physical entity is shown in Figure 3.2.

The next concept is that of a Virtual Entity: The idea of a virtual entity is closely related to the concept of a *virtual counterpart*, as introduced by Römer et al [323]. A virtual entity is the digital counterpart of the physical entity that exists in an information processing system. The third *-entity concept is that of an augmented entity. An augmented entity is the composition of a Physical Entity with its associated Virtual Entity. The relationship between the three entity types is shown in the middle of Figure 3.2.

Figure 3.3 shows the complete domain model as developed in the IoT-A project. We will now describe how services interact with devices. Devices are computing units (any hardware that is able to do ”something”). In the domain model [33], the device is the superclass for the more specific components. Devices can be physically attached to Physical Entities, like an RFID-tag. They can also be somewhere in the surroundings of a physical entity — for example, a temperature sensor in a container. The three most common device-types are tags, sensors, and actors. Tags identify physical entities. They can be read by sensors. Sensors monitor physical entities, whereas actors can act on physical entities.

Resources are defined as software components that implement functionalities [34]. Resources can be hosted on a device or any other place in the network or in the cloud. A cloud resource, for example, could derive higher level information by analyzing data provided by other resources hosted on sensor devices. We will use the term ”resources” for means to get information about the environment. Resources provide the technical links to physical entities. They could be more than just ”dumb” sensors. For example, the temperature of a room (the physical entity we are interested in), could be observed by one ore more mobile phones of people who are actually in the room. In this case, the resources are the mobile phones. The resources then access temperature sensors on the phone. The sensors are the devices.

While resources provide functionality they do not offer any means of accessing these functionalities. A service exposes the functionality of a resource via an interface that can be invoked by the user. The service has internal knowledge on how to access the functionality provided by a resource. The definition of a service is technical in nature at this point. In Chapter 3.5, we will generalize this to form a more comprehensive definition of an IoT-service. The IoT-A is very generic in nature. It offers many design choices to a system designer. For example, it does not specify how services are described, how the interface of the service can be derived, or what kind of physical entity it interacts with. However, it suggests the use of semantic services.

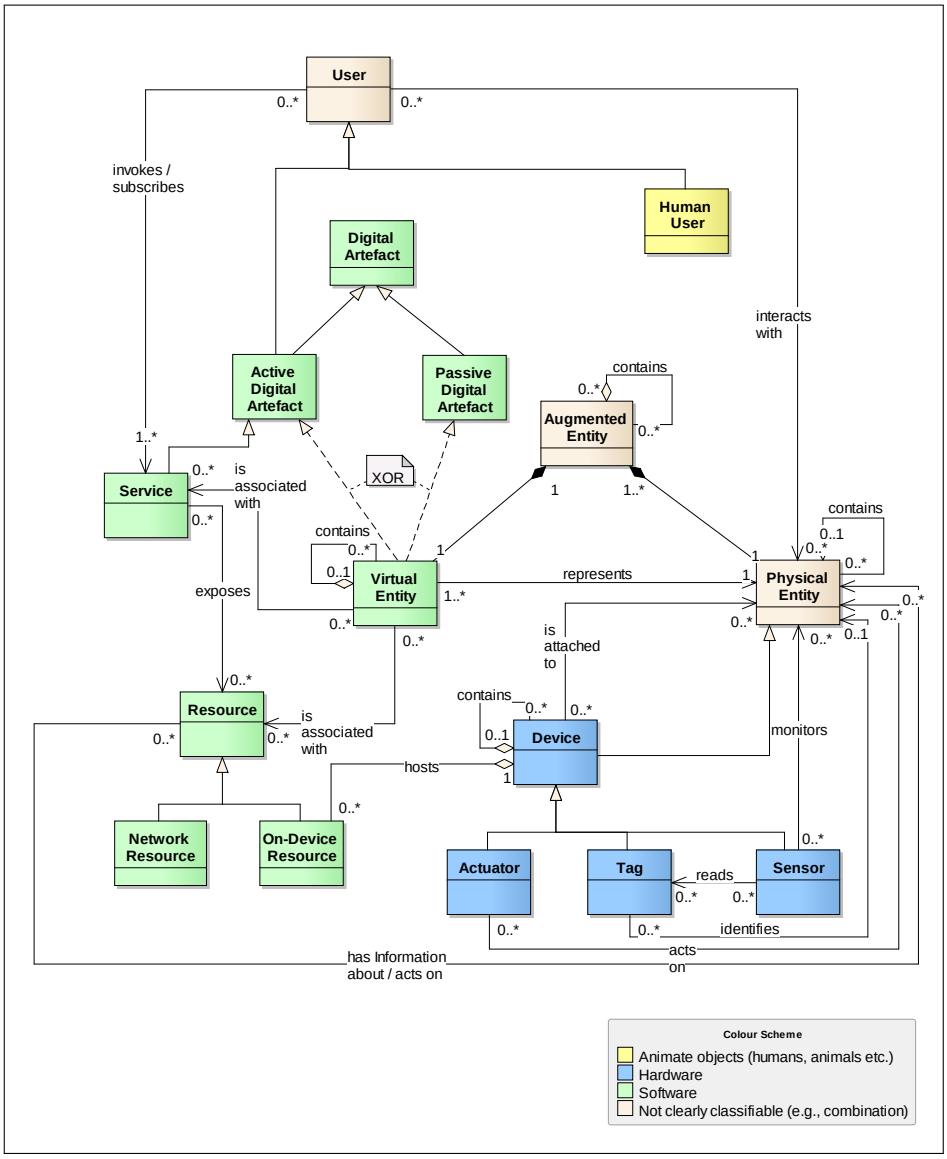


Figure 3.3: IoT-A domain model (complete view) [33]

3.5 Definition and Classification of IoT-Services

As discussed in Chapter 3.3, most people use the term IoT-service for accessing the functionality (for example: sensing) of a device (a sensor) or resource via low level services. It is rarely used as a high level concept and, if so, the relationship to general purpose services, as known from Enterprise IT or the Internet of Services, is not entirely clear. As explained in Chapter 3.4, we take a perspective based on the physical entity and resources.

As ”service” is an ambiguous term, we give a definition that is not too limiting (for example, by defining a service as a software component only), but that is still restrictive enough so that IoT-services can stand as a field of their own. We used the term *transaction* as introduced in the work of Barros and Oberle [29], but do not limit ourselves to commercial transactions.

> An *IoT-Service* is a transaction between two parties, the service provider and the service consumer. It causes a prescribed function enabling the interaction with the physical world by measuring the state of entities or by initiating actions which will cause a change to the entities.

In the following, we propose a simple but comprehensive classification of services along two dimensions:

1. Relationship with the entity
2. Service life-cycle

We define IoT services as services enabling interactions with the real world, and thus as the superset of the more specifically defined services outlined in Table 3.1. Integrated services are conceptually used to form a bridge between IoT-specific services and external services/non IoT specific services.

When we classify along the relationship with the physical entity (see Table 3.1), one of the key concepts we explicitly introduce are the *Entity service* and *integrated services* as higher level of abstraction that utilize simple resource/device and other (IoT-external) services as primitives, thereby hiding the complexity of dealing with them from modeling experts, developers and users. This is the most natural way to look at an IoT-thing entity, because it is more intuitive for domain experts as they do not need to work with some low-level sensor service interface. This has to be abstracted for being usable, for example, in enterprise environments. Additionally, this, of course, also reduces the complexity for development in general and allows the integration into service-oriented environments.

Table 3.1: Classification based on relationship with the Entity

Low-level service	The low-level services make the capabilities of the devices or the resources accessible to entity services or integrated services
Resource service	Resource services provide the observations the resource is capable to make or provide the actions a resource is capable to execute.
”Entity”-Service	Entity services are the heart of IoT systems. These are the services provided by the entities and they are often, but not necessarily, compositions of low-level services.
Integrated service	Integrated services work with entities. They usually are based on entity services and compose them with non-IoT services.
IoT-Service	An *IoT-service* is a transaction between two parties, the service provider and the service consumer. It causes a prescribed function enabling the interaction with the physical world by measuring the state of entities or by initiating actions which will cause a change to the entities.

Table 3.2: Classification based on the service life-cycle

Deployable	A service that is not yet in the field, but that is generally deployable. The according service description exists in a service repository, but an appropriate runtime environment is not yet assigned. Thus a service locator is not available in the service registry.
Deployed	A deployed service is already in the field, but not yet ready for use. Further steps are necessary to make it operational. These further steps could be technical or economic (like paying for the service)
Operational	An operational service is already deployed and ready to use. The service is associated with an entity and the association is known to a resolution infrastructure.

Another important classification of services is according to the service lifecycle. Apart from the different Quality of Information (QoI) and Quality of Service (QoS) constraints of IoT-services, the other very special thing is that they are bound to and run on a large variety of devices, and they have to be aware of physical entities. This complicates the service management compared to traditional service management frameworks. In an Enterprise context, it is therefore a necessity to have a closer look at the different states in the service lifecycle: This is a necessary precondition for the service management, as well as for enterprise service orchestration and service choreography engines as discussed by Haller et al. [157] and Sperner et al. [362]. An IoT-service is usually bound to a physical entity and working together with a service environment or within a general service ecosystem. In general, we distinguish between three different states: deployable, deployed and operational. We call a service deployable, if it is generally ready to be deployed in the field. Commonly, this means that the service is either not yet on its destination or not yet part of an appropriate service runtime environment as part of the service ecosystem. Once the service is in the field (i.e. at its target destination), we call it *deployed*. A deployed service is not yet working and serving requests. Further steps might be necessary. These steps can be either technical or economic. This could mean that first batteries need to be charged to become operational or that it is waiting for someone to pay for its services. Finally, we call a service *operational* if it can serve requests. The service is typically associated to a physical entity and is known to a service ecosystem, that means it can be found in a service repository and resolved to a technical endpoint.

3.6 Conclusions

The term ”IoT-service” is used inconsistently in literature. Most projects use the term intuitively. While it might sound obvious that there are specific differences between services in a classic sense and services that allow interaction with the physical world, a conceptual frame defining them is necessary. First, in Chapter 3.2, we argued that a comprehensive definition of the term ”service” in the Internet of Things is needed. In Chapter 3.3 we performed a literature survey and concluded that many works are based on a rather implicit definition of service and that no generally accepted definition exists. In Chapter 3.4 we then showed how services interact within an IoT framework. We introduced a conceptual view on IoT-services, which allows easy integration of IoT entities into the service-oriented world. Based on existing definitions, in Chapter 3.5, we suggest a definition based on the term ”transaction” that is consistent with definitions found in service science. The classification and the surrounding concepts are centered on the entity and not that much on the technical representations or means of realization through low-level services. We explained the entity-concept and mapped our service-definitions to it. In the next chapter, we will discuss how those entities and services can be integrated into enterprise IT systems.

Chapter 4

Enterprise-embedded IoT-Services

This chapter discusses the special needs of enterprises with regard to IoT-services. We present the opportunities for an enterprise that lie in the general use of semantics and service descriptions for interoperability [381, 375, 382, 373, 374]. We contribute abstractions — such as — the semantical physical business entity [382] to enable business modeling using semantic concepts. Furthermore, we present a novel architecture based on linked service descriptions to enable interoperability with IoT-devices. Its novelty is the combination of Linked Services and the distribution of services, which combined, serve the needs of both enterprises and constrained IoT-devices. We argue that (distributed) Linked Services are especially well suited for IoT-applications, given their limited battery power, as well as storage and processing constraints. Thus, Linked Services do not only enable interoperability, but also suit the needs of constrained devices.

4.1 Introduction

One of the main challenges of future enterprise IT systems is using data collected from the real world in real time, contextualizing it, and providing the user of these systems with the best possible up-to-date information on which to base business decisions. The vision of context-aware and real-world aware enterprise IT systems is often associated with the term ”Sensing Enterprise” [335, 57, 319]. A recent approach to describe services in a service-oriented environment are semantically enriched service descriptions, based on semantic web technologies. This chapter presents our vision of the sensing enterprise based on these semantically-enriched services. We use them to access sensor devices, which are able to describe themselves, thus enabling a sensing enterprise that seamlessly integrates into today’s enterprise world. Furthermore, we will discuss an integration platform as an architectural proposal that can be used to connect ERP systems to the physical world.

Semantics and even ontologies do already exist in nowadays enterprise IT systems. Nonetheless, they are often hidden and not explicitly stated as such — at least compared to what is seen in the semantic web movement. To enable interoperability between

different components of an enterprise system, often a common vocabulary is used. The specific modules are then based on such common datatypes. For instance, SAP ERP systems use a Global Data Type (GDT) [334] directory that represents business related content SAP wide. All elements of service-oriented services provided by SAP systems are described (typed) by GDTs. A GDT is more than just an integer or a float. It might be something rather generic like an invoice or something very domain-specific like AirCargoSecurityStatusCode. This implicit knowledge provided by the GDT directory currently specifies more than 5100 different data types and it is documented on more than 16.000 pages. This shows the potential of semantics in enterprises that could be leveraged if it were made more explicitly available.

4.2 Key Drivers

We see four key drivers in the sensing enterprise context. The four key drivers are:

Interoperability: In the past (and even nowadays, but at a lower level) ERP vendors used to base their system on proprietary protocols. Interoperability, as a result, meant implementing custom connectors to these services. The connectors had to ensure interoperability on a technical level and also map business concepts. To some degree, this was caused by historical and technical reasons, and the lack of agreed-on standards. Nowadays, in the days of coopetition [58, 41] interoperability has become more important. Thinking further into the future, especially in the sensing enterprise, where we might deal with all kinds of smart items from several vendors, which have to integrate into various backend systems, flexible and smart interoperability is a must. Furthermore, even when we talk about enterprise interoperability today, there is specific knowledge of the used protocol and the data required. Future enterprise systems need to target interoperability at a semantic level as well.

Standardized Technologies for Direct Access of IoT-enabled Entities: This key driver is closely connected with the general problem of achieving interoperability. Enterprise systems move away from their proprietary protocols towards standardized technologies, like TCP/IP, HTTP or OData. Previously, IoT-integration platforms typically were middleware solutions, where the enterprise system only communicated via an interface with the middleware but not directly with the IoT devices. We anticipate a shift towards the use of standardized (Internet-) technologies that allow communication between enterprise systems and IoT-devices directly, or over transparent proxies.

Enablement of Sense-making: Future enterprise backend systems will have to do reasoning on data from various sources. The description of these services should allow semantic annotations that can be understood and processed by enterprise

IT systems. As we will describe later, we are following a pragmatic approach here, without the need of being fully reasonable in a theoretical sense.

Enablement of Real-time Business Decision Support: In many industries, the back-end systems are still disconnected from what is actually happening. Often, the information is gathered a-posteriori and deviations from the planned state are detected late. Integrating real-time decision support into these systems will enable a business to run more efficiently, react in a timely manner to the changes in the business process, and allow proper exception handling. While this is closely related to sense-making, real-time business decision support does not only rely on sense-making. Traditional rule based complex event processing systems are already in use today.

Two emerging technologies could soon enable the sensing enterprise to become a reality:

Real time Big Data Analytics: A typical enterprise generates large and diverse data sets from its distributed business locations. Besides OLAP data, the enterprise might also record data produced by social networks, surveillance devices, or third party systems owned by business partners. These massive amounts of detailed data can be combined and analyzed by predictive analytics, data mining, or statistics. Doing this in real-time — for example, by using in-memory data processing [118, 119] – creates a business advantage for the company by giving insight into the real-world dynamics of their business.

Sensor Networks and Near Field Communication: Sensor networks are starting to complement the already existing RFID (Auto ID) technologies that are already available on the market.

4.3 Semantic Service Descriptions

A service description is used to describe the characteristics of a service. This may include non-functional properties as well as a technical interface. It is important to distinguish between the deployed service itself (in a technical sense) and its description. The description, for example, does not need to be hosted on the same device as the service. A service description can exist without a deployed service. This is a necessary precondition for supporting reconfigurability and reprogrammability.

We define a service description as follows:

> A service description is a description of all essential properties of a given service, as well as the means to access it. A service description is independent of an actual implemented callable service.

As emphasized earlier, it is important to distinguish between service descriptions and the actual implemented (callable) service on a device. The technical interface to this callable service is called endpoint. Please be aware that the term endpoint might have different meanings in specific actual protocols specifications or technologies. Unless stated otherwise, we will always refer to the generic concept of an implemented callable technical interface. The process of linking a service description to an endpoint is called binding, while finding services for specific items of interest is called discovery. The means of access a service is called resolution and delivers an endpoint description.

4.4 Linked Services

Our architecture was designed with the use of Linked Services [297, 299] in mind. As we will describe we extend the definition of linking to also include a distribution aspect and relax the constraints on technology. Furthermore, we assume a pragmatic view on the usage of semantics. In a nutshell, the idea behind Linked Services is to base service descriptions on standard technologies known from the semantic web whereby their inputs and outputs, their functionality, and their non-functional properties are described in terms of (reused) light weight RDFS vocabularies and exposed following Linked Data principles [297]. The linked data principle was outlined by Berners-Lee [42]. He suggested the following four simple rules for publishing data on the web, thus creating a single data space – the web of data:

1. Use URIs as names for things
2. Use HTTP so that people can look up those names
3. When someone looks up a URI, provide useful information, using the standards
4. Include links to other URIs, so that they can discover more things

The linking aspect, here, essentially means linking vocabularies through reusing lightweight RDFS vocabularies. In our work we extend the term to also include the possibility of linking parts of one service together, thus distribute parts of the service description, through means of standard technology. Furthermore, we also relax the constraint of semantic web technology to not only include the traditional standards (RDF, SPARQL) but any semantic technologies like, for example, OData. Additionally, we also have a pragmatic view on semantics in general and differentiate between enterprise use of semantics and the more ambitious web of data.

Compared to the very ambitious idea of the *web of data* [50] our vision is way more focused on the interoperability between enterprise systems, and between enterprise systems and IoT devices. We currently see reasoning as applicable on a domain level only. In contrast to many ongoing research on the semantic web, we do not want to model or understand the whole world. Our mid-term goal is semantic interoperability in very specific domains (e.g. in retail). For this, we foresee the use of light-weight

ontologies. We would even allow manual steps, like hard-coded rules by domain experts, in the processing of these services. Research on interoperability has shown the need for *semantic interoperability*, this is sometimes complemented by the need for *pragmatic interoperability* [301]. Our approach does not solve all the problems that can arise from wrong assumptions on either the semantic or the pragmatic level, but the use of semantic technologies and the restriction on a domain-level should reduce the risk of making wrong assumptions on both sides of the communication channel.

Therefore, we have a more pragmatic view on the web-of-things and its technologies. For example, in Chapter 6 we will use the Constrained Application Protocol (CoAP) instead of HTTP. Any REST-based protocol is suitable for the implementation of Linked Services. Furthermore, as emphasized already, we are not aiming for understanding or describing the "whole world". Our aim is simply to leverage on semantic technologies to the extent needed to achieve our goals.

4.5 Semantic Physical Business Entities

We now introduce the concept of Semantic Physical Business Entities, which in conjunction with ontologies can be used to make this implicit knowledge explicit and directly accessible. First, we define the term Physical Business Entity (PBE) that makes the difference to a semantic physical business entity explicit:

> A *Physical Business Entity* is a conceptual representation of a real-world entity processed by one or more enterprise IT systems.

At this point we limited ourselves to *physical entities* in the real-world. Nonetheless, our approach can be generalized to other types of entities, which can be observed by all kinds of sensors. This includes even high-level concepts such as social networks.

In the literature there is no common agreement on the definition of the terms business object and business entity. Some authors use them interchangeably; others define business objects as entities with logic. As we want to emphasize the relationship with the entity concept as found in the IoT, we choose the term semantic physical business entity, which we define as follows:

> A *Semantic Physical Business Entity* is a conceptual representation of a real-world object processed by one or more enterprise IT systems. Information about it is discoverable. It is described through well-defined vocabularies that make internal and external relationships explicit.

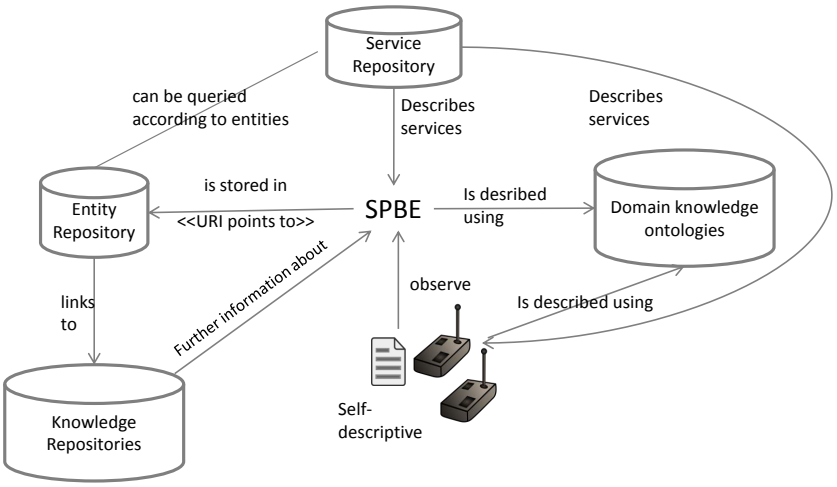


Figure 4.1: Knowledge repositories and distribution

The decoupling between the PBEs and the devices observing it is important: An enterprise IT system's user is usually not interested in the value of sensor no. 0815 or sensor no. 4711, but in the temperature of some given good or class of goods, which could be monitored by several sensors. This abstraction, moving away from the pure technical view concentrating on sensing devices, towards the "things" they monitor is one of the main ideas in the IoT community. The SPBE is just a concept. It manifests itself through the use and combination of several semantic repositories. The decoupling and the dependencies between semantic repositories that form the concept of an SPBE are shown in more detail in Figure 4.1. The link from SPBEs to an entity in the physical world is through an entity repository and self-descriptive IoT-nodes. Both, the entity description and the service descriptions of the IoT-nodes are stored in semantic repositories. An entity is further described in general and domain-specific knowledge repositories. The services running on an IoT-device are also further refined in domain-specific ontologies. Further services, that are used to build an integrated service are also stored in the service repositories. In Chapter 4.6 we show how our platform supports this abstraction.

4.6 Enterprise Integration

In our opinion the most important benefits of using service descriptions are seen at the level of the enterprise IT integration. This section presents a high-level architectural overview on how linkable service descriptions and enterprise IT systems can be combined using the service-oriented paradigm. We first present requirements derived from a supply-chain and retail use case and then an integration scenario.

4.6.1 Requirements

In the following, we briefly present what we consider the most important requirements for enterprise integration, namely the modeling of service characteristics, linking services, use of standardized technology, service discovery, constrained resources and reconfigurability and reprogrammability.

4.6.1.1 Modeling of Service Characteristics

To integrate a service into an enterprise system it is necessary to have a service endpoint and at least a technical description on how to invoke it. In addition, it is desired to have also non-functional aspects described. The execution engine can take these into consideration in the binding and execution phase of a business process.

4.6.1.2 Linked Services

The architecture should be able to use linked services as introduced in Chapter 4.4. It should be able to follow links and download service descriptions.

4.6.1.3 Standardized (Internet) Technology

The architecture should leverage on standardized Internet technologies instead of producing home-grown protocols. Nonetheless, the support of existing proprietary protocols should be possible, if needed.

4.6.1.4 Service Discovery

In a traditional service-oriented environment a service registry or repository is sufficient for discovering services within an enterprise. For those sensor networks that are more or less static in nature, where most of the business logic is performed in the enterprise backend system, or if only limited business logic is executed on the nodes, a repository approach is sufficient as well. Nonetheless, in ad-hoc or self-organizing scenarios, where a lot of business logic is executed on the nodes and a backend system might not even exist, self-description of services is essential [395]. Especially, in a non-static transportation setting there is often the problem that one or more sensor nodes join a different enterprise context, depending on their location. In such transportation scenarios, for example, a sensor network would monitor the goods along the complete supply chain. In case of food it could monitor humidity and temperature. A recent research project in this context is Intelligent Container [136]. In such a setup, it is possible to run small business processes on the sensor nodes, for example, calculating the final price the customer has to pay based on SLA and pricing models stored on the sensor nodes.

4.6.1.5 Constrained Resources

It is essential for the wide adoption of sensing technology in the enterprise IT that related technologies come at a low cost. Therefore, we are dealing with devices that are constrained in terms of memory, computation and communication. In an industrial setting the use of constrained devices is desired and often enforced to reduce the total cost.

4.6.1.6 Reconfigurability and Reprogrammability

As more sensors are deployed by enterprises, the evolution [306], shared use and reuse of already deployed sensor networks will play a crucial role. In a typical enterprise IT system a sensor network is used for one or multiple tasks for some time. Changing requirements and cost pressure lead to the need of constant reconfiguration and shared use of resources. Applications that time share a sensor network, might need to reflash a node to perform its task, as sensor nets are usually memory constrained. Therefore, it is essential that the devices can easily be reconfigured and reprogrammed.

4.6.2 Application Scenario

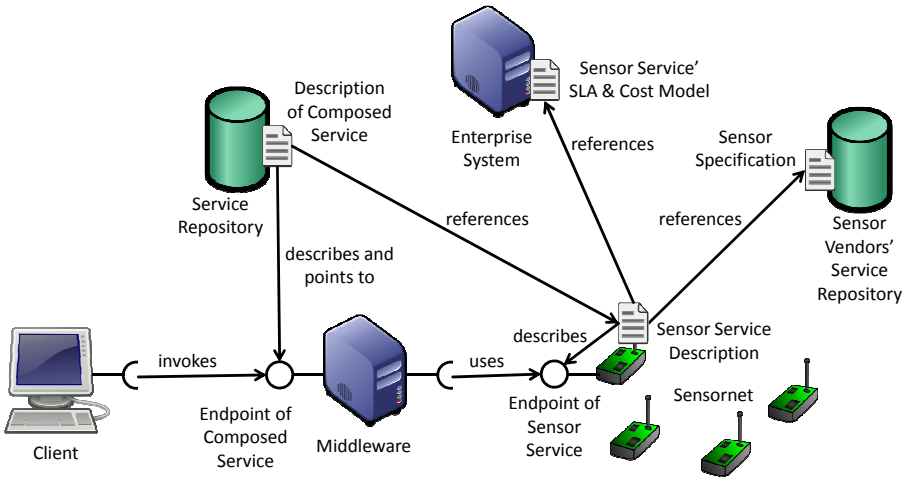


Figure 4.2: Typical sensor network integration scenario in an enterprise environment

In the following we will present a motivating application scenario for the use of Linked Services. Figure 4.2 shows a possible partitioning of a service deployed on the sensor nodes with different references (linking) to other sources, where more information about this particular service can be retrieved. The composed service that forms a sensor network service has its accessible service endpoint on an enterprise middleware system. It refers to the sensing services on the individual sensor nodes.

The sensor service descriptions on the individual nodes describe their technical service endpoint, which is on the same IoT (sensor) device in this scenario. Nonetheless, this is completely flexible. The service description there, for example, may link to some service endpoint on the gateway and to a service description on the gateway.

Furthermore, the sensor nodes' service descriptions link to further information on an enterprise system (e.g. SLAs) and even to detailed comprehensive sensor specifications at some repository of the sensor vendor.

The entire architecture follows service-oriented principles. The sensor network services up to the middleware follow a classic repository-based approach. The sensor network beyond the gateway also follows service-oriented principles. Service descriptions and service endpoints allow, for example, the usage of individual sensor nodes and compositions thereof just like any other web service from the perspective of a business process execution engine.

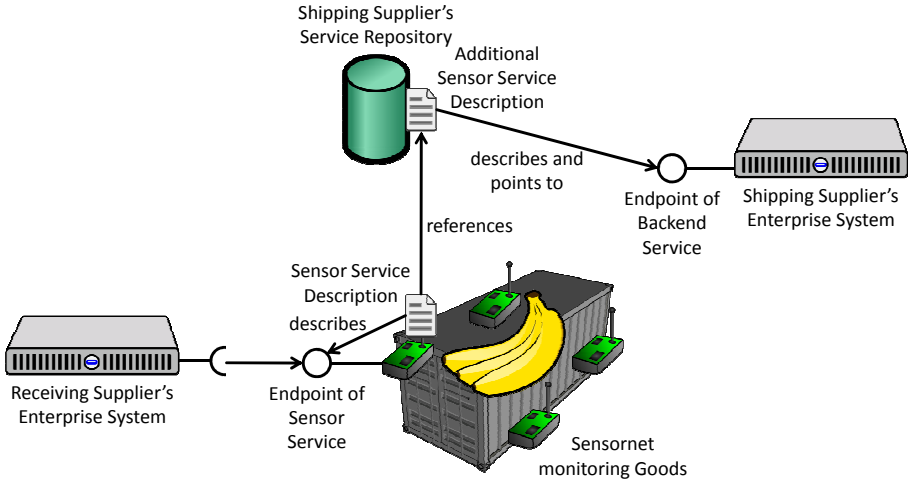


Figure 4.3: Sensor network migration in an enterprise context

Figure 4.3, shows another scenario. The ERP processes queries and accesses the sensor nodes directly. There is no explicit, non-transparent, gateway. This scenario is common in a sensor network migration context. In the transportation of goods, for example, we might have a sensor network monitoring one or more shipping containers. These containers move along different stations in the supply chain; therefore sensor networks need to be integrated into several enterprise backend systems. These backend systems get direct access to the sensor network and integrate them into their own network. In addition, the description might refer to the original vendors backend system for additional information. For example, the service level agreements could be stored there and accessed for billing. Nonetheless, the sensor network needs to remain accessible even when this additional information is not available. In the next section, we will generalize the concepts we used intuitively in the application scenario. What we called middleware in this section will be refined and introduced as integration

platform. The integration platform is typically transparent, but could also be used to host composed services.

4.7 Enterprise Integration Platform Architecture

In the following we present an architectural proposal based on the principles and ideas discussed in Chapter 3 and Chapter 4. We provide an overview of how those concepts can be used to enable the sensing enterprise. We present the architecture of a Linked Services enterprise platform and describe how we apply light-weight service descriptions to smart items and sensor networks. We are not going into too many details on how Linked Service integration is carried out in regular enterprise networks and service marketplaces, but present our extension to the concept. More information on Linked Service integration is given by Cardoso et al. [75, 77] and Razo-Zapata et al. [315].

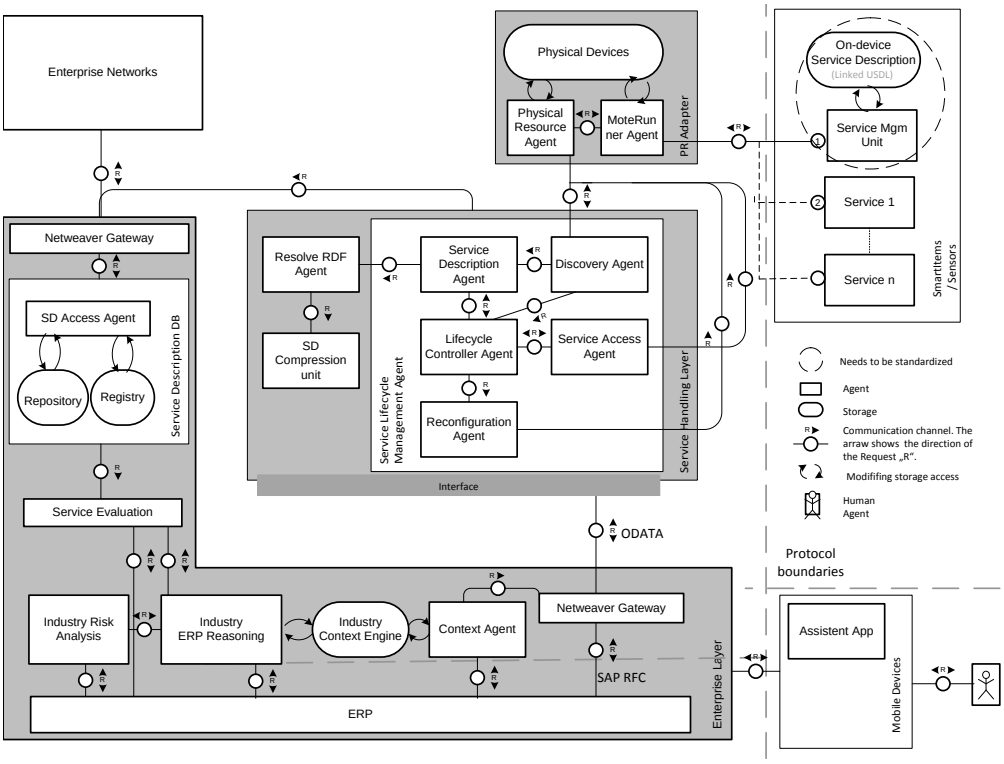


Figure 4.4: Architectural block diagram of the Linked Service enterprise platform.

Our platform, as shown in Fig. 4.4, enables an enterprise IT system to access and evaluate Linked Services. We distinguish between two main building blocks: the enterprise layer and the integration platform. The integration platform (IP) consists of the service handling layer and the physical resource adapters (PR-adapters).

The *enterprise layer* can access both the integration platform and the smart items via the service-oriented paradigm. As we consider IoT devices to be first-class entities in our service-oriented architecture, they are able to describe themselves through *Linked Services descriptions*. The actual *services* are accessed via the *service management unit*. Interoperability is achieved via a standardization of the interface to the management unit and of the service description language.

The ERP accesses the *services* provided by the IoT devices in the registry. As the *service description* is standardized it can evaluate its contents using the *service evaluation* component. The business information encoded in the service description is domain-specific and it triggers further actions in the business processes running within the ERP. The actual means of access is described in the service endpoint description. In our system the endpoints are described in Linked USDL for IoT (see Chapter 5). Nonetheless, the architecture is not bound to Linked USDL. Any other solution can be used as well. Depending on the protocol used and the capabilities of the ERP system either a direct access is possible, the service-oriented integration platform could serve as a protocol gateway, or the integration platform itself could be the service endpoint [381].

The communication between the enterprise backend system and the integration platform is commonly carried out via a *gateway component* (e.g. SAP Netweaver Gateway) that separates the *ERP* and auxiliary services. As the integration platform is just a service for the ERP system, communication takes place through a standard interface known to the enterprise SOA environment. Nowadays, ERP systems still use a lot of heavy-weight proprietary protocols, e.g. BAPI or SAP RFC, whereas the communication with the SOA platform can be done via a standardized protocol. The industry is currently moving towards the Open Data Protocol (OData) as specified by the OData consortium [274]. At an application layer, CoAP and HTTP seem to be good candidates in addition to Linked Services. This allows different ERP vendors to implement their own integration platform without losing interoperability. These self-description capabilities allow the smart items to join an ERP backend system in order to integrate into the system automatically based on the semantic information found in the *service description*, trigger backend actions and access the *services* on the smart item.

The *Integration Platform* takes care of the actual handling of services on smart devices. It is exposed to the enterprise layer through a service-oriented compatible interface. Thus, it gets integrated into an enterprise' service-oriented environment. Within the Integration Platform there are several agents which we will describe briefly: The discovery agent (DA) interacts with the physical resource adapter (PRA) and discovers new services available on new smart items. Communication with and monitoring of smart items is typically done via the physical resource adapter. New service descriptions are first completed by the *semantic resolution agent* and (in case of compression) uncompressed by the *service description compression* unit. The service description agent is then responsible for adding the service to the ERPs service registry and repository,

thus making it available to the enterprise. Communication between the ERP and the services on the mote either can be directly, for example for HTTP or CoAP-based services in an IPv6 network, or with the help of a s*ervice access gateway* agent that serves as gateway.

The deployment view of our envisioned semantic enterprise integration platform is shown in Figure 4.5 Several integration platforms called integration platform instances (IPI) can be used to talk to one or more wireless sensor networks. Specialized physical resource adapters, for example for the Mote Runner system, are responsible for the communication between the sensor networks and a specific instance of the integration platform. The communication between the ERP and the individual device can be either direct (transparent) or indirect. In an indirect scenario a service endpoint can be placed on the IPI via the specialized resource adapters. The only requirement we have towards the motes is that the IPI is notified when motes join or leave the WSN. Optionally, the motes can have a service description stored on them. This is necessary when third-party smart objects join the domain of an enterprise. Information from the motes is then used in the information and reasoning parts of the enterprise system, supported by ontologies and domain knowledge.

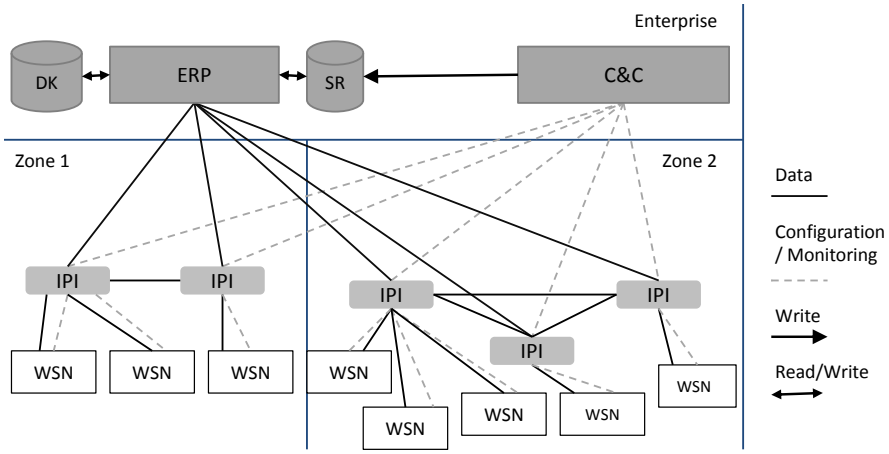


Figure 4.5: Deployment view

The actual IPIs are deployed to different (geographical or logical) zones and control one ore more motes. They are responsible for reading data from them or setting their state in case of an actuator.

Considering interoperability as it is currently done, one observes that almost always a device-centric approach is used: Specific modules within the enterprise system interact with the sensing devices through proprietary protocols. Recently, the situation has changed towards the more widespread use IP-based protocols and standardized application level protocols.

We suggest moving away from the device-centric approach by leveraging on the SPBE conceptual framework as introduced in Chapter 4.5. The integration platform

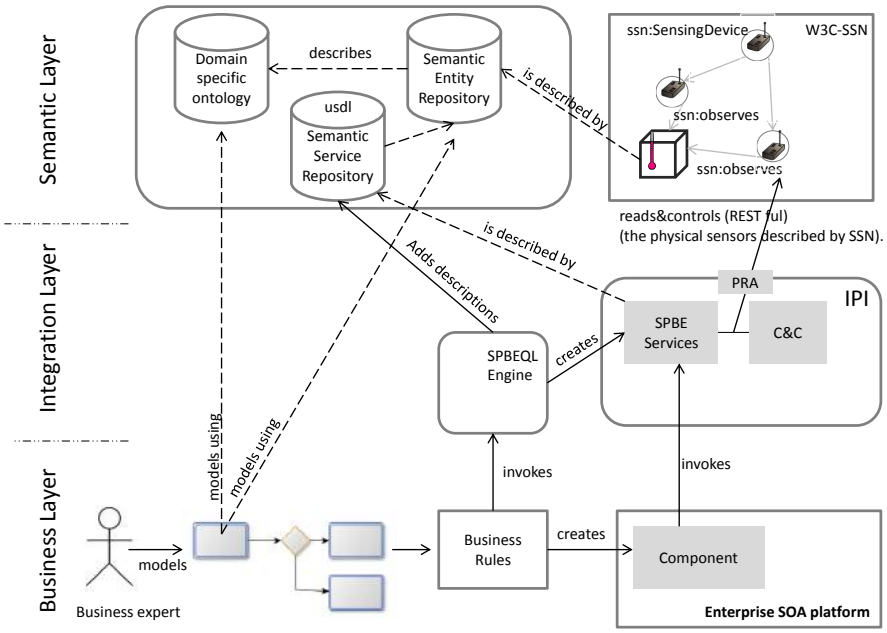


Figure 4.6: SPBE in an enterprise modeling framework

needs support from the business layer/enterprise layer. Modeling tools allow to model IoT-services on semantic repositories as shown in Figure 4.6. A semantic enterprise modeling framework can be divided into three layers: a semantic layer, an integration layer and a business layer. The integration layer in Figure 4.6 refers to semantic integration. The integration platform is only a part of it. As the image shows, the IPI can be used to realize an SPBE integration layer. The components of the integration platform are abstracted as command&control (c&c) unit. All information that the business expert needs to model an IoT system is stored in semantic repositories. The business expert uses a modeling tool that can access all the relevant information stored in semantic repositories. The semantic repositories can be further divided into domain-specific knowledge, semantic representations of physical entities and services related to those entities. A transition from legacy systems to such a framework is possible: Business objects are already stored in all kinds of data stores. The introduction of identifiers addressing these semantic repositories is necessary. Existing data can be made available through service interfaces using semantic descriptions. This would not introduce any changes to existing code. Access to all sorts of semantic entities is accomplished through semantically described services as introduced in this work.

The business expert can formulate different rules that are expressed using SPBEs. In our vision, the running business model is then to be executed by a (future) semantically-enriched business process execution engine. We call this execution engine SPBEQL engine. It can either access IoT-devices directly or it can create combined higher level

(integrated) services that run on the IPI. Those newly created services are added to the semantic repositories and can be reused by business experts.

4.8 Conclusions

Semantics are omnipresent in nowadays enterprises. In future, leveraging on all this semantic information will be crucial for enterprises. We identified the key drivers that will drive enterprises in the coming years. Namely: interoperability, use of standardized technologies, enablement of sense-making and realtime business decision support. All those drivers, especially interoperability, sense making and real-time business support would benefit from semantic service descriptions and Linked Services. We suggested an architecture that allows to integrate all kind of IoT-devices, based upon service descriptions. We argue that a semantically- enriched platform and semantically-enriched Linked Services allow moving away from a device-centric view, to an entity or business centric view. This not only eases the integration process and enables semantic interoperability, it also allows easier modeling. To provide further support to a business modeler, or business domain specialist, we suggest an abstraction named semantic physical business entity that can be used in enterprise architectures and especially in modeling environments. It would enable the business modeler to navigate from entities to services easily. Semantic physical business entities can be used as part modeling BPMN modeling frameworks, for example, the approach of Meyer et al. [267, 265] that has been introduced in Chapter 1. Furthermore, a non-disruptive migration path is critical, because one of the main challenges of innovation for an enterprise software vendor with a huge user base is to cope with the myriad of already existing code. Vendors prefer innovation that has a clear integration path into already existing systems, even for more disruptive technologies. Since most enterprise systems already use service repositories as an integrated part of their SOA environment, the integration of semantic service descriptions would not change the paradigm of how software is written today. Semantic services thus could be added to enterprise software in an incremental manner without the need of disruptive changes. As explained in Chapter 4.1 there is already a lot of semantics in enterprise systems. The GDT in SAP systems can be seen as a kind of ontology. What is needed is to make the implicitly encoded knowledge more explicit. This would not introduce significant changes to nowadays systems. Systems utilizing the business information and reasoning layer could provide additional higher level services based on SLAs, reasoning (e.g. ontology based virtual sensors) or the connection of entities and (virtual) sensor data. This will allow new applications at an enterprise level by leveraging on the semantic repositories filled by the integration platform.

Part III

Implementation, Service Descriptions and Protocols

In the third part of this work we discuss two service descriptions: Linked USDL for IoT and the Open Data Protocol (OData). In Chapter 5 we discuss Linked USDL for IoT. We contribute four new vocabularies to Linked USDL. Each of these vocabularies targets a specific aspect of the Internet of Things. The covered aspects are events, quality of information, technical endpoints and the REST paradigm. Chapter 6 describes our CoAP and OData implementation. We also discuss IoT-related modeling options for OData and compare OData to traditional semantic web technologies, including strategies to merge the two worlds.

Furthermore, in Chapter 7, we discuss our Sleepy Nodes protocol and its implementation. We present a scheduling framework to be used within IoT-systems and three different scheduling strategies: A simple first fit, an exhaustive strategy and a strategy based on the observation that it is possible to combine measurements under certain circumstances.

Chapter 5

Linked USDL for IoT

This chapter presents an IoT-extension [378, 259, 382, 187] for the Linked USDL [76] service description language. We contribute four new vocabularies to Linked USDL to support Internet of Things applications. Each of these vocabularies targets a specific aspect of the Internet of Things. The covered aspects are events, quality of information, technical endpoints and the REST paradigm. We introduce each of these vocabularies in detail. We call the set of vocabularies, in combination with the Linked USDL core , as Linked USDL for IoT[1]. In the context of a semantics-aware enterprise architecture, as described in Chapter 4, Linked USDL for IoT is the service description that is stored on the mote and can be stored in one or more service description repositories. Linked USDL for IoT does not prescribe any technical protocol. It is a bottom-up approach with respect to enterprise IoT integration.

5.1 Introduction

Linked USDL for IoT, similar to Linked USDL [76], follows the Linked Services philosophy [297]. We already elaborated shortly the basic ideas of Linked Services in Chapter 4.4. In simple terms, Linked Services are based on two main principles: publishing (semantically-enhanced) service annotations and creating services that follow Linked Data principles. Linked Data means essentially four things (based on Berners-Lee [50, 42]):

1. Use URIs as names for things [50]
2. Use HTTP so that people can look up those names [50]
3. When someone retrieves information by accessing an URI useful[2] information based on standards should be returned.
4. Include links to other URIs, so that they can discover more things[3] [50]

[1] available online: http://www.iot4bpm.de/usdl4iot.ttl

[2] Berners-Lee does not specify what "useful" means. This has to be seen in the context of the application. The original formulation, which has been generalized here is: When someone looks up a URI, provide useful information, using the standards (RDF, SPARQL) [50] [SIC]

[3] *Things*, in this context, means any information related to the original URI. It does not necessarily

We relax the original principles, without altering their meaning. We do not limit ourselves to (semantic) web standards. For example, instead of HTTP, any RESTful protocol could be used to look up URIs.

Following the Linked Services idea (see also Chapter 4.4) of publishing (semantically-enhanced) service annotations, the service descriptions should describe their semantic and/or technical inputs and outputs, as well as their non-functional properties in terms of (reused or reusable) light-weight vocabularies. In the Internet of Things, as shown in the motivating example in Chapter 4.6.2, the possibility of linking to other URIs is also particularly useful for partitioning service descriptions in a way that less space is consumed on the device, if needed. Only the parts of a service description that are really needed are transmitted.

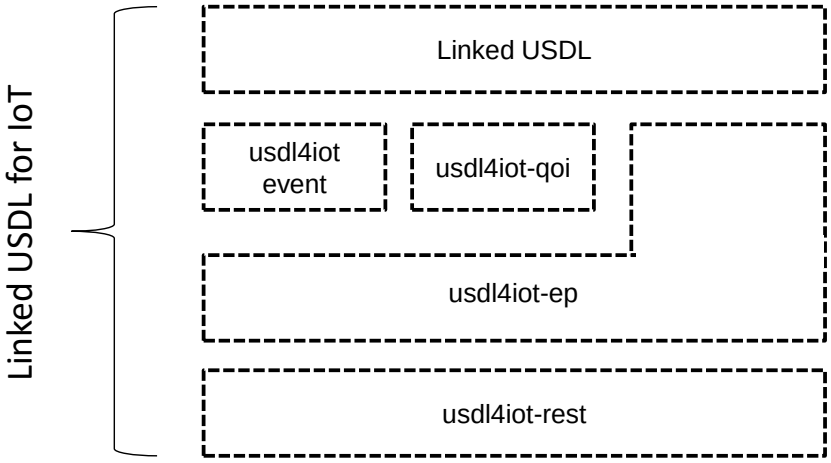


Figure 5.1: Linked USDL for IoT (building blocks)

Linked USDL for IoT, short LUSDL4IoT, is an implementation of the Linked Service idea consisting of the following building blocks (see also Figure 5.1):

Linked USDL Core The Linked USDL core is the base of all semantic modeling in Linked USDL for IoT. It covers the high-level concepts that are central to a service, such as basic service descriptions, service offerings descriptions, business entities involvement, and service delivery communication by defining communication channels and interaction points.

usdl4iot-event The event vocabulary can be used to model a publish-subscriber interface.

usdl4iot-qoi The qoi vocabulary is used to model *Q*uality *o*f *I*nformation (QoI) aspects.

mean Things as in Internet of Things.

usdl4iot-ep The ep (*e*nd*p*oint) vocabulary is used to describe the endpoint of a service.

usdl4iot-rest This vocabulary is used to describe the properties of an endpoint using the RESTful paradigm.

In the following we briefly discuss the application fields each of these vocabularies are used for and also introduce briefly further Linked USDL extensions that could be used in conjunction with Linked USDL for IoT.

Traditionally, web service communication follows a request-response communication scheme. This is sufficient for some applications, but many IoT applications need communication patterns that are more sophisticated. In this work we support three types of communication patterns:

Request/Response (R/S) The service consumer issues a request, which the service providers answers either synchronously or asynchronously.

Publish/Subscribe (P/S) The publish/subscribe pattern allows a consumer to subscribe to particular events. As soon as this specific event occurs, the provider triggers a notification to the consumer. A subscriber-based communication model is a necessary precondition for running parts of business processes on smart objects, as these often need to react to specific events.

Time triggered While it can be argued that time-triggered is just a subclass of P/S in IoT there are many sense and send applications. This specific pattern is so common that it deserves to be a category of its own.

The R/S pattern is supported by Linked USDL. To support events we added a small event vocabulary, called usdl4iot-event. Based on existing studies of event-based systems [174] we identified the following subset of typical P/S operations that we support in usdl-event: (i) register event subscriptions, (ii) remove event subscriptions, (iii) decouple resources from events and (iv) specify means of delivery, for example, callbacks.

Quality of Information is a subset of the more general concepts of Quality of Service and uncertainty of information. In the Internet of Things domain Quality of Information is of special interest when dealing with sensors and actuators. We have only imperfect sensing technologies, and insufficient and/or incomplete contextual knowledge. This means that sensors and actuators inherently have the problem that the information provided or the action performed might not be correct, imprecise or depend on some statistical model. Therefore the uncertainty of the data a service can provide has to be sufficiently described.

In the Internet of Things domain a common approach is to look at two specific Quality of Information aspects: Quality of Sensing and Quality of Actuation. We modeled the vocabulary based on previous work on quality of information parameters, as shown in Table 5.1

Property	Description
Precision Sheikh et al.[348], Buchholz et al.[64]	How exactly the provided context information mirrors the reality[64], Granularity with which context information describes a real world situation [348] *Typical metrics:* Depends on subject, e.g. position precision of a localization in cm
Freshness Sheikh et al.[348], Buchholz et al.[64]	Time that elapses between the determination of context information and its delivery to a requester [348] *Typical metrics:* Time span / Age of data, in seconds
Uncertainty of Information, Probability of correctness, Trust worthiness, Confidence, Certainty Sheikh et al. [348], Buchholz et al.[64], Laskey et al.[228], Gu et al.[146], Loke[239]	Probability that a piece of context information is correct [64], How likely is that the provided information is correct. [64] Probability that an instance of context accurately represents the corresponding real world situation, as assessed by the context source, at the time it was determined [348] *Typical metrics:* percentage values, levels, probability function

Table 5.1: Quality of Information: Overview of Concepts

The four Linked Data principles map very well to the RESTful paradigm [50], which is currently predominant in the web and has great chances of dominating the Internet of Things as well [149]. Linked USDL for IoT was therefore designed with the primary goal of supporting the RESTful paradigm.

The Linked USDL core vocabulary can be extended by the following further domain-specific ontologies [76]. They can be used to extend Linked USDL for IoT.

usdl-price usdl-price can be used to describe price structures in the service industry. According to the maturity classification as described in [76] it is regarded as ready to use and validated.

usdl-agreement usdl-agreement [135] covers functional and non-functional quality of service. It is classified as having past the proof-of-concept phase and is now in validation [76].

usdl-sec This module covers security aspects of a service. Service providers can use it to describe security related features of their services. It is classified as being in the proof-of-concept phase [76].

usdl-ipr This USDL module covers legal aspects of accessing a service, such as access rights and copyrights. It is classified as being in the proof-of-concept phase [76].

5.2 Design Principles

The Linked USDL vocabulary for IoT (USDL4IoT) follows the same lightweight ideas as Linked USDL does. Our vocabulary is developed in an iterative use-case driven approach [283]. The goal of Linked USDL for IoT is to allow publishing and retrieving of IoT-Services in heterogeneous environments and to standardize the common core, i.e. the properties all these services have in common. We aim to capture the most important concepts only. Domain-specific details are delegated to additional ontologies, following one of the core ideas of Linked Services. We specified the communication patterns up to a degree that allows to model typical IoT-service interactions. Nonetheless, it would be possible to extend it to also support full-blown web-services and more complicated interaction schemes.

Considering the original design decisions of Linked USDL [296], and aiming for a becoming a regular member of the Linked USDL family of vocabularies, we identified four principles that guided the design of Linked USDL for IoT: lightweight, coherency, extensibility and compatibility. We choose them in order to ensure a seamless integration of Linked USDL for IoT into the Linked USDL core vocabulary and the already existing extensions. In the following we will briefly describe the four principles and their main properties:

1. **Lightweight**: We wanted to cover the most important aspects only, avoiding the possibility of over-specifying and, therefore, limiting the applicability of the vocabularies. In particular, domain-specific knowledge needs to be specified by (external) ontologies. This is inline with the original design methodology of Linked USDL that was centered on the reuse of widely adopted vocabularies [296].

2. **Coherency**: Our extension needs to integrate well into Linked USDL. Concepts and modeling style should follow the one used in Linked USDL to ensure that it seamlessly integrates into the existing Linked USDL vocabulary.In our opinion introducing a modeling-style that differ from what is used in the core ontology would lower the acceptance of the IoT extension.

3. **Extensibility**: The vocabulary is designed with evolution and extensibility in mind. It needs to integrate well with existing well-known external ontologies. The concepts need to be generic enough to provide extension points for domain-specific vocabularies.

4. **Compatibility**: We aim for compatibility with existing (high-level) concepts and for easy integration into existing systems.

5.3 Linked USDL

Linked USDL is one specific realization of the Linked Services concept [76, 231]. To maximize interoperability, Linked USDL adopts, whenever possible, existing RDF(S) vocabularies such as QUDT [171]. It creates explicit ontological links to domain-specific ontologies. While this is the origin of the name Linked Services, we foresee a lot of potential in actually linking one part of the service description to more detailed information defining the very same service. A brief overview on how Linked USDL for IoT can be embedded into further related vocabularies is shown in Chapter 5.6.

To our knowledge Linked USDL is the only standardization effort driven by large corporations with the goal of expressing not only purely functional aspects of a service, but also the business and operational aspects. A comprehensive introduction into each of these aspects can be found in Barros et al. [29].

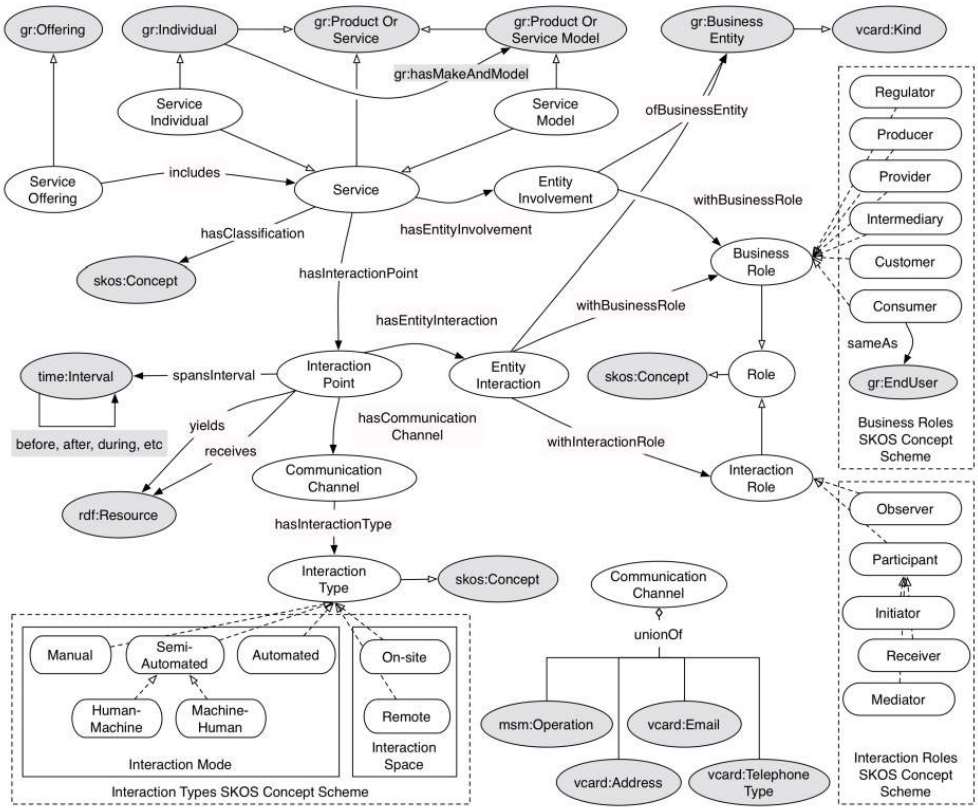


Figure 5.2: Excerpt from the Linked USDL ontology (based upon [296]); see http://www.linked-usdl.org for the complete ontology

An excerpt of the Linked USDL ontology is shown in Figure 5.2. In the following we will introduce the concepts of Linked USDL as specified in [231] and released under

a Creative Commons license.

usdl:Service

A usdl-service describes a service. It is the initial entry point for service consumers. The description contains functional properties of the service, described by the interaction protocol as well as non-functional properties described by qualitative or quantitative values [231].

Listing 5.1: Linked USDL service definition

```
usdl:Service a rdfs:Class, owl:Class;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "Service"@en ;
    rdfs:subClassOf gr:ProductOrService .
```

Linked USDL for IoT bases all of its service descriptions on usdl:service. Every Linked USDL for IoT service is, therefore, also a Linked USDL service.

usdl:ServiceOffering

A ServiceOffering is an offering made by a gr:BusinessEntity to its customers. An offering is part of the business side of Linked USDL. It usually includes a price, legal terms, and service level agreements.

Listing 5.2: Linked USDL Service offering

```
usdl:ServiceOffering a rdfs:Class, owl:Class;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "Service Offering"@en ;
    rdfs:subClassOf gr:Offering .
```

An actual service is added to a particular ServiceOffering using the include property. usdl:ServiceOffering allows specifying bundles of services traded as one entity. It is not meant for creating composite services. All Linked USDL for IoT service offerings are modeled with usdl:ServiceOffering.

Listing 5.3: Linked USDL relationship between service and service offering

```
usdl:includes a rdf:Property;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "includes"@en ;
    rdfs:subPropertyOf gr:includes;
    rdfs:domain usdl:ServiceOffering;
    rdfs:range usdl:Service .
```

usdl:ServiceModel

A ServiceModel is used to model 'classes' of services, which means services that share some characteristics. The property usdl:hasServiceModel is used to connect a service to a service model. A software vendor can use the service model to classify its services into IoT and non-IoT services.

Listing 5.4: Linked USDL Service Model

```
usdl:ServiceModel a rdfs:Class, owl:Class;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "Service Model"@en ;
    rdfs:subClassOf usdl:Service, gr:ProductOrServiceModel .

usdl:hasServiceModel a rdf:Property;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "has service model"@en ;
    rdfs:domain usdl:Service;
    rdfs:range usdl:ServiceModel;
    rdfs:subPropertyOf gr:hasMakeAndModel .
```

usdl:InteractionPoint

When thinking about accessing a service, an InteractionPoint is an actual step in accessing and performing operations of the service. As Linked USDL was designed to represent not just pure technical services, but services of any kind, this could be an interaction between two applications, but it could also be an interaction between two or more humans. On a technical level, this could translate into calling a SOAP Web Service method, or, in the case of Internet of Things, a CoAP endpoint. The usdl-interaction point is, therefore, the attach point for an usdl4-iot:event.

Listing 5.5: Linked USDL InteractionPoint

```
usdl:InteractionPoint a rdfs:Class, owl:Class;
        rdfs:subClassOf usdl:TimeSpanningEntity;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "Interaction Point"@en ;
```

InteractionPoints can have temporal relationships between them. Therefore, they are a subclass of usdl:TimeSpanningEntity. TimeSpanningEntitiy is used to introduce temporal ordering between interaction points, such as precede and succeed relationships.

Listing 5.6: Linked USDL hasInteractionPoint

```
usdl:hasInteractionPoint a rdf:Property;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "has interaction"@en ;
    rdfs:domain [a owl:Class; owl:unionOf (usdl:Service usdl:
        ServiceModel)];
```

```
    rdfs:range usdl:InteractionPoint .
```

The semantical link to the input and output of an interaction point is modeled with the properties usdl:receives and usdl:yields. The range of both of them is rdfs:class, so it is possible to establish a link to an arbitrary domain specific ontology.

Listing 5.7: Linked USDL receives and yields definition

```
usdl:receives a rdf:Property;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "receives"@en ;
    rdfs:domain usdl:InteractionPoint;
    rdfs:range rdfs:Class.

usdl:yields a rdf:Property;
    rdfs:isDefinedBy <http://linked-usdl.org/ns/usdl>;
    rdfs:label "yields"@en ;
    rdfs:domain usdl:InteractionPoint;
    rdfs:range rdfs:Class.
```

The communication taking part at an interaction point is modeled as CommunicationChannel. The term was chosen to be rather generic because Linked USDL was designed to support all kinds of communication. Linked USDL itself does not define any specific CommunicationChannels. In case of human contacts, it could refer to vCard (e.g. email, phone). Operations in Linked USDL for IoT are modeled as communication channels.

5.4 IoT-specific Vocabularies

In the following, we discuss the actual implementation of Linked USDL for the IoT (in short, LinkedUSDL4IoT). We generally distinguish between a semantical and a technical part. The semantical part is mainly used to model the semantic relations, whereas the technical part is mainly used to model the specific parts of a technical endpoint.

5.4.1 Vocabulary: Endpoint and Application-layer Support

We model the supported application-layer protocol as *ApplicationProtocol* class. Protocol layers below the application layer are out of scope, but they could be modeled in an orthogonal way, if needed.

Listing 5.8: Linked USDL for IoT: Application Protocol class

```
usdl4iot-ep:ApplicationProtocol a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "Application Protocol"@en ;
    rdfs:comment "Technical high-level application protocol"@en .
```

We support the two most common REST-based application layer protocols, namely HTTP and CoAP. Our main focus in this thesis was CoAP, hence all the experiments were conducted based on a CoAP application layer. HTTP could be an option for systems based on less constrained devices. As a third option, we foresee the use of custom (REST-like) protocols. We define a separate application protocol endpoint for that case (APCustom). Custom protocols are a subclass of APCustom.

Listing 5.9: Linked USDL for IoT: Application layer protocol definitions

```
usdl4iot-ep:APCoAP a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "CoAP Application Protocol"@en ;
    rdfs:comment "CoAP Application Protocol"@en ;
  rdfs:subClassOf usdl4iot-ep:ApplicationProtocol .

usdl4iot-ep:APHttp a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "HTTP Application Protocol"@en ;
    rdfs:comment "HTTP Application Protocol"@en ;
  rdfs:subClassOf usdl4iot-ep:ApplicationProtocol .

usdl4iot-ep:APCustom a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "Custom Application Protocol"@en ;
    rdfs:comment "Custom Application Protocol"@en ;
  rdfs:subClassOf usdl4iot-ep:ApplicationProtocol .
```

Independent of the underlying application-layer protocol, we will follow the notions of operations and parameters. The abstraction chosen for modeling services can be used for any underlying application-layer protocol that can be mapped to operations with accompanying input and output parameters.

Listing 5.10: Linked USDL for IoT: Application layer protocol definition - Operation

```
usdl4iot-ep:Operation a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "Operation"@en ;
    rdfs:comment "An operation offered by the service, yielding an
         output for a given input"@en ;
  isSubclassOf usdl:CommunicationChannel.

usdl4iot-ep:Parameter a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "Parameter"@en ;
    rdfs:comment "A parameter conceptually describes the input or
        output of an operation"@en .
```

Based on that abstraction, we define the properties that define an operation – namely hasInput, hasOutput, hasType, isOptional, representAn and hasValueLink. The first two further define an operation and its parameter. hasInput connects an operation with a

parameter and marks it as an input parameter; hasOutput works similarly for output parameters. A datatype is attached to a parameter by using hasType. Furthermore, a parameter can be marked as optional.

The remaining two properties *representAn* and *hasValueLink* are used for semantical linking. *representAn* links to the semantic concept a parameter represents. For example, a temperature parameter could have a datatype of xsd:float. All that is known up to this point is that there is a parameter and that it expects or return a float value. To interpret this value at higher level, the *representsAn* semantical link can be used to, for example, link this parameter to a temperature ontology and more specifically to the entity TemperatureCelcius. Now, from a semantic point of view, it is obvious that the parameter represents a temperature in degree Celcius and that it is (technically) encoded as a float value.

Listing 5.11: Linked USDL for IoT: Application layer protocol definitions

```
usdl4iot-ep:hasInput a rdf:Property ;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
  rdfs:domain usdl4iot-ep:Operation ;
  rdfs:range usdl4iot-ep:Parameter .

usdl4iot-ep:hasOutput a rdf:Property ;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
  rdfs:domain usdl4iot-ep:Operation ;
  rdfs:range usdl4iot-ep:Parameter .

usdl4iot-ep:representsAn a rdf:Property;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:comment "Connects a parameter to the high level concept
        it represents"@en ;
  rdfs:domain usdl4iot-ep:Parameter ;
  rdfs:range rdfs:Class .

usdl4iot-ep:hasType a rdf:Property;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:comment "The parameter value's type, e.g. xsd:int for
        integers"@en ;
  rdfs:domain usdl4iot-ep:Parameter ;
  rdfs:range rdfs:resource .

usdl4iot-ep:isOptional a rdf:Property ;
  rdfs:domain usdl4iot-ep:Parameter ;
  dfs:comment "Specifies whether parameter is required for the
      operation"@en ;
  rdfs:range xsd:boolean .

usdl4iot-ep:hasValueLink a rdf:Property ;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
  rdfs:comment "Specifies the parameter's concrete value in the
      output document"@en ;
  rdfs:domain usdl4iot-ep:Parameter ;
```

```
  rdfs:range rdf:Literal .
```

5.4.2 Vocabulary: Event Support

The usdl4iot-event vocabulary is used to describe events and a generic publish/subscribe communication pattern. We decided not to support any further refinement of events, as this is to be defined by the application scenario. For example, an event could be raised if the temperature of a given good is above a given threshold. The vocabulary therefore can get by with two classes: An *Event* and a *Publisher*.

Listing 5.12: Publiser/Event interface in Linked USDL4IoT

```
usdl4iot-event:Event a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-event>;
    rdfs:label "Event"@en ;
    rdfs:comment "Any kind of event occuring within the scope of
        the service"@en .

usdl4iot-event:Publisher a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-event>;
    rdfs:label "Publisher"@en ;
    rdfs:comment "Actor which autonomously generates events"@en .
```

On a semantic level, we attach the publisher to the interaction point of the usdl-core ontology. Each publisher can support an infinite number of events via the hasEvent property.

Listing 5.13: Publiser/Event interface in Linked USDL4IoT

```
usdl4iot-event:hasPublisher a rdf:Property;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-event>;
  rdfs:domain usdl-core:InteractionPoint;
  rdfs:range usdl4iot-event:Publisher.

usdl4iot-event:hasEvent a rdf:Property;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-event>;
  rdfs:domain usdl4iot-event:Publisher;
  rdfs:range usdl4iot-event:Event.
```

On an operational level, we need to connect the events generated and the subscribe and unsubscribe operations to endpoint operations. Each event is connected to a subscribe (*hasSubscribeOperation*) or unsubscribe operation (*hasUnsubscribeOperation*) and can generate one or more output parameters (*generates*).

Listing 5.14: REST verbs in Linked USDL4IoT

```
usdl4iot-event:hasSubscribeOperation a rdf:Property ;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-event>;
  rdfs:domain usdl4iot-event:Event ;
```

```
  rdfs:range usdl4iot-ep:Operation .

usdl4iot-event:hasUnsubscribeOperation a rdf:Property ;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-event>;
  rdfs:domain usdl4iot-event:Event ;
  rdfs:range usdl4iot-ep:Operation .

usdl4iot-event:generates a rdf:Property ;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-event>;
  rdfs:comment "The output generated each time the event occurs"
      @en ;
  rdfs:domain usdl4iot-event:Event ;
  rdfs:range usdl4iot-ep:Parameter .
```

5.4.3 Vocabulary: Quality of Information Support

The Quality of Information (QoI) concept has been introduced in Chapter 5.1. In the spirit of Linked USDL we reuse the agreement vocabulary to implement a small lightweight Quality of Information vocabulary. The Agreement vocabulary is visualized in Figure 5.3. We briefly introduce the main concepts that we build on, based on the original specification of the Linked USDL - Agreement [135].

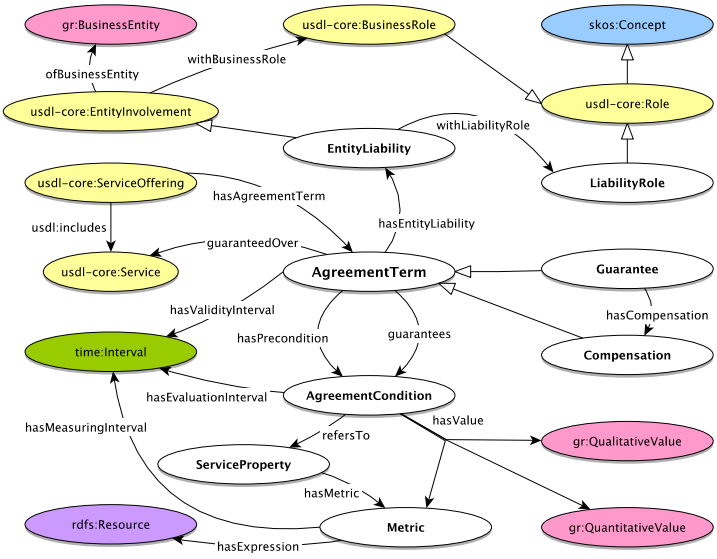


Figure 5.3: Linked USDL Agreement

AgreementTerm A single term of an SLA.

Guarantee An agreement term of an SLA.

AgreementCondition A constraint or axiom within a SLA. It is connected with a concrete ServiceProperty via the *refersTo* association.

ServiceProperty The property of a given service, e.g. freshness.

Metric A metric defines how a ServiceProperty is measured.

We model the following properties that we identified in Chapter 5.1 and summarized in Table 5.1: Precision, Freshness and Certainty.

The three QoI properties are of type ServiceProperty in Linked USDL-Agreement terminology. The properties do not standalone, they typically need to refer to some subject. The QoI property freshness, for example, could relate to the sensing interval of a specific temperature sensor. The integration of the QoI properties into usdl-agreement is visualized in Figure 5.4.

Listing 5.15: Quality of Information in Linked USDL4IoT

```
usdl4iot-qoi:Precision a rdfs:Class, owl:Class
    rdfs:label "Precision"@en ;
    rdfs:comment "Precision QoI property"@en .
  rdfs:subClassOf usdl-agreement:ServiceProperty .

usdl4iot-qoi:Freshness a rdfs:Class, owl:Class
    rdfs:label "Freshness"@en ;
    rdfs:comment "Freshness QoI property"@en .
  rdfs:subClassOf usdl-agreement:ServiceProperty .

usdl4iot-qoi:Certainty a rdfs:Class, owl:Class
    rdfs:label "Certainty"@en ;
    rdfs:comment "Certainty QoI property"@en .
  rdfs:subClassOf usdl-agreement:ServiceProperty .
```

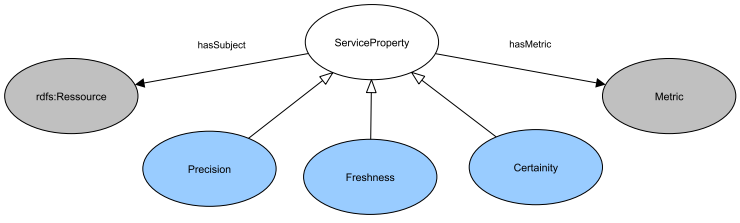


Figure 5.4: Quality of Information ServiceProperties

The metric for each QoI property depends on the subject of the ServiceProperty. Considering precision, for example, Sheikh et al. [348] lists four possibilities:

Boolean The precision can be represented by true or false only.

Numeric Information that can be represented by numerical values, for example the precision of a temperature reading.

Incremental Sets A series of sets, which progressively represent higher information value.

Weighted Sets Sets of discrete values that cannot be arranged in a series of increasing precision. An assigned weight shows the information value of each set.

Given the wide range of possible subjects and their metrics, we follow the Linked USDL principle of delegating them to specialized ontologies. From a SLA point of view, for Internet of Things services, standardization in terms of what the possible QoI properties are is more important than specify each and every detail of its metric. Based upon our initial usdl-qoi more quality of information ServiceProperties could be specified, if needed.

5.4.4 Vocabulary: REST Support

The usdl4iot-rest vocabulary further refines the endpoint specification by mapping it to the RESTful paradigm.

We support REST-based systems using the well-known methods/verbs GET, PUT, POST, DELETE, HEAD, TRACE, and CONNECT. These verbs are supported by HTTP. CoAP supports only the subset GET, PUT, POST and DELETE.

Listing 5.16: REST verbs in Linked USDL4IoT

```
usdl4iot-rest:Method a rdfs:Class, owl:Class
    rdfs:label "Method"@en ;
    rdfs:comment "A RESTful verb"@en .

usdl4iot-rest:POST a rdfs:Class, owl:Class ;
    rdfs:label "POST"@en ;
    rdfs:comment "POST"@en ;
  rdfs:subClassOf usdl4iot-rest:Method .

usdl4iot-rest:PUT a rdfs:Class, owl:Class ;
    rdfs:label "PUT"@en ;
    rdfs:comment "PUT"@en ;
  rdfs:subClassOf usdl4iot-rest:Method .

usdl4iot-rest:GET a rdfs:Class, owl:Class ;
    rdfs:label "GET"@en ;
    rdfs:comment "GET"@en ;
  rdfs:subClassOf usdl4iot-rest:Method .

usdl4iot-rest:DELETE a rdfs:Class, owl:Class ;
    rdfs:label "DELETE"@en ;
    rdfs:comment "DELETE"@en ;
  rdfs:subClassOf usdl4iot-rest:Method .
```

```
usdl4iot-rest:HEAD a rdfs:Class, owl:Class ;
    rdfs:label "HEAD"@en ;
    rdfs:comment "HEAD"@en ;
  rdfs:subClassOf usdl4iot-rest:Method .

usdl4iot-rest:TRACE a rdfs:Class, owl:Class ;
    rdfs:label "TRACE"@en ;
    rdfs:comment "TRACE"@en ;
  rdfs:subClassOf usdl4iot-rest:Method .

usdl4iot-rest:CONNECT a rdfs:Class, owl:Class ;
    rdfs:label "CONNECT"@en ;
    rdfs:comment "CONNECT"@en ;
  rdfs:subClassOf usdl4iot-rest:Method .
```

The uniform interface, as introduced in Chapter 2.4, is defined by URI templates. URI templates (and similar concepts) have been defined early on as part of several specifications, including WSDL [88] and WADL [154]. Subsequently, it has been standardized. We support a simplified subset of URI templates as defined in RFC6570 [144]. URI templates are used for describing Uniform Resource Identifiers through variable expansion. They provide an easy mechanism for abstracting resource identifiers in such a way that the variable parts can be easily identified and described. RFC6570 defines URI templates as follows:

> URI Templates are similar to a macro language with a fixed set of macro definitions: the expression type determines the expansion process. The default expression type is simple string expansion, wherein a single named variable is replaced by its value as a string after pct-encoding any characters not in the set of unreserved URI character. [144]

In usdl4iot-rest, we define two main classes as entry points for describing the uniform interface: URITemplate and URIBinding. The URITemplate represents a pattern to model an URI containing placeholders to be replaced by input parameters. The URIBinding binds a certain placeholder in a URI template to the parameter whose value should replace it.

Listing 5.17: URI templates

```
usdl4iot-rest:URITemplate a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "URI Template"@en ;
    rdfs:comment "A pattern to model an URI containing
        placeholders to be replaced by input parameters"@en.

usdl4iot-rest:URIBinding a rdfs:Class, owl:Class;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-ep>;
    rdfs:label "URI Binding"@en ;
```

```
    rdfs:comment "Binds a certain placeholder in a URI template to
        the parameter whose value should replace it"@en.

usdl4iot-rest:hasPlaceholderName a rdf:Property;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-rest>;
  rdfs:comment "Name of the placeholder to be replaced in the
     template, without the curly brackets'"@en;
  rdfs:domain usdl4iot-rest:URIBinding ;
  rdfs:range rdf:Literal .

usdl4iot-rest:hasPlaceholderParameter a rdf:Property;
  rdfs:isDefinedBy <http://research.sap.com/IoT/usdl4iot-rest>;
  rdfs:comment "Defines the parameter whose value should be used
     for the replacement"@en;
  rdfs:domain usdl4iot-rest:URIBinding ;
  rdfs:range usdl4iot-ep:Parameter .
```

The *usdl4iot-rest:hasPlaceholderName* property binds a placeholder name to a literal and *hasPlaceholderParameter* is used to define the parameter whose value should be used as a replacement for the placeholder (called string expansion [144] in RFC6750 terminology). Therefore, it makes the implicitly available information about placeholders within the URI template explicit. For each of these bindings, (semantic) information about the meaning of the parameter is added.

5.5 Representations

The representation of resources for REST-based and non REST-based systems with and without semantics has been a research topic that received a lot of attention during the last years. Linked USDL4IoT, as a service description language, does not prescribe any specific data representation. Nonetheless, at the time of writing, JSON and its semantic counterpart JSON-LD seems to be the upcoming de-facto standards. JSON is currently used in many new RESTful enterprise developments. Moreover, the need for JSON (JavaScript Object Notation) [60, 114] to be extended to support semantics as well has been identified by the W3C. In 2014 it released a W3C Recommendation called JSON-LD [226].

Linked USDL for IoT can easily be integrated into JSON/JSON-LD, XML-based formats such as WSDL, as well as part of a pure textual description.

JSON-LD was designed so that developers would not need to know about any other semantic web technology [226]. It is, quite close to regular JSON. It is syntactically fully compatible with JSON; hence, all tools and preexisting toolchains can be reused. In the same way as our architecture provided a smooth path towards semantics, JSON-LD aims for a smooth integration into already existing tools. One of the design goals of JSON-LD was to lower the entry barriers for enterprise developers. JSON-LD, like Linked USDL for IoT, is not a complete technology stack. According to the authors (JSON-LD) *needs ontologies to express domain semantics* [226]. LinkedUSDL4IoT

and JSON-LD complement each other, and combined with the IoT-stack and domain-specific ontology form a complete technology stack.

JSON/JSON-LD and Linked USDL for IoT can be integrated in two different ways: First, in the case of only JSON (without -LD) a semantical link from each returned data to the semantics can be established by using the unique *ValueLink*, which can be attached to parameters. The processing system now make the link from the data representation to the service description. This approach can also be followed for JSON-LD. Furthermore, JSON-LD can be used independently by referring to the same ontologies as Linked USDL for IoT, thus giving the same references to the same entities in its response.

To connect WSDL/SOAP with Linked USDL, we leverage on ideas from WSDL-S [6]. WSDL-S based references (modelReference) from a WSDL description to a high level ontology is currently used by most semantic approaches, including the OWL-S [249, 251, 250] and WSMO [122, 322] approaches. OWL-S and WSMO are introduced in Chapter 2.11.2 and Chapter 2.11.3.

WSDL-S Schema definition excerpt based on [6]

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s
     /examples/purchaseOrder.wsdl"
 xmlns:wssem="http://lsdis.cs.uga.edu/projects/meteor-s/wsdl-s/
     examples/purchaseOrder.wsdl"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<attribute name="modelReference" type="anyURI" use="optional"/>
[...]
</schema>
```

Notably, we use WSDL-S and its wssem:modelReference to connect operations and parameters with Linked USDL for IoT endpoints, and encode the same semantic information there. Furthermore, parameters can be mapped to their respective Linked USDL for IoT description by using the *ValueLink* again. In that case, the link between an endpoint and higher level semantics is solely done through usdel4iot-ep:representsAn and, on a service level, the usdl:yields and usdl:receives properties.

To sum up, we now have a vocabulary for specifying the actual means of accessing an IoT-service by specifying the application layer protocol, operations the service can perform, as well as the data encoding. Moreover, links to domain-specific ontologies can be specified. We would like to emphasize here that this description is meant to be used for constrained devices. More complex services that demand features going beyond the current capabilities of our vocabulary, are not in the scope of Linked USDL for IoT. The structure of the vocabulary is designed to be extensible though. We expect it to evolve over time and incorporate many more features known from today's enterprise architectures.

5.6 Relationship to further Ontologies

Linking vocabularies instead of a one-fits-it-all approach is one of the main design goals of Linked USDL. In this section we briefly introduce some ontologies that could be used with Linked USDL for IoT to form a larger service ecosystem. Linked USDL4IoT can be embedded into a more comprehensive framework, in conjunction with other ontologies, to describe the Internet of Things domain fully. Of course, domain specific aspects of an IoT-application are not covered by these and external ontologies are still needed.

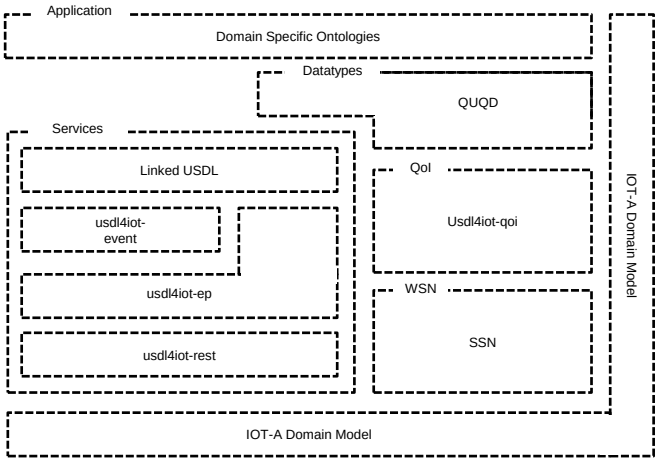


Figure 5.5: Relationship to Further Ontologies

The most prominent ontology in the sensor network field, and one of the few that has reached consensus among many researches, is most likely the Semantic Sensor Network (SSN) ontology[4]. The SSN ontology provides a rather high-level vocabulary to describe sensor network landscapes, that is, sensor nodes and sensor observation and measurement data. It does not provide any service support, generic IoT-enablement nor any support for units or quantities of sensor measurements. The service part is covered by Linked USDL and Its IoT extensions. Considering the Internet of Things one can leverage on the IoT-A domain model[33] and its concepts. It introduces the concept of Physical Entities to describe real-world things, devices and resources connected to it, as well as services as means to access information about physical entities or to manipulate them.

Data interoperability will require the use of standardized ontologies beyond our service vocabularies. Neither SSN, Linked USDL nor Linked USDL4IoT do prescribe any quantitative datatypes. A recent approach driven by NASA and TopQuadrant for standardization of quantities, units, dimensions, and data types is QUDT[5]. It is one of

[4] http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

[5] http://www.qudt.org/

the most comprehensive approaches for standardizing datatypes and quantities, and it has the potential to be the de-facto standard in the coming years.

5.7 Illustrating Example of a Sensor Service Modeled in Linked USDL4IoT

We will now model a specific service for a temperature sensor. The service description is modeled in Linked USDL4IoT. The mote provides a temperature service (temperatureMoteService) and supports a publish/subscribe interface. The technical interface is realized with CoAP (usdl4iot-ep:APCoAP). The REST-endpoint is described by URITemplate and URIBinding. It has the URL /sensor/number, where number is an exemplary placeholder for a sensor. The mote can support different sensors. It returns its data in Celsius (measuredTemperatureParameter).

Listing 5.18: Linked USDL4IoT temperature service description

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dcterms: <http://purl.org/dc/terms/> .
@prefix gr: <http://purl.org/goodrelations/v1#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix usdl-core: <http://www.linked-usdl.org/ns/usdl-core#> .
@prefix : <> .  # Same as base URI
@prefix usdl4iot-ep: <http://research.sap.com/IoT/usdl4iot-ep#> .
@prefix usdl4iot-event: <http://research.sap.com/IoT/usdl4iot-
    event#> .
@prefix usdl4iot-rest: <http://research.sap.com/IoT/usdl4iot-rest
    #> .

:temperatureMoteService a usdl-core:Service;
  usdl-core:hasInteractionPoint :temperatureMoteInteractionPoint;
  usdl4iot-ep:hasEndpoint :temperatureMoteCoAPEndpoint.

:temperatureMoteInteractionPoint a usdl-core:InteractionPoint;
  dcterms:title "Temperature Interaction Point";
  usdl-core:yields :temperatureConcept;
  usdl-core:hasCommunicationChannel :measureTemperatureOperation;
    usdl4iot-event:hasPublisher [a usdl4iot-event:Publisher;
    usdl4iot-event:hasEvent :temperatureChangeEvent].

:temperatureConcept a skos:Concept; #should be replaced with some
    real ontology
  skos:prefLabel "Temperature".

:measureTemperatureOperation a usdl4iot-ep:Operation;
  usdl4iot-ep:hasInput :sensorNumberParameter;
  usdl4iot-ep:hasOutput :measuredTemperatureParameter.

:measuredTemperatureParameter a usdl4iot-ep:Parameter;
```

```
  dcterms:title "Temperature";
  gr:unitOfMeasurement "CEL";
  usdl4iot-ep:representsAn :temperatureConcept;
  usdl4iot-ep:hasDataType xsd:float.

:sensorNumberParameter a usdl4iot-ep:Parameter;
  dcterms:title "Sensor number";
  usdl4iot-ep:hasDataType xsd:int.

:temperatureMoteCoAPEndpoint a usdl4iot-ep:Endpoint;
  dcterms:title "Temperature Endpoint";
  usdl4iot-ep:hasApplicationProtocol [a usdl4iot-ep:APCoAP];
  usdl4iot-ep:hasOperationImplementation :
      measureTemperatureOperationImplementation,
      :subscribeTemperatureOperationImplementation,
      :unsubscribeTemperatureOperationImplementation.

:measureTemperatureOperationImplementation a usdl4iot-ep:
    OperationImplementation;
  usdl4iot-rest:restMethod [a usdl4iot-rest:GET];
  usdl4iot-rest:hasURI :sensorURI
  usdl4iot-rest:implementsOperation :measureTemperatureOperation.

:sensorURI a usdl4iot-rest:URITemplate;
  usdl4iot-rest:hasTemplate "/sensor/{number}";
  usdl4iot-rest:hasBinding :sensorURIBinding.

:sensorURIBinding a usdl4iot-rest:URIBinding;
  usdl4iot-rest:hasPlaceholderName "number";
  usdl4iot-rest:hasReplacementParameter :sensorNumberParameter.

:temperatureChangeEvent a usdl4iot-event:Event;
  dcterms:title "ChangeEvent";
  dcterms:description "The measured temperature value has changed
      ";
  usdl4iot-event:hasSubscribeOperation :
      subscribeTemperatureOperation;
  usdl4iot-event:hasUnsuscribeOperation :
      unsubscribeTemperatureOperation;
  usdl4iot-event:generates :measuredTemperatureParameter.

:subscribeTemperatureOperation a usdl4iot-ep:Operation;
  dcterms:description "Temperature Subscription Operation";
  usdl4iot-ep:hasInput :sensorNumberParameter.

:unsubscribeTemperatureOperation a usdl4iot-ep:Operation;
  dcterms:description "Temperature Unsubscription Operation";
  usdl4iot-ep:hasInput :sensorNumberParameter.

:subscribeTemperatureOperationImplementation a usdl4iot-ep:
    OperationImplementation;
  usdl4iot-ep:followsStandard true;
```

```
    usdl4iot-ep:hasContentType "application/json-ld";
    usdl4iot-rest:hasURI :sensorURI;
    usdl4iot-rest:implementsOperation :subscribeTemperatureOperation
        .

:unsubscribeTemperatureOperationImplementation a usdl4iot-ep:
    OperationImplementation;
    usdl4iot-ep:followsStandard true;
    usdl4iot-rest:hasURI :sensorURI
    usdl4iot-rest:implementsOperation :
        unsubscribeTemperatureOperation.
```

5.8 Conclusions

In this Chapter we introduced Linked USDL for IoT. Linked USDL for IoT is a service description language tailored to the needs of the Internet of Things. It is based on Linked USDL and consists of four different vocabularies: an endpoint and application-layer support vocabulary, event support, quality of information and a REST-layer support vocabulary. They address different aspects of IoT-services. As Linked USDL for IoT is based on Linked USDL, it can use all of its extensions for modeling high-level service properties. Furthermore, as it is currently being actively developed further extensions are likely. Linked USDL for IoT has been designed with RESTful IoT systems in mind, but it can be used for custom projects as well. It is flexible enough to be used in very constrained environments with data representations that just transmit a minimal necessary amount of data.

The advantages of Linked USDL for IoT are manifold: It decouples the technical interface and its service description. It enables interoperability not only at a technical layer, but also on a semantic level. Furthermore, it allows several endpoint technologies and data representations. Compared to, for example, WSDL, it is not tightly coupled to a particular technical stack. It could be used with XML and JSON in the same way. A more in-depth comparison and evaluation of Linked USDL is presented in Chapter 8.5. In the end, not only the technical interfaces form the Internet of Things, but the combination of technical services and (distributed) service descriptions. The fact that descriptions can be distributed has several advantages. First, the service description on the IoT-device can be kept limited to a minimum, if desired. Second, Linked USDL for IoT can be used as an entry point for discovering more information about all entities involved in the service. The service description can therefore be used as part of an enterprise business modeling framework.

Chapter 6

CoAP and OData

In this chapter we contribute two protocol implementations OData [274] and the *Co*nstrained *A*pplication *P*rotocol (CoAP) [353] on very constrained devices. To our knowledge, we are the first that target constrained devices with OData [379]. Furthermore, we contribute a discussion of semantic modeling options for OData enabled systems and relate OData to traditional semantic web technologies. We also discuss lessons learned from using Java on such an embedded platform.

6.1 Introduction

Following our strategy, as outlined in Chapter 1, of looking into a bottom-up and a top-down approach towards the integration of semantics-aware protocols into Enterprise systems, we implemented and evaluated two protocols on the Mote Runner environment: CoAP and OData. CoAP is an IETF standard targeting REST-based systems on constrained devices. A detailed introduction into CoAP is in Chapter 2.5. OData is an enterprise-level REST-based protocol that is semantics-aware. It gained a lot of attention recently and is the de-facto standard related to semantics-aware enterprise protocols.

CoAP can be used as application-layer protocol for the bottom-up approach, since CoAP endpoints can be described with Linked USDL for IoT. We described the bottom-up approach in detail in Chapter 5. Therefore, no further semantic information needs to be provided. Meanwhile, OData is already semantics-aware and it does not necessarily need an external description language.

We use OData to illustrate a possible top-down approach for IoT-devices. To our knowledge we are the first to use OData on such constrained devices. We explain why OData can be considered semantics-aware. We also discuss some implementation decisions that were taken, as well as show usage scenarios and discuss the use of OData in a comprehensive semantics-aware IoT context.

In the following, we first relate the two protocols to our overall strategy of exploring a bottom-up approach and a top-down approach towards semantics-awareness. We then describe our core CoAP implementation and the supported extensions. CoAP is used as

REST-based interface for OData. We briefly introduce OData and then continue with describing its usage on constrained devices.

6.2 CoAP

This section presents a CoAP implementation for a reactive VM-based OS. The implementation is also used as the underlying protocol for the OData implementation that we present in Chapter 6.3. Our codebase was not only used in this thesis, but also as part of the EU PPP FI-WARE. The CoAP codebase developed for the FI-WARE project is available as open source[1]. We focused on the base CoAP specification as outlined in RFC 7252 [353], and two extensions: block-wise transfer [352] and observe [162]. Block-wise transfer provides application layer fragmentation support. The observe option extension allows applications to specifically subscribe to changes in measured data. The CoAP protocol, the block-wise, and observe extensions are introduced in Chapter 2.5.

6.2.1 CoAP on a Reactive VM-based OS

CoAP is a light-weight REST-based protocol that shares some similarities with HTTP. Due to the verbosity of HTTP for constrained devices, CoAP has been suggested and standardized by the IETF[354]. CoAP uses an interaction model similar to HTTP, but typically acts in server and client roles. As CoAP has been specifically designed for constrained environments, it has low overhead and low parsing complexity. Compared to HTTP, it is designed to work with a non-reliable transport layer (UDP) and it does not rely on the transmission control provided by transport-layer protocols such as TCP. A more through description of CoAP, including the message formats and its interaction model, can be found in Chapter 2.5.

Supporting CoAP on a very resource constrained (class 0) devices is a non-trivial task that combined with a reactive VM based operating system poses some interesting challenges:

1. Serving requests should be implemented in an energy-efficient way that permits the motes' battery depletion, as these devices usually operate for long periods of time without recharging. The battery could be drained if the sensors are not started, suspended, and resumed properly.

2. The implementation must guarantee that responses are sent to the client in a timely manner, as timeout periods have to be obeyed. This is not always trivial because motes have limited computing and network capabilities, and a slow request could potentially interfere with newer ones. That leads to the need of supporting concurrent requests from different clients so that they can all experience the same quality of service.

[1] https://github.com/MR-CoAP/CoAP

3. Scaling is one of the main issues, as the overhead generated by the metadata that describes the clients and their requests has to be stored in a limited memory space. Storing and updating metadata on flash memory is inefficient as increases computation time and energy consumption.

In the following, we present our CoAP (Constrained Application Protocol) implementation for the Mote Runner Operating system. First, we give a brief overview of our implementation of the Constrained Application protocol and two of its extensions: block-wise transfer and observe. We then proceed with how redirections can be implemented in a semantical context, before we discuss some of our experiences with Java on very constrained devices.

6.2.2 Implementation

In the following we present our CoAP implementation. It collects real-time information from sensors installed on the mote. To enable communication via the 6LoWPAN protocol among the motes and between the motes and the gateway, we make use of the MRv6 assembly (Mote Runner 13 and later) or 6lowpan assembly (Mote Runner 11) that come shipped with Mote Runner. The assembly is the first to be loaded on the motes during the simulation which also defines which motes acts as a gateway and what kind of topology will be used. A more in-depth description of MRv6 can be found in Chapter 2.3.3.

The functionality of the CoAP implementation is split over five main files:

CoapSocket Main entry point for applications.

Message.java Protocol processing. CoAP messages are encoded and decoded here.

Request.java Request allows abstracting requests from actual CoAP messages.

ObserveRequest.java Abstracts a request with observe option set.

Core.java Provides a basic implementation of the ”well-known” CoRE functionality [280]. It has to be extended by an application with its provided resources.

The main message flow of our CoAP implementation is shown in Figure 6.1. The main interface towards our CoAP service is the CoAP Socket class. It extends the UDP socket interface. All the message decoding is performed within the message class. It contains all the required functionalities for message decoding and encoding. The actual business logic is implemented by a ServiceHandler. As soon as the CoAP service class ensures that a valid packet has arrived, it is delegated to the appropriate service handler. We will now describe the main functionality in detail:

When a client makes a request, the incoming packet is received by the MRv6 or v6lowpan assembly and a Mote Runner application can process it by overriding the onPacket method of the UDPSocket class. The CoapSocket class overrides this

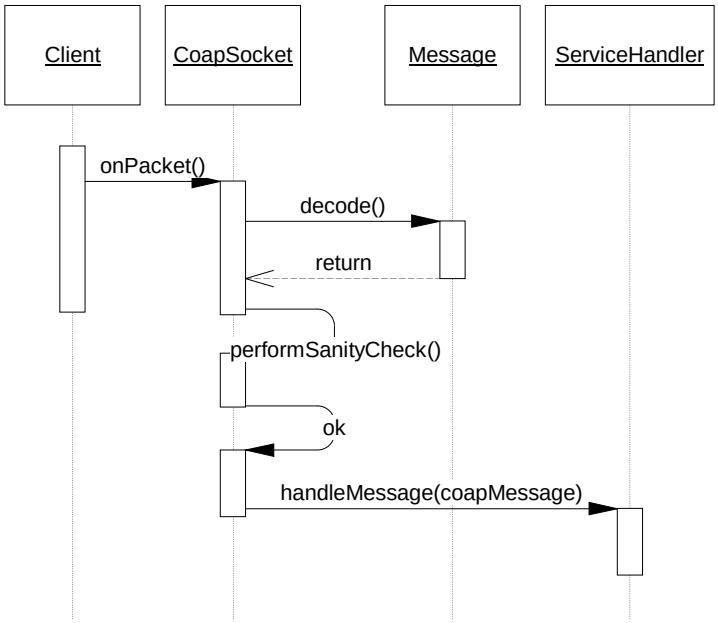


Figure 6.1: CoAP implementation: Core interactions

onPacket method and expects any packet arriving over this particular UDP port to be CoAP-encoded. The actual handling of CoAP packets is application dependent. In the following, we describe the extended packet management as implemented by us. Different applications, for example, on even more constraint devices or when timing and amount of requests is known could implement a different strategy.

The strategy we implemented needs to have a buffer for incoming packets. Packets that are still being processed could cause others to fill the incoming buffer, if processing takes longer than expected or if a burst of incoming traffic occurs. To avoid such delays we opted first to examine the packet about the type of information it carries, for example which sensor data to retrieve and to store a stripped-down version of it in a queue located in the applications heap memory where it remains until the entire response has been sent back to the client. The queue also contains some metadata that identifies the network location of the client and the type of the submitted request. This approach scales, as the number of clients the mote can store information about, is linear to the amount of memory available. As long as there are still packets in the queue, we periodically process packets from the incoming queue. The mote exhibits pseudo-concurrent behavior. It is able to concurrently serve multiple requests of different types and the clients do not have to wait for others to be served first. We also need to keep a queue of already sent CON messages, as these could get lost and may need to be sent again in case of a lost packet.

Most of the actual CoAP processing is performed in the Message class. It provides *decode()* and *encode()* messages for decoding and encoding messages. Parsing the

header is not a complicated task. The main effort of the whole code goes into option handling. Options are delta encoded and, thus, we provided an Iterator-like interface to scan through the options. This interface is tailored towards a use, where the applications scans all options once. Finding an option later is an expensive task, as all the options have to be examined from the beginning. Applications that create a new CoAP message should also add options according to their numbers. Otherwise, a lot of recalculation and memory operations have to be performed which decrease performance significantly.

6.2.3 Redirections

Redirections are not automatically supported in CoAP. To help implementing our distributed descriptions, there are several possibilities:

Reintroducing 3.xx message codes The 3.xx message code group, which in HTTP is responsible for redirections, is not available in the current CoAP draft. The 3.xx message code group was part of earlier drafts, but was removed before standardization. Reintroducing the group solves the problem of redirection, but it would result in a non standards compliant use of CoAP.

CoAP options CoAP options could be used to define an application-layer redirection. They are application-specific, but protocol-compliant. The solution of using CoAP options has the disadvantage of needing application support.

Content type A special CoAP option, content-type, can be used to define an application layer redirection. The type of the content would be x-redirect and it would signal a redirection content. The advantage of this approach is that the content-type can easily be translated to HTTP and, thus, it is usable in any kind of bridging or conversion between CoAP and HTTP.

Data encoding The forth option is to use linking functionality defined in the data representation itself. In case of RDF, for example, the *owl:sameAs* functionality can be used to model that the two URIs actually model the same entity.

The only standard compliant version that does not need application support is encoding the redirections whenever possible within the resulting data. This has the disadvantage that some result has to be delivered. It is not possible to redirect without actually replying to the request. If this is not desirable, we suggest to use application layer redirection as content type. In the long term, reintroducing the 3.xx messages codes into CoAP would be, from our point of view, the optimal solution.

6.2.4 Validation

We validated our implementation with a third party implementation, Californium [218], for automated testing, and Copper [216] as its browser based incarnation. Californium,

written at ETH Zurich, is a cloud-ready implementation of CoAP. Compared to many other implementations it was not written with any constraints in mind. It is written in Java and works only on rather powerful devices. Copper is a browser-based CoAP client. It offers easy testing with an interface that can be used by human users. Californium comes with example applications that also test exceptional states and illegal message exchanges. Californium participated in several ETSI plug-tests [232, 84] which ensures conformability with standards. We used Californium for (automatic) interoperability testing using its plug-test code to test the implemented features.

6.2.5 Embedded Java: Experiences and Lessons Learned

The CoAP code was completely written in the Mote Runner Java dialect. The main goal of using Java on an embedded platform is to ease development of embedded applications on a memory-managed high level embedded platform. Typically, the development was quite straight forward. We didn't experience a steep learning curve that some report for other competing embedded platforms like TinyOS and its nesC language [365, 384, 387]. Anyone knowing the basic fundamentals of the Java programming language was able to write applications within hours. The Mote Runner environment proofed to be easy to use for researchers and developers proficient in the Java programming language and, thus, confirmed our initial assumption of its easy applicability within an enterprise environment.

While developing a more bit-and-byte oriented protocol, such as CoAP, we encountered some unexpected difficulties: Java does not provide unsigned data types. This is a particular problem when translating a specification based on unsigned datatypes to such a platform. This caused some readability problems and made bit-shifting operations more complicated than necessary. Many bugs in our initial implementation were solely due to such conversion errors. For future versions of embedded Java platforms, we would suggest adding at least unsigned versions of bytes and integers.

Furthermore, we experienced some out-of-memory situations. As memory is not explicitly allocated in Java, the feeling for how much memory is allocated got sometimes lost. For performance reasons the Mote Runner documentation suggests reusing objects whenever possible.

6.3 OData

In the following we introduce a top-down approach for getting a semantics-aware protocol on embedded IoT-devices. We did all our experimental research with OData Protocol Version 3.0 (short: OData v3), and we will henceforth refer to that version as OData. We will also explain (semantics-related) advancements of the more recent OData Protocol Version 4.0 (short: OData v4) and focus on how it could improve the current system where appropriate. First, we briefly introduce OData and its growing importance in the market, explaining why OData is a good candidate for a semantics-

aware protocol. In the following, we first give a brief introduction into OData in Chapter 6.3.1, before we will present our implementation in Chapter 6.3.2. In Chapter 6.3.3, we show how OData and semantics can be combined in an IoT-context. We conclude, in Chapter 6.3.3.2, by analyzing the differences between OData and the Resource Description Framework (RDF).

6.3.1 Introduction

Recent advances in the typical protocol stack of Wireless Sensor Networks (WSNs), particularly the use of IP technology, and the demand of businesses for real-time monitoring and real-time decision support has increased the need for enterprise systems to interoperate directly with wireless sensor nodes. One of the major benefits of 6LoWPAN based networking is the use of standard technologies and common and well-understood architectures to integrate smart objects into enterprise systems.

In a typical 6LoWPAN-based [351] WSN protocol stack, as shown in Chapter 1.4, there is a trend towards applying existing application-level protocols and paradigms. In a (networked) enterprise architecture as it exists today, one can observe two main paradigms: (Web-) services, like SOAP and standardized by a variety of standards known as WS-*, and REpresentational State Transfer (REST)[127]. The REST paradigm is explained in Chapter 2.4. REST architectures have become particularly important in Internet of Things applications, as sensors and actuators can often be naturally represented as resources identified by URIs. Both protocols, CoAP and OData, follow the REST-paradigm.

OData [274] is a data access protocol based on widely-used technologies (HTTP, AtomPub and JSON). Traditionally, either custom interface were offered or SOAP-based services. Later on, with the rise of Internet, often custom REST-based APIs were added. A typical SAP ERP system, for example, offers the following interfaces towards an user:

Business Application Programming Interface (BAPI) BAPI is a proprietary protocol for accessing business objects within an SAP system. They represent the way of exposing services in the early, non standardized, days of computing where every vendor had its own way of accessing data and interoperability was mainly achieved through custom built adapters

Enterprise Services Enterprise services are SOAP/WSDL-based web services. They reflect the second wave of technology introduced in the early 2000s

REST-based services REST-based services were introduced in late 2011 and allow interoperability on a semantic level based on Web technologies.

OData, compared to the formerly predominant SOAP services, follows a REST-based approach, aims for semantic interoperability, and follows a more lightweight approach than traditional XML-based web services. Recognizing the growing demand

for exposing services to customers and partners as part of a networked industry, many IT vendors have extended their offerings with OData support. In general, one observes a trend towards OData in many IT organizations. SAP, for example, makes its whole Business Suite available through OData additionally to already existing APIs aiming towards making OData the sole standard for all of its products. Windows Azure, the cloud platform of Microsoft, can be accessed through OData [205, 219] as well as their ERP offering Dynamics [39] among many others. Other than these examples, showing the acceptance of OData in the ERP and the cloud sector, many more offerings available that can produce and consume OData [367].

The OData protocol was not designed to work with constrained devices in the first place. It was originally designed to work with more heavy-weight clients, like in communicating enterprise systems or communication between an enterprise system and a mobile phone. The OData standard defines two ways for data representation (sometimes called serialization), from which one is rather heavyweight: ATOM/AtomPub [282] and JSON [98]. The two formats are used to represent the result of a service call. They are also used to model the service itself, using the common schema definition language. A more detailed overview into the data representation in OData is given in Chapter 2.6.4.

Semantic modeling in OData follows a close-world assumption, where everything is supposed to be known. Its entities (the objects to be modeled) and the relations are stored in the Entity Data Model (EDM). In the case of a temperate sensor, for example, an OData model would need to have the entities for sensors and temperature with a specialization of TemperatureInCelcsius. Nonetheless, the entity model can be shared among parties.

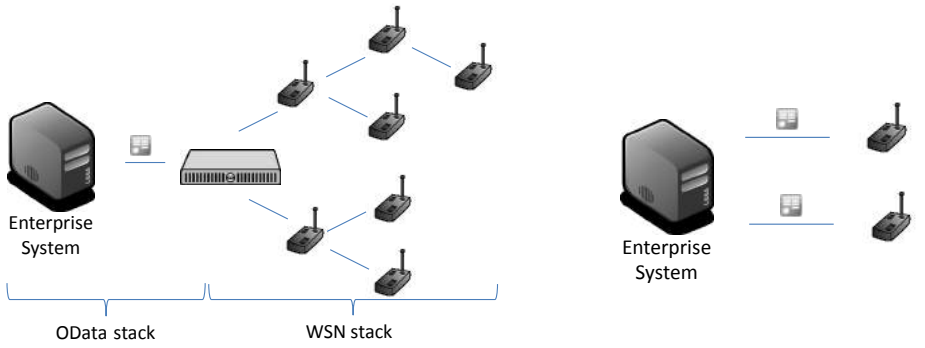
The general concept shares some similarities with the Semantic Web our definition of Linked Services. Concepts for mapping OData to the (RDF-based) semantic web exist. The data models can be described in EDM terms using the conceptual schema definition language (CSDL) [271]. A more detailed introduction into OData, with a broader overview of its data model can be found in Chapter 2.6.

6.3.2 OData on Wireless Sensor Motes

In the following we will, in Chapter 6.3.2.1, first briefly elaborate on possible deployment options and then continue with discussing data representation in Chapter 6.3.2.2, followed by the protocol stack in Chapter 6.3.2.3.

6.3.2.1 Deployment Options

For exposing single IoT-devices or whole sensor networks towards enterprise systems with OData several deployment options are possible. The two most common options are illustrated in Figure 6.2. In this section, we will discuss the deployment considerations of implementing an OData-enabled system.



(a) Intermediate OData Gateway

(b) Direct access to single motes

Figure 6.2: Deployment Options

The first option is illustrated in Figure 6.2a: OData can be used to access whole sensor networks (Entity Model), with a gateway as entry point. The sensor network itself communicates internally with a different protocol stack that is typically tailored to low energy consumption or low latency. The application logic for making requests and setting up the sensor network to satisfy a request resides in the gateway. The OData gateway can be either a sensor node itself, or a platform with more computing power.

The second option is shown in Figure 6.2b. The enterprise system interacts directly with single motes over the OData protocol. This pattern is usually applied when single-board sensor platforms or embedded devices are used. We will consider only this approach, as one of the main visions of the future Internet of Things is to be able to communicate with and directly address independent motes. The first approach is no different from any other gateway or proxy approach, as the sensor network is completely independent from the enterprise communication. We will not explore this option further. Instead, we will only consider only scenarios where the motes themselves communicate through OData.

6.3.2.2 Data Representation

OData supports two representations: XML (as in ATOM) and JSON. An introduction to the OData data representation is given in Chapter 2.6.4.

We investigated both options, as the current standards up to OData v3 require that OData services must support the ATOM encoding and should support the JSON encoding. The upcoming v4 of the standard loosens those in favor of more lightweight clients, as it specifies that OData service must support at least one of the two.

In our experiments we included a compressed version of both XML and JSON. We opted to use general purpose compression algorithms. Previous research at the CDS working group by Dolfus and Braun [110] showed that standard compression algorithms have a quite decent compression performance, even compared to specialized

algorithms. A comparison of compression results, memory, and CPU resource consumption that motivated our work, as well as a thorough discussion of the feasibility of different compression approaches on very constrained devices are given in Chapter 2.8. Furthermore, implementations are widely available on all platforms for Class 0 devices. We used a standard implementation that uses flash memory for paging and caching, thus circumventing the problem of limited RAM. The data was generated when sensing and cached, and in case compression did not yield to better results the original was used. In cases where nothing, for example, in a service description, or only parts of the compressed data changes information can be pre-computed. The data dictionary can be pre-computed up to the point where the data changes and is stored on the mote to increase compression speed.

The use of EXI for very constrained platforms was considered, but its feasibility remains unclear. Reported memory consumptions for an embedded EXI implementation [78] are higher than for LZW-based formats. To our knowledge, only one implementation on an 8kb mote exists. Caputo et al. [68] describe such a platform for Contiki and 8kb platforms, nonetheless without the support of EXI schema encoding or any decoding. No further information on compression restrictions or about the energy or memory consumption is given.

6.3.2.3 IoT OData Stack

OData was originally designed to work with HTTP. Nonetheless, every HTTP-like CRUD-based protocol is a suitable option for OData, as long as it can be mapped to HTTP. A CoAP to HTTP mapping has been demonstrated by Castellani et al. [79]. In this work, we will base our implementation on CoAP as a protocol to communicate with the motes directly. HTTP could be used in an Enterprise system for communication with an proxy.

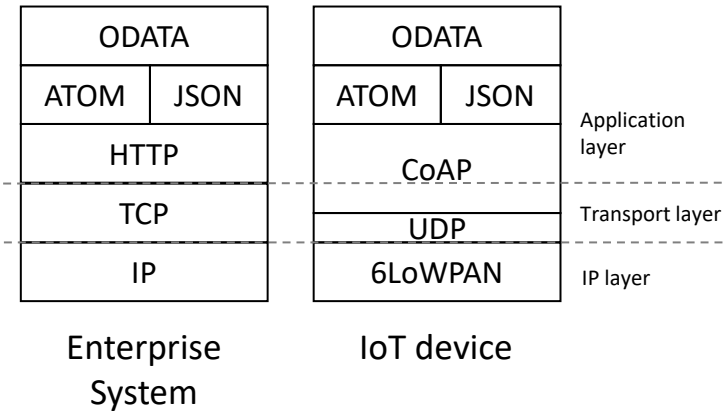


Figure 6.3: OData stack on enterprise system vs. OData stack on mote

In Figure 6.3, the standard OData stack is compared with our CoAP-based stack. The network layer is in both cases IP-based: IPv4/IPv6 vs. 6LoWPAN in the Internet of Things. The transport layer in a typical enterprise OData stack is the TCP protocol, whereas in our IoT-Stack it is a combination of UDP for transport and (parts of) CoAP for transmission control. On top of that, there is HTTP or CoAP, respectively. Considering the traditional ISO/OSI stack, CoAP can be seen as a cross-layer protocol providing more or less the same functionality and also incorporates elements of TCP (transmission control, message deduplication). On top of HTTP/CoAP the stack is identical for enterprise systems and smart objects. Data is transported in either ATOM/XML or JSON format while the data and resource handling is done as specified by the OData protocol itself. An OData query in CoAP notation looks as follows:

$$\underbrace{\text{coap://}}_{\text{protocol}}\underbrace{\text{[a:b:c:d:e:f]/service.svc}}_{\text{IPv6 address of mote and service}}\underbrace{\text{/sensors/temperature?}}_{\text{resource}}\underbrace{\text{\$filter=temp gt 20}}_{\text{query}}$$

In our implementation, instead of HTTP, we use the CoAP protocol. We directly address the mote through its IPv6 address which is then routed over 6LoWPAN. The service caller is unaware that its request goes to a mote. All other parts of the request stay the same. The only limitation is that each request needs to fit into one CoAP and, thus, one UDP packet. Responses can be larger and are transmitted block-wise.

6.3.3 OData and Semantics

In the following, we first introduce how OData and semantics can be combined in an IoT-context. An overview of the general modeling approach of OData is given in Chapter 2.6.4. We then compare RDF and OData from different viewpoints. First, we give a brief overview on the differences between RDF and OData from a conceptual and from a technical point of view. We then continue with discussing first approaches for integration of OData and RDF that are currently being discussed in the OData community.

6.3.3.1 Semantic Annotations in an IoT-Context

OData, especially OData v3, is considered by some as ”semi-semantical”. Real-world implementations often depend on additional knowledge that is stored and implemented in the enterprise system. As described in Chapter 8.7, additional knowledge about the returned values need to be provided by the system. OData allows to model return values with EDM native types or by introducing a type system. In addition, it allows the use of annotations. The annotations can be used to decorate the model and allow the OData system to specify additional information about the value. The W3C open Data on the Web group uses them to map [4] OData on RDF types.

The number of vocabularies and support of semantic annotations was very limited before OData v3. Therefore, the term ”semi-semantical” could be considered as justified.

Before the introduction of annotations some vendors, for example, SAP had their own semantic extensions to express business logic.

Listing 6.1: Annotations: OData vendor specific extensions

```
<Property Name="temperature" Type="Edm.Int16" sap:unit="Celsius"
    />
```

In OData v3, the concept of annotations was introduced. They were, similar to the original SAP concept, unstructured information to store any kind of annotations, including semantics. As shown in Listing 6.2, terms and strings are used to express semantic relationships. The annotation is added to the description of the property.

Listing 6.2: Annotations: OData v3

```
<property name="temperature" type="Edm.Int16">
  <ValueAnnotation Term="Measures.Unit.Temperature" String="
      Celsius">
</property>
```

OData v4 enhanced the concept of annotations and made them first-class citizens in the OData world. OData allows annotations both on a metadata level (Metadata annotations), as well as on an instance level (Instance annotations). Metadata annotations are valid for the whole service. They can be attached, for example, to a property. Instance annotations are used for just that instance of the return set.

Listing 6.3: Annotations: OData v4

```
<Property Name="temperature" Type="Edm.Int16">
  <Annotation Term="Measures.Unit.Celsius" />
</Property>
```

Different modeling options are possible. For example, the above could also be implemented as a string.

Listing 6.4: Annotations: OData v4 - as string

```
<Property Name="temperature" Type="Edm.Int16">
  <Annotation Term="Measures.Unit" String="Celsius" />
</Property>
```

In addition, the annotations can be made available via hyperlinks. This means that even the annotations and the semantic information can be made available as part of a HATEOS-driven engine.

Listing 6.5: Annotations: OData v4 - as path

```
<Property Name="temperature" Type="Edm.Int16">
  <Annotation Term="Measures.Unit" Path="sensorUnit" />
</Property>
```

```
<Property Name="sensorUnit" Type="Edm.String" />
```

Of course, different, and more complicated vocabularies can be build with annotations. A comprehensive introduction is given in the OData v3 [271] and OData v4 [286] specifications.

With OData v4 [287] more vocabularies are introduced, which are interesting for IoT: for example, a core vocabulary that can be used to annotate the capabilities of the OData service. This is especially relevant for OData services that are used in the Internet of Things, as they allow specifying what a service running on a constrained device can actually do. For example, some capabilities of OData (i.-e. some queries) might not be supported or might be supported only in a limited scope. OData v4 also supports a measures vocabulary. The measures vocabulary is used to describe amounts and measured quantities. Obviously, this vocabulary is of special interest in the IoT domain.

To sum up, prior to OData v3, vendor-specific extensions had to be used. Only starting with OData v3 did the protocol have capabilities to describe services semantically. OData v3 already supports relating things with each other (as RDF does) and querying data (as SPARQL) does. It lacks (standardized) vocabularies though and offers limited support of annotations. The situation changed with OData v4. It provides more extensive support of annotations and started to introduce vocabularies. As soon as OData v4 use becomes more widespread and if some standard vocabularies evolve, it could become the enterprise standard for semantically-enriched data services.

6.3.3.2 OData and RDF: A Comparison

We now briefly compare the properties of OData and RDF. Of course, they were created from different backgrounds and with different underlying goals. Nonetheless, they share some key similarities which we will briefly discuss. An overview of the different properties of OData and RDF and their similarities and differences is summarized in Table 6.1. Understanding the differences is important to be able to compare Linked USDL for IoT and OData in a larger context. Linked USDL and Linked USDL for IoT are typical representatives of the RDF-based service description family.

OData and RDF share the same conceptual model based on an entity-attribute-value graph data model. OData prescribes a data representation format, while RDF does not. Typically, RDF is represented as N3, Turtle or more recently JSON-LD. As discussed in Chapter 6.3.3.1, OData previously lacked – to some degree – semantic modeling capabilities. One of the main issues interoperability faces is the availability of standardized vocabularies. Nonetheless, the number of vocabularies is increasing.

Property	RDF	OData
Model	Entity-Attribute-Value Graph	Entity-Attribute-Value Graph
Main data representation format(s)	Non mandatory: Mainly triples (N3), Turlte, JSON-LD	ATOM (XML), JSON
Operations	CRUD	CRUD
Data storage	Not mandatory, but RDF and the development of triple stores are closely connected. Commonly, triple stores are the backbone of RDF implementations.	None mandatory, OData was designed to be seen as an abstract interface to any type of data. OData coined the term
Extensibility	Open world assumption	Closed-world assumption
Semantics	Semantic Annotations (v3,v4), vendor specific (v2)	inherent
Vocabularies	many	few
Standardization	W3C	OASIS

Table 6.1: OData vs. RDF comparison

6.3.3.3 OData and RDF/SPARQL

An early approach for integration of OData and RDF-encoded knowledge was presented by Microsoft Research and DERI. They expose data stored within a RDF triple store through OData [163]. A generalized RDF to OData converter was developed. The basic schematics of the architecture is shown in Figure 6.4. The converter accesses the RDF data through a SPARQL interface and maps it to OData. The mapping configuration was stored in user-defined rule sets and not automated.

The rule-set generates both data-feeds for the actual data (services accesses) and the metadata that describes the service and the semantic information based on the original RDF triple store.

Further research at SAP is currently underway that aims for integration of OData into RDF. Kirchhoff and Geihs [203] propose a semantic extension of the Service Metadata Document to make OData available for integration with the semantic web. They defined a mapping from the Entity Data Model (EDM), which is described with CSDL, to RDF graphs. Similarity to the Microsoft Research approach, but more advanced, templates are used for mapping.

For querying OData with RDF technologies Kirchhoff and Geihs also present a SPARQL interface for OData [204]. The general architecture of their proposed system is shown in Figure 6.5. A client issues SPARQL queries to the system. The Query Engine receives the request and evaluates it using the templates that are stored in their service registry. In case the data exposed by one of the services is relevant for

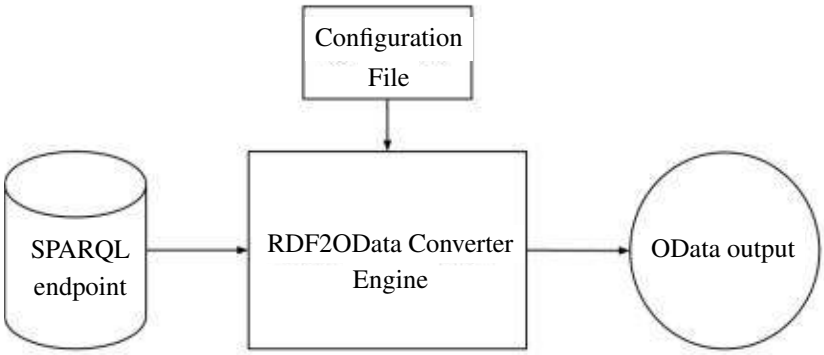


Figure 6.4: Overview of RDF2OData architecture [163]

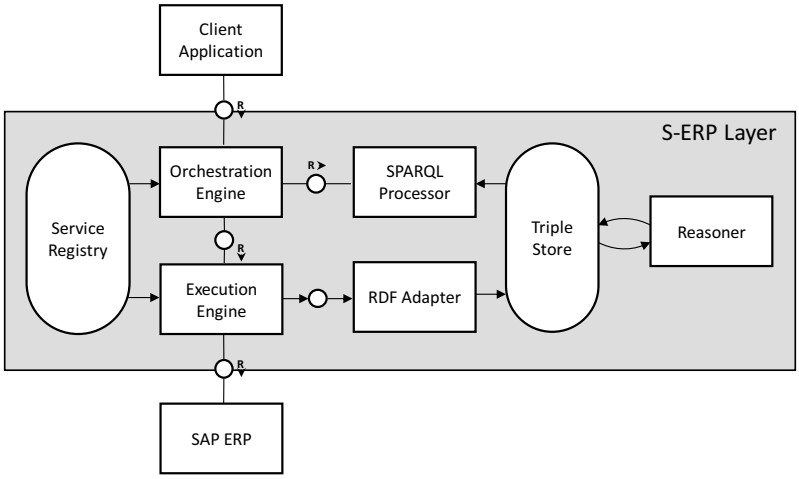


Figure 6.5: SPARQL-OData-Layer architecture [204]

the successful evaluation of the query, the Query Engine creates an URI that is then forwarded to the execution engine. The execution engine performs the actual call to the OData service. The OData service responds with ATOM or JSON documents that are transformed into RDF graphs (RDF Adapter). The RDF graphs can be used in rule-based reasoning engines (Reasoner). A SPARQL processing engine executes the SPARQL query against the data stored in the semantic data storage (triple store). The final result is passed to the Query Engine which returns it to the client. The complete architecture and its logical foundations can be found in [202].

6.4 Conclusions

In Chapter 1.3, we introduced our general idea of investigating a bottom-up and a top-down approach. We introduced a CoAP implementation and an OData implementation for constrained devices. Furthermore, we discussed several properties of OData and related them to the needs of a semantics-aware enterprise utilizing IoT-devices. CoAP in conjunction with Linked USDL can be used to implement a bottom-up approach. OData, coming from the enterprise-world, can be seen as an top-down approach.

We presented our implementation of the Constrained Application Protocol (CoAP) on a reactive VM-based operating system (see Chapter 2.2). We implemented the base CoAP specification, as outlined in RFC 7252 [353] and two extensions: block-wise transfer [352] and observe [162]. Implementing the CoAP protocol on a Class 0 [54] (see Chapter 1.4) device exposes some specific challenges. First, we deal with very constrained devices both in computing capabilities and memory. Especially, a memory-aware implementation of the options as defined by CoAP in its latest versions, needed some effort to reduce the number of processing steps without allocating too much memory. Previous versions of CoAP used linearly ordered options, which are way easier to handle on-mote, but consumed more bytes when transferred. Here the design of the CoAP protocol should be easier to implement in Class 1 (and above) motes. CoAP implementations for Class 0 devices should refrain from using too many options.

In Chapter 4.7 we already mentioned the need of redirections to support linking parts of a service description together. CoAP does not have this property at the moment. We therefore discussed four possible solutions: 3.xx message codes, CoAP options, Content Type and Data encoding. We concluded that, while the 3.xx family might be the most desirable, currently the most promising solution is to move the problem to the application itself. In our opinion this is a gap in the standard that needs to be addressed in future CoAP releases.

We implemented all of our work in Java. Using Java was quite an advantage as every team member was able to start after a minimal ramp-up phase. While implementing a protocol like CoAP some downsides emerged: The CoAP specification, like most protocol specifications, assumes a C-like 8bit unsigned byte datatype or, more generally, the availability of unsigned datatypes. It was therefore not straightforward to implement the specification and lead to some hard to track errors and unnecessary calculations and type casts. We believe an embedded version of Java should support unsigned datatypes to ease implementation of such protocols.

We introduced an IoT stack for OData. Instead of using HTTP, the more recent CoAP protocol can be used. OData was used as a representative of a new class of service description languages. It follows a different approach than RDF-based service description languages, such as Linked USDL for IoT. It combines semantics-awareness with a RESTful protocol that does not need external semantic annotations or extensions. Linked USDL for IoT separates the service description from the representation. There is

no representation prescribed and minimal data representations can be used. Furthermore, Linked USDL for IoT is better integrated into the semantic world, leveraging on semantic technologies and linking well known vocabularies. The advantage of OData is that it can be easily integrated into Enterprise systems without the need for any adoptions. It is standardized and used by a large number of corporations. Existing tooling can be reused, as well as programming frameworks and libraries. OData follows a close world assumption. Semantic-annotations are possible and likely to be further extended in upcoming OData versions. The amount of vocabularies will most likely increase as well. OData does not need to stay in its ”silo”. OData and RDF-based semantics do not need to be mutually exclusive. First approaches for integrating OData with RDF exist.

OData responses are typically rather large, at least compared to what has been used in the sensor network community so far. Compression helps reducing transmission times and, thus, total energy consumption. As far as compression is concerned, based upon previous research as discussed in Chapter 2.8, we opted for a standard compression scheme. These have shown to work reasonably well compared to more specialized compression algorithms. They work for both ATOM and JSON representations, as well as any kind of data that might be collected by the system. Furthermore, they have shown low computational and memory overhead, and, thus, are suitable for very constrained devices. Moving away from very constrained devices to a platform with more memory and computational power, the choice might have to be reconsidered.

Interoperability with enterprise systems is one of main arguments for using OData. On the downside, one has to stay within the ”closed world” of OData. OData currently offers less semantic modeling abilities and was coined ”semi-semantic” by some people, because of its lack of integration into the semantic web. This seems to be justified for OData v2, but less justified for OData v3 and with OData v4, semantic annotations are fully supported. The situation on standard vocabularies also changes currently. OData v4 already has some standardized vocabularies, for example, for core capabilities and measurements. Given the standardization power of the OData supporters within OASIS, the OData community could soon have its own set of standard vocabularies. This would enhances OData’s interoperability capabilities and, at some point, might enable the breakthrough of semantic interoperability in enterprises.

Chapter 7

REST Sleepy Node Integration

Sleepy Nodes, in an IETF context [311], are a recent research area that aim to not only use energy-aware protocols on the networking, routing, or hardware layer but also take the application layer into account [311, 312]. Towards a user, it has the unique property that it always needs to appear as connected to the network. We contribute to sleepy nodes' research by introducing key components needed to utilize them in a semantics-aware enterprise [372]. First, we present a light-weight REST-based sleepy node protocol. Second, we describe a measurement framework that implements different strategies to map sensing requests to actual nodes. The general goal of the framework is to maximize the network lifetime by explicitly sending nodes to sleep based on available information about the applications properties. It tries to combine requests that can be served together, without violating deadlines. We compare an optimal strategy, based on exhaustive search, a random first-fit, and a heuristic that is based on the observation that subsequent requests are more likely to be combined than those that not follow each other time-wise. We call this heuristic dynamic partitioning. Third, we present an energy model that is tailored towards the needs of such a measurement framework. It enables a scheduler to decide on sleeping times based on the additional energy needed to serve requests, the time and energy needed to change to the sleep state, and the estimated energy savings due to sleeping. Furthermore, some external support (e.g. by a platform) is necessary for sleepy nodes, because, by definition, they are required to appear as connected to the network. We show that our measurement framework integrates very well in our architecture as presented in Chapter 4. Last, we discuss how mid- to longterm sleeping can be integrated into an existing hardware platform. We extend the 6LoWPAN protocol of the Mote Runner system, considering the limitations of imprecise clocks. In this context, we argue about the advantages of a modern platforms with more advanced deep sleep modes and very precise hardware clocks.

7.1 Introduction

In addition to service-based interoperability one of the main issues in industrial use of small battery-powered devices is energy saving. The integration of enterprise systems and Internet-of-Things (IoT) devices based on semantically enriched protocols and services enables IoT devices to act according to additional knowledge available on a semantic or application layer.

In the past, the problem of energy saving on the protocol-level [59], routing [178, 345], in the MAC layer [13, 400, 385], or on the actual hardware [309, 304, 300] has been studied extensively. While the MAC-layer or routing support for sleepy nodes (or sleep states) performs quite well, these functionalities are typically unaware of any timing or operational aspects of the application layer and, therefore, cannot leverage on that.

In the following, we introduce an application-layer REST sleepy node implementation and measurement framework for saving energy [372]. We define, based upon the IETF definition [311], a *sleepy node* as follows:

> A sleepy node is a node that may sometimes go into a sleep mode to save energy and that temporarily suspends all protocol communication. A sleepy node will otherwise remain in a fully powered-on state where it has the capability to perform any protocol communication. To the user, the sleepy node always appears as connected, thus being part of the network.

Sleepy nodes that perform mid- to longterm sleeping can be used whenever these sleeping times can be calculated by the application layer. In the following, we will look into a monitoring framework with soft real-time constraints. Usually, the design space of real-time applications is divided into three disjunctive classes [356]:

Hard real-time: A deadline is called hard, if the consequence of not meeting it is a failure and there are potentially severe consequences.

Firm real-time: A deadline is called firm, if the data produced ceases to be useful as soon as the deadline expires, but the consequences of not meeting the deadline are not severe.

Soft real-time: A deadline is called soft, if it is not hard or firm and the utility of the results decreases over time after the deadline expires. Compared to a hard real-time system, in a soft real-time system the miss of a deadline has no catastrophic consequences. The data produces is still to some degree relevant, even if the deadline has expired.

We are solely considering *near real-time* IoT systems [289] with soft quality of service requirements. Most enterprise applications that are part of the IoT have soft real-time requirements. In many cases, the data has some value even when it is late. Hard

real-time applications are not considered in this work. The delays and unpredictability of over-the-air transmissions make hard real-time difficult to achieve [83], if the acceptable deadlines are small. Körber et al. [213] and Lill/Sikora [237] report success under ideal conditions in the 5ms to 8ms hard-real requirements. An overview by Paavola et al. [290] concludes that current wireless solutions, such as IEEE 802.11, IEEE 802.15.1, IEEE 802.15.4 and ZigBee are inapplicable for hard real-time applications with trigger times around 5ms. Nonetheless, such applications can rarely be considered as part of the Internet of Things. Moreover, it is even more unlikely that they can profit from mid- to long range sleeping.

We will build on the application scenario as introduced in Chapter 4.6.2. The implementation is the monitoring of goods throughout the supply chain. For example, a container may be monitored very infrequently while being transported in a cargo ship, more frequently while staying at a harbor and again less frequently while being transported in a truck.

In the following, we first present the application REST API that can be used to put a node to sleep. To comply fully with the definition of a sleepy node, we need some platform support. We describe a possible platform in Chapter 7.3. The platform needs to determine if there is a potential energy consumption benefit in putting a node to sleep. For this we propose an energy model in Chapter 7.4. Next, as a second perquisite for the successful implementation of the platform we need to adopt the network layer. We explain our modifications to the MRv6 protocol in Chapter 7.5. Finally, we put all the pieces together and present a measurement framework that can be used as scheduler in a sleepy node aware platform.

7.2 REST API

The REST API for accessing the sleepy node functionalities is kept simple and light-weight. It consists of a resource called sleepy that each node has to support and only two operations. The first operation is for putting a mode to sleep. The second operation is for retrieving its status. The resource and the operations are shown in Table 7.1. We support a JavaScript Object Notation (JSON)[60] representation with the parameters as key-value pairs. We decided against using CoAP options, which would have had the advantage of being more compact as they are stored in the message header itself. Nonetheless, such a design would be tightly coupled to CoAP. The use of specialized resources allows easier integration into non CoAP REST IoT-systems, like those built on top of HTTP.

The get statement is special in some sense. In case of an accessible node it can return status information, like the next scheduled sleep period or the time already spent sleeping. If the mote is sleeping, it can, for obvious reasons, not answer the request. In that case platform support is needed. It lets the node appear connected to the network and replies to the request instead of the node.

Main Resource: sleepy		
Op	Parameter	Description
POST	duration	Puts a node to sleep for a given amount of time (in ms)
GET	duration state	Returns the remaining sleep time or the state of the node.

Table 7.1: Sleepy Nodes: REST API

7.3 Integration of Sleepy Nodes into the Enterprise Integration Platform

To comply fully with the previously given definition of a *sleepy node*, some support of an external system is needed. According to the definition of a sleepy node, to the user the node has to always appear as connected. Operations on the mote, therefore, need support if the node is sleeping. This can be achieved with an external integration platform such as the one we described in Chapter 4. We will now describe how such supported can be implemented as part of our integration platform.

The high-level schematics of the needed new components are shown in Figure 7.1. The new components integrate seamlessly into our architecture (as described in Chapter 4). The following additions are made: A command and control (C&C) module issues commands, such as sleep requests, to the sensor nodes. As a sleepy node always appears connected to the network, the platform acts as a transparent proxy, intercepting calls to sleepy nodes and returning the state and/or cached values in case they are sleeping. As our sleepy node implementation is to be used as part of a semantically-enriched IoT platform, the state of the sensor network (in this particular case, sleeping or not sleeping) is stored as part of the sensor network description.

The following three main interactions between a client requesting data and the node that provides this data are possible: (1) The node is not sleeping and the client can directly work with the node. (2) The node is sleeping and a client requests data. The request is intercepted by the transparent proxy. It returns the requested data together with the age of the information and an indication that it came from a proxy. The client can now either work with this data or query the sleeping time of the node and schedule a new query later on. (3) The node is sleeping, but a request that cannot be handled by the proxy arrives. This could be the case as no data has been cached, because the node requested that this particular resource is not to be cached, or because it is a request that has side effects. In such cases, the proxy will indicate the unavailability of the node and the client has to query about when the node will be available again and schedules its queries accordingly. In each case, the data can then be retrieved either by polling or via a notification (e.g. CoAP observe)

As outlined in Chapter 7.3, we are working with semantic descriptions instead of a resource directory [350]. Furthermore, we envision the use of an integration platform

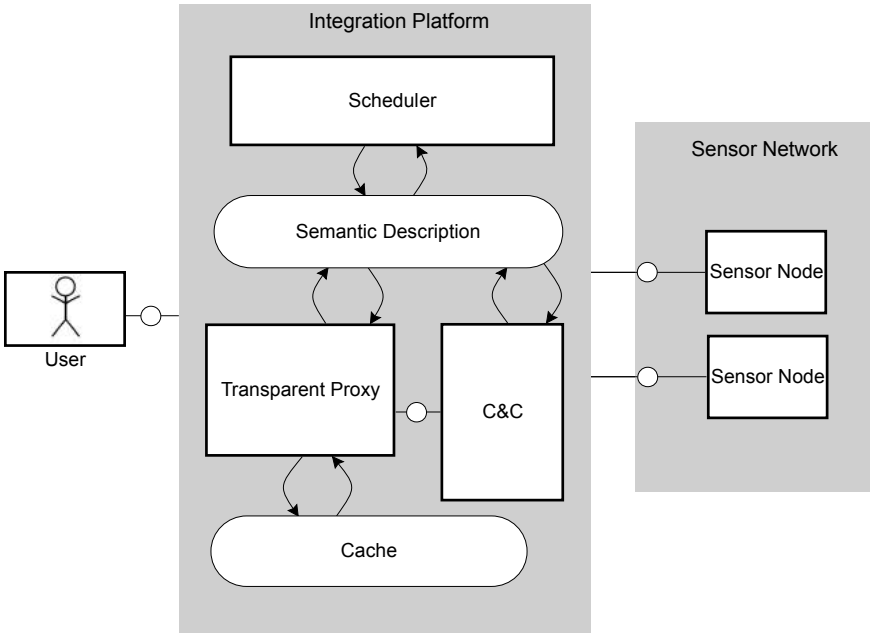


Figure 7.1: High-level schematics of a REST integration platform

that manages and issues the sleep states. The usage of an integration platform has the advantage of letting the sleepy node always appear connected to the network and reachable, even when there is access only to cached values.

A standalone implementation where the user, acting as a client, queries the network description (i. e. SSN) directly and issues calls only when the node is awake is possible. Alternatively, a scheduler can be used as an intermediate client.

A scheduler is particularly useful in use cases like the monitoring of goods throughout a supply chain or a warehouse (see Chapter 4.6.2). The user may request a new monitoring task from the scheduler, which can then leverage on knowledge available from the semantic description repository as well as from being aware of other already scheduled monitoring tasks, to handle it more efficiently. Depending on the objective at hand, different task allocation strategies can be used.

To demonstrate our framework's potential, we will present a windowing-based measurement framework in Chapter 7.6, with our goal being energy saving to maximize the network lifetime. To assess the strategies, we use an energy model that will be briefly introduced in the next section. In Chapter 8.8 we evaluate the sleepy-node implementation in conjunction with a supporting framework and a number of task allocation strategies.

7.4 Energy Model

The measurement framework needs to be able to determine if there is any benefit in sending a node to sleep. For this, it relies on an (analytical) energy model that we will introduce here.

7.4.1 Introduction

Energy estimations, if not measured directly, are typically made by estimating the time an application stays in a specific energy mode or by counting the number of messages sent and received. Given that the radio is often the most energy-consuming part of an IoT device, the message counting approach is considered good enough for many applications. Often, a three state radio model is assumed in which listening (receive), transmit and sleep states are considered [179, 161, 400, 302]. Models with higher levels of sophistication also consider the states of the CPU or of specific sensing devices [341, 200, 346].

7.4.2 Energy Model

We use a hybrid approach, in the sense that we consider the duration spent in some energy modes (idle and sleep) and the energy needed to react to different events (processing, sending and receiving). It is more accurate than message counting, but less accurate than measuring hardware states. Given the warm-up and cool-down times of devices, the presented strategies will look for large savings. Therefore, the objective of the energy model is not to get highly accurate solutions, but to define situations in which sleeping is more energy efficient than idling.

For every node, we distinguish between the following modes, each with a different level of power consumption:

- Protocol Idle: The node is connected to the network and participates only in required communication protocol activities.
- Protocol Sleeping: The node does not participate in any communication protocol activity. The nodes' radio is turned off. Infrastructure support is needed to make a sleepy node appear reachable to users.
- Sensing: The node is collecting data from its sensor
- Computation: The node performs computations
- Communication: The node is transmitting or receiving application data packets through its radio

To present our model properly, we first make the following assumptions and definitions: E_{sense}, the energy consumed by a sensing request, and E_{sleep}, the energy needed for processing and executing a sleep request, have negligible variance. Packet payloads do not vary considerably either. We call the amount of energy spent for receiving one packet and then transmitting one, E_{comm}. In *protocol idle*, a node is only consuming (on average) power P_{idle} for necessary protocol activity. Similarly, in *protocol sleeping*

a node is consuming constant power P_{sleep}. P_{sleep} and P_{idle} should not be confused with the CPUs idle or sleep energy states.

We can now express an approximation of the energy E_n consumed on node n with specific numbers of requests m, and the duration a node n stays in a specific state t^n_{state} as:

$$E_n = \underbrace{t^n_{idle} \cdot P_{idle} + t^n_{sleep} \cdot P_{sleep}}_{time\ based} \underbrace{+m^n_{sense} \cdot E_{sense} + m^n_{sleep} \cdot E_{sleep} + m^n_{comm} \cdot E_{comm}}_{event\ based}$$

P_{idle} is determined by the lower-layer communication protocol. In a beacon-based protocol (like MRv6) it depends on the number N of connected edges, namely the number of neighbour nodes it is connected to:

$$P_{idle} = P^{base}_{idle} + N \cdot P^{neighbour}_{idle}$$

where P^{base}_{idle} is the power consumed by participating in the communication protocol regardless of the number of connected nodes (neighbours) and $P^{neighbour}_{idle}$ is the extra power required for each connected node.

It is important to differentiate between time-based and event-based energy and power consumption. The relationship between time-based and event-based is as follows: P_{sleep} is time-based, while E_{sleep} is event-based. E_{sleep} is the energy needed to send a node to sleep, wake it up again, and reattach it to the network. The energy consumed between going to sleep and waking up is $t^n_{sleep} \cdot P_{sleep}$.

7.5 Implementation of Mid- and Longterm Sleeping in MRv6

In the following, we will briefly introduce how we extended MRv6 [180], the 6LoWPAN implementation that ships with Mote Runner, to support mid- and longterm sleeping periods. A brief introduction into 6LoWPAN is given in Chapter 2.3.2. The MRv6 protocol is described in Chapter 2.3.3. The extension of MRv6 came with some challenges that we had to overcome. The main challenge was to wake a node up before it was needed so that it would be available on time, taking into account the time it needs to attach to and leave the network as well as imprecise drifting clocks. First, we present our modifications and extensions of MRv6 in Chapter 7.5.1. Then we go on to discuss clock-drift in Chapter 7.5.2.

7.5.1 MRv6 Extensions for Sleepy Node Functionality

In order to support the application-layer sleepy node protocol, we had to extend the MRv6 protocol to enable the sleep states. As discussed in Chapter 7.4.2, the Mote Runner VM decides in which states the devices go. Therefore, we had to design the protocol so that it stops executing even the MAC layer protocol. This enables the Mote

Runner system to put the radio device (among others) into sleep mode. The upper layer protocol had to be modified such that the disassociation functionality of the original MRv6 protocol does not query the children nodes, as they are unable to answer in a planned deep sleep (protocol off) state.

Every (non-edge) mote that is loaded with the MRv6 component alternates between parent and child roles. Both roles keep track of their own state, reflecting the association or communication stage they are currently in. Either may change several states within a single communication period. Each of these states has several expectations on the behavior of the counterpart (parent or child) that we had to consider.

After a mote has been successfully associated with a parent, it starts participating in the super-frame (see Chapter 2.3.3). At the beginning of the communication period the mote adopts the child role in STATE_CHILD_RECEIVE_BEACON. It schedules a radio receive operation in order to listen for a beacon from its parent. The timestamp when the beacon is expected to arrive is calculated by adding BEACON_INTERVAL_MILLIS to the timestamp of the last received beacon. If the node is sleeping, none of this happens. The node cannot act as parent for its children and the children cannot communicate with the parent.

Our sleeping functionality is exposed to Mote Runner applications as an MAC layer interface. Any application running on the mote can put it to sleep for m milliseconds by calling $Mac.scheduleSleep(ms)$. The MAC layer will then calculate the sleeping time for the radio and then instructs the child-part (see Chapter 2.3.3) of the MRv6 layer to prepare for sleeping by also saving the timestamp that the next beacon is expected by calling the newly introduced function $Child.handleSleepRequest$, so that it does not have to go through the energy-costly and time-consuming procedure of reassociating with its parent afterwards (see Chapter 2.3.3.3 for the association process). First, the number of communication periods (π) that the requested sleeping time amounts to is calculated:

$$\pi = \left\lfloor \frac{msSleep}{BEACON\_INTERVAL\_MILLIS} \right\rfloor$$

Then, π beacon intervals are added to the timestamp of the last received beacon to form the next timestamp that we will be listening to for a parent beacon:

$$nextBeaconTs = lastBeaconTs + \pi \cdot BEACON\_INTERVAL\_MILLIS \quad (7.1)$$

After ensuring that there are no more pending packets to be transmitted by the mote, the MAC layer switches the radio off and sets a timer callback that will go off after the sleeping duration is over. The total time that the timer will count is calculated by subtracting an offset from the total requested sleeping time, in order to compensate for the radio state switches overhead and the mote wake-up (t_{ovh}) time.

$$wakeUpTs = lastBeaconTs + \pi \cdot BEACON\_INTERVAL\_MILLIS - t_{ovh} \quad (7.2)$$

Furthermore, clock drift has to be considered. We will take a closer look into clock drift in the next section.

7.5.2 Clock-Drift Considerations

The clock on most platforms is rather inaccurate and might drift over time. This means that two clocks on two nodes might not run at the exact same speed compared to each other. The clock drift problem, although a major issue in many distributed algorithms for WSNs, is not as dramatic in a coarse-grained sleepy node implementation as ours, as no exact timing is necessary. We only have to ensure that the node is awake at a given time. In sleeping periods in the magnitude of minutes and hours the impact of waking up motes some seconds before it is actually needed is negligible, compared to the savings of long term sleeping. Nonetheless, we want to keep the extra energy needed to compensate the clock-drift small.

Clock drift is measured in parts per million (ppm). Nowadays, a typical (non compensated) crystal clock has a drift of around 20ppm [63]. Nonetheless, boards that are more recent sometimes already have compensated crystals. The IEEE 802.15.4 standard specifies a maximum drift of 40 ppm [184]. 20ppm translate to

$$\begin{aligned} D_{percent} &= \frac{D_{ppm}}{10^6} * 100 \\ &= \frac{20}{10^6} * 100, for D_{ppm} = 20 \\ &= 0.002\% \end{aligned} \quad (7.3)$$

So, given one day with 86400 seconds, the expected clock drift per day is 1.78 seconds, or around 400ms for a 10 hour period. As shown in Maroti et al. [248] most clocks lag behind in time. As a consequence, the drift makes them timeout earlier instead of too late. We will see later on (Chapter 7.6) that in the window task allocation scheme we are using, all nodes wakeup periodically at the beginning of a window. We assume window sizes in the range of multiple hours as an absolute maximum. Therefore, the expected clock drift is within a range that can be handled by the protocol without too much additional energy waste while waiting for the beacon.

Recent platforms come with more precise clocks. The Waspmote Pro platform, for example, uses the DS3231 [255] from Maxim Integrated. It is a highly accurate realtime clock with an integrated temperature-compensated crystal oscillator (TCXO). It has an accuracy of only +-2ppm. In common temperature ranges its accuracy is even close to 0ppm. Figure 7.2 shows the drift ppm (delta frequency) of a DS3231 compared

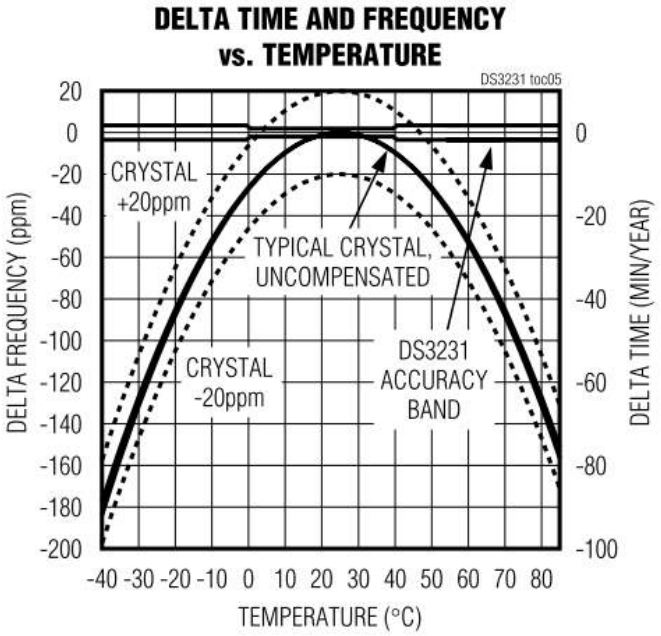


Figure 7.2: MAXIM DS3231 drift [255]

to crystals that are not temperature compensated. Therefore, on the Waspmote Pro platform, the clock drift we have to compensate for can be calculated as follows:

$$\begin{aligned} D_{percent} &= \frac{D_{ppm}}{10^6} * 100 \\ &= \frac{2}{10^6} * 100, for D_{ppm} = 2 \\ &= 0.0002\% \end{aligned} \tag{7.4}$$

These bounds can be improved by sophisticated estimation schemes that can be plugged into our implementation. Brzozowski et al. [63] investigated clock drift behavior with typical MicaZ motes by applying drift prediction based on samples gathered during runtime with moving average and linear regression algorithms.

Furthermore, to increase accuracy (thus sacrificing energy) the motes could turn on in intervals and synchronize with the global time base (beacons). Maroti et al. [248] have shown that in MicaZ modes, which can be considered similar to IRIS or Waspmote Pro, linear regressions models can be used to predict the clock drift. In experiments with lengths of 8 hours and 18 hours, the average absolute error was 2.24μS in the first experiment and 1.48μS in the second. The maximum absolute error was 8.64μs in the first experiment and 6.48μS in the second. Dai and Han implemented a similar approach as part of their TSync protocol [99].

If there is significant clock drift, after waking up, the mote might start listening for its parent beacon either sooner or later than it actually arrives. In the first case, there will be an extra energy overhead, proportional to the clock drift value, as this is the time that the mote keeps its radio on waiting for the parent beacon redundantly. In the latter case, the beacon we aimed for might be missed. Therefore, the mote will again have to wait with its radio on, until the next parent beacon arrives. In the worst scenario for both cases, the time that the mote spends with its radio unnecessarily on, will be equal to a whole beacon interval.

To reduce the possibility of loosing the parent beacon in the case of positive clock drift, we can wakeup one or two intervals earlier, by subtracting a value θ. Furthermore, we need to account for the overhead caused by waking up and turning the radio on again (t_{ovh}).

$$\begin{aligned} nextBeaconTs = lastBeaconTs \\ + (\pi - \theta) \cdot BEACON\_INTERVAL\_MILLIS \\ - t_{ovh} - t_{drift} \end{aligned} \tag{7.5}$$

There is a tradeoff when calculating good safety offset values that ensure that the node wakes up before the targeted beacon and can join the network quickly: If we use a bigger offset than needed, energy is wasted unnecessarily. But, if the offset is not big enough to cover the potential drift, then the radio will have to stay on for a whole beacon interval, which is even worse. If the clock drift magnitude, as well as an approximation of the average sleeping time that we expect to be requested are available, we can calculate a good safety offset as follows:

$$t_{drift} = t_{sleep} \cdot p_{drift} \tag{7.6}$$

For example, in case we rarely expect to request a mote to sleep for more than 2500 seconds and the measured clock drift is less than 2ppm, then by applying the above formula we get a 5ms safety offset.

In case of even lengthier sleep durations when large clock drifts are present, the delay until the mote becomes responsive again might even exceed one beacon interval. Regardless of the effect on energy consumption, it has to be accounted for during the higher-level sleeping time calculation.

When the sleep timer expires, the radio is switched back on and the mote adopts the child role again, but this time starting in a new state that we added to the MRv6 state model: STATE_CHILD_SYNC_BEACON. This state is identical to STATE_CHILD_RECEIVE_BEACON, except for the fact that it listens to the global superframe to get in sync with the network again.

The sleep routine was designed to be invoked from the application layer, so that the mote may offer a REST go-to-sleep service.

7.6 Measurement Framework

We introduce a measurement framework that solves a soft real-time task allocation problem in a time-sliced (windowed) environment. We present three windowing based task allocation (WBTA) strategies that map sensing requests to actual nodes. The proposed system shows the potential of a centralized sleep planning system by leveraging knowledge available at the application layer. The objective of the WBTA is to measure the entities at the required intervals, while aiming for an enhanced overall network lifetime. We first give a detailed problem formulation in Chapter 7.6.1. We then introduce the task allocation strategies in Chapter 7.6.2. Next, in Chapter 7.7, we explain the dynamic partitioning heuristic in a step-by-step example.

7.6.1 Problem Formulation

We assume a given static WSN topology and a list of entities that need to be measured on defined time periods. These measurements can only be performed by defined sensor sets. Our main objective is to generate a close-to-optimal ("best effort") querying schedule that accommodates all given entity measurements, considering latency and energy saving requirements. We define $\mathbb{S}$ to represent the set of all sensors and $\mathbb{E}$ to be a set of entities, whose properties we want to monitor over time. First, we introduce the following terms as part of our notation:

- $Measurement$: A measurement for an entity is a defined period in time (timestamp and time tolerance) at which a measurement should be performed. More than one sensor may be eligible to perform it.
- $Query$: A query is a specific request to a sensor of the WSN to perform a measurement. It may combine more than one measurements, as long as they are within time constraints and the assigned sensor is eligible to perform all of them. A query can be seen as an instance of one or more measurements.

Furthermore, for each entity $e_i \in \mathbb{E}$, we define the tuple (S, ω, δ) with

- S: $S \subset \mathbb{S}$ set of sensors that can measure the property of the entity
- ω: time period between subsequent measurements
- δ_i: the time tolerance within which a measurement is acceptable

We leverage on the high-level sleeping services, as introduced in Chapter 7.3, taking into account the period ω, the tolerance δ, and the set of possible sensors S, to send nodes into a sleep mode in order to save energy and achieve an enhanced network lifetime. A visualization of the scenario is shown in Figure 7.3. In the illustration, there are four nodes in the network (N_0,N_1,N_2,N_3) and three entities (E_1, E_2, E_3) representing some goods to be monitored . E_1 is in the sensor range of nodes N_1 and N_2. E_2 is in the range of the nodes N_3 and N_1. E_3 is only in the range of node N_3.

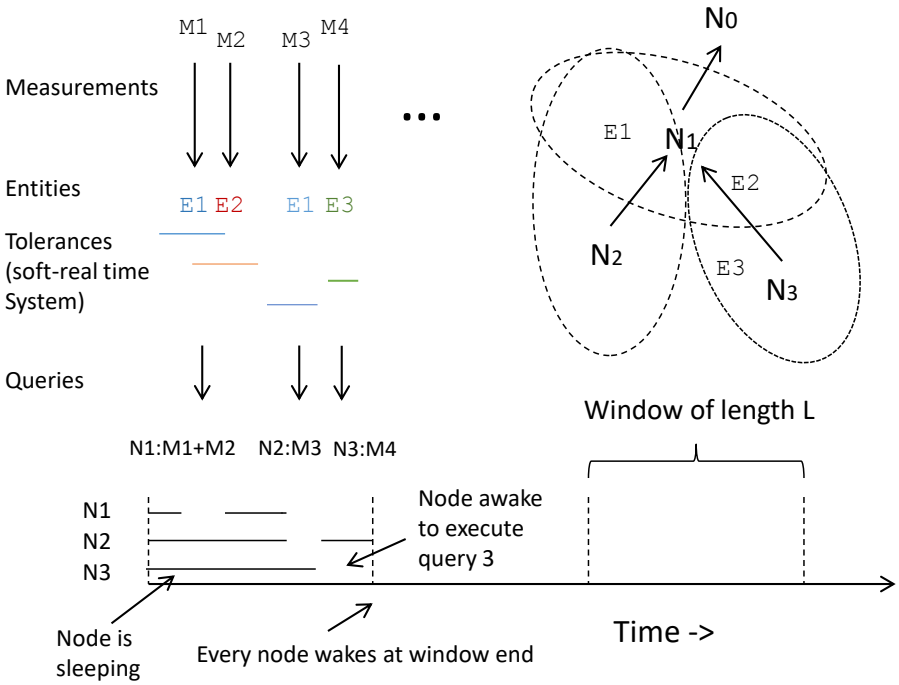


Figure 7.3: Measurement Framework: Illustration of the problem formulation

We assume that within the first window four measurements M_1, M_2, M_3 and M_4 have to be performed. M_1 and M_3 measure a property of E_1. M_2 measures a property of $E2$ and M_4 measures a property of E_3. Each of these measurements has a tolerance associated to it. The tolerances of M_1 and M_2 overlap, so the system could decide to combine the two measurements into one actual query for node N_1. The tolerances of M_3 and M_4 overlap as well, but there is no common sensor. So, no decision has to be taken and M_3 is mapped to a query of N_2 (in this particular example, N_1 could also have been selected). M_4 is realized as query to N_3. The resulting sleeping times are shown in the bottom left corner. In the beginning all nodes can sleep, then N_1 needs to be awake to answer the query. It then can go to sleep again. Later, the remaining two queries have to be performed. Node N_2 awakes to execute the second query and, finally, N_3 will respond to its query.

7.6.2 Implementation of the Windowing-based Task Allocation Framework

In the following, we present the implementation of the WBTA and the task allocation strategies.

The scheduling task can be roughly expressed as mapping a set of required mea-

surements to a sequence of scheduled queries. Our base scheduler uses a priority query queue (soonest timestamp first).

There is no explicit sleeping schedule. The sleeping requests are issued immediately after the sensing queries have been completed. Calculation of the sleeping time for each node is performed by the scheduler as follows: The scheduler examines the queries queue, searching for the soonest timestamp where the node will be needed next either for sensing or for communication. It then calculates the time interval from the current timestamp to this timestamp and subtracts a communication overhead. This is the maximum duration that the node can be allowed to sleep. Sleeping is omitted if this duration is less than a defined threshold, where the energy overhead for sleeping is bigger than the gain. The sleeping times are always constrained by the window length. At the end of every window, all nodes in the network are awake.

The basic querying loop is shown in Algorithm 1, where the sleeping times calculation is provided by the procedure CalcSleepTime.

The scheduling occurs in time windows of length L and is described in Algorithm 2. At the beginning of each window w, M_w, the set of measurements that have to be performed within this particular window, is calculated. Procedure CalcWindowQueries is then called to map these measurements to a queue of queries. We present a number of different strategies for this in Chapter 7.6.3.

In order to balance the power load over the nodes, we use a control policy based on penalties for their utilization. We evaluate the sensors used for a measurement from the set of possible sensors, according to their expected effect on the energy footprint of the whole network. The penalty p^n for each sensor S_n is calculated as the quotient of the average energy (m) over all nodes by the nodes' residual energy (ε^n_{res}). We use the penalty to prevent over-utilization of one sensor to increase the network lifetime.

$$\begin{aligned} \varepsilon^n_{res} &= \varepsilon^n_{init} - \varepsilon^n, \forall n \in S \\ p^n &=\leftarrow \frac{m}{\varepsilon^n_{res}}, \forall n \in S \end{aligned} \tag{7.7}$$

A penalty with a value larger than 1 means that the respective node has already consumed more energy than its counterparts. Further selections of this particular sensor will result in an even higher utilization and, thus, in larger penalized costs. This will cause the scheduler to avoid choosing that node in favor of one that yields lower penalized cost.

Using the aforementioned penalty rule when choosing a sensor has two main advantages:

- Modeling of heterogeneous networks with different initial energy reserves for different nodes is possible.
- As a nodes' energy residual approaches zero, the denominator becomes very small, effectively yielding a very high penalty for the utilization of the node.

Algorithm 1 Base querying loop

```
1: while true do
2:     NEWWINDOW
3:     for all n ∈ S do
4:         t^n_sleep ← CALCSLEEPTIME(n)
5:         Issue sleep request for t^n_sleep on node n
6:     end for
7:     while QueryQueue not empty do
8:         nextQuery ← QueryQueue.remove()
9:         wait until timestamp of nextQuery
10:        execute nextQuery
11:        for all nodes that were used do
12:            t^n_sleep ← CALCSLEEPTIME(n)
13:            Issue sleep request for t^n_sleep on node n
14:        end for
15:    end while
16: end while
1: procedure CALCSLEEPTIME(n)
2:     for all q ∈ QueryQueue do
3:         if q uses node n then          ▷ either for sensing or for communication
4:             t_diff ← q.ts − ts_now
5:             t_wind ← windowEnd − ts_now
6:             t_sleep ← min(t_diff, t_wind) − t_comm
7:             if t_sleep ≤ t^thresh_sleep then
8:                 return t_sleep
9:             else
10:                return 0
11:            end if
12:        end if
13:    end for
14: end procedure
```

Hence, any choice that risks a node being depleted will be harshly penalized and, thus, avoided.

The calculation of these penalties is done as part of Algorithm 3. EvalQueries is given a query queue as input and it returns an energy cost for its potential realization. Calling SimulateConsumption will produce the states we expect the networks' nodes to be in, after executing the given queries within the current window.

GetEnergyMap is then called to provide the energy that has been consumed on each sensor within the windows time interval according to these states. GetEnergyMap calculates the consumed energy using the linear energy model and scaled with the

Algorithm 2 New Window and penalty update

procedure NEWWINDOW
    $windowStart \leftarrow windowEnd$ ▷ previous window's end
    $windowEnd \leftarrow windowStart + L$
    UPDATEPENALITES
    $M_w \leftarrow \{m \in M : windowStart \leq m.ts < windowEnd\}$
    $QueryQueue \leftarrow$CALCWINDOWQUERIES(M_w)
end procedure

procedure UPDATEPENALTIES
    $\vec{\varepsilon} \leftarrow$ GETENERGYMAP(0, $currentTimestamp$)
    $m \leftarrow mean(\vec{\varepsilon})$
    $\varepsilon^n_{res} = \varepsilon^n_{init} - \varepsilon^n, \forall n \in S$ ▷ ε^n_{init} indicates the initial energy reserve of sensor n
    $p^n \leftarrow \frac{m}{\varepsilon^n_{res}}, \forall n \in S$
end procedure

penalty vector $\vec{p}$. The penalties are updated at the beginning of each window, as shown in procedure UpdatePenalties.

Algorithm 3 Base Evaluation of Queries

procedure EVALQUERIES(Q)
    SIMULATECONSUMPTION(Q)
    $\vec{\varepsilon} \leftarrow$ GETENERGYMAP($windowStart$, $windowEnd$)
    $penalizedEnergy \leftarrow \vec{\varepsilon} \cdot \vec{p}$
    return $penalizedEnergy$
end procedure

7.6.3 Scheduling Strategies

In the following, we present three strategies for scheduling the tasks within a window: an exhaustive strategy (Chapter 7.6.3.1), a memoization strategy (Chapter 7.6.3.2) and first fit (Chapter 7.6.3.3). The memoization strategy, as we will show, performs only slightly worse than the exhaustive strategy. The exhaustive strategy, also known as exhaustive search or bruteforce, guarantees finding the optimal solution for the clustering problem, with respect to our given window and query evaluation system. It has exponential complexity and works only for small problem sizes within reasonable time bounds. The first fit strategy tries to compensate for overusing power resources by randomly choosing a sensor, without any optimizations or combinations.

For every window w, a set of measurements M_w that are to be mapped to queries is given. Consider an arbitrary partition of M_w, consisting of a family of sets called clusters C, with

$$\bigcup C = M_w, 0 \notin C, \forall C_1, C_2 : C_1 \neq C_2 \rightarrow C_1 \bigcap C_2 = 0 \tag{7.8}$$

We call the partition C *valid* if all $C_i \in C$ can be mapped to queries. For a cluster C_i to be mapped to a query, the following constraints need to be satisfied:

1. There is at least one sensor eligible to perform **all** measurements.
2. There is an overlap of the time tolerance intervals among all of its measurements.

In order to map a cluster that fulfills the above constraints into a query, we extract a common timestamp and sensor for it. The common timestamp is calculated as the mid-point of the common intersection of the time tolerance intervals of all the clusters' measurements. The sensor selection depends on the strategy and is described in the respective sections. We will refer to the above conditions as time tolerance and sensor constraints, respectively, in the context of combining measurements into queries.

7.6.3.1 Exhaustive Strategy

The optimal solution for the clustering problem, within our windowing and query system, is applying an exhaustive approach (backtracking with constrained satisfaction). An exhaustive approach considers all possible partitions of the set M_w and evaluates them. The exhaustive approach is outlined in Algorithm 4. First, all possible partitions are calculated and then each one is evaluated. The evaluation (and sensor selection) in procedure EvalPartition is performed as follows: The clusters are examined in reverse chronological order. First, the intersection of the sets of eligible sensors of all the measurements is calculated. Afterwards, for every sensor in the resulting set, a query is constructed and evaluated according to its energy profile in procedure EvalQueries (as defined in Algorithm 3. The sensor that yields the lowest penalized energy cost is selected and the resulting query is added to the queries queue. After all clusters have been mapped to queries, the resulting query queue is evaluated and the outcome is returned. [1]

The exhaustive approach has exponential time complexity, as the total number of possible partitions for a set of n measurements is the Bell number[2] $B_n = \sum_{i=0}^{\infty} \frac{B_i}{i!} n^i = e^{e^n - 1}$. In practice, it may only be applied to small windows and, thus, will only be used for benchmarking.

7.6.3.2 Dynamic Partitioning

In the following, we present an alternative to the exhaustive approach. Instead of considering all possible partitions, we order the measurements M_w according to their timestamps and check only the ones that combine neighboring measurements using a greedy strategy. It calculates a local optimum and never reconsiders its choice. This

[1] This is equal to the cost that was returned by the evaluation of the sensor selected for the last query added. Therefore this second evaluation need not be actually done.

[2] Named after Eric Temple Bell, the Bell numbers count the number of partitions of a set. B_n counts the number of different ways to partition a set of n elements

Algorithm 4 Exhaustive Window Queries Calculation

```
1: procedure CalcWindowQueries(M_w)
2:     SP_w ← all possible partitions of window measurements set
3:     P_best ← argmin_{P∈SP_w} EvalPartition(P, M_w)
4:     return generated queries for partition P_best
5: end procedure
1: procedure EvalPartition(P_w, M_w)
2:     Q_w ← empty queue
3:     for all cl ∈ {P_w sorted backwards} do
4:         M_cl ← M_w(cl)                          ▷ get measurements in this cluster
5:         ts ← mean timestamp over M_cl
6:         S_cl ← {∩(S ∈ M_cl)}                    ▷ ∩() represents reduction with intersection
7:         s_best ← argmin_{s∈S_cl} EvalQueries( [ new Query(ts, s, M_cl) , Q_w ] )
8:         Q_w.add(new Query(ts, s_best, M_cl))
9:     end for
10:    return EvalQueries(Q_w)
11: end procedure
```

problem can be solved by applying memoization [96]. We call this heuristic dynamic partitioning.

Next, we briefly present the main steps of this strategy: The approach is shown in detail in Algorithm 5. As we are following a memoization approach, we examine previously solved subproblems and combine their solutions during the construction of a $|M_w|\,x\,|M_w|$ matrix representing subpartitions. The upper triangular part of the matrix, excluding the diagonal, is not used. An illustration of such a matrix can be seen in Table 7.2. Within the lower triangular part of the matrix, the (x,y) coordinates represent a sequential set of measurements within M_w, starting at measurement M_x and ending at measurement M_y, with $x \leq y$. This means that cell (x,y) holds the best known valid partition of all measurements $(M_x, M_{x+1}, .., M_y) \in M_w$. By populating the matrix step-by-step for each subset of sequential measurements we end up with the partition that, after being mapped to a query queue, has the lowest known cost for the whole window. This approach does not necessarily yield the optimal solution; nonetheless, in practice, it performs quite well as measurements that can be combined appear often in sequence when ordered by timestamp. Experimental results are presented in Chapter 8.8.

The matrix is evaluated column-wise starting at column x = $|M_w|$. At the beginning of each column-wise evaluation, y equals x, because the upper triangular part of the matrix is not used. Next, y is increased as long as y $< |M_w|$. Afterwards, the next column is evaluated. The diagonals are filled trivially, as a set of only one element has only one possible partition and thus can be filled in directly. For each non trivial sequence of measurements $M_x, .., M_y$ we select one element after the other as a pivot

element dividing the sequence into two. The two sequences are then evaluated, either by computation or by matrix-lookup and concatenated into one solution. The best valid concatenation is added to the table. Repeating this iteratively will end with the desired solution in cell (1,$|M_w|$). In Chapter 7.7 we illustrate this strategy within a comprehensive example.

Algorithm 5 DP Window Queries Calculation

procedure CALCWINDOWQUERIES(M_w)
  $n \leftarrow |M_w|$
  $P_{table}[i][i] \leftarrow [i], \forall i \in [1, n]$
  for $fr \leftarrow n-1, 0$ **do**
    for $w \leftarrow 1, n-fr-1$ **do**
      $to \leftarrow fr + w$
      $piv_{best} \leftarrow \text{argmin}_{piv \leftarrow fr,to}$ EvalSplit(fr, to, piv, P_{table})
    end for
    if $piv_{best} = to$ **then**
      $P_{table}[fr][to] \leftarrow [\ range(fr, to)\ ]$
    else
      $P_{table}[fr][to] \leftarrow [\ P_{table}[fr][piv], P_{table}[piv+1][to]\ ]$
    end if
  end for
  return generated queries for partition $P_{table}[1][n]$
end procedure

procedure EVALSPLIT(fr, to, piv, P_{table})
  if piv = to **then**
    $subP \leftarrow [\ range(fr, to)\ ]$
  else
    $subP \leftarrow [\ P_{table}[fr][piv], P_{table}[piv+1][to]\ ]$
  end if
  if to $<$ n **then**
    $P \leftarrow [\ subP, P_{table}[to+1][n]\ ]$
  else
    $P \leftarrow subP$
  end if
  if P breaks time tolerance or sensors constraints **then**
    return ∞
  else
    return EVALPARTITION(P)
  end if
end procedure

7.6.3.3 First Fit Strategy

We compare the dynamic partitioning strategy with two alternative strategies as benchmarks. The first benchmark we use to evaluate the dynamic partitioning strategy is the exhaustive strategy that we already introduced in Chapter 7.6.3.1. As a second benchmark for comparison, we use a first-fit algorithm operating within the same windowing framework. It creates a new query for every measurement, picking one of the eligible sensors at random. The first fit algorithm is shown as Algorithm 6.

Algorithm 6 First Fit Window Queries Calculation

```
1:  procedure CalcWindowQueries(M_w)
2:      Q_w ← empty queue
3:      for all m ∈ M_w do
4:          S ← eligible sensors for m
5:          s ← random sample from S
6:          ts ← timestamp of m
7:          Q_w.add(new Query(ts, s, m))
8:      end for
9:      return Q_w
10: end procedure
```

7.6.4 Dynamic Measurement Request

Up to this point, we assumed that the system has to consider only periodic measurements known in advance for a given time window. This is a reasonable assumption in many enterprise environments. We will now relax this constraint and describe how the system can be adapted in order to react to dynamic, not pre-known, measurement requests. Our schedulers now need to accommodate incoming measurements that are submitted externally to our system in a dynamic, perhaps unknown, fashion. First, in Chapter 7.6.4.1, we describe how the system can react to arbitrary (previously unknown) measurement requests. We also propose a countermeasure that is based on a probability distribution to schedule probable measurements and, thus, can proactively turn on a sensor. We call this strategy "virtual measurements", because from a systems point of view they are not real measurements at the beginning of a window, but just expected measurement requests. Virtual measurements are explained in detail in Chapter 7.6.4.2.

7.6.4.1 Arbitrary Measurement Requests

We assume that on top of the static periodic measurements in each window, we may also get measurement requests for any entity at arbitrary times. This means we weaken the assumption that all measurements are known a-priori. As described in Chapter 7.6.1, the scheduling takes place at the beginning of the window, which means that

sensors will only be awake and available for the intervals they are needed for the already scheduled queries. Now, assuming that an incoming request has to be served that was not previously scheduled at the beginning of the window, the scheduler handles it as follows: First, the scheduler iterates the queue of the remaining scheduled queries for that particular window to find sensors that could also perform the new measurement. If such a query is found, then the measurement can be added to this query and nothing else has to be done; the latency of the response depends solely on the time until this query. Alternatively, the scheduler can look for eligible sensors in the communication paths from the sink to the nodes used by the remaining scheduled queries. If such sensors are found, that is, if one of the nodes in the communication path can be utilized for sensing while being awake for enabling communication anyway - then a new query is scheduled for this node. The timestamp is chosen to be similar to the one from the original query. Finally, in case neither is possible, we schedule the measurement for the next window. A countermeasure that decreases latency at the cost of energy is described in Chapter 7.6.4.2. We call it virtual measurements.

7.6.4.2 Virtual Measurements

In order to reduce the latency of the dynamic incoming requests as described in Chapter 7.6.4.1, we can choose to include some "virtual measurements". These virtual measurements require the scheduler to reserve nodes that can be used to measure an entity, by keeping them awake for a defined time interval within the window. We call these measurements virtual as they do not necessarily represent a real measurement, but are treated by the scheduler as such and, therefore, are virtual until a real measurement is added to the system.

For each entity, this interval reflects the expectation about when it is most probable to receive an actual incoming measurement request. Virtual measurements have a start and end timestamp, and they are processed by the scheduler in the same way as regular measurements. The scheduler tries to combine the virtual measurements along with real ones, by sharing sensors and using the ones that have consumed less energy, in the exact same fashion as it would if only real measurements were used. When assigning actual measurements to queries, separate virtual queries are created for each virtual measurement. Finally, when the time comes to execute a virtual query, no real query is issued, but the associated node and all required nodes in the communication path are kept awake for the time interval defined by the virtual measurement.

To define those intervals, we assume an underlying probability distribution for each entity. In our experiments we used a Gaussian distribution, but any other distribution could be used as well. The distribution describes the expected timestamp of the next dynamic measurement request. It is obvious that every virtual measurement induces an energy overhead equal to the energy that would be preserved if the nodes were sleeping instead. This energy-latency trade-off can be controlled by adjusting the interval lengths of the virtual measurements. We will refer to that as the confidence level.

The lower the (required) confidence level of an incoming query the better it is

expected to fit the estimation of the underlying distribution. A required confidence level of 0 means that we believe all incoming messages will arrive exactly when expected, a confidence level of 0.1% means that we choose an interval such that 10% of the measurements are expected to be covered, while a (theoretical) confidence level of 1 means that we will be extending the waiting window according to the underlying distribution, in order to be 100% sure that we will be able to answer the message.

For example, if we assume a normal distribution for all arbitrary incoming messages, we can say we expect the next measurement for an entity to arrive in t+30 minutes from now, with a standard deviation of 5 minutes. To be 70% confident that we will be able to answer the request right away - resulting in low latency - we will have to schedule a virtual measurement covering the time interval from 24.818 to 35.182 minutes. As those numbers already suggest additional 10min of awake time results in a considerable potential additional energy consumption. The total amount of additional energy in idle-mode compared to sleep-mode depends on the energy characteristics of the used hardware and protocol stack. The actual energy consumption depends on when the measurement request actually arrives.

Therefore, the confidence levels should be chosen reasonably with respect to the properties of the assumed distribution.

7.6.5 Packet Loss

In case a query is not answered by the mote for any reason – for example, because of packet loss due to interference – it is repeated immediately a number of times before being dropped. The default maximum number of retries we used was three. The delay resulting from the retransmissions might cause a measurement to miss the deadline defined by the related entity's tolerance. In a near-real time system this behavior is tolerated. Of course, every packet loss will cause an additional energy overhead.

7.7 Example

This section focuses on applying the dynamic partitioning heuristic (Algorithm 5) by examining an example, based on a some small exemplary data set. We will first describe the setup of the example before we proceeding to demonstrate our heuristic, the dynamic partition strategy. We conclude the example by discussing the sensor selection.

7.7.1 Scenario

The setup is illustrated in Figure 7.4. We consider the third window, starting at t_{start}=100 and ending at t_{end}=150. We assume four entities and one measurement each (see Chapter 7.6.1). The measurements that have to be scheduled within the window are also listed in Table 7.2.

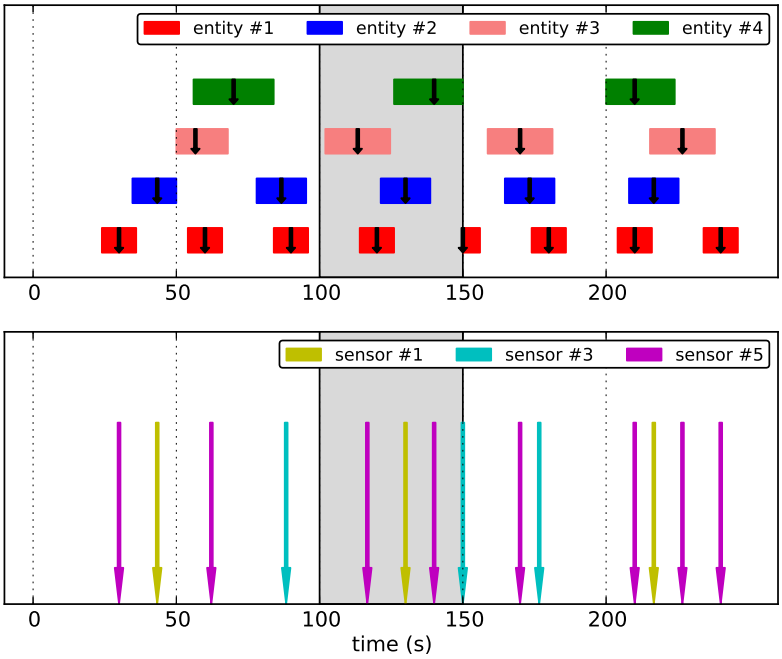


Figure 7.4: Measurements arrival and assignment to queries

We introduce the following notation in accordance with the definitions presented in Chapter 7.6.3

- To refer to each measurement, we will use the measurement numbers (#) given in Table 7.2. Notice that the measurements are already sorted according to their timestamps : $ts_1 \leq ts_2 \leq ts_3 \leq ts_4$.
- We will represent an ordered set (sequence) of measurements, using curly brackets. For example, the sequence of our window's measurements is $M_w = \{1, 2, 3, 4\}$
- To refer to clusters within partitions, we will use square brackets. [2 3], for

Measurement #	ts	Entity #	tolerance (δ)	avail. sensors (S_{pi})
1	113	3	11	[4, 5, 6]
2	120	1	6	[3, 5]
3	130	2	8	[1, 3]
4	140	4	13	[5]

Table 7.2

Measurement #	1	2	3	4
1	[1]			
2	-	[2]		
3	-	-	[3]	
4	-	-	-	[4]

(a)

Measurement #	1	2	3	4
1	[1]			
2	-	[2]		
3	-	-	[3]	
4	-	-	[3] [4]	[4]

(b)

Measurement #	1	2	3	4
1	[1]			
2	-	[2]		
3	-	[2 3]	[3]	
4	-	-	[3] [4]	[4]

(c)

Measurement #	1	2	3	4
1	[1]			
2	-	[2]		
3	-	[2 3]	[3]	
4	-	[2 3] [4]	[3] [4]	[4]

(d)

Measurement #	1	2	3	4
1	[1]			
2	[1 2]	[2]		
3	-	[2 3]	[3]	
4	-	[2 3] [4]	[3] [4]	[4]

(e)

Measurement #	1	2	3	4
1	[1]			
2	[1 2]	[2]		
3	[1 2] [3]	[2 3]	[3]	
4	-	[2 3] [4]	[3] [4]	[4]

(f)

Measurement #	1	2	3	4
1	[1]			
2	[1 2]	[2]		
3	[1 2] [3]	[2] [3]	[3]	
4	[1 2] [3] [4]	[2 3] [4]	[3] [4]	[4]

(g)

Table 7.3: `DP window queries scheduling` . Entries that result from full partitions are highlighted in red. When a concatenation of smaller subpartitions is used, it is highlighted in blue and the participating subpartitions in yellow.

example, is a cluster containing measurements #2 and #3. We use the same notation to refer to the measurements that a query combines, as we already defined a one-to-one mapping from clusters to queries.

- Finally, we use square brackets again to group clusters, thus producing partitions of the respective sequences. For example, [[1] [2 3] [4]] is a partition of M_w. This means that measurement #1 is executed as a query. Measurement #2 and measurement #3 are combined into one query. Measurement #4 is again executed as a query on its own.

7.7.2 Dynamic Partitioning

This section illustrates the behavior of the dynamic partitioning algorithm based on some small exemplary data set. A visualization of the measurements and their tolerances

is shown in Figure 7.4. As explained in Chapter 7.6.3.2, the goal of the dynamic partitioning algorithm is to find a good valid partition of M_w.

The algorithm fills in a 4x4 matrix – here called P_{table} – with the best-known valid partitions for each sequence within M_w. Cell $P_{table}[columni][rowj] = (i, j)$ will be filled in with the best known valid partition of $\{i, .., j\}, i \leq j$, ignoring the measurements previous to i and using the best known valid partitions already computed for $\{j + 1...n\}$. The final cell to be filled in – in this example $(1, 4)$ – will contain the desired solution.

Following this, we use the term "subpartitions" to refer to partitions of the respective subsequences of M_w that correspond to each cell of P_{table}. To follow the example, please refer to Table 7.3, where each step is illustrated. The energy values are calculated using the linear energy model as described in Section 7.4. The values used in this example are not artificial. They are based on a real hardware platform as used in the evaluation section. We will be going through the algorithm step by step, by filling in the respective cells in the matrix.

The diagonal cells are filled trivially, as there is only one possible partition of a set with one element – that is, we cannot combine a measurement with any other. This sufficiently bootstraps the algorithm (see Table 7.3(a)). We start with the first non-trivial cell, $(3, 4)$.

Cell (3,4): The entry of cell (3,4) is the best possible valid partition of $\{3\ 4\}$. Only two partitions are possible [[3] [4]] and [[3 4]]. The dynamic partitioning heuristic determines the possible partitions, by choosing one distinguished element of the set: the pivot element. The pivot element is chosen one by one from left to right. The first pivot element, at 3, yields the partition [[3] [4]]. This means a separate query is needed for the measurements #3 and #4. The energy cost returned by the evaluation function here is assumed to be 2.21. Next, we move the pivot element to the right. The second possible pivot, at 4, yields the partition [[3 4]]. At this point, we need to evaluate if the two measurements can be combined. For this, the partition needs to be valid, as defined in Chapter 7.6.3. The two measurements satisfy the tolerance constraints and could be combined. A combination is possible, because the possible interval of M#4 is $t_4 = 140 \pm 13$and $t_3 = 130 \pm 8$. For example, t=130 would be a time where both measurements could be performed within their tolerances. Nonetheless, M#4 can only be performed by sensor 5. M#3 can be measured by sensor 1 and sensor 3. As the sets of eligible sensors do not overlap the partition is not valid. It has a cost of ∞. As there is only one valid partition, this partition is added to the table. As shown in in Table 7.3(b), [[3] [4]] is added into the matrix. We proceed to the next column.

Cell (2,3): The cell (2,3) is processed similarly as cell (3,4). The two possible pivot elements yield the partitions [[2] [3]] and [[2 3]]. To calculate the energy consumption and update the penalty map, the algorithm also takes the remaining measurements until the end of the window into account – in this case measurement 4. A lookup reveals the

best respective partition. Here it is the trivial one, [[4]].

Measurements 2 and 3 have one common eligible sensor - sensor #3 - meaning that since they also satisfy time tolerance constraints, they can be combined into one query. To evaluate partition [[2][3]], three queries with measurements [2] [3] and [4] are calculated. We assume this returns a total cost of 2.56. To evaluate the performance of subpartition [[2][3]], two queries with measurements [2 3] and [4] are considered. The assumed cost is 2.42. [2 3] is added in the matrix as shown in Table 7.3(c).

Cell (2,4): In cell (2,4) the number of elements in the set increases. There are now three elements in the set (2,3,4). For pivot element 2, a lookup at cell (2,2) retrieves the trivial partition [[2]]. A second lookup at cell (3,4) contains the partition [[3] [4]]. As illustrated, the algorithm is greedy and does not reconsider a decision it made once. The best partition calculated in cell (3,4) is used, even if in this particular situation a different partition combing (2,3,4) might be more optimal. The concatenation of the lookups in cell (2,2) and (3,4), [[2] [3] [4]], has a total cost of 2.56. Similarly, for pivot element 3 and partition [[2 3] [4]] a cost of 2.42 is returned. Pivot element 4 yields partition [[2 3 4]]. This combination is encountered for the first time; hence, no entry in the matrix exists for it yet. However, as [[3 4]] was not a valid partition, it is obvious that [[2 3 4]] is also not valid. Therefore, [[2 3] [4]] is added to the matrix. The result of this step is shown in Table 7.3(d). We continue with the next column.

Cell (1,2): The two pivot elements lead to the two partitions [[1] [2]] and [[1 2]]. For the remaining measurements until the end of the window, a lookup of cell (3.4) is necessary: It contains subpartition [[3][4]]. The resulting partitions, after appending to the aforementioned are respectively: [[1] [2] [3] [4]], which yields a cost of 2.91 and [[1 2] [3] [4]], which yields a cost of 2.56. [[1 2]] is entered into cell (1,2), as shown in Table 7.3(e).

Cell (1,3): Similarly to (3,4), we get 1, 2 and 3 as pivot elements. They yield the following partitions and respective costs:

[[1] [2 3]] $\rightarrow$ 2.77

[[1 2] [3]] $\rightarrow$ 2.56

[[1 2 3]] $\rightarrow \infty$

[1 2] [3] is added to the matrix, as shown in Table 7.3(f).

Cell (1,4): We now proceed to the final cell. The pivot elements are 1, 2, 3 and 4. The resulting partitions and their costs are:

[[1] [2 3] [4]] $\rightarrow$ 2.77

[[1 2] [3] [4]] $\rightarrow$ 2.56

$[\ [1\ 2\ 3\ 4]\ ] \rightarrow \infty$.

As the partition $[\ [1\ 2]\ [3]\ [4]\ ]$ has the lowest cost it is added into the matrix. As this is the last cell to be filled in, the algorithm returns this partition as a solution for the whole window.

7.7.3 Sensor Selection

If more than one eligible sensor exists for a query combining one or more measurements, one has to be selected. This is always done as part of any partition's evaluation. It is described in detail here in the context of our example.

Assuming partition $[\ [1\ 2]\ [3]\ [4]\ ]$ for the window we discussed above, the sensor selection starts by examining the clusters in reverse order.

The last cluster, and hence the first to be examined is [4]. Sensor $\#5$ is the only one eligible to perform measurement $\#4$. So the decision here is trivial.

The next cluster is [2]. There are two sensors eligible to perform measurement$\#2$ – namely $\#1$ and $\#3$. The energy model yields a cost of 2.21 for sensor $\#1$, while usage of sensor $\#3$ yields 2.42; therefore, we will query $\#1$ for this measurement.

The final cluster is [1, 2]. The intersection of the sensors eligible to perform these measurements only contains sensor $\#5$; therefore, we again select it trivially.

7.7.4 Final queries

Summing up all of the above, the arrival of the measurements along with the tolerances of their respective entities and the actual assignment to queries can be seen in Figure 7.4. The four measurements due in the window between 100s and 150s, are performed by issuing three queries: The first is for sensor $\#5$, and it combines measurements $\#1$ and $\#2$. The second one is for sensor $\#1$ and performs measurement $\#3$. The last one is for sensor $\#5$ again and performs measurement $\#4$. One can observe that the queries indeed satisfy the tolerances of the entities they measure.

7.8 Conclusions

Energy management and saving of energy is not only a network-layer or hardware problem, but can also be tackled at the application-layer. The presented sleepy node implementation is leveraging on semantics that are stored in an enterprise system. The nodes with mid- and longterm sleeping capabilities, which adhere to the definition of a sleepy node, provide a simple REST API. To unleash the full power of sleepy nodes, platform support is needed. This is, first to ensure that the node appears as connected to the user. Second, in our case, it enables the centralized scheduling. The integration into a semantics-aware enterprise platform is very promising because it can directly leverage on the semantic information stored within the platform. The knowledge of measurements allowed us to create a system that can only be reconfigured

after a specified amount of time. We call this a window. The achievable sleeping time depends on this window size. In addition, we tried to increase the sleeping time and the number of sleeping nodes by considering soft real-time tolerances. We combine multiple measurements into one query, considering the overlapping of their tolerances. We added three different possible algorithms to our system: An exhaustive strategy that tries every combination. It yields the optimal solution, but has exponential complexity. We added a greedy heuristic based upon the fact that measurements that can be combined into one query typically follow each other time wise. Thus, instead of evaluating all solutions, only the ones following each other are potentially considered. Finally, we also added a very simple first fit algorithm. The first fit and the exhaustive approach are used as benchmarks in our evaluation. They represent the most optimal solution, within our constraints, and a naive solution.

The actual integration into the MRv6 protocol and the Mote Runner environment required the adaptation of the beaconing and heartbeat functionalities for leaving the network without being recognized as lost and rejoining of the network. For the in-time or ahead-of-time rejoining the network and the calculation of the sleeping times the clock drift is a problem that has to be considered: For non TCXO crystals, as used in the IRIS platform the clockdrift can become considerable over time. The situation for TCXO-based systems, as the Waspmote Pro platform, is quite better and less conservative wakeup times can be achieved. Later on, when we show the evaluation results, it is demonstrated that the network lifetime greatly benefits from the advanced (deep) sleep mode of the Waspmote Pro platform. The evaluation results of our implementation are presented in Chapter 8.8.

Part IV

Evaluation

In the fourth part of this thesis we present evaluation results. The evaluation itself is divided into an empirical evaluation and an experimental evaluation. We first describe the evaluation framework used, before we present the actual results. On the empirical side we present the results of a survey on semantics and IoT, and an evaluation comparing Linked USDL for IoT with related approaches. We apply a software architecture analysis method on our proposed architecture. On the experimental side, we present our results of using OData and CoAP on the Mote Runner environment. We then conclude with describing the results of our sleepy nodes implementation on IRIS motes and on Waspmote PRO.

Chapter 8

Evaluation

This chapter describes the evaluation of our contributions. We apply two different evaluations methodologies. First, we perform an empirical evaluation based on a surveys, stakeholder workshops, and architectural evaluation methods to evaluate our enterprise integration architecture (Chapter 4) and Linked USDL for IoT (Chapter 5). Second, we perform an experimental evaluation of the REST-based protocols OData and CoAP (Chapter 6) and the Sleepy Node implementation (Chapter 7).

In both cases, for the empirical and the experimental evaluations, we first describe the evaluation framework used. The evaluation framework introduces the theoretical background, as well as the used tools and measurement methods. For each part, we then present and discuss the actual evaluation results.

8.1 Empirical Framework

The architecture, as described in Chapter 3 and Chapter 4 and the general Linked USDL for IoT approach, described in Chapter 5, are evaluated using empirical methods.

The empirical evaluation framework consists of three parts:

- An architecture evaluation framework that is used to evaluate the architecture presented in Chapter 4.
- A survey among industry experts and academia with regard to the potential of semantic IoT-integration, including the used and expected technology stack.
- An evaluation of different types of service descriptions based upon existing literature, internal requirements and stakeholder workshops.

We leverage on three different data sources. The three sources were:

1. The drivers we identified in our own work – namely interoperability, enablement of sense-making and enablement of real-time business decision support (internal source)

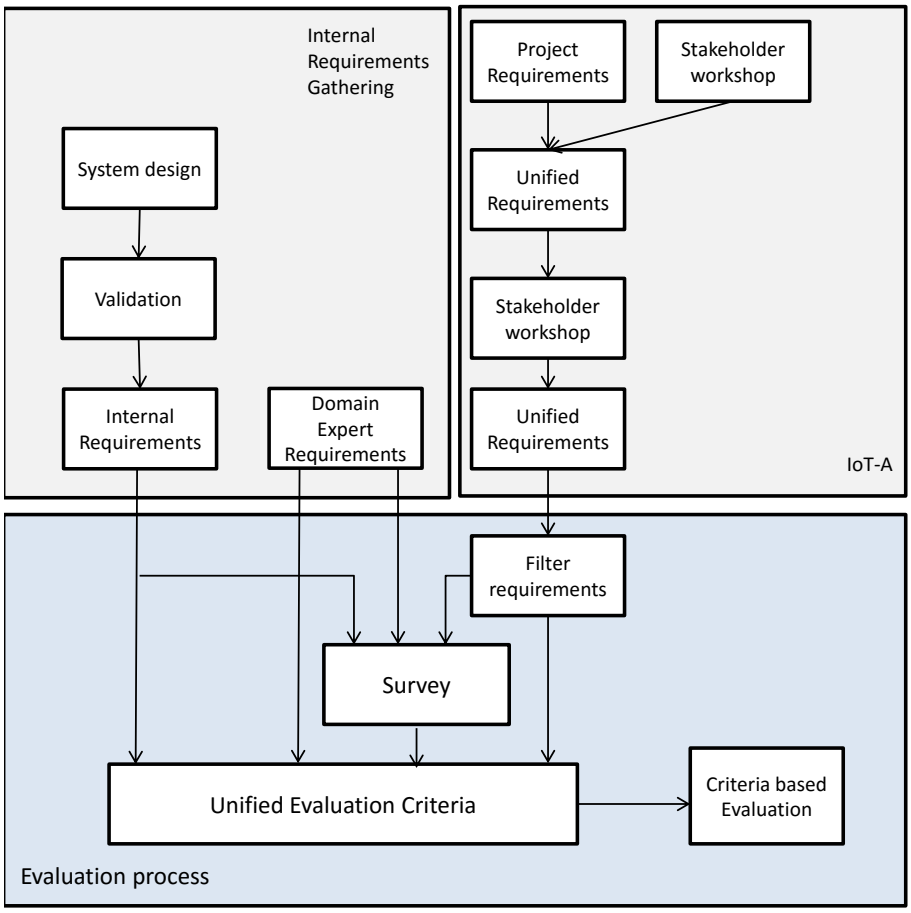


Figure 8.1: Evaluation process

2. The Internet of Things-Architecture (IoT-A) project (external source)

3. Domain experts (internal and external sources)

The evaluation process is illustrated in Figure 8.1. Based on this requirement mining process, we were able to consolidate nine core requirements that an ideal service description language should fulfill. One important first source are the requirements we have towards a semantic enterprise integration system. As these requirements might be biased, we also included further internal and external experts. To gain more insight into the needs of the industry, we gained access to experts knowledge from within SAP and partners. We leveraged on the results of the Internet of Things-Architecture (IoT-A) project. This allowed us to access expert knowledge not only within SAP, but also from our partners and from a selected IoT stakeholder group. The information we gathered from our internal assessment, as well as the insights from the stakeholder workshops

[34, 259] were used as an input for a survey we started in mid-2013 and concluded in early 2015 [376, 377]. This allowed us to broaden the basis of our evaluation. Bassi et al. [34] provide an overview of the general findings of the IoT-A project, including a comprehensive description of the requirements engineering process used in the project. Within the IoT-A project, requirements from a wide variety of stakeholders of IoT-based systems have been collected. Based on this feedback the IoT-A project derived 184 unified requirements[1]. Out of these, we distilled the ones related to semantic services and IoT. These three data sources led to a comprehensive view on the contemporary needs of IoT-system users and providers. First, Chapter 8.2 will present a scenario-based architectural evaluation and its methodology. Chapter 8.4 will present the setup of the survey, its methodology and the results. We will conclude the empirical evaluations in Chapter 8.5 by presenting the foundations of the service description evaluation and then proceed with the actual results.

8.2 Architecture Evaluation Framework

We will use an architecture evaluation framework to assess the proposed architecture that we described in Chapter 4. Evaluation of software architectures is a lively research field that has a large industrial impact [22, 199]. Several software architecture evaluation methodologies have been developed, especially in the last 20 years [24, 347]. Software architecture evaluation can be performed at different points during the software development life cycle. Basically, two different types of evaluations can be distinguished: early evaluation and a late evaluation [238]. Early evaluations are performed during or shortly after the specification phase, but typically before implementation. Late evaluations are performed after the system is implemented and running. They are used to compare against earlier versions of the system or to evaluate changes. Maranzano [245] further distinguishes between an *architectural discovery* and an *architecture review*. Architectural discovery is performed very early in the process with high-level requirements and typically focuses on the desired functionality attributes. Later, but still before implementation, an architecture review can take place to closer examine quality attributes such as performance, portability, and usability more closely.

8.2.1 Architecture Evaluation

A number of software architecture evaluation methods have been suggested. The objective of any software architecture evaluation method is to ensure that the architecture fulfills its requirements. Any evaluation is therefore a validation of architectural decisions with regard to given requirements. Nonetheless, there is little consensus on the technical and non-technical quality attributes that such an assessment method should address.

We perform an architectural discovery based on two methodologies:

[1] available online: http://www.iot-a.eu/public/requirements

- *Software Architecture Analysis Method* (SAAM) [198]. SAAM is is one of the oldest and best-known software architecture evaluation methods. It originates from the Software Engineering Institute (SEI) of Carnegie Mellon University (CMU). It is widely used in academia and industry [105, 30]. SAAM is based on scenarios. Requirements and quality properties are mapped onto scenarios and evaluated. SAAM was originally developed to assess the modifiability of software architectures, but it was soon extended to also cover further quality attributes such as extensionability, scalability, and functionality.
- *Recommended Best Industrial Practice for Software Architecture Evaluation* [1]. This summarizes best practices and lessons learned in software architecture evaluation. It is also published by the SEI of CMU. It is based upon real-world experiences in applying methods as SAAM.

Method	Generality	Level of Detail	Phase
Questionnaire	General	coarse	early
Checklist	domain-specific	varies	middle
Scenarios	system-specific	medium	middle
Metrics	general or domain-specific	fine	middle

Table 8.1: Evaluation Approaches (shortened, [1])

Possible evaluation approaches are shown in Table 8.1. Questionnaires are general in nature and they can be applied to any kind of architectures. Checklists are already more detailed and domain-specific. Checklists in an organization evolve over time and document an organizations past experiences in a domain.

Abowd [30] recommends the sole use scenarios, if there are no questionnaires or checklists available. Questionnaires and checklists for an architecture have to evolve over time. They are often not unavailable in the architectural discovery stage.

SAAM in its original form is a seven-step process where the steps have to be passed through sequentially [199, 23], as illustrated in Figure 8.2. The steps are:

Architecture description The architecture or architectures under investigation should be described in an architectural notation that is well-understood by all involved parties.

Scenario discovery / collection In the first phase scenarios are collected and refined. The SEI, inventors of SAAM, propose [199] to use brainstorming [90] or similar techniques to ensure a broad range of possible scenarios.

Classification of scenarios and prioritization In the previous phase scenarios were only collected. In this phase the collected scenarios are categorized, and a prioritization has to be done. In this phase it is decided which scenarios are to be evaluated.

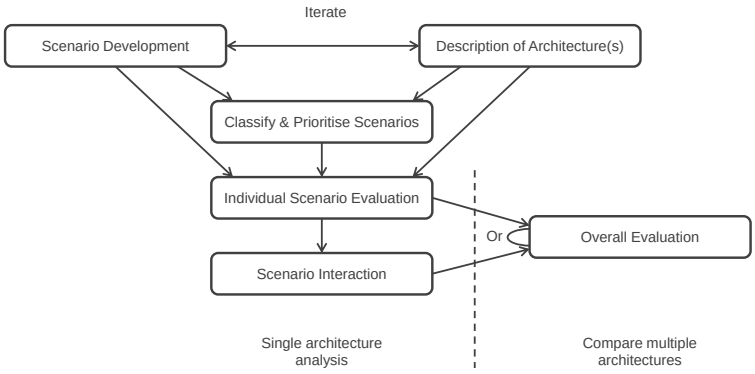


Figure 8.2: Software Architecture Analysis Method (SAAM) [23]

Individual scenario evaluation SAAM evaluates each discovered scenario by mapping it onto the relevant parts of the architecture. It investigates whether the architecture supports the scenario directly. In that case it is called a *direct* scenario. If it does not support the scenario directly, it is called an *indirect* scenario.

Assess interactions of scenarios Different scenarios, which may necessitate changes to the same component are called to scenarios that interact in that component or connector. Scenario interaction measures the extent to which the architecture supports an appropriate separation of concerns [197].

Overall assessment During this phase, the scenarios and their interactions are collected. A weight is assigned to each scenario based on how much it contributes to the overall success of the system. Alternatives for major ”pain points” in the architecture can be collected, if proposed during the process or if there were architectural alternatives.

8.2.2 Architecture Evaluation Methodology

In the following we present the applied methodology based on SAAM and its industrial application. We follow an approach similar to SAAM combined with lessens learned from its industrial application. The approach follows six phases:

1. Development of Business Use Cases
2. Requirement deduction
3. Ranking of Requirements
4. Development of Scenarios and mapping to Ranked Requirements
5. Assessment of Scenarios

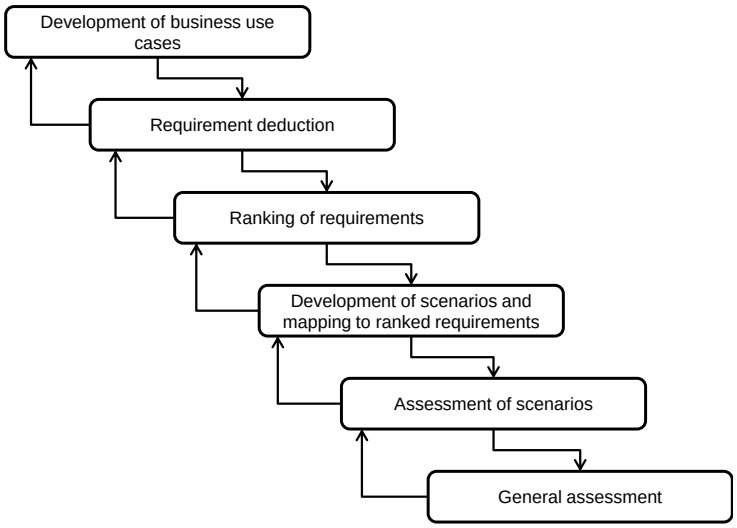


Figure 8.3: Scenario-based Evaluation process used for evaluation

6. General Assessment

The phases are illustrated in Figure 8.3. The phases are sequentially traversed in a waterfall-like process model. If, in a later process phase, an issue with an earlier process step is discovered one has to go one phase back and from there on follow the waterfall again. In the following sections we will discuss each of these phases.

8.2.3 Development of Business Use Cases

It is important for every evaluation to understand the underlying business drivers or business goals and the overall context in which the architecture will be used. We are using business stories as use-cases, similar to the way stories are used in agile methods such as SCRUM [91] or Extreme Programming [254]. Business Use Cases are at a higher level of abstraction than scenarios. Business use-cases should explain the "*why*" and the business context and not the concrete "*what*" and "*how*".

8.2.4 Ranked Requirements

Requirements are used to scope the evaluation. Industrial experience suggests not using more than three to five requirements [30]. The ranked requirements are on the output of the architectural evaluation process.

8.2.5 Scenario-based Evaluation

We perform a scenario-based evaluation. Before describing the scenario development phase in detail, however, we must define what a scenario is and what kinds of scenarios

exist. A scenario is a particular use-case that represents a typical or anticipated use of the system. We use scenarios in the same way that we employed use-cases and user-stories to describe high-level business use cases. Scenarios allow catching not only functional but also non-functional quality properties. Typically, one can distinguish between three kinds of scenarios [26]:

Use case: This it the typical or anticipated use of the final system. Such scenarios reflect the normal state or operation of the system. If the system is yet to be built, these would be about the initial release. The architecture should be able to fulfill a scenario without any modifications to the scenario.

Growth scenario: A growth-scenario is an expected and probable modification of the system.

Exploratory scenario: Exploratory scenarios consist of further, typically deeper, modifications of the architecture. Exploratory scenarios might involve extreme changes to the system.

8.2.6 Scenario Development and Mapping to Requirements

In the scenario development phase typical user-stories are generated. They describe the anticipated use of the system from a users point of view. User is a rather generic term here and be either a human or technical user (software component).

8.2.7 Scenario Assessment

In the scenario assessment phase the scenarios are mapped onto the architecture. For each scenario it is evaluated how the architecture contributes to realizing the particular scenario. Use cases should be supported directly; otherwise, a gap in the architecture has been identified. Growth scenarios should be addressed by the architecture with minor modifications or additions. Exploratory scenarios can help to find the boundaries of the architecture and to name its limitations.

8.2.8 Overall Assessment

This phase collects the scenarios and their interactions and puts them in relation with each other. Furthermore, all other findings during the evaluation are collected and put into context.

8.3 Software Architecture Evaluation

We will now evaluate the architecture as described in Chapter 8.3.1 using the methodology of Chapter 8.2. In Chapter 8.3.1, we first present the business drivers, then describe the requirements and rank them in Chapter 8.3.2. In Chapter 8.3.3 we deduce

scenarios and evaluate them. Eventually, in Chapter 8.3.4, we conclude with the overall assessment.

8.3.1 Business Case

To evaluate our software architecture we use two typical and closely connected IoT use cases: supply chain management and retail. In particular, we will consider the two following cases:

- Transport monitoring with smart load carrier
- Sensor-based quality control

Transport monitoring with a smart load carrier and the Sensor-Based Quality Control are based on the corresponding IoT-A use cases [34]. Having already sketched parts of the transport monitoring scenario in Chapter 4, we will now present its overall context.

The business case assumes a truck on the road or a ship. It illustrates the usage of sensor technology in the supply-chain in order to monitor the quality of perishable goods. The scenario is typically associated with the terms *intelligent container* [222, 108, 241] and *smart logistics* [48, 326]. The sensor information is directly linked to an alerting functionality in order to allow human personnel to react to critical events related to the temperature of the monitored goods. Most important for a deployment in real business situations, the cool chain history is also linked to an electronic delivery note that is exchanged with the retailers backend system upon goods receipt. The business-case shows how sensor-enabled smart load carriers (e.g. containers) may prevent the transported goods from being damaged due to environmental influences. To monitor the frozen goods, for example, while driving on the road, each load carrier is turned into a smart item with the help of a mobile sensor environment. In this context the load carrier is called smart because it is equipped with a sensor node that may communicate with other devices wirelessly. With this hardware every load carrier continuously measures its environmental parameters. A truck driver, for example, may transport frozen goods from a distribution center to a retail shop. The goods reside within or on top of smart load carriers, which know the critical temperature ranges of the food specified as part of an Service Level Agreement (SLA). Upon the loading of the load carriers on the truck, the truck driver is linked to these and is now responsible for them. During the commissioning of each pallet, several batches of goods are loaded on top of it and linked to the pallet identifier. Therefore, the pallet knows which kinds of goods are put on top of it and which environmental values have to be monitored. Drivers load pallets on to their truck according to a loading list given to them.

The sensor-based quality control scenario shows how sensors monitor perishable goods in a store or warehouse. The sensor infrastructure and its measurements are used to estimate the quality of a rare and expensive form of good. Depending on the luminance, humidity, and temperature of the environment, the estimated future quality

of the good is calculated. Prices are reduced, even before a perceivable degradation of quality occurs. Applying this sensor-based quality control and combining it with a dynamic pricing strategy [403, 123], ensures that the store can sell the goods before quality degradation could to occur. Dynamic pricing is a real-time tool for price optimization strategies that has always been crucial for profit maximization. With IoT technologies dynamic pricing is no longer performed on static information such as best before end dates in the transaction data of the backend ERP system, but it is based on real time data gathered from a sensor infrastructure. Up to 20% of perishable goods never reach the consumer [170]. They are disposed of earlier, either in the store or in the supply chain as shown in Table 8.2. The utilization of sensing technology is, therefore, an interesting concept for implement quality control of perishables, thus, reducing waste and increasing profits at the same time.

Type	Percentage	Total (in 10^6 kg)
Fruit	9%	2604
Fresh	14%	1919
Processed	4%	686
Vegetables	6%	3123
Fresh	10%	2314
Processed	3%	809
Dairy products	11%	4245
Fluid Milk	12%	2967
Other	10%	1728
Meat/Fish	3%	1236
Total	10%	19474

Table 8.2: Retail food loss (Estimated food loss in the USA in 2008 [data from [170]], Data for selected goods and total)

The dynamic pricing system is shown in Figure 8.4. Such a system has been developed and is running in the SAP Future Retail Center. It compromises of the following main components:

i Sensing Devices: The sensing devices gather information about the physical world. We use MEMSIC IRIS motes running IBMs Mote Runner system [73].

ii Electronic Shelf Labels (ESLs): The ESLs are used as actors in the system. They show the current price of a given good and are to be attached to the shelves.

iii Linked Services (in a service repository): The services provided by the sensing devices are stored in a service repository, as well as those provided by the Electronic Shelf Labels (ESL). All services are modeled in Linked UDSL. The sensing devices

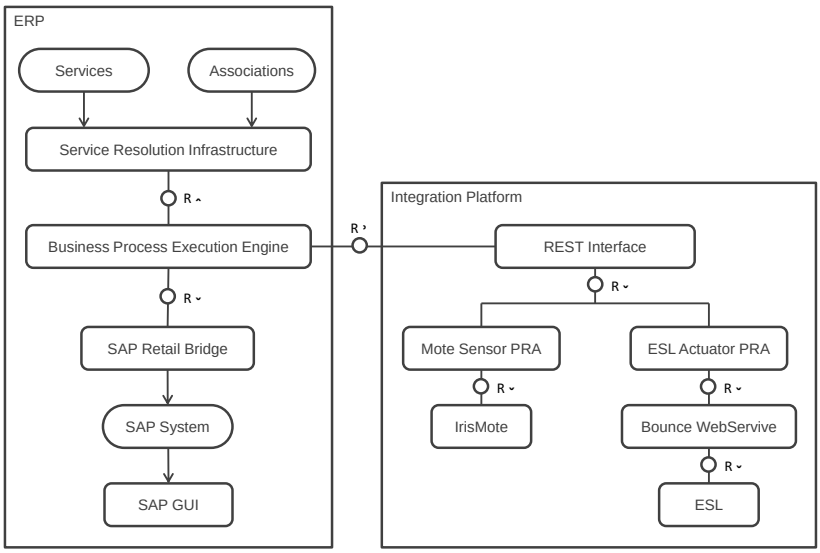


Figure 8.4: Technical Realization of a Dynamic Pricing business process

have a CoAP endpoint, while the ESLs use an HTTP REST interface at application level.

iv Service Resolution Infrastructure: We are using a resolution infrastructure [165] for discovery services. The resolution infrastructure contains the associations between sensing devices, actors and the corresponding service information. It also offers location-based and semantic service discovery functionalities.

v Integration Platform: An prototype integration platform, based on SAP RWIP [242] for communication with SAP systems enhanced with the ability to produce semantics-aware Linked USDL based services (see Chapter 4.6 and Chapter 5), is used in in order to make the wireless sensor nodes and the ESLs accessible. In Figure 8.4 only the essential parts of the prototype are shown.

vi Business Process: The adaption of the retail price, as shown by the ESLs, is managed by an IoT-aware business process [267]. A prototype IoT-aware BPMN modeler was used. It measures the environmental parameters and adjusts the price as needed. The process model contains declarative descriptions of the particular good for which it can be executed, as well as the IoT-related operations (sensing or acting). The execution engine then generates a SPARQL query based on these descriptions and queries the resolution infrastructure for the corresponding IoT services. Upon the execution of the process, the engine accesses the service endpoints. In addition, the execution engine writes the environmental parameters and the prices into the SAP retail system.

8.3.2 Ranked Requirements

The requirements have already been detailed out in Chapter 4.6.1. We briefly repeat the requirements and rank them starting with the most important first. As noted in Chapter 8.2.4 and recommended by Bass et al. [30] we will limit ourselves to the five most important requirements.

#R1 Linked Services: The architecture should be able to use linked services as introduced in Chapter 4.4. It should be able to follow links and download service descriptions.

#R2 Constrained Resources: The architecture must be able to deal with devices that are constrained in terms of memory, computation and communication.

#R3 Reconfigurability and Reprogrammability: Changing requirements and cost pressure will lead to the need of a constant reconfiguration and shared use of resources. Therefore, it is essential that the devices can be reconfigured and reprogrammed easily.

#R4 Service Discovery: The ERP system should be able to discover services provided by the constrained devices.

#R5 Standard Technologies: Use of standard technologies for direct access of IoT devices

8.3.3 Scenarios

In the following we present eight scenarios that were derived from the business drivers in conjunction with the ranked requirements. Five scenarios consider typical use-cases, two are growth scenarios and one exploratory scenario. We use the integration platform as described in Chapter 4 combined with experiences from the business use case, and prototype described in Chapter 8.3.1 as subject under evaluation.

8.3.3.1 Scenario 1

Scenario # 1 is a *use-case* that addresses Linked Services (R#1), constrained resources (R#2) and service discovery (R#4)

Description

The container is arriving at different harbors or at the retailer as described in the business scenario. Linked Service descriptions need to be exchanged, they are stored on the mote. The ERP system later on wants to utilize the semantic data stored on the smart device for reasoning.

Architectural Mapping

On the mote there is a technical endpoint, that runs, for example, CoAP. The service description is stored on-device and can be retrieved upon arrival at the harbor. On the integration platform an agent for the Mote Runner system is running an adapter for the MRv6 protocol. Later on, this allows, direct access to the motes. The discovery agent, in cooperation with the Mote Runner adapter, detects a new mote and adds it to the list of newly detected motes. It is now monitored by the lifecycle controller. The integration platform now downloads its service description. The enterprise system stores the service description in a service repository. From that point on, its available to the industry-specific reasoning engines within the enterprise system.

Conclusions

Based on the architectural mapping the scenario can be fulfilled by the proposed architecture. The major risks discovered are not of architectural nature, but come from standardization. The interface for accessing the service description needs to be either standardized, or at least, known to all adapters.

8.3.3.2 Scenario 2

Scenario #2 is a *use-case* that addresses Linked Services (R#1) and constrained resources (R#2).

Description

In this scenario we assume a very constrained device. We assume that it has some memory to describe its basic services, but not even have enough memory available to store all related service information. It still needs to be able to attach to the enterprise system. A service level agreement and further sensor-specific information is stored on the repository of the shipping company that owns the container.

Architectural Mapping

The scenario can be implemented by the architecture. This scenario shows how linked data principles can be used to obtain further information about almost anything, including hardware specification. The basic service description is processed as in scenario #1. The main difference is that the service now can be completed by the service description agent and the resolve RDF agent. It is assumed that an Internet connection is available. The Linked Services approach allows using a domain-specific ontology to describe service characteristics. For example, the hardware specifications of the sensors involved (luminance, humidity, and temperature) could be described via an ontology and downloaded from the manufacturers homepage, following the links in the original service description. The data can be loaded on-demand. For example, when the service

description agent tries to retrieve the service level agreement, instead of getting the data delivered by the mote, it gets a redirection. It follows the links in this redirection until it has all the data it needs. The data could be stored in an enterprise system of the shipping company. The respective service is stored in a service repository and made available to its partners in the enterprise network.

The description of a basic service also allows using the mote in situations, where no Internet connection is available. In that case, the mote could still be used for monitoring tasks and be synchronized only at harbors.

Conclusions

The scenario can be fulfilled. The challenges and risks concerning linking different sources together are more an a management and standardization level than on a technical level: all involved parties need to agree on some ontologies in order to be able to link services. Availability and correctness needs to be guaranteed, as well.

8.3.3.3 Scenario 3

Scenario # 3 is a *use-case* that addresses reconfigurability and reprogrammability (R#3)

Description

A container should get a new configuration. It has to be reconfigured by replacing the technical service on the mote and its service description. Both are stored on the mote and should be replaced *on-the-fly*.

Architectural Mapping

The enterprise system needs to instruct the integration platform to reflash the device. The lifecycle-controller ensures that the device is marked as invalid for the time being and it is currently reconfigured and reprogrammed. The physical resource adapter of the system communicates with the service management unit of the device to store new data. This can be either done before or after the actual reflashing of the software on the device. It is highly hardware-specific. For the integration platform to perform this operation a (vendor-specific) PRA is necessary, or alternatively some standard has to be established. After the reflashing of the device it is reattached to the integration platform. Its service description is downloaded and integrated into the system.

Conclusions

This scenario can be generally fulfilled by the architecture. Some technical preconditions have to be fulfilled: The mote must be able to receive a new service description, and store it on the mote. Before or after that it also must be able to be ”reflashed”, preferably with minimal to no human interaction.

8.3.3.4 Scenario 4

Scenario # 4 is a *growth-scenario.* It addresses Linked Services (R#1) and constrained resources (R#2), and service discovery (R#4).

Description

A new device technology should be incorporated into the system. It uses a proprietary protocol to attach and reattach to the network and for transportation. The communication at the application-layer is done via a standardized protocol.

Architectural Mapping

The first step is that a new physical resource adapter for this particular device/protocol combination has to be written. The device would need to support reconfiguarability and reprogrammability on the device to support R#4. Otherwise, it is not possible to support that requirement. In addition, access to the network needs to be achieved by providing software drivers for the IPI. The actual communication with the device can then happen via a standardized technology.

Conclusions

This scenario can be generally fulfilled by the architecture. It depends on the actual characteristics of the hardware, if and how many of the requirements can actually be fulfilled. This is not a limitation of the architecture though. In order to improve interoperability of different hardware devices, it would be necessary to further standardize those aspects.

8.3.3.5 Scenario 5

Scenario # 5 is a *growth-scenario* that addresses Linked Services (R#1), constrained resources (R#2), and service discovery (R#4).

Description

The growth-scenario is shares some similarities with Scenario #4. A new device technology should be incorporated into the system. It uses a proprietary protocol to attach and reattach to the network as well as for transportation. The protocol is proprietary and it is not possible that an ERP (or the Internet in general) at any point can directly connect to it. Nonetheless, the ERP should be provided with a service via a standardized technical interface.

Architectural Mapping

The first step is that a new physical resource adapter for this particular device/protocol combination has to be written. The device would need to support reconfiguarabilty and reprogrammability on the device to support R#4. Otherwise, it is not possible to support that requirement. In addition, access to the network needs to be achieved by providing software drivers at least for the adapter. To allow access from the ERP or the Internet in general a proxy solution needs to be written. The integration platform could create a simple proxy, accessible through standard web technologies like HTTP or CoAP. The actual technical interface on the device is then called by the physical resource adapter.

Conclusions

The architecture could support this scenario as well, with minimal changes. One possible way would be to generate a proxy out of a given service description. The service description would have to specify the technical interface towards the ERP. The device needs to provide a means to access the actual functionality. Next, automatic proxy generation could be applied. As long as the device access can be mapped to a RESTful protocol this could be done automatically. If the protocol itself is also proprietary handwritten glue code might be necessary. The ERP would then access the technical interface on the IPI and not on the device directly.

8.3.3.6 Scenario 6

Scenario # 6 is a *use-case* that addresses Linked Services (R#1) and constrained resources (R#2).

Description

The dynamic pricing scenario include ESLs (Electronic Shelf Labels) as third-party actuators. The architecture should be able to include actuators.

Architectural Mapping

Out of scope of the architecture is currently real-time support. The actuators need to be accessible via standard Internet technologies. This currently comes with some limitations: delay, jitter and packet loss needs to be considered, as well as drifting unsynchronized clocks. A scenario like the described electronic shelf label can be integrated easily, because its updates rates are typically not time-critical.

Conclusions

The architecture supports this behavior. Real-time support with immediate response is currently out of scope of the architecture. It can support soft realtime and firm realtime, but hard realtime cannot be achieved without architectural modifications.

8.3.3.7 Scenario 7

Scenario # 7 is a *use-case* that addresses Linked Services (R#1), constrained resources (R#2) and modeling of service characteristics (R#5).

Description

We are assuming a very constrained device. The device is not able to store even a small service description. It can hold some data though.

Architectural Mapping

Even, if no service description is stored on the mote the architecture can still accommodate this situation. The minimal requirement is that there is a endpoint available; running software that can at least return one redirection. This redirection links to an external resource – for example, on an enterprise system of the container shipping company. The enterprise system can connect to the device through an adapter. Now, the service description can be processed as if it were available directly on the sensor mote. This scenario requires the availability of an Internet connection. In rural scenarios or on a ship without a working Internet connection, it would not be possible to connect to the device.

Conclusions

The use-case can be fulfilled. It has imitations though, when no Internet connection is available. As a general recommendation, all motes should provide at least a minimal service description that allows to interconnect with the device. Nonetheless, for the motes to be self-descriptive in all scenarios, a minimal service description on the mote is necessary.

8.3.3.8 Scenario 8

Scenario # 8 is a *exploratory-scenario*.

Description

In future, it might well be that more and more computation (business process decomposition [158]) is happening on the device. As a result, the interactions between the devices and the integration platform or the enterprise backend will increase.

Architectural Mapping

The problem of supporting this scenario does not directly involve the integration platform. Orchestration of the services needs to be done on the ERP side and it does not directly involve the integration platform. In future, additional semantic descriptions

might support such an orchestration engine and it might even allow automatic matching of services.

Conclusions

Business process decomposition [158] support can be achieved by using the integration platform. Nonetheless, all the logic of orchestrating this additional logic needs to be done in the ERP system.

8.3.4 Overall Assessment

We performed a scenario-based assessment of the proposed architecture. The architectural mapping has shown that the proposed architecture in principle is capable of fulfilling the defined requirements and their underlying business drivers. The use of linked services and semantics are on an architectural level able to enable interoperability and scalability. Devices and information from different vendors can be incorporated. Some risks have been identified as well. Given the assumed technology stack (REST-based services, IP/6LoWPAN based, CoAP) it is pretty obvious that the architecture is currently targeting only soft-realtime systems. A move towards firm-realtime or even hard-realtime seems basically possible, but most likely will require changes to the architecture and the protocol stack. Very constrained devices that lack self-description capabilities can be integrated as well. Nonetheless, in such scenarios an Internet connection is mandatory.

8.4 Survey

We conducted a survey on semantics with an emphasis on the potential of semantics in enterprise IT systems. Semantics are currently being intensively researched by both industry and academia. Our objective was to identify the actual needs of IoT with regard to semantic support and to identify current shortcomings. We also investigated reasons for the perceived ”semaphobia”. Lanthaler and Gütl [225] coined the term semaphobia to describe the fear of an average developer to use semantic technologies. We first describe the methodology in Chapter 8.4.1, then the threats to its internal and external validity in Chapter 8.4.2, before we proceed to discuss the results of the survey in Chapter 8.4.3.

8.4.1 Methodology

The survey was distributed among domain experts, from industry (among others: SAP, IBM, NEC, Orange, Telefonica) and academia. While some experts were recruited directly, the majority of the respondents were self-selected. The survey was conducted online, with anonymity being guaranteed and technically enforced by the system. As the IoT is a very broad field, we did not ask questions about mobile phone development,

and limited protocol related questions specifically to systems where an ISO/OSI-like stack [406] is used. The questionnaire was designed to be rather short, as we did not expect professionals to spend more than five to ten minutes on it. To further speed up the process the survey mostly contained multiple-choice and ranking questions and only very few free text entries. The survey included a control question: the question if semantics will play a significant role in future IoT was repeated. Incomplete questionnaires and those failing the sanity check were not taken into consideration.

Some questions ask for agreement or disagreement with statements about the IoT or semantics. We chose a 4-point Likert [236] scale. The Likert-scale allows participants to specify their level of agreement or disagreement, typically on an agree-disagree scale, for a number of different statements. In literature, especially in psychology, there is plenty of research on the optimal number of Likert items and the value of a neutral answer. It is widely acknowledged that the number of "negative" options, should equal the number of "positive" options. Furthermore, the number of people choosing a neutral option decreases with the number of options given [253]. The optimal number of answer options depends on the subject of the survey. For example, to express feelings, having more options, closer to ten, seem to be more favorable [305]. When choosing to agree or disagree with statements, a lower number might be more suitable, as it otherwise remains unclear how different subjects perceive the difference between two choices [81]. Therefore, we decided to use a four-point Likert scale without a neutral option to force people to form an opinion, even if it is only a slight tendency towards one or the other side. The scale is quite intuitive and should not leave the subjects much room for interpretation. They can agree or disagree, or they need to give a tendency in one or the other direction.

We asked the participants specifically to focus on constrained networks and wireless projects, and more specific on IEEE 802.15.4 and similar based protocols. We particularly asked not for mobile (phone) development, as for example an IPad-based shopping assistant. The participants were asked to focus on communication with constrained devices and the constrained devices themselves.

8.4.2 Threats to internal or external validity

As the study was conducted anonymously, it is not possible to validate that the claims made are valid. No incentives were given for participating. Most industrial participants worked with IoT-systems in industrial automation, retail or logistics. The responses by participants from academia were (if a sector was chosen) mainly from automation and logistics, and the broad areas of smart city. The survey was mainly distributed in an environment that is close to enterprise systems and enterprise resource planning, or to semantic systems. Furthermore, most of the projects and the research in the surveyed community are traditionally based on IEEE 802.15.4 and similar. As this was our main research focus and we were particularly interested in the needs and views of this group. We did not survey other IoT-fields – for example mobile development based upon IEEE 802.11. Therefore, some of the results might not be generalizable to other

problem domains or communities. Semantics is one of the major concerns in nowadays enterprise systems and the participants of the survey are probably from a community that is more likely to use semantic technology and more verbose application-layer protocols, than for example, a more ”bit-and-byte”-orientated community, such as the automotive sector. Nonetheless, we assume the results to be generalizable to any enterprise (ERP-) and semantics-prone community. For other communities, further research is suggested.

8.4.3 Results

In the following, we present and discuss the results of the survey. We categorize the results into four groups: (i) general statistical information about the participants and their skillsets, (ii) protocols, (iii) semantics and (iv) enterprise integration.

8.4.3.1 Participants

The total number of participants was 178. Their experience levels, origin and skillsets are detailed in Table 8.3. There were twice as many participants from industry as from academia. The majority of participants had at least three years of professional experience and a more than basic understanding of IoT and semantics. Most participants had at least some experience with semantics and IoT. Naturally, the expertise in enterprise systems and enterprise interoperability was higher for industry participants.

8.4.3.2 Protocols

In the following we give an overview on our results with regard to the used protocol stack. We survey the currently used protocols on the network layer, transportation layer and application-layer. We were mainly interested in the penetration of REST-based protocols and the progress of standardized protocols. In the past two decades, especially 802.15.4 networks were often associated with custom made, specialized and often proprietary, protocols. Industry used a lot of vendor-specific, non standardized protocols. This started to change with initiatives such as ZigBee, 6LoWPAN and a general industry-wide trend towards more integration and standardization.

The views of the community with regard to the current IoT-protocols have been surveyed with a 4-point Likert-style questionnaire. The Likert items and the responses are illustrated in Figure 8.5. First, we asked if current application-layer protocols are considered sufficient for the special needs of IoT-applications. In total, there was a small majority towards more work needs to be done on the application layer. A large majority believes that network management protocols need more work to adopt them towards IoT, suggesting the need for further research in this area. There seems to be consensus that most future IoT-applications will be IP-based to some degree, and a bias towards REST-based architectures. In total 87% of the participants opted for a future IP-based Internet of Things. Especially, the participants from industry (90%) expressed

Participants					
Origin		*Career level*			
Total	178	Entry	23	Professional	64
Industry	114	Industry	9	Industry	47
Academia	64	Academia	14	Academia	17
		Advanced	91		
		Industry	58		
		Academia	33		
Experience (years)					
<=2	23	6-9	44	>=15	5
Industry	9	Industry	34	Industry	4
Academia	14	Academia	10	Academia	1
3-5	93	$10-14$	13		
Industry	58	Industry	9		
Academia	35	Academia	4		
Sector					
Industrial automation	26	Home automation	12	Retail	31
Industry	20	Industry	7	Industry	29
Academia	6	Academia	5	Academia	2
Transportation and logistics	12	Smart City	7	Healthcare	3
Industry	8	Industry	4	Industry	2
Academia	4	Academia	3	Academia	1
Vehicular communications	5	IoT-in general, other	82		
Industry	2	Industry	42		
Academia	3	Academia	40		
Skills (self-assessment)					
IoT		*Semantics (general)*		*Enterprise Systems*	
Beginner	28	Beginner	42	Beginner	37
Industry	17	Industry	28	Industry	8
Academia	11	Academia	14	Academia	29
Some experience	59	Some experience	80	Some experience	43
Industry	28	Industry	48	Industry	19
Academia	31	Academia	32	Academia	24
Advanced	81	Advanced	49	Advanced	84
Industry	61	Industry	34	Industry	74
Academia	20	Academia	15	Academia	10
Expert	10	Expert	7	Expert	14
Industry	8	Industry	4	Industry	13
Academia	2	Academia	3	Academia	1

Table 8.3: Participant group: Experience and Skills. All numbers are total numbers, typically further splited into academia and industry participants. Skill-sets are based on self-assessment.

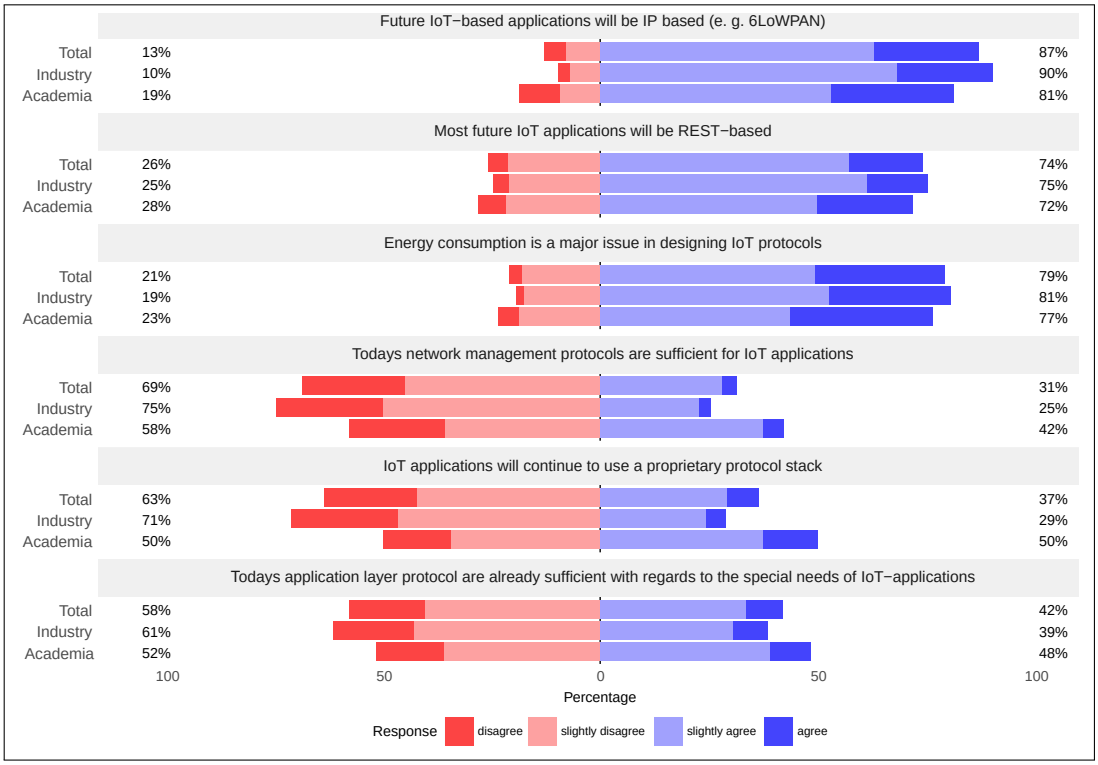


Figure 8.5: Usage and potential of IoT-protocols (on a 4-point Likert scale)

the opinion that the IoT will be largely IP-based. This matches the results given in Table 8.4. We always asked for the protocols that are used at the moment, the planned protocols and the expected industrial use.

Curr Please select the protocols you are currently using in your project(s). Please consider only the protocols you are actually using for communication with your IoT-device.

Planned Considering the near- to midterm future of your project(s) (3-5 years outlook). Please check the protocols that you would most likely use for any new projects, or advancements of your current projects. What protocols would you use right now if you would have to start a new project? Please consider only the protocols you are actually using for communication with your IoT-device

Expected Please select the project you think new projects will use or you generally expect the industry to use in future. What do you think the industry will/should choose in general for future IoT-projects? In which direction do you think the industry will move?

The planned usage and the further expected usage of standard protocols is very high. The HTTP protocol family is the most frequently used in industry. CoAP,

as a second RESTful protocol, also attracted quite some attention. But this was mainly in academia and not as much in industry. MQTT [177] was also mentioned by quite a few participants. Interestingly, the expected usage of CoAP and MQTT is significantly higher than the planned. This could be due to the recent attention that both protocols have received. AMQP [390], an alternative to MQTT was mentioned only very infrequently. WebSockets did not attract widespread use either. WebSockets had to be manually entered by the participants though, so the numbers for the expected use are most likely not conclusive. However, what can be said is that WebSockets did not make it into a generally accepted IoT concept – at least, not to a degree where our participants would naturally have mentioned it "as to be used" protocol. SOAP, which is otherwise widely used in enterprises [132], does not seem to play a major role in the IoT. Surprisingly, ZigBee is not that widely used either. Nonetheless, this could be explained by the surveyed community. The OData protocol is not yet considered to be an "IoT-protocol".

On the network and the transport layers, the answers showed a trend towards standardization as well. The data indicates that two types of IoT-streams coexist at the moment: Developments based upon small very constrained devices with specialized protocols and already quite some developments on platforms capable of running less-constrained protocols, such as IPv6 and TCP. This development is most likely driven by the general need for standardization, in conjunction with advances in hardware, a significant drop in hardware prices and the recent advent of many new platforms. The numbers show that most people expect standardization towards the high-level protocols or their IoT adoptions (such as 6LoWPAN). This correlates well with the expectations of our participants regarding REST-based and IP-based IoT-developments in future.

8.4.3.3 Semantics

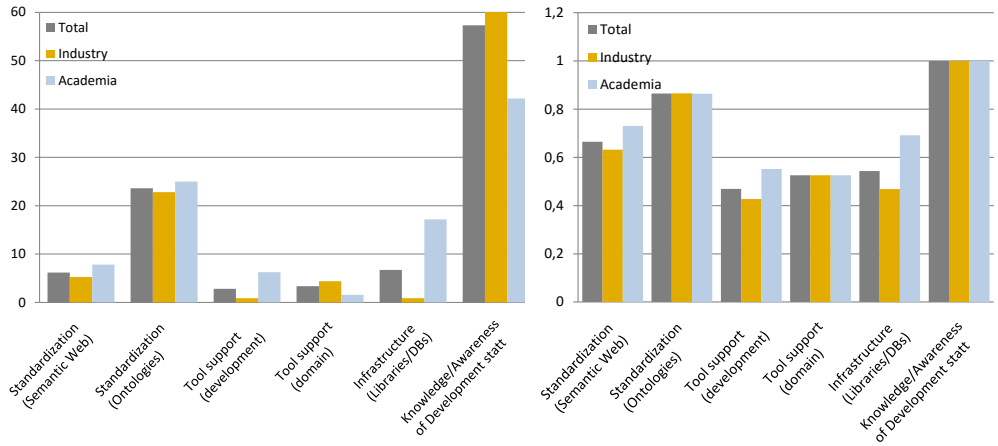
Apart from the used protocols, we were mainly interested in the views and the general attitude of the community with regard to semantic technologies and semantics in general. Semantic technologies and semantics have been around for some time now, but they have not yet gained the widespread use that was once predicted. We asked our partiticpants what are, in their opinion, the main obstacles hindering the widespread adoption of semantics in the Internet of Things. The participants were given a list of six typical reasons that might be an obstacle for the wide-spread use of semantics. They were asked to rank these reasons from the most influential reason to the least influential.

In Figure 8.6(a) we show the main obstacle – that is, the one the participants ranked first. The most frequently mentioned issues were knowledge / awareness of development staff and standardization, followed by development tool support. The differences between the responses of industry participants and academic participants were small. Only the infrastructure seems to be viewed more critically by academia than by industrial participants. Figure 8.6(b) shows the same numbers in weighted and normalized ways. As mentioned earlier, the participants were asked to rank the six possible obstacles from most important to least important. We multiplied all "most

Network layer	Curr	Plan	Exp	Transport layer	Curr	Plan	Exp
IPv4	16.29%	7.86%	6.17%	UDP	25.84%	25.28%	51.66%
Industry	21.92%	11.40%	7.89%	Industry	34.21%	34.21%	56.14%
Academia	6.25%	1.56%	3.12%	Academia	10.93%	9.37%	43.75%
IPv6	29.21%	56.17%	85.39%	Reliable UDP (non CoAP)	6.74%	6.17%	20.24%
Industry	35.08%	64.91%	86.84%	Industry	5.26%	3.50%	14.03%
Academia	18.75%	40.62%	82.81%	Academia	9.37%	10.93%	31.25%
6LoWPAN	45.50%	80.33%	97.75%	Reliable UDP (CoAP)	23.59%	36.51%	68.53%
Industry	49.12%	80.70%	98.24%	Industry	20.17%	36.84%	67.54%
Academia	39.06%	70.31%	92.18%	Academia	29.68%	35.93%	70.31%
Custom 802.15.4 protocol	49.43%	33.14%	20.22%	TCP	47.19%	60.11%	76.40%
Industry	38.59%	19.29%	10.52%	Industry	60.25%	77.19%	78.94%
Academia	68.75%	57.81%	37.50%	Academia	23.43%	29.68%	71.87%
Custom (other)	21.91%	23.03%	10.11%	Custom/other (TCP-like)	36.51%	29.77%	18.53%
Industry	19.29%	21.05%	7.89%	Industry	30.70%	21.92%	12.28%
Academia	26.56%	26.56%	14.06%	Academia	46.87%	43.75%	29.68%
Zigbee	8.98%	9.55%	51.68%	Custom/other	42.13%	39.32%	16.29%
Industry	10.52%	10.52%	53.50%	Industry	33.32%	29.82%	10.52%
Academia	6.25%	7.81%	48.43%	Academia	57.81%	56.25%	26.56%

Application Layer	Curr	Plan	Exp		Curr	Plan	Exp
CoAP	23.59%	40.44%	62.92%	AMQP	1.12%	1.12%	1.12%
Industry	12.28%	32.45%	59.64%	Industry	0.87%	0.87%	0.87%
Academia	43.75%	54.68%	68.75%	Academia	1.56%	1.56%	1.56%
HTTP(S)	54.94%	55.06%	76.40%	KNX	6.17%	5.61%	11.79%
Industry	63.15%	65.78%	75.43%	Industry	7.89%	7.01%	9.64%
Academia	39.04%	35.93%	78.12%	Academia	3.12%	3.12%	15.62%
SOAP	12.35%	8.42%	16.85%	MODBUS	7.86%	8.42%	7.86%
Industry	18.42%	12.28%	20.17%	Industry	10.52%	10.52%	7.89%
Academia	1.56%	1.56%	10.93%	Academia	3.12%	4.68%	7.81%
ODATA	2.80%	3.93%	5.05%	MQTT	20.78%	23.03%	49.43%
Industry	3.50%	5.26%	6.14%	Industry	28.94%	32.08%	57.01%
Academia	1.56%	3.12%	3.12%	Academia	6.25%	6.25%	35.93%
CAN	6.17%	6.17%	5.06%	WebSockets	2.24%	2.24%	1.68%
Industry	7.89%	7.89%	6.14%	Industry	2.63%	2.63%	1.75%
Academia	3.12%	3.12%	3.12%	Academia	1.56%	1.56%	1.56%
Other/Custom	29.77%	29.21%	29.77%	Zigbee	13.48%	13.48%	38.20%
Industry	20.17%	15.78%	23.68%	Industry	15.78%	16.66%	47.36%
Academia	46.87%	51.56%	40.62%	Academia	9.375%	7.81%	21.87%
None	4.49%	4.49%	1.12%				
Industry	0.00%	0.00%	0.00%				
Academia	12.50%	12.50%	3.13%				

Table 8.4: Used Protocols, in percentage of the participants (in total, and per group) choosing the respective protocol. Participants could choose more than one protocol. The responses are categorized into current usage (*curr*), planned knowledge (*plan*) and the anticipated or expected use of the industry in future (*exp*)



(a) Main (percentage)

(b) All (weighted, normalized)

Figure 8.6: The main obstacles for not using semantic technologies (”semaphobia”). Participants had to rank the reasons from most important to least important. Results in percent of main reason (top ranked) and all reasons weighted by importance

important” entries with six, the second most important with five and so on. Here, it becomes even more evident that education of developers and standardization is considered the main obstacle for the use of semantic technologies. Tools support, for both domain experts and developers did not rank very high. Academia considered tool support for domain experts irrelevant, while industrial participants gave it only a minor importance. We found this quite surprising as semantic tool support for business specialists is still considered a weak spot in the business modeling community [47, 169, 55]. Nonetheless, the participants’ views might be due to the fact that system designers are not yet fully convinced of semantic technologies, and therefore the importance of business users is not yet particularly high and therefore not visible yet.

The attitude of the community towards semantics, and more explicitly semantics in the IoT was surveyed with a four point Likert-style questionnaire. The individual Likert items and the distribution of the answers are shown in Figure 8.7. Most participants agree that semantics will play a role in future IoT systems. Some think that it is too bloated/an academic toy, or as one of the participants stated in a personal mail to us that it is ”a hype from bored academics that no one will remember in a few years”

Next, we proceeded to ask the participants about the actual usage, opportunities and main advantages of using semantic technologies. We presented the participants with a list of seven opportunities of semantics in the Internet of Things as identified by us. The seven opportunities are:

Self-healing If an overall system is able to detect the situation that individual parts of it do not work as intended and automatically takes measures to either repair or replace these parts, the system is said to be self-healing [186]

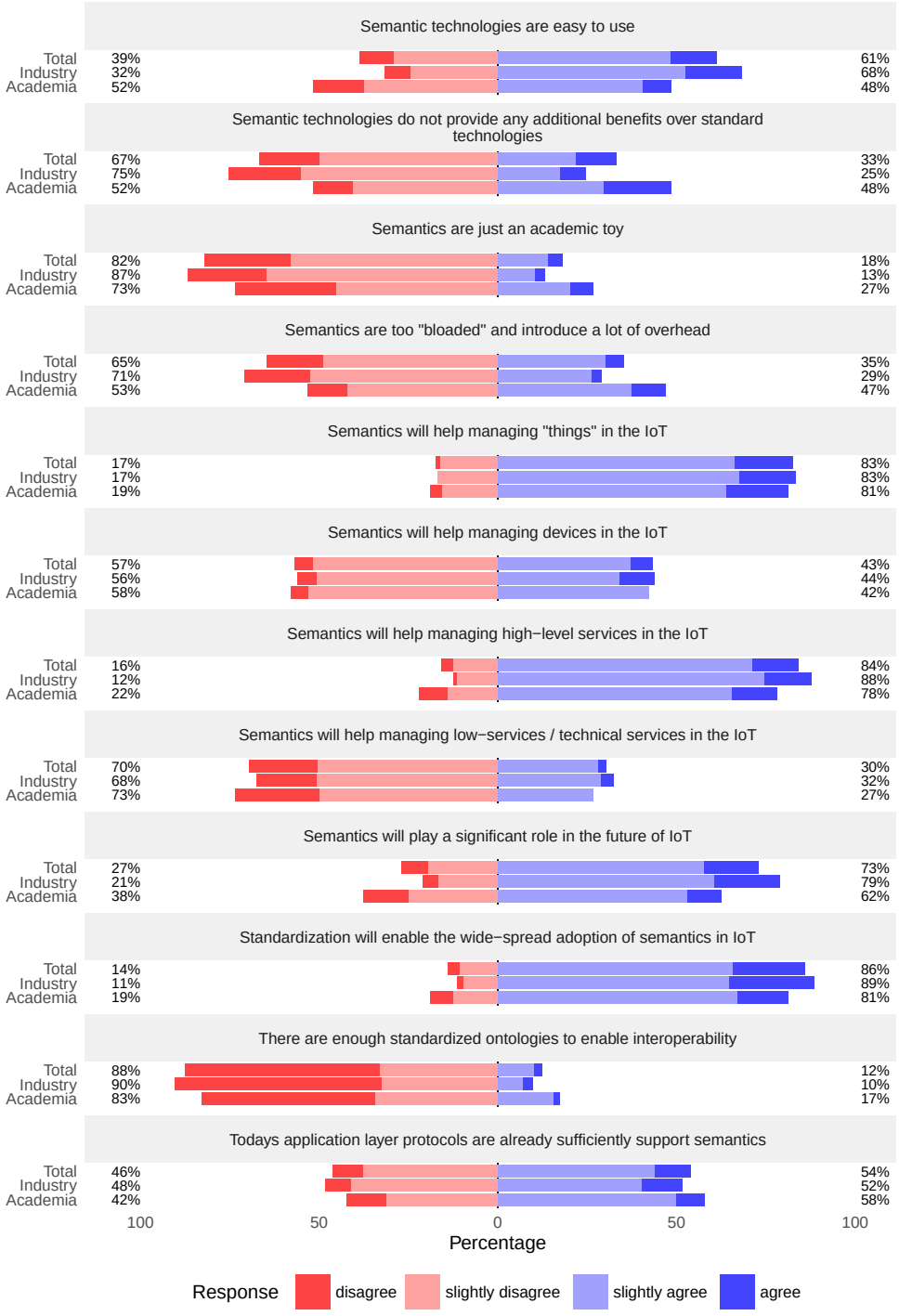


Figure 8.7: Usage and potential of semantics (on a 4-point Likert scale)

Self-configuring A self-configuring system can be described as having the property that it does not require manual configuration before it is ready to be used. In other words, a self-configuring system automatically adapts to its environment and to the needs of the user. [186]

Reasoning/Sense Making Reasoning enables backend systems to automatically perform reasoning on data coming from various sources. Semantic annotations can to be understood and processed by enterprise IT systems.

Device Management Device Management is concerned with information about the used devices and their capabilities.

Management of Things Things, or entities, are the subjects of the Internet of Things. This can be any physical object of interest to a system. Semantics make properties of and connections between things explicit.

Interoperability at technical level Interoperability at a technical level is concerned with the description of the technical interface of services, e.g. what calls can be made to a technical endpoint, how the resources look like.

High-level interoperability Interoperability is not restricted to the technical layer only. Interoperability at a semantic level allows systems to not only know that the received value is a float, but that it is a float representing a temperature with a given precision and that it is monitoring a specified good.

The participants were given the same short definitions. They were asked where they see the most opportunities for semantics to make an impact on IoT. They could choose from a range of none to any. The results are presented in Figure 8.8. The main advantages of semantics are seen in high-level interoperability and reasoning, followed by the management of things. Interoperability at an endpoint level, a topic that is also addressed by our integration platform, is not yet seen as an area where semantics can contribute a lot. The differences between academia and industry were negligible.

Given that most people think there is potential in semantics, it is interesting to see its actual usage: We summarized the results in Table 8.5. Participants could choose from a list of widely-recognized uses of semantics and select what they are using it for today, if they do. They were asked to express what their short- to midterm plans are and what they think the industry will move towards. In this case, the question was not where they see most benefit or most opportunities, but to which degree they are using the technology and what they think will happen.

Reasoning seems to be one of the most important topics discussed currently. Given that research on protocols and energy efficiency has been conducted for more than two decades now it is, in our opinion, just the logical consequence that IoT-research moves one layer or two layers up and is spending more resources on application topics. Considering the huge interest in reasoning, it could also be the main driver behind describing things, services and devices.

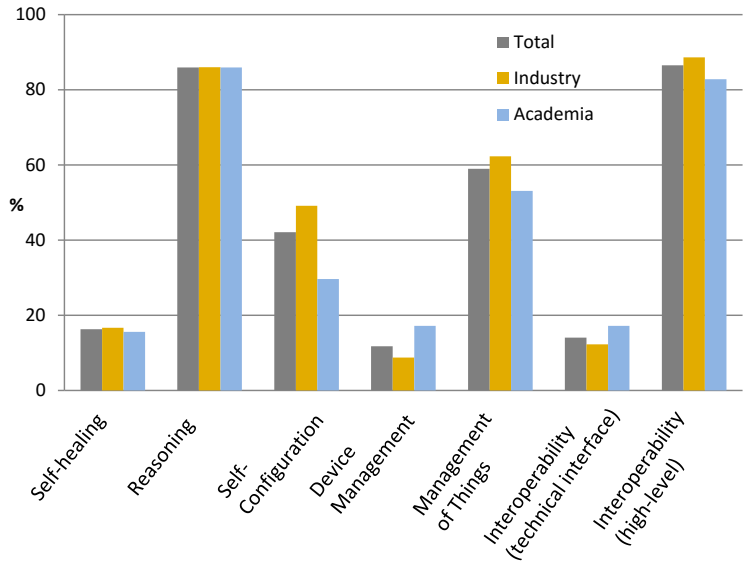


Figure 8.8: Attitude towards opportunities/usages of semantics in IoT (in percentage of participants per item, multiple selections were possible)

This part of the survey was followed by a question that asked for any specific IoT-related ontology people could think. There were some mentions of enterprise internal ontologies. The only ontology that was mentioned more often was SSN.

8.4.3.4 Enterprise Integration

We were also interested in enterprise or, more generally, ”outer world” integration. The Internet of Things vision, to some degree, assumes direct use- and addressable smart objects. We call that direct access, which means with only transparent network devices, if any. A network device is called transparent, if other devices on a network do not need to be aware of its existence. On the other side of the spectrum are gateway solutions, where communication is done via some intermediary that also offers the services. In such a setup, for example sensors, do not appear directly and they are not directly addressable or known to the service consumer. We call this a gateway access. A typical example for a gateway solution is some enterprise monitoring tool, where the sensors are connected through a specialized protocol to the gateway. On the gateway the date is collected and the gateway is the interface towards the outer world. Also, many home automation systems follow the same principle. The participants had to choose if their (main-) product/prototypes fall into or are closer to the first or second scenario. In total, direct access is used more often than a gateway solution (57.86% vs. 42.13% of all participants). Industry participants choose direct access (64.9%) more often than participants from academia, while academia had a slight bias towards gateway (56.2%).

Usage			
	Curr	Plan	Exp
Description of Things	28%	46%	82%
Industry	32%	51%	86%
Academia	22%	37%	73%
Description of high level (semantic) services	14%	33%	73%
Industry	15%	43%	76%
Academia	11%	14%	68%
Description of low level (technical) services	6%	14%	28%
Industry	4%	9%	25%
Academia	8%	20%	34%
Description of Devices	11%	28%	61%
Industry	11%	29%	69%
Academia	11%	23%	48%
Reasoning	38%	69%	88%
Industry	42%	77%	91%
Academia	29%	46%	82%
Configuration	3%	5%	25%
Industry	3%	4%	24%
Academia	5%	6%	26%

Table 8.5: Usage of semantic technologies for the mentioned reason (in total, and per group) in percent. Participants could choose more than one reason. The responses are categorized into current usage (*curr*), planned knowledge (*plan*) and the anticipated or expected use of the industry in future (*exp*)

Direct Access seems to be more common now, than gateway solutions. In this question, the participants were forced into one option and should think about their current main project. Nonetheless, in our opinion both options are valid in the Internet of Things. A sensor node that is accessible via HTTP over a more powerful gateway is in our belief not necessarily worse than using direct access. Gateway solutions help reducing the total energy consumption by applying specialized protocols and many solutions working or doing research with such specialized protocols might still choose a gateway approach.

We were also interested in how the IoT-devices are configured, for example if they can automatically connect to a network without any manual work, or if the end user has to configure the device or if some kind of administrator has to configure the devices first. We differentiated the following four scenarios:

Fully-automatic: The configuration of the IoT devices is automatic, after an optional initial setup or one-time network configuration. Devices can potentially be added and removed with further configuration steps.

End-user: The end-user is responsible for the configuration of the device – for example

by entering connection data.

Administrator, on-site: A technically skilled administrator has to be on-site to add and remove devices and to execute some configuration steps that go beyond what an (less experienced) end-user can do.

Administrator, semi-automatic: A technically skilled administrator has to execute some configuration steps on a regular basis that go beyond what an (less experienced) end-user can do.

Fully-automatic and semi-automatic configurations are used most often. On-site configuration, where a technically skilled administrator has to work on-site, is also common in industry to some degree. There is a huge gap between academia and industry in manual end-user configurations. It is widely used by academic participants and only seldom by industrial participants. It seems likely that the end-user, in many cases, is the research team itself.

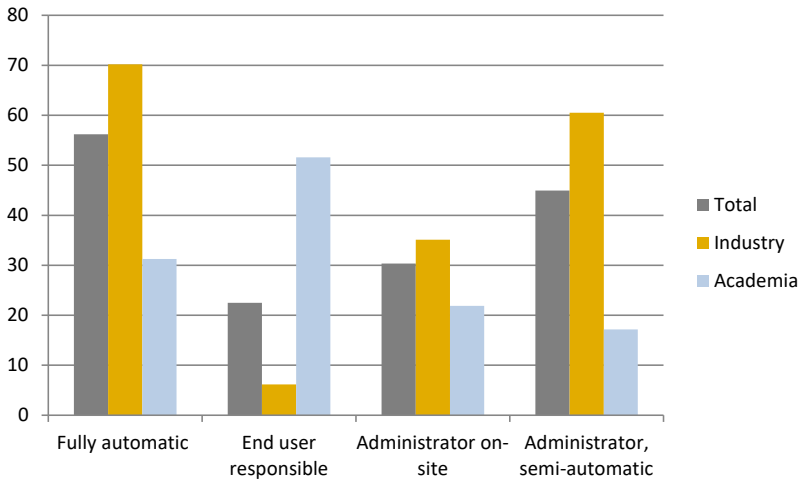


Figure 8.9: Internet of Things - Configuration (in total, and per group) in percent of all participants selecting this option. Participants could choose no, or more than one option.

Finally, we asked two questions regarding data gathering, data aggregation and data processing for monitoring tasks. Participants could choose to answer one or both questions, or to skip the questions. First, we asked where the processing of the monitoring data or the data aggregation takes place. We offered three different possibilities: on the device, on the gateway or in a backend-system. Second, we asked where the view of the system materializes and how it is gathered. The options were: device level (autonomous), centralized (event-based or time-based push), centralized (poll), or centralized (stream). The results are summarized in Table 8.6. Data processing and aggregation is mainly happening on a backend-system or on a gateway. Typically,

it seems that devices push information to the monitoring application in either an event-based or a time-based way. Stream processing is not yet widely used.

Processing of tasks / data aggregation	%	Overall control and view on application	%
device level	11.79%	device level (autonomous)	15.16%
on a gateway	26.40%	centralized, push, time-interval	14.04%
on a backend-system	34.26%	centralized, push, event-based	25.28%
		centralized, poll	11.79%
		centralized, real-time stream	8.98%

Table 8.6: Data processing and overall system (application) view (non mandatory question), answers in percentage of all participants choosing this option

8.4.4 Conclusions

We conducted a survey among domain experts from industry and academia. On one hand, to get an overview of the state of semantics, the Internet of Things and its relation to enterprises. On the other hand, the survey served as an input to our evaluation of Linked USDL for IoT. First, we surveyed the general feeling about Internet of Things and Semantics. It seems to be widely anticipated that the (future) Internet of Things will be largely REST-based and IP-based. This might not be surprising at a first glance. Nonetheless, abstracting from the buzzword IoT and taking the history of the wireless sensor networks into account, this was not a compulsory development, especially in an enterprise environment. First, because a considerable part of nowadays IoT community developed out of the sensor and actor community, as well as automation. In the past, both communities were to a considerable extend grounded in (sometimes proprietary) protocols that were neither IP-based nor REST-based, but still many solutions have been advertised as being part of the IoT. The IP or Internet part in such solutions, if at all, was often not more than a gateway that provided some services. The services themselves were technically realized through specialized protocols between the gateway and some constrained devices. Second, also the enterprise community, which has only very recently moved towards REST-based protocols. In the past, many enterprise companies based their products on either SOAP-services or proprietary protocols, such as RFC and BAPI. Specialized protocols still have its share in the current IoT, but most of our participants are already using standardized technologies or plan to move towards standardized technologies, so a convergence towards REST-based and IP-based protocols can be expected. It is interesting that Enterprises move towards IoT-protocols, instead of moving enterprise protocols (as SOAP) towards IoT. CoAP, to some degree, seems to be in a special position. It was used by many participants from academia, but not by many from industry. Nonetheless, many industry participants nonetheless stated that they will consider it for future projects. In our opinion, it remains open whether

the protocol really will get a considerable industrial usage, or if the expected usage is mainly based on the recent hype the protocol has received in the IoT community. The surveyed community was split (see Figure 8.5) when asked if todays application-layer protocols sufficiently address the special needs of IoT. The used protocols might change in future, with increasing hardware capabilities: The use of standard Internet protocols, suggests that more capable hardware is used now, something that was not always assumed as given in the past. This might further influence the protocol stack towards the use of4 existing and mature technologies, or the new upcoming developments in the Internet such as HTTP/2 [40] or SPDY [116]. Even SOAP might experience a revival then. Applications could also use a more powerful node as application gateway that uses a specialized protocol for communication with sensor nodes and a standardized protocol for communication with the outer world. It seems that the aspect of direct communication with devices, as predicted by the IoT vision, is becoming increasingly common. Significantly less participants were using a gateway approach than the number direct communication. Combined with the expected IP-protocol use, this number can be expected to grow in the upcoming years.

The overall picture with regards to semantics is even more diverse. The general applicability and reasonability of semantics seems not to be denied. Nonetheless, the use of semantics is still not that high, a phenonemon coined ”semaphobia”. The main current and expected usage of semantic information seems to be for reasoning and for increasing interoperability. This is not surprising as sensor data is predestined for basing business decisions on them. As the semantic technology has not been surveyed, it is unclear if there is an end-to-end semantic scenario or if the sensor data is transformed into some semantic representation to be able to do reasoning. Description of Things ranked second in usage, most likely this is due to its usage in reasoning applications. Still, the reluctance to use semantic technologies in general was also observed by us. Quite some participants agree to the statement that semantic technologies do not provide additional benefits over standard technologies. Reasoning, for example, does not necessarily need to use RDF or other (standard) semantic (web-) technologies. The top reason for ”semaphobia”, is seen in a lack of training and awareness, followed by a lack of standardization in terms of vocabularies and ontologies. Standardization of the actual semantic technologies or support from tools and infrastructure (e.g. triple-stores) did not rank high. We assume that with the next generation of computer scientist joining the industry the acceptance of semantics will raise, as many of those will already have had contact with machine-learning, reasoning and semantics during their studies.

8.5 Linked USDL for IoT

In the following section, we will evaluate Linked USDL for IoT and several other description languages with regard to nine selected evaluation criteria.

8.5.1 Introduction

We present a qualitative evaluation comparing Linked USDL for IoT with more traditional technologies like WSDL, and competing semantic approaches like SA-WSDL and OWL-S. A brief introduction into those technologies and related concepts can be found in Chapter 2. First, we briefly introduce the used methodology and present the nine evaluation criteria that we use to evaluate the different service description languages. We then evaluate the semantic and non-semantic service description languages, including Linked USDL for IoT, with regard to this evaluation criteria. Finally, we present the conclusions that can be deduced from our evaluation.

8.5.2 Methodology

We perform a criteria-based evaluation based upon an internal requirements-gathering process, the IoT-A stakeholder workshops and our survey (unified evaluation criteria). The applied evaluation process is shown in Figure 8.1. The internal requirements and the results of the IoT-A stakeholder workshops influenced the survey and they were also used in the unified evaluation criteria. Based on the requirements mining process, as described in detail in Chapter 8.1, we distilled the following nine properties that a service description has to fulfill and six characteristics that a corresponding IoT-system should have. First, we introduce the nine service description related characteristics:

Endpoint agnostic: Endpoint agnostic means that the service description should be as independent of any specific technical endpoint technology as possible. The different technical endpoints in a sensing enterprise are typically heterogeneous. There exist proprietary protocols, like RFC or BAPI, webservices (SOAP), semantic web-enabled interfaces (often RDF-based), as well as specialized protocols like CoAP to mention only a few. Endpoint refers to any of those technical realizations of a service. Any service description language that is chosen needs to support such formats and to describe them in a semantically interoperable way.

Support of Business Aspects: A service description needs to provide more information than just the mere technical details on how to call a technical interface. Therefore the selected service description language should be able to express business aspects like Service Level Agreements (SLAs) or pricing.

Distributed Descriptions: Nowadays, services can be composed of parts from different vendors. We foresee the need of storing parts of a composed service description at different locations. For example, the Quality of Information parameters of a specific sensor node can be stored on the ERP of the sensors' vendor. The service description should be able to capture such dependencies.

Self-Description capabilities: Service descriptions should be able to allow smart items to describe themselves. The service description should ideally be stored on the smart item itself.

Quality of Information: In sensing environments is the capability of describing quality of information (QoI) parameters particularly important . Sensors often have a limited sensing range or work different in special conditions. The service description should be able to express such QoI parameters.

Semantic Extensions possible: To enable reasoning and interoperability on a semantic level the service description needs should be customizable/extendable for specific domains.

Infrastructure support: Infrastructure and tool support is essential in an enterprise environment. A service description, which does only exist in paper, but without tooling support for using it as part of a programming language or store it in appropriate repositories is not sufficient for professional use.

Standardization: Standardization of one of the key drivers of interoperability. Therefore, we evaluated the service description languages also towards standardization or standardization efforts.

Discovery of further information / knowledge base: A service description can be used as a base to find further information from a knowledge base, for example about the sensed entity.

We compared the Linked USDL approach first with other service description languages, both semantic and non-semantic. We compared Linked USDL for IoT with WS-*, hRESTS, WADL, OWL-S and SEREDASj. The WS-* family was selected because of its widespread use. OWL-S is represents a semantics-based language mainly using WSDL. SEREDASj represents a recent approach based on JSON-LD.

We carefully examined each service description language. To control an examiners bias we used the survey to determine the importance of these properties for possible users of a service description language. The results of our evaluation are presented in the next section.

8.5.3 Evaluation

In the following, we present the results of our evaluation. As described in the previous section we identified nine properties related to service description languages. We used these properties as evaluation criteria. First, we present a grading of the nine evaluation criteria to show their perceived importance. Next, we perform a criteria-based evaluation. Finally, we combine the findings of the grading and the criteria-based evaluation into a multi-variant comparison.

8.5.3.1 Evaluation Criteria Grading

As part of our survey, the participants where asked to grade each item by their perceived importance, as shown in Table 8.7. The scale ranked from one point (low importance)

up to seven points (high importance). Each item was graded separately. No ranking between items was enforced. We then classified the nine properties based on the average values into three classes: high importance, medium importance and low importance. The average values formed three clusters. One outlier with an average value of 2.69. The remaining eight properties formed two clusters: four of them are between 4.6 and 5.2, the other four have average values larger than 5.5. The classification and the average values are depicted in Table 8.7.

	Avg	Importance
Endpoint agnostic	2.69	Low
Business Aspects	5.08	Medium
Distributed Descriptions	5.19	Medium
Self-description capabilities	4.67	Medium
Quality of Information	4.83	Medium
Infrastructure	5.62	High
Semantic Extensions	5.52	High
Standardization	5.88	High
Discovery of further information / knowledge base	5.56	High

Table 8.7: Grading of the different evaluation characteristics (external, survey)

8.5.3.2 Criteria-based Evaluation

We now present the results of the evaluation. We compared Linked USDL for IoT with WS-*, hRESTS, WADL, OWL-S and SEREDASj. The results are summarized in Table 8.8. Each attribute was given a value on an nominal scale ranging from does "not fulfill requirement at all" [-], "can be supported by additional modules or is generally supported by the chosen paradigm" [0], "supports requirements with some constraints or limitations" [(x)], to "fully supports requirements" with little or no limitations [X]. In the following we will examine each property individually.

Endpoint-technology Agnostic The property "endpoint-technology agnostic" was the only one to be given a low priority in our survey. For further analysis we divided the property into three sub-properties: We distinguished between SOAP and REST, where REST stands for all kinds of REST-based system regardless of the actual technology in which the endpoint was realized (e.g. CoAP or HTTP) or its maturity (see Chapter 2.10.3). Furthermore, we looked into how far the service description could be used to model any arbitrary endpoint technology or paradigm. Linked USDL for IoT, OWL-S and to some extend the WS-* family support different endpoint technologies. Linked USDL for IoT was designed with REST-based endpoints in mind, but can support SOAP/WSDL and integrates well with a variety of data representations. OWL-S,

	L-USDL4IoT	WS-*	hRESTS	WADL	OWL-S	SEREDASj
Endpoint-technology agnostic	X	(x)	0	0	X	0
SOAP	(x)	X	-	-	X	-
REST	X	(x)	X	X	X	X
Arbitrary Paradigm	X	-	-	-	(x)	0
Business Aspects	X	(x)	-	-	(x)	0
Service Level Agreements	X	X	-	-	-	-
Pricing	X	-	-	-	-	-
Business concepts integration	X	0	0	0	0	0
Quality of Information	X	0	-	-	0	0
Self-description capabilities	X	X	0	X	X	X
Distributed Descriptions	X	0	0	0	X	0
Semantic Extensions	X	0	0	0	X	X
Infrastructure	0	X	0	0	0	0
Standardization	0	X	-	0	0	0
Discovery of further information	X	0	0	0	X	0

Table 8.8: High-level qualitative analysis of the capabilities of different service description languages

supports WSDL/SOAP by default, but could also be used in conjunction with REST. The WS-* family is a special case: the widely used, and widely with WS-* associated WSDL 1.1 [88] specification, only supports SOAP-like interfaces. WSDL 2.0 [86] added support for REST-based systems, mainly through the HTTP binding extensions [85, 244]. However, it did not gain widespread use. In theory, other extensions could be provided. However, no such attempts are known to us. All the other service description languages are either bound to SOAP or the REST-paradigm. hRESTS, WADL and SEREDASj were build for RESTful, mainly-HTTP based, technical interfaces.

Support of Business Aspects Support of business aspects is a requirement mainly influenced by our definition of IoT-service as something that goes beyond a pure technical interface. In the evaluation this favors languages that already have a more comprehensive view of the term ”service”. All semantic approaches would be extensible to support business aspects. We again selected three sub-aspects of the general requirement: (i) service level agreements, with the restriction that it should also be

possible to model non-technical requirements (ii) pricing, as an example of a pure business aspect and (iii) general business concept integration as an indicator of whether concepts from any business domain could be added.

As linking ontologies was one of the main design goals of Linked USDL, the strength of our Linked USDL for IoT approach is to seamlessly integrate all kinds of business aspects. As the original research goal of its predecessor USDL was to support business aspects, its successor Linked USDL also comes with modules for Service Level Agreements and Pricing. OWL-S and SEREDASj generally can support business aspects. Several ontologies exist that can be used with OWL-S for describing service level agreements, for example DAML-QoS [404] and QoSOnt[109]. Compared to Linked USDL for IoT, which is able to describe the base actors of service interactions and which was designed to connect service actors and further ontologies, OWL-S would need more work to model general business concepts. Nothing specific is known to us with regards to SEREDASj. Nonetheless, it is in principle capable of using business-related ontologies as well. The whole modeling would happen outside of SEREDASj, though. The WS-* family supports the description of some business aspects. WS-Agreement [12] and WS-Policy [25] could be used to model service level agreements.

Self-description Capabilities All description languages generally provide self-description capabilities. The only exception is hRESTS being a microformat that usually needs to be delivered in conjunction with a describing webpage. Hence, the effort to provide self-description capabilities is therefore considered a magnitude higher than with one of the other approaches. The issue that typically needs to be solved is the size of description, in case the smart item does not provide enough storage. One way is to compress the descriptions; the other is to store as much as possible on the smart item and distribute most (maybe optional, or advanced) parts on the Internet.

Distributed Descriptions All semantic approaches generally allow service descriptions to be distributed. Typically, they do not have to be stored on a single repository. Linked USDL for IoT, OWL-S and SEREDASj can therefore support distributed descriptions. The Linked Services Architecture (Chapter 4), which uses Linked USDL, supports this property as it is one of its main design goals. WS-* follows more an ”all or nothing” mentality, but some parts can be distributed. For example, a WSDL can be combined with a corresponding WS-agreement file. hRESTS and WADL do not specifically support the distribution of descriptions. To our knowledge, only Linked USDL for IoT particularly emphasizes the distribution and linking aspect to decrease the amount of data to be stored on the mote and to enable interoperability of IoT devices.

Semantic Extensions While USDL and OWL-S started as conceptual languages for describing services at a higher level, and not necessarily technical endpoints, WSDL and WADL started as interface descriptions. WSDL, as well as its REST-counterpart WADL,

were originally designed to describe technical endpoints in a syntactical way, and only to some degree in a semantically interoperable way. Therefore, semantic extensions were designed to annotate semantic information to these languages. SA-WSDL (Semantic Annotations for WSDL) or SA-REST are examples of such extensions.

Infrastructure and Tooling WS-*, in particular WSDL, is currently the most widely used service description language for describing web services. It also has the most complete infrastructure and tooling support. The core standard, WSDL, has excellent tooling support. This does not necessarily include all further standards and extensions. Support of semantic modeling with SAWSDL, for example, often has to be implemented manually on top of general WSDL libraries. WADL and hRESTS do not have significant tool support. Semantic Web based languages (RDF/OWL) have gained significant attention recently. Tooling support has increased a l3ot over recent years and libraries to manipulate RDF/OWL files are now available for all major languages. Nonetheless, it is still much behind the infrastructure and tooling support that the WSDL-family offers. Furthermore, while RDF/OWL tooling support is increasing, semantic service support is not. There is a Protege[2] Plugin for OWL-S[3] [115]. Although, active development seems to have stopped around 2007. Only one maintenance release has been published since. Linked USDL provides a web-based service editor[4]. Service marketplaces and service repositories are also being developed [107], leading to a complete service eco-system. Some tools support WADL, for example the GlassFish EE Application server [131]. No extended tooling or infrastructure support is known for hRESTS. SEREDASj can, to some extend, built upon the tooling and infrastructure support of JSON. Nonetheless, no extended tooling or infrastructure support is known. As only WS-* has a rich set of specialized tooling and infrastructure, combined with a very lively community we rated all but WS-* with a rating of 0. WS-* clearly is the gold standard regarding this criteria.

Standardization The W3C family, especially WSDL, is currently the most important standard used to describe services. All the large enterprise systems provide SOAP support to some degree. In total, there are more than 150 specifications with a different degree of standardization as part of the W3C or OASIS. All other service descriptions are not even close to that degree of standardization. A W3C unified service description language incubator group [196] was initiated in 2010. It concluded its work with a reworked USDL specification [28]. The standardization efforts so far have not lead to a W3C (or other) standard USDL. Meanwhile, its successor Linked USDL has some enterprise sponsors that could eventually lead to a further standardization efforts or the establishment of a de-facto standard. For this reason standardization was ranked with a ”0”, as it is an ongoing effort and backed by a community. WADL was submitted by SUN Microsystems to the W3C. Nonetheless, the community did not pick-up the

[2] http://protege.stanford.edu

[3] http://owlseditor.semwebcentral.org/index.shtml

[4] http://www.linked-usdl.org/ and https://github.com/linked-usdl/usdl-editor

submission. There also seem to be no further investments by Sun up to this point. OWL-S is also a W3C submission, but standardization has not progressed since 2004. hRESTS was brought into an STI working group [210]. No further standardization efforts are known for hRESTS and SEREDASj.

Discovery of further information Next, we looked at how easy it is to connect "things" based on the service description language. We were mainly interested to see if it is possible to break out of the service description and connect different repositories that may be part of the Internet of Things. Linked USDL for IoT, SEREDASj and OWL-S support such linking of knowledge repositories, because the underlying technology already supports that. WADL and hRESTS could be used in such a way, but it is not really part of their specifications and both would need further extensions. WS-*, to some degree, can be used to link repositories. However, this would be complicated as WS-* was not designed with such a goal in mind. Typically, it is used as part of a closed world – for example in an enterprise – to link business models and services, as in BPMN/BPEL setups.

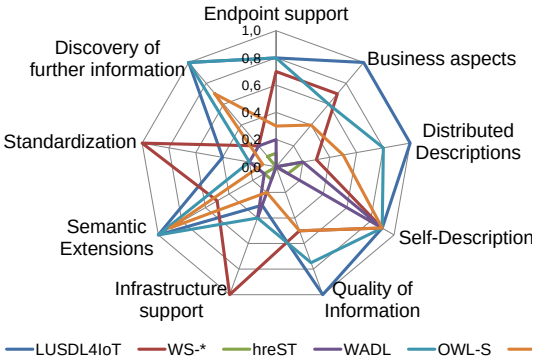
8.5.3.3 Multivariate Comparison

Based on the previous analysis we did a multivariate comparison of the different approaches. First, we quantified the qualitative results. A property that did not fulfill the requirements was mapped to zero. The nominal value [0] was mapped to the range $]0; 0.5]$, [(x)] to $[0.6; 0.7]$ and [X] to $[0.8; 1.0]$. The results are visualized in Figure 8.10 as radar charts [292]. Service description languages with a similar fulfillment of the evaluation criteria have approximately the same distance from the center point. The larger the corresponding area of a service description language in the graph is, the better the overall fulfillment of the evaluation criteria will be. We visualized all approaches and, for better visual comparison, also only the semantic approaches. Furthermore, we also take the importance weighting (see Table 8.7) into account. The weighted scores accentuate differences on criteria that are heavily weighted [310].

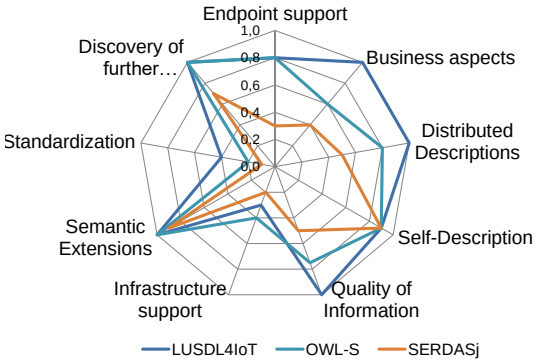
The semantic approaches generally fulfill the evaluation criteria better, but lack in standardization and infrastructure support. Considering only the semantic approaches, it can be seen that Linked USDL for IoT fulfills the requires better than or as well as OWL-S and SEREDASj. Only, the infrastructure and tooling is less mature than for OWL-S.

8.5.4 Conclusions

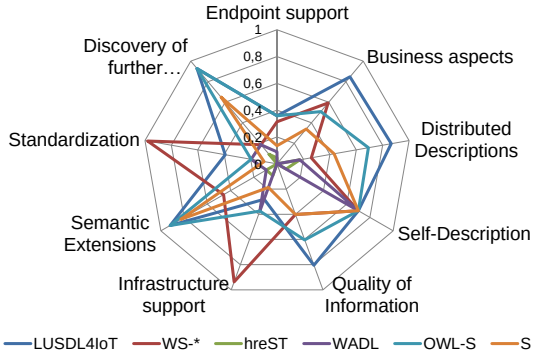
The evaluation criteria used to examine Linked USDL for IoT are based upon an internal requirements engineering process and the IoT-A stakeholder workshops. In total, we considered nine evaluation criteria. We compared Linked USDL for IoT against five related approaches: The WS-* family, hRESTS, WADL, OWL-S and SEREDASj.



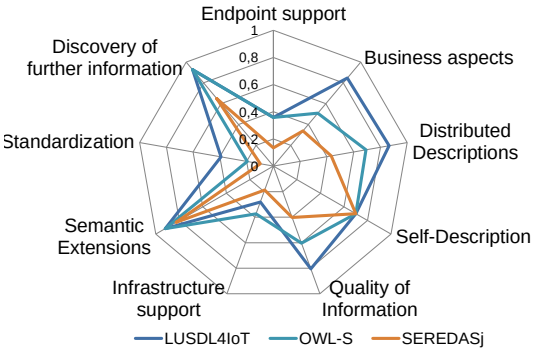
(a) All approaches



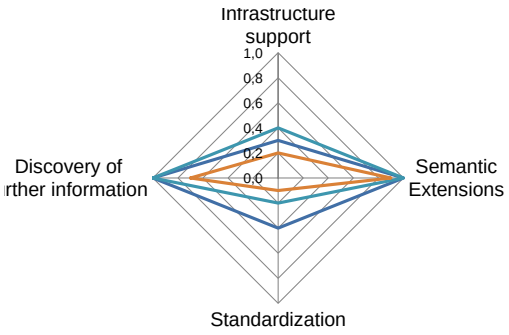
(b) Semantic Approaches only



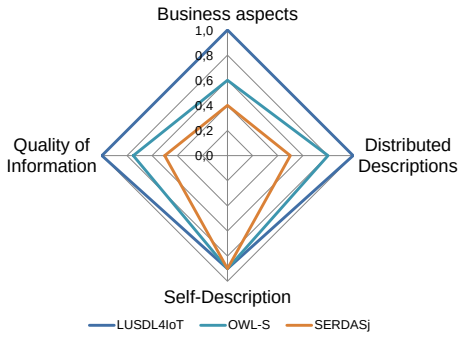
(c) All approaches (weighted)



(d) Semantic Approaches only (weighted)



(e) Important features, Semantic Approaches only



(f) Medium features, Semantic Approaches only

Figure 8.10: Radar charts of the different approaches and the degree they fulfill the evaluation criteria

We showed that Linked USDL for IoT, as a part of the Linked USDL family, fulfills most requirements. Compared to the current gold standard, – that is, the WS-* family – it lacks infrastructure and tooling support, as well as standardization. In those two aspects WS-* is clearly superior to all other service description languages. Linked USDL comes with some specialized tooling and can use all RDF-based tools. Nonetheless, the lack of infrastructure and tooling is blatant when compared to the rich tooling of WS-*. Only, WS-* has gone through a standardization process. The work on Linked USDL itself was initiated within the W3C USDL Incubator Group. It is mainly driven by SAP, the Knowledge Media Institute of The Open University, and the University of Coimbra.

Linked USDL for IoT is stronger in the area of modeling business aspects. Furthermore, it is – naturally – more tailored towards the needs of Internet of Things applications: It supports distributed descriptions, comes with a Quality of Information vocabulary, and generally offers a better support for semantic extensions. The WS-* protocol family, representing a non-semantic means of describing services, of course lacks any semantic modeling abilities. Extensions for WSDL exist (see Chapter 2.11), but do not have gained widespread adoption.

In comparison with its direct competitors – OWL-S and SEREDASj – it is either on par (OWL-S) or even more advanced (SEREDASj). Both, OWL-S and SEREDASj, lack support for business aspects. OWL-S is not specifally tailored towards the needs of the IoT, although some research exists. OWL-S did not develop into an integrated suite of complementary languages, but is more based on isolated research. Development of the OWL-S generally seems to have stalled, recently. Tooling support, for example the Protegee plugin has not been further development since 2007. Linked USDL has a web-based marketplace and service editor and currently is under active development. SEREDASj is a relatively new development; it is not specifically made for the IoT as well. It is tightly coupled to JSON and is only a description language (similar to WSDL) and does not come with any vocabularies of its own. Linked USDL for IoT and OWL-S are not tightly coupled to the RESTful paradigm, while SEREDASj solely is based on a RESTful design.

For industry-wide adoption of Linked USDL for IoT it needs to improve on tooling and standardization. Furthermore, with increasing adoption it would need to be strictly decided which parts of a service-ecospace should be standardized as part of the Linked USDL family of vocabularies, and which parts should be left for external (linked) vocabularies. An open standardization process would prevent users from fearing a vendor lock-in and strengthen their belief in a further development of the language.

8.6 Experimental Evaluation Framework

We designed a (semi-) automatic toolchain for measuring different properties of IoT applications and protocols within the Mote Runner simulation environment. As discussed in Chapter 2 the Mote Runner system comes with a simulator called Saguaro. For the application it is transparent if it runs on a simulated platform or on real hardware. Within the simulation environment measurements of energy consumption and memory consumption are possible. Measuring round trip time (RTT) and service access time (SAT) can be done either on hardware or also within the simulation. The definition of Service Access Time is introduced in Chapter 8.6.3. In the following we will present the main parts of the experimental evaluation framework.

8.6.1 Simulation Environment

The Mote Runner simulation is a real-time simulation with actual code execution. The simulation environment was used to gather the results for the OData evaluation. Furthermore, it was used to derive the energy estimations in the sleepy node implementation and for validation of the algorithms.

8.6.2 Memory Usage measurements

The Mote Runner environment provides two tools to measure the memory consumption of an application. Heap and stack usage can be measured over time. It is possible to stop the execution of the Mote Runner assembly at any possible event and retrieve memory usage data. As the Mote Runner environment is controlled and manipulated a script like the one shown in Listing 8.1 is used. It attaches itself to the (memory) frame creation event. Whenever this event occurs the registered listener is called and it retrieves memory information.

Listing 8.1: Memory usage measurements in the Mote Runner simulation environment

```
for (var i = 0; i < this.motes.length; i++) {
  var mote = this.motes[i];
  var conditions = Bugz.args2conds(Saguaro.EV_VM_FRAMESTART, 1);
    mote.getImpl().programHaltConditions(conditions, BLCK);
}

[...]

var listener = function(/** Sonoran.Event */hev, /** Saguaro.
    Connection.EventListeners */eventListeners) {
[...]
  if (evname !== Saguaro.EV_FRAMESTART) {
     return;
  }

  for (var i = 0; i < _this.motes.length; i++) {
```

```
    var mote = _this.motes[i];
    var data = mote.getImpl().inspectResources(BLCK).getReply().
        data;
    var stack = data.vmstack[1];
    var theap = data.theap[1];
    if (stack != -1 && stack < _this.freeStack[i]) _this.freeStack
        [i] = stack;
    if (theap != -1 && theap < _this.freeTheap[i]) _this.freeTheap
        [i] = theap;
  }
[...]
```

The memory measurements produce a human-readable result and, of course, can also be redirected to disk. The human readable output looks as follows:

Listing 8.2: Memory usage measurements in the Mote Runner simulation environment

```
---------------------------------------
Maximum Resource Usage:
---------------------------------------
                          | STACK | THEAP
02-00-00-00-00-5A-E3-D1 |    52 |   536
                    FREE |   912 |  3256
               AVAILABLE |   964 |  3792
02-00-00-00-00-5A-E3-D0 |    52 |   536
                    FREE |   912 |  3256
               AVAILABLE |   964 |  3792
---------------------------------------
```

The memory usage measurements should not be used together with time-based measurements, as it increases the execution time while recording memory usage data. Therefore, the data gathered for memory and all time-based measurements (energy, access times) were retrieved in different runs.

8.6.3 Service Access Time Measurements

We usually measure the service access time. The service access time is defined as follows:

Service Access Time (SAT): Service access time (SAT) is the time it takes for a service request to be sent to the recipient, processed, sent back and decoded by the service requester.

The SAT can be seen as the service-oriented equivalent of the round trip time. For CoAP calls, and therefore also for OData, we used Californium [218], a CoAP implementation for more powerful devices, from ETH Z ürich. Californium is written in Java using heavyweight technologies. Its design goal was not particularly the use on embedded devices, but ease of use from a developers point of view.

8.6.4 Energy Measurements

The Mote Runner platform provides a tool that writes a trace of the power consumption. Basically, there are two approaches for estimating the energy consumption: hardware (attached or in circuit) measurements and simulation-based approaches. Approaches based on hardware rely on additional components on the mote (to provide run-time measurement of energy consumption) or it means the complicated and time consuming use of oscilloscopes and multimeters. Due to the nature of the used devices these measurements are quite accurate. Hurni et al. [179] studied the accuracy of software based estimation compared to sophisticated (in circuit) hardware measurements. They reached the conclusion that the while due to inaccuracies in the production of electronic components, differences of more than 4% are possible. Protocol generic, per node calibrated parameters could reduce the difference to as low as $1.13\% \pm 1.15\%$. The paper conveys that software-based energy estimation can be a valuable alternative to using sophisticated measurement hardware. Therefore, software based power consumption measurements are more than feasible when comparing protocols and yield results that are very close to in circuit measurements.

The energy tracer outputs a given point in time for a specific mote as triple (t_n, p_n, r_n), where t_n is a timestamp given in nanoseconds starting from the beginning of the simulation, p_n is the current draw in nano-ampere and r_n is the reason for changing the energy state based on the internal state machine. As soon as the state in the simulation changes and, thus, the power consumption (current draw), a new triple (t_{n+1}, p_{n+1}, r_{n+1}) is generated.

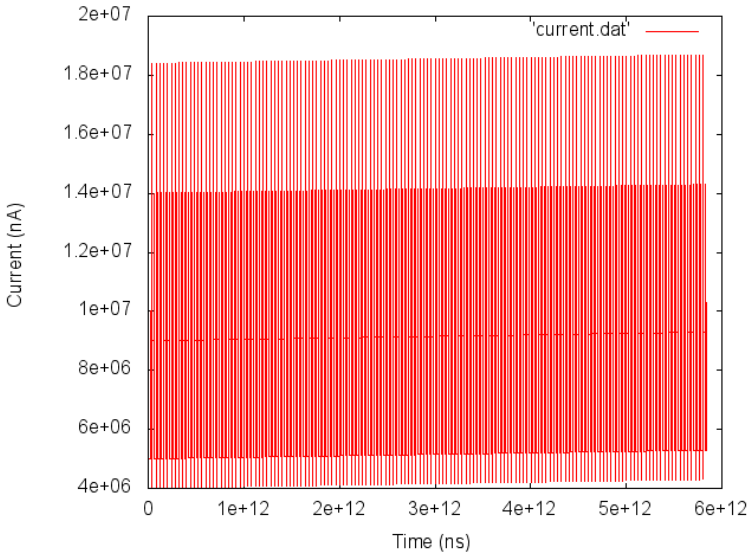


Figure 8.11: Visualized current trace as produced by the Mote Runnner simulation environment

A trace, as written by the Mote Runner simulation environment is shown in Listing 8.3. The energy traces of different energy states are shown in Figure 8.11. The figure shows the energy trace of a periodic application doing some calculations periodically every 10 seconds and sending data after every second periode. The radio is in sleep mode in between. The peaks at around $1.8 * 10^7$nA are the short transmitting phases, the calculations are around $1.4 * 10^7$nA, while the sleeping in between is around $5 * 10^6$nA. The Mote Runner simulation environment, in conjunction with our evaluation framework, allows an a-posteriori analysis and simulation of different hardware characteristics. We used that feature to simulate different behavior of the Waspmote Pro for scenarios with deep sleep and without deep sleep.

Listing 8.3: Power measurements in Mote Runner Simulation environment

```
[nS]         [nA]     [State]
12221068740 17602000 RUN
12221100870 7612000  IDL
12222352067 17602000 RUN
12222370157 12602000 HLT
12223346701 17602000 RUN
12223346701 12602000 HLT
12223346701 17602000 RUN
12223346701 20402000 TXING
12223351561 10412000 IDL
12223922701 20402000 RUN
12223922701 17602000 ACTIVE
12223925941 7612000  IDL
12223925941 17602000 RUN
12223947001 16002000 IDLE
12223961581 6012000  IDL
13222630406 16002000 RUN
13223624403 14002000 LED off
```

Within a given time interval $[t_a, t_b]$, using an energy consumption function $\Psi(t)$ and an ordered set of energy level change points $p = (t_0 = t_a, t_1, t_2, ..., t_n = t_b)$ the overall energy consumption $E(t_a, t_b)$ can be calculated via discrete integration:

$$\begin{aligned} E(t_a, t_b) &= \int_{t_a}^{t_b} \Psi(t)dt \\ &= \sum_{x=0}^{n-1} \Psi(t_x) * (t_{x+1} - t_x) \end{aligned} \tag{8.1}$$

We wrote a tool in Java that takes the output of the Mote Runner simulation environment, reads it, performs some operations on the data and outputs the energy consumption. The framework is shown in Figure 8.12. First, the power measurements triples (t_i, p_i, n_i) are read, then some processing can be done one the data, like filtering for different motes or applying a different energy profile. This is particularly useful for

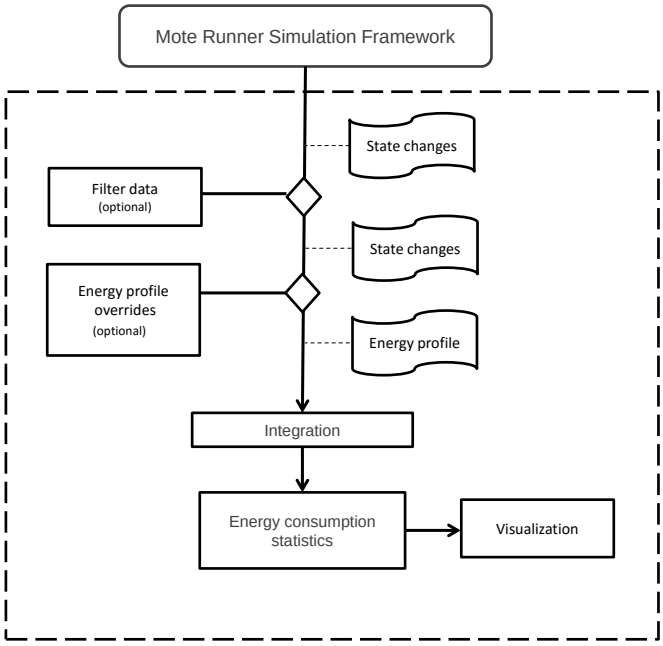


Figure 8.12: Energy evaluation framework

performing ”what-if” analysis without having to run the simulation again. For example, the impact of a different sensor technology could be evaluated based on datasheet information or the impact of different sleep modes can be evaluated. We used this to conveniently simulate the impact of different sleep modes – for example, the hibernate mode on the Waspmote Pro platform or to see how a different radio would have affected the total energy consumption. The total energy consumption can be either based on the internal values of the Mote Runner plugin or via state-based plugins for the what-if simulations of other platforms. The output of Mote Runner energy measurements are typically given in mAs in all Mote Runner documentation and publications. We are following this convention.

8.7 OData Stack

In this section we perform a quantitative analysis of our OData implementation and, consequently, the CoAP implementation. We are using the software (Mote Runner) and hardware (IRIS) as described in Chapter 2. First, we present the experimental setting that was used to perform our experiments (Chapter 8.7.1). As outlined in Chapter 6.3.2.1 we are communicate with the mote directly. The enterprise system has direct access to the mote and communicates with it without an intermediary gateway.

8.7.1 Experimental Setting

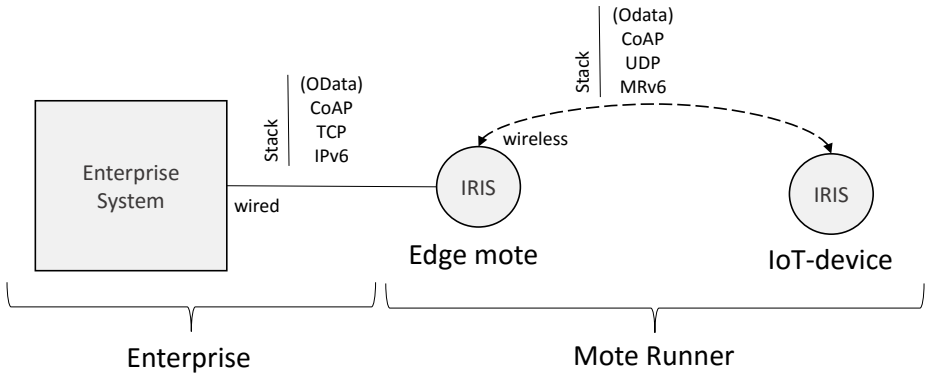


Figure 8.13: Experimental setting

All experiments were conducted with IRIS motes. The technical details and specifications of the IRIS mote can be found in Chapter 2.1.2. All measurements were performed within the Mote Runner simulation environment. We were only interested in scenarios, in which the mote communicates directly with the enterprise system following the Internet-of-Things vision. No application-layer gateway was used. The experimental setting is shown in Figure 8.13. The motes were running the Mote Runner v11 with a 6LoWPAN network stack, as shown in Figure 6.3. The Mote Runner v11 6LoWPAN stack is tailored towards predictability and not optimized for low energy consumption or high throughput. Some optimization-potential has been addressed with later versions of MRv6. Please refer to Chapter 2.3.3 for more details on MRv6. We assume our results to be generalizable to other platforms and protocols as well, because the ratios between sending, receiving, sensing and using CPU time should be comparable on all platforms.

8.7.2 Mote Setup

The OData service we are exposing consists of one mote with three different sensors (temperature, humidity and light), where each has a unique ID, a name and can return a

value (data). As outlined in Chapter 6.3, service discovery is done through the service description and the metadata information. The service description of our service looks as follows:

Listing 8.4: ”OData Service”

```
<?xml version="1.0" encoding="utf-8" ?>
<service xml:base="http://tmpsvc.sap.com/OData.svc/" xmlns="http
    ://www.w3.org/2007/app" xmlns:atom="http://www.w3.org/2005/
    Atom">
  <workspace>
    <atom:title>Default</atom:title>
-   <collection href="Sensors">
      <atom:title>Sensors</atom:title>
    </collection>
  </workspace>
</service>
```

The JSON representation is more compact and therefore more suitable for constrained devices:

Listing 8.5: ”OData Service - JSON”

```
{"odata.metadata":"http://tmpsvc.sap.com/OData.svc/\$metadata","
    value":[{"name":"Sensors","url":"Sensors"}]}
```

The $metadata keyword, which provides information about the sensors looks as follows:

Listing 8.6: ”OData Service - metadata”

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/
    ado/2007/06/edmx">
<edmx:DataServices m:DataServiceVersion="3.0" m:
    MaxDataServiceVersion="3.0" xmlns:m="http://schemas.microsoft.
    com/ado/2007/08/dataservices/metadata">
<Schema Namespace="Mote" xmlns="http://schemas.microsoft.com/ado
    /2009/11/edm">
<EntityType Name="Sensor">
<Key><PropertyRef Name="ID" /></Key>
<Property Name="ID" Type="Edm.Int32" Nullable="false" />
<Property Name="Name" Type="Edm.String" m:FC_TargetPath="
    SyndicationTitle" m:FC_ContentKind="text" m:FC_KeepInContent="
    false" Nullable="false" />
<Property Name="temp" Type="Edm.String" m:FC_TargetPath="
    SyndicationSummary" m:FC_ContentKind="text" m:FC_KeepInContent
    ="false" Nullable="true"
  <ValueAnnotation term="Measurements.Unit.Celsius">
</property>
```

```
<Property Name="humi" Type="Edm.String" m:FC_TargetPath="
    SyndicationSummary" m:FC_ContentKind="text" m:FC_KeepInContent
    ="false" Nullable="true" />
  <ValueAnnotation term="Measurements.Unit.Humidity">
</property>

<Property Name="lght" Type="Edm.String" m:FC_TargetPath="
    SyndicationSummary" m:FC_ContentKind="text" m:FC_KeepInContent
    ="false" Nullable="true" />
  <ValueAnnotation term="Measurements.Unit.Light">
</property>
<Property Name="data" Type="Edm.String" m:FC_TargetPath="
    SyndicationSummary" m:FC_ContentKind="text" m:FC_KeepInContent
    ="false" Nullable="true" />
  <ValueAnnotation term="Measurements.Unit">
</property>
<EntityContainer Name="APPService" m:IsDefaultEntityContainer="
    true">
<EntitySet Name="Sensors" EntityType="Mote.Sensor" />
</EntityContainer>
```

The metadata encodes that our mote has entities (in OData notation) of type *Mote.Sensor*. These entities have certain properties (ID, name, and data) as well as corresponding datatypes. Furthermore, the set Sensors as specified in the service description is further defined to be of type Mote.Sensor. The $metadata request in OData v3 is only specified for ATOM/XML. OData v3 does not support the metadata document as part of a JSON representation. As per OData v4 the JSON representation is also supported now. The responses of the motes are based on that schema.

Listing 8.7: ”OData service- sample response: ATOM”

```
<d:temp xmlns:d="http://sap.com/dataservices" xmlns:m="http://coap
    /metadata">24</d:temp>
```

The same result in JSON verbose format [272] would look as follows:

Listing 8.8: ”OData service - sample response: JSON”

```
{"odata.metadata":"[::::1]/OData.svc/$metadata#Edm.String","temp
    ":"24"}
```

As baseline for a system that is not OData-enabled, we choose pure CoAP as application layer protocol with a small data representation. This baseline could be used as data representation in an external service description such as Linked USDL for IoT. In the following we will refer to this baseline simply as CoAP, in comparison to JSON and ATOM. The message representation in the CoAP baseline is then in the form of comma delimited property:value pairs, as shown in the following code fragment:

Listing 8.9: ”OData payload - sample CoAP response”

```
id:<sensor_no>,n:<sensor_name>,d:<value>
```

A means of discovery resources is the CoRE Link Format[349], accessible through the *.well-known/core* interface. There is a subtle difference between the *./well-known/core* and OData service discovery: the CoRE Link Format supports *Resource Discovery*, while OData aims for *Service Discovery*. The CoRE Link Format provides web linking as specified in RFC5988[281] and can be used to discover the links hosted by a CoAP server. It returns information in a link-header style format [281]. A minimal resource description for a similar resource based access to the mote could look as follows:

Listing 8.10: "OData Service - metadata in Core Link Format"

```
</temp>;rt="temperature";ct=0;if="sensor"</hum>;rt="humidity";ct
    =0;if=sensor,</light>;rt="light";ct=0;if="sensor"
```

First, the resource is named – in this case "/temp", for temperature sensor. Then the resource type (rt) and the content type (ct). The rt attribute is string used to assign an application-specific semantic type to a resource [349]. In case the CoRE-Link Format is used in conjunction with Linked USDL for IoT, it can be used for linking the two. The interface "if" specifies the interface to be used, in our case sensor. This is also an application specific string. The ct attribute specifies the content type, as described in the CoAP specification [354]. A content type of zero means plain text.

Q_1	`GET coap://[]:1024/OData.svc?$metadata`
Q_2	`GET coap://[]:1024/OData.svc/sensors`
Q_3	`GET coap://[]:1024/OData.svc/sensors(0)`
Q_4	`GET coap://[]:1024/OData.svc/sensors(0)/ID`
Q_5	`GET coap://[]:1024/OData.svc/sensors(0)/Name`
Q_6	`GET coap://[]:1024/OData.svc/sensors(0)/temp`
Q_7	`GET coap://[]:1024/OData.svc/sensors?$filter=Data gt 42`
Q_8	`GET coap://[]:1024/OData.svc/sensors?$filter=Data eq 60 and Name eq humidity`

Table 8.9: Queries

8.7.3 Results

In the following we investigate a typical IoT-scenario in which a backend system is directly communicating with the mote. The queries used to evaluate the properties of the system are listed in Table 8.9. The computational complexity increases with each Q_i. In the rest of the work we will refer to these queries as Q_i^m or Qmi, where i is the query number as listed in Table 8.9, and m is either A for OData/ATOM, J for OData/JSON and C for CoAP, with or without the compression suffix CP

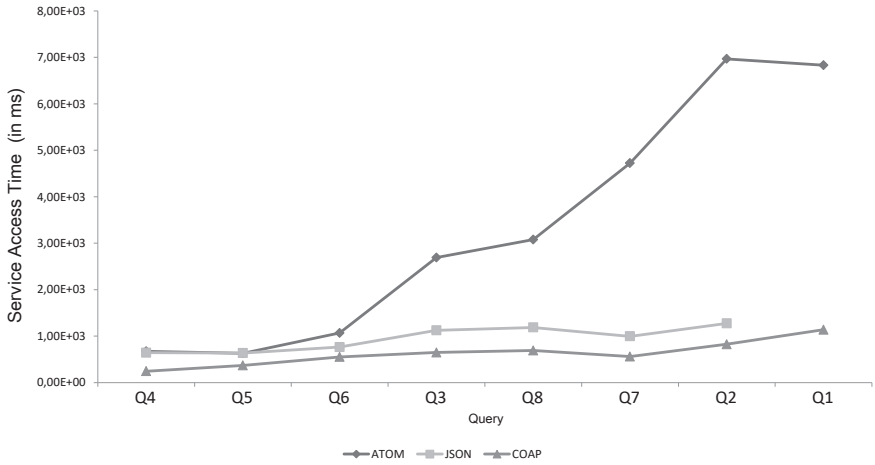


Figure 8.14: Service access time (in ms, blockwise-transfer with block size 64 Byte) for queries Q_1 to Q_8 sorted by Q_n^A payload size

Query	ATOM	ATOM/CP	JSON	JSON/CP	COAP
Q_1	1627	1164	—	—	115
Q_2	1643	906	191	156	63
Q_3	604	443	110	105	23
Q_4	99	96	74	74	4
Q_5	94	88	85	84	13
Q_6	87	82	76	76	4
Q_7	1199	734	148	100	39
Q_8	761	523	108	108	18

Table 8.10: Payload size (in Bytes), compressed (CP) and uncompressed

The resource consumption of the on-mote implementation is shown in Table 8.11. It is seen that the memory consumption of the OData/ATOM implementation is larger than the memory consumption of JSON. Nonetheless, the JSON implementation is comparable to a pure CoAP implementation in terms of resource consumption. We did not optimize for resource consumption – for example, some memory blocks were pre-allocated to have a big enough table available for sensor data and the dictionary, and might not be actually used. More aggressive implementations are possible. Only stack and heap are critical because the IRIS platform offers only 8kb of which nearly half is already used by the Mote Runner VM. Flash is not that much of an issue on the hardware platform. The IRIS mote offers 128KB of program flash and another 512KB of data flash. The actual payload of each query is shown in Table 8.10.

For each query $Q_1...Q_8$ we measured the service access time, that is, the time from a request being issued by the the service consumer until the answer has been decoded. The results (averaged over 100 runs) are shown in Figure 8.14 for uncompressed data

	COAP			JSON			ATOM		
	Stack	Heap	Flash	Stack	Heap	Flash	Stack	Heap	Flash
Q_1	232	3271	3850	—	—	—	292	3543	4255
Q_2	232	3284	3850	292	3460	3931	292	3552	4255
Q_3	232	3252	3850	292	3432	3931	292	3476	4255
Q_4	232	3244	3850	292	3436	3931	292	3408	4255
Q_5	232	3248	3850	292	3440	3931	292	3496	4255
Q_6	232	3248	3850	292	3448	3931	292	3416	4255
Q_7	232	3316	3850	292	3500	3931	292	3524	4255
Q_8	232	3404	3850	292	3528	3931	292	3548	4255

Table 8.11: Memory consumption (maximum, in bytes)

and in Figure 8.15 for compressed data. For small result sets the difference between the three formats is negligible. Larger data sets change the situation. Compression can decrease the service access time considerably for large ATOM requests and, to some degree, the JSON requests. CoAP and JSON remain at low service access times, but the ATOM format increases the service access time rapidly because of the amount of data to be transmitted. Compression does help in case of ATOM, but is still considerably worse than JSON. It reduces the energy consumption and the service access times, but not to a degree where it could compete with JSON or even CoAP. Devices that are less constrained would profit from more advanced compression schemes. Unfortunately, OData v3 does not support a metadata JSON representation, so at least once a download of the metadata is necessary.

The CoAP block size also affects the service access time. A 6LoWPAN packet, even if fragmented, is typically more efficient than the CoAP block option in settings that do not suffer packet loss and when WSN links do not suffer from congestion often [240]. In such cases they show a better performance than the CoAP block option. Nonetheless, in cases of packet loss or when fragmentation is not available or exceeds the capabilities of the system, CoAP blockwise transfer can be used. Furthermore, block-wise transfer has the additional advantage that allows the server to be stateless. A block-wise transfer does not need any connection setup or memory of previously sent blocks. Most highly constrained devices are running on so few memory that the handling of additional state and possible caching of packets would exceed their capabilities. In our implementation we therefore rely mainly on the CoAP block option. The relationship between CoAP block sizes and the resulting service access times are shown in Figure 8.16. Early negotiation of block sizes is assumed. The simulation was idealistic as we assumed no packet loss, and minimal to no application-related overhead (e.g. sensing) that might cause later transmission. Due to limitations of the MRv6 protocol [180], we did not take the overhead of the IP-layer fragmentation or the adaption-layer fragmentation into account, but only the transmissions and receptions of packets. The VM-based approach of Mote Runner might have added some delays as we do not control the hardware. It is

possible that the VM delays sending by a slot or two (e.g. garbage collection, different tasks running). Other than that, the response times typically did not vary by more than a few slots for the same amounts of packets send when running with the MRv6 protocol.

We experimented with non-standard block sizes (see also Chapter 2.5) to estimate the effects of an unconstrained block size negotiation. As shown, the effects were relatively low. So, the restriction is sufficient. The main factor reducing the response time is the number of new block requests. A block size of 32 bytes, of course, requires twice as many new block requests than a 64 byte block size.

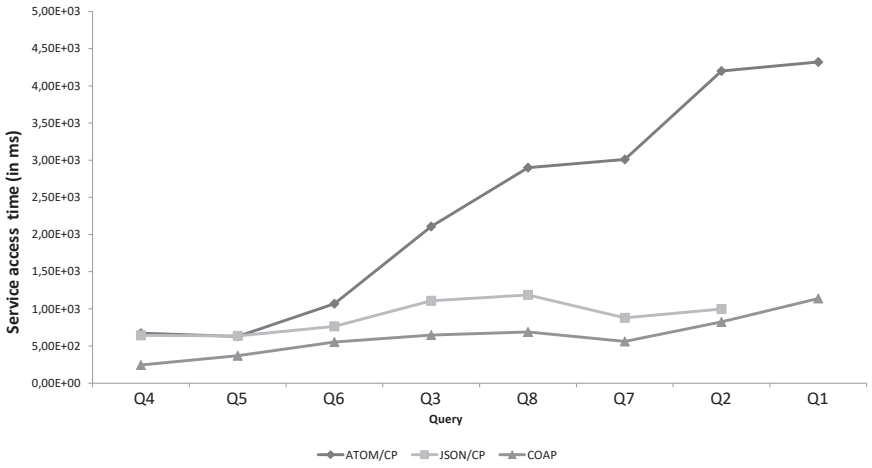


Figure 8.15: Service access time (in ms, blockwise-transfer with block size 64 Byte) for queries Q_1 to Q_8 (compressed) sorted by Q_n^A payload size. CoAP (see also Figure 8.14) as baseline for comparison (always uncompressed)

The energy consumption evaluations were performed within our simulation environment. In Figure 8.17 and Figure 8.18, we show the energy consumption of OData/ATOM, OData/JSON and CoAP once for a block size of 64 bytes and 256 bytes. As can be seen, the advantages of a more machine-readable and machine-interpretable data representation is payed with higher energy consumption. The ATOM response needs most energy, for the obvious reason of more air time, but also as there are many transfers from flash memory to RAM. The energy consumption of JSON, and especially JSON/CP, is comparable or only slightly higher than the CoAP baseline. Smaller block sizes lead to more data transfer for the block requests and responses, as well as more computation to process these requests and thus an increase in energy consumption. The advantage of a smaller block size is increased reliability and in case of very lossy network increased throughput. Furthermore, they avoid fragmentation on lower layers.

8.7.4 OData versus Linked USDL

In the following we will briefly compare OData and Linked USDL for IoT. They follow different design philosophies, which leads to certain strengths and weaknesses. The

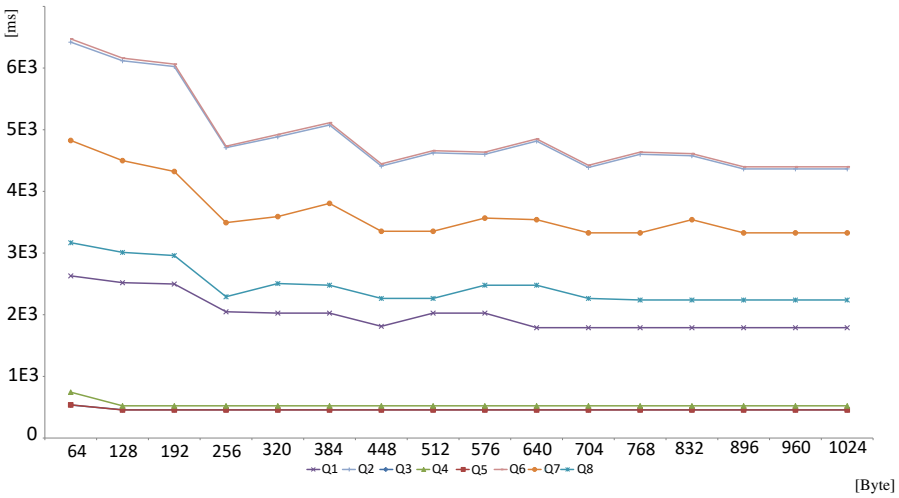


Figure 8.16: Service access times of queries Q_1^A to Q_8^A with different CoAP block sizes (simulation, idealistic values)

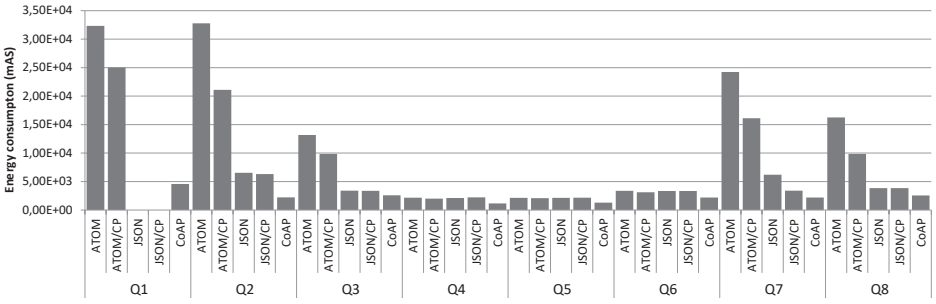


Figure 8.17: Energy consumption (in mAs, serving a series of 100 requests, blocksize 64 Byte)

main advantage of OData is its industrial support. Unlike Linked USDL, it prescribes a data representation. Linked USDL for IoT provides far richer modeling possibilities. It has a more comprehensive service idea – for example, it allows model business aspects, human work, and interaction patterns among other things. As discussed in Chapter 6.3.3.2 for reasoning purposes, a mapping from OData to RDF can be performed. OData is more than just a service description, it is also a query language. Linked USDL for IoT is not designed to work on that level. Extending Linked USDL for IoT with complex query capabilities might also not be desirable. However, further extensions of Linked USDL for IoT that allow modeling such query interfaces easily are possible.

A Linked USDL implementation that uses JSON-LD as data representation would have approximately the same resource consumption and general behavior as we observed in the OData/JSON case. Linked USDL for IoT is able to ”semantify” our CoAP

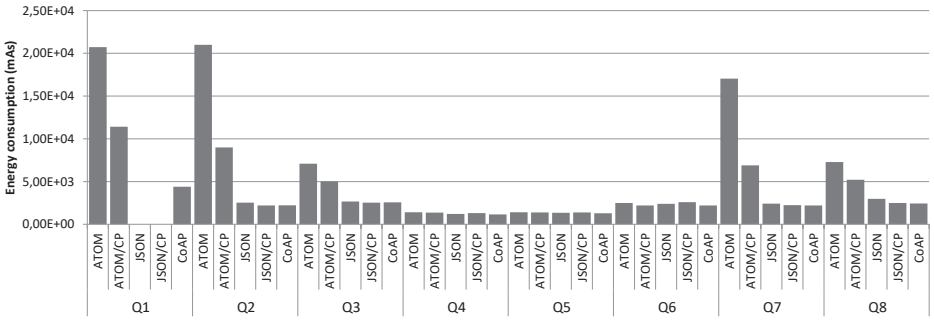


Figure 8.18: Energy consumption (in mAs, serving a series of 100 requests, blocksize 256 Byte)

baseline as well. Each mote has a core link format description and a Linked USDL service description, similar to Listing 5.18. Thereby, the energy and service access times could be reached by using Linked USDL as external service describing, linking the result sets to their respective semantics. In case of more complex result sets, of course, the size of the Linked USDL payloads and the necessary modeling effort would also increase.

8.7.5 Conclusions

We performed an experimental feasibility study on the applicability of OData on Class 0 devices. Based on the terminology we introduced in Chapter 1.1, this represents a top-down approach for enabling REST-based interoperability in a semantics-aware enterprise. As shown, there is a considerable energy overhead of ATOM/XML over the CoAP baseline. The SATs are also much higher and they render the ATOM representation as an almost impossible choice for battery-powered constrained devices. Compression significantly improved both the SAT and the energy consumption. In case of a Class 0 device, it is still not feasible unless the device communicates very infrequently. Given our experimental platform, we could not improve much on this. Class 1 and higher devices could benefit from more advanced ATOM/XML representations using XMI and other technologies. More details can be found in Chapter 2.8. Nonetheless, compared to JSON the advantage of the baseline shrinks. The difference between JSON/JSON-CP and CoAP is the price to be paid for an enterprise protocol on an IoT-node. The overall energy consumption was comparable. The CoAP baseline did consume significantly less energy than ODATA, but typically not significantly less than the JSON or JSON/CP representation. In terms of SAT, the JSON representation was between 1.38 and 2.62 times the respective SAT of the COAP baseline. JSON/CP had SATs between 1.20 and 2.62 times the baseline. Of course, as shown in Chapter 2.8, real performance gains from the compressed versions are to be expected for larger data sets when lot of sensor data has to be transmitted. For example $Q7$, where the amount of data was reduced to three thirds of the original, the SAT time immediately decreased

by around the same. The critical $metadata information, one of the weak points of OData v3, decreased from around 6.0 times the baseline to 4.0 times the baseline. From a pure SAT and energy point of view, this makes the bottom-up approach using Linked USDL for IoT more attractive. In contrast, the use of an industry standard increases interoperability immediately, without the need for Linked USDL. One is then, of course, limited to the semantic expressiveness of OData. A major drawback of OData v3 is the fact that the $metadata, thus the place were all the semantics is provided, has no representation in JSON and thus always needs to be transmitted as ATOM/XML. The situation changed with OData v4 so that future OData implementations on IoT systems can benefit from a $metadata JSON representation.

8.8 Sleepy Nodes

In the following, we present the evaluation results of the sleepy node framework as described in Chapter 7. First, we describe the experimental setup. We proceed by determining the constants in our energy model and discussing the impact of clock drift. Afterwards, we evaluate our implementation and the three allocation strategies within the windowing framework.

8.8.1 Experimental Setup

We used a network of seven nodes ($n_0 ... n_6$) – including the sink (n_0) – in a binary tree layout running the MRv6 protocol (see Chapter 7.5). We simulated Waspmote PROs and IRIS Motes with hardware capabilities as explained in Chapter 2. If not specified otherwise, the following default parameters were used to run the experiments:

In most experiments we assumed the energy state for all the nodes as follows $(n_0, ..., n_6) = (\infty, 2E_{leaf}, 2E_{leaf}, E_{leaf}, E_{leaf}, E_{leaf}, E_{leaf})$. For leaf nodes we assumed either $E_{ES1} \equiv E_{leaf}$=10J, $E_{ES2} \equiv E_{leaf}$=100J or $E_{ES3} \equiv E_{leaf} = 2000J$. The results scaled well with different energy values. The sink was assumed to have infinite power reserves and therefore the power consumption on it was ignored.

A variable number of entities was used, 8 ± 3, the entities measurement periods varied from 3*10^3 seconds to $7 * 10^3$ seconds. The experimental setup is a scaled down version of a supply chain or warehouse scenario, with rather frequent monitoring of perishable goods, or medicine that allows time for longer sleeping periods. The measurement tolerance per entity was up to 20% of its measurement period. The window size used was equal to the mean measurement period. 50% of the entities could be measured by 2 sensors, 30% by 3 and 20% by 4. The assignment of eligible sensors to entities according to the above percentages was done by sampling from a uniform random distribution. All the values presented in the results are typically averages among a number of runs (up to 15), using the same parameters. For every run the random number generator was initialized with a different seed. We used the Monte Runner environment v14.

8.8.2 Energy Model

To be able to compare different sensor selections for some given measurements, we have to be able to calculate the energy that would be spent on each node of the WSN. We approximate this by using the linear energy model (Chapter 7.4). From a computational point of view, as this is only an estimation based on a linear model, the calculation takes virtually no time. The energy model, as shown in the following formula, was already discussed in Chapter 7.4.

$$E_n = \underbrace{t^n_{idle} \cdot P_{idle} + t^n_{sleep} \cdot P_{sleep}}_{time\ based} \underbrace{+m^n_{sense} \cdot E_{sense} + m^n_{sleep} \cdot E_{sleep} + m^n_{comm} \cdot E_{comm}}_{event\ based}$$

In this chapter, we present the actual values we derived for the two hardware platforms. The simulation framework allows us to measure durations, states and current draw. Furthermore, we can control the input parameters, in particularly the number of requests, and the time in protocol sleep and protocol idle mode; therefore, we can use ordinary least squares (OLS) regression analysis [314] to determine the unknown variables in our linear equation.

The general idea is to model a variable (E_n) as a linear function of others. OLS minimizes the sum of squared errors between the estimations by applying the model and observations as measured in simulation. E_n has been measured, as well as t_{sleep} and t_{idle}. The number of requests m_{sense}, m_{sleep}, and m_{comm} have been varied. For both platforms (IRIS and Waspmote), we performed a large number of (automated) experiments over several hours to get enough data to fit our model. In the end, we got hundredths of Megabytes of Data. Every experiment was repeated in four different modes of network operation, so that we could extract correlated and uncorrelated features for approximating the different energy components of our model. The different modes of operation were:

- Querying and Sleeping: Normal mode of operation. Nodes are put to sleep between measurements.
- Querying Only: Only measurement queries are issued. The nodes are not put to sleep between measurements.
- Sleeping Only: Only sleep requests are issued. No measurements are performed.
- Idle Only: The network is initialized and left in the idle state for the whole duration of the experiment. Neither queries nor sleeping requests are issued for any of the nodes. There is no network activity other than the one imposed by the MRv6 protocol itself.

For all components given in Equation 8.8.2 we derived the following energy and power values:

$$\left.\begin{aligned} E_{sense} &= 4.62891 * 10^{-3} J \\ E_{sleep} &= 3.45422 * 10^{-3} J \\ E_{comm} &= 6.82133 * 10^{-4} J \\ P_{sleep} &= 3.60 * 10^{-5} W \\ P_{idle}^{base} &= 1.70 * 10^{-2} W \\ P_{idle}^{neighbor} &= 4.64 * 10^{-4} W \end{aligned}\right\} \quad \text{IRIS}$$

$$\left.\begin{aligned} E_{sense} &= 1.99402 * 10^{-3} J \\ E_{sleep} &= 6.30984 * 10^{-3} J \\ E_{comm} &= 5.27001 * 10^{-4} J \\ P_{sleep} &= 6.00 * 10^{-6} W \\ P_{idle}^{base} &= 1.71 * 10^{-2} W \\ P_{idle}^{neighbor} &= 4.61 * 10^{-4} W \end{aligned}\right\} \quad \text{Waspmote Pro}$$

We used two-thirds of the data to determine the parameters in our equation and one-third of it for validating our model with two metrics: the mean absolute percentage error and the coefficient of determination. The mean absolute percentage error (MAPE) is defined as

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{V_i - \overline{V}_i}{V_i} \right| * 100 \tag{8.2}$$

where V_i is the actual value of the observation and $\overline{V}_i$ its statistical forecast. MAPE expresses its accuracy as a percentage and describes how large the mean absolute percentage error is. Its lower bound is zero and it has an unlimited upper bound. A lower value means a better match of the actual observation and its statistical forecast.

The coefficient of determination R^2 indicates how well the data fits its statistical forecast as well. R^2 is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (V_i - \overline{V}_i)^2}{\sum_{i=1}^{n} (V_i - AVG(V))^2} \tag{8.3}$$

The coefficient ranges from zero to one. The closer the coefficient is to one, the better the fit. A coefficient of one means a perfect fit.

We calculated for IRIS motes:

$$\begin{aligned} \text{MEAP}_{\text{IRIS}} &= 1.09 \\ R^2_{\text{IRIS}} &= 0.99951 \end{aligned} \tag{8.4}$$

and for the Waspmote Pro Platform:

$$\begin{aligned} \text{MEAP}_{\text{Waspmote}} &= 1.71 \\ R^2_{\text{Waspmote}} &= 0.99903 \end{aligned} \tag{8.5}$$

Both metrics show, for both platforms, a good fit between the expected value and the simulated value. In the following, the simulated sleepy node algorithms use the energy model with those derived parameters.

8.8.3 Clock-Drift and Energy Considerations

We have already discussed clock drift and its consequences in Chapter 7.5.2. Now, we quantify its impact on energy consumption. Clock-drift is important because it prevents a node to wake-up at the exact calculated moment. Depending on the clock-drift the calculated and the actual clock-drift can differ substantially. Calculations are presented in Chapter 7.5.2.

At this point, we are solely looking at the impact on energy consumption. The energy consumption of a mote that is sleeping and about to wake up and rejoin the network is shown in Figure 8.19. The least energy is consumed if the node wakes up at exactly the right time, catches the beacon and immediately joins the network. This is illustrated in the graphs at point ppm=0. Assuming a clock-drift of, for example, 10ppm an additional 0.008J is spent on the IRIS mote (see Figure 8.19). In an experiment we observed that the maximum energy overhead caused by drift is in the order of $1 * 10^{-2}$J per 5000 seconds sleep request. As can be seen, clock-drift that causes the mote to wake up earlier than required is typically better than clock-drift that will cause missing the beacon. A miss just after the beacon forces the mote to keep listening for a nearly full beacon interval.

8.8.4 Sleepy Node Implementation

In the following, we present the results of the conducted experiments based on the three strategies introduced in Chapter 7: first Fit, an exhaustive approach, and dynamic partitioning. The three strategies can be summarized as follows. For further details please refer to Chapter 7.6.3

Exhaustive Strategy (ea): The optimal solution for the task allocation problem, with regard to our energy model and within the given window, is computed by applying an exhaustive approach (backtracking with constrained satisfaction). An exhaustive approach considers all possible partitions of the set M_w and evaluates them. It has exponential time complexity.

First Fit (ff): As a second benchmark for comparison, we use a first-fit algorithm operating within the same windowing framework. It creates a new query for every measurement, picking one of the eligible sensors at random.

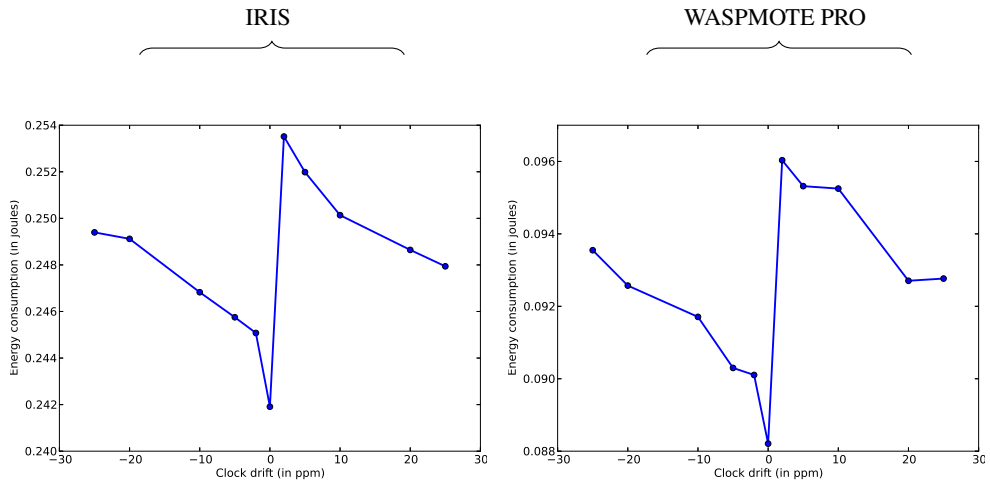


Figure 8.19: Energy consumption for node answering a 5000s sleep request, sleeping for that amount of time and then waking up. The chosen beacon interval here is 250ms. For the largest drifts of 25 ppm presented, the clock offset over 5000 seconds is 125 ms.

Dynamic Partitioning (dp): Instead of considering all possible partitions, we order the measurements M_w according to their timestamps and check only the ones that combine neighboring measurements in a greedy way. The rationale for this strategy is the observation that measurements that can be combined often appear in sequence.

In the presentation of the results we normalized the window sizes (ws) and tolerances according to the mean entity measurement period. For ws=0.5, therefore the actual length of a window in seconds (L_{window}) was half of $mean(\omega)$. For $ws = 2$, L_{window} was twice $mean(\omega)$.

$$ws =\leftarrow \frac{L_{window}}{mean(\omega)}$$

$$tolerance_e \leftarrow \frac{e.\delta_i}{mean(\omega)} \forall e \in \mathbb{G}$$

To compare the two platforms in a better way, we typically normalized the energy and network lifetimes with respect to the maximum value of the IRIS mote. As described in Chapter 7.6, the task-allocation algorithm tries to take advantage of the knowledge of the measurement timings and sensor distribution, in order to reduce energy overhead by combining measurements into the same query. Therefore, the capability of a possible combination relies on two factors (see Chapter 7):

1. the entities tolerances
2. the sensors overlapping

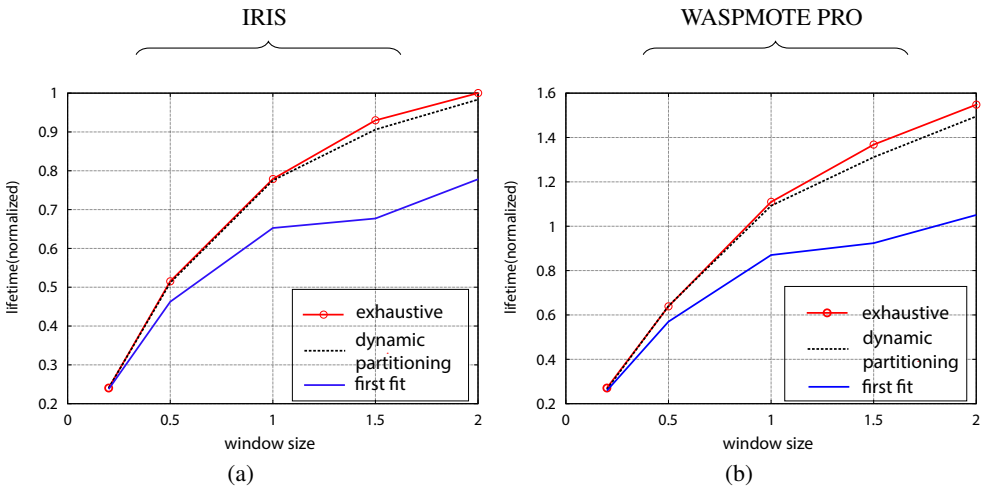
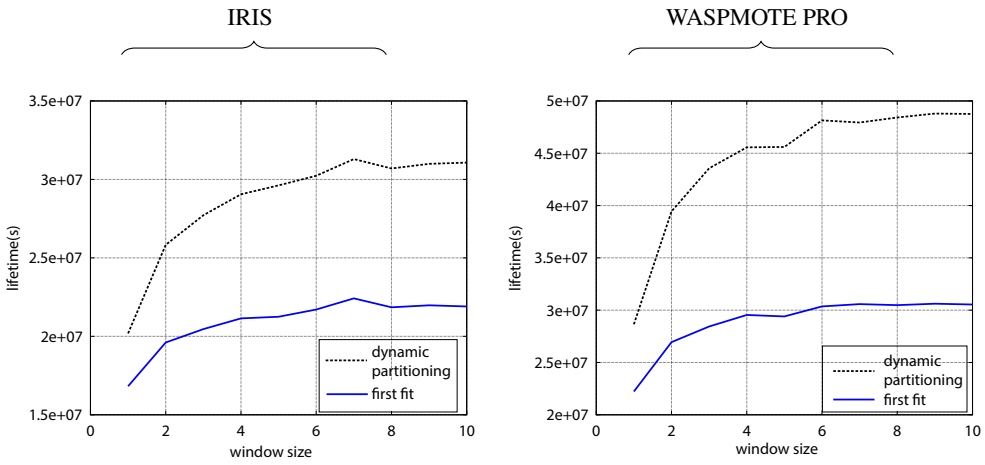


Figure 8.20: Comparison of the different task allocation algorithms, IRIS Platform (left hand side) and Waspmote Pro platform (right hand side), E_{ES1}. The window size is expressed relatively to the mean entity measurement period

Figure 8.20 presents a comparison of the allocation strategies (first fit, dynamic partitioning, exhaustive). The metric for lifetime was defined as the time until the first node ”dies” because of energy depletion. The simulated lifetimes are idealistic. The battery technology, timing and intensity of the applied load and the physical environments have an impact on the actual lifetime, as demonstrated by Feeney et al. [121].

The exhaustive algorithm is only slightly better than the dynamic partitioning approach. This renders the latter a very good approximation. The lifetime gain for longer windows is substantial even in the case of the first fit algorithm, where no measurement combinations occur. This is due to all the nodes waking up at the end of each window, which induces considerable energy costs. The longer the windows are, the more rarely the scheduling has to be performed. Hence, the nodes can sleep for longer periods before they have to wake up again for the scheduling of the next window. This also explains why the other two strategies do not perform significantly better than the first-fit strategy for small window sizes (< 1): For very small window sizes there was not much of a difference between the three strategies, because there were not enough opportunities to go to sleep at all. For larger window sizes ($>= 1$) the first-fit strategy reached significantly lower lifetimes than the other two strategies. On the Waspmote Pro and with a window size of two, for example, first-fit only reached around 60% of the lifetime of the other two strategies.

It is obvious that the Waspmote platform is superior in terms of network lifetime due to its better energy characteristics and better real-time clock. The lifetime of the application on the Waspmote Pro platform was around 1.5 times the lifetime on the



(a) (b)

Figure 8.21: Comparison of the different task-allocation algorithms, IRIS platform, E_{ES3}. The window size is expressed relatively to the mean entity measurement period

IRIS platform.

A simulation of a long-running experiment is shown in Figure 8.21. The window sizes varied between one and ten. Due to the computational complexity of the exhaustive approach we were running the experiment only with the dynamic partitioning and the first fit strategies. Up to a window size of five the lifetime increases significantly, due to longer sleeping periods. From that point on, the increase in lifetime starts to level off. On both platforms starting from a window size of five or six no significant lifetime increase was observed. It has to be noted though, that longer window sizes do not allow the system to adopt to change, as discussed in Chapter 7.6.4.

We now quantify the impact of tolerances on the lifetime. Figure 8.22 shows how lifetime varies with window size for different tolerance values. This experiment used only the dynamic partitioning strategy. As expected, larger tolerance values result in lifetime improvement, as more combinations are possible. The effect increases with an increase of the window size. The impact on the Waspmote Pro platform seems higher than on the IRIS platform. This is explained by the very low energy consumption on Waspmote Pro while sleeping. On that platform, additional sleeping time contributes more significantly to the network lifetime than on the IRIS platform. Of course, very large tolerances lead to best results, but only rarely reflect real-world scenarios. Between the very large tolerance scenarios and the very low tolerance scenarios there is a group in the middle with a considerable lifetime increase compared to the low tolerance group.

Next, we evaluated the impact of sensor overlap. Naturally, a high amount of overlapping will prolong the network lifetime due to our penalty based scheduler. Figure 8.23 presents the lifetime of the network for varying number of eligible sensors and percentage of overlapping occurring among them. Lets call n_s the number of sensors

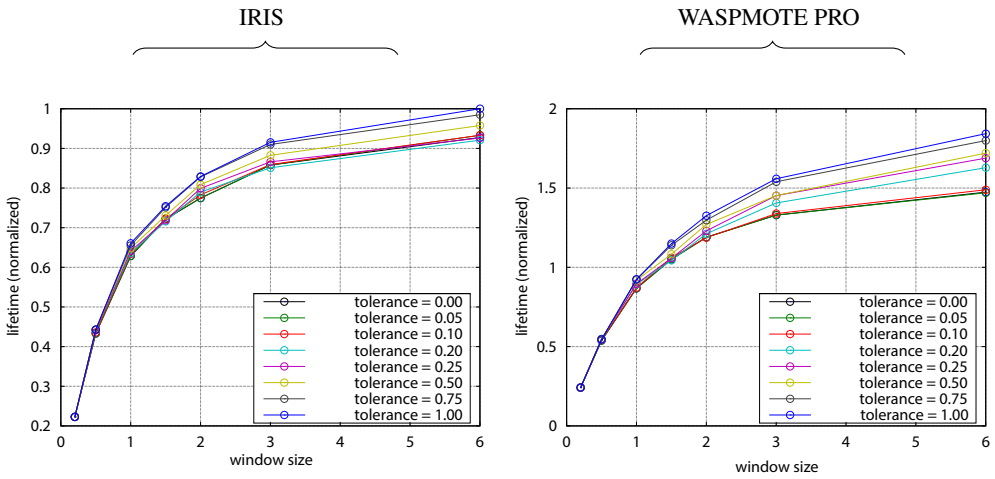


Figure 8.22: network lifetime vs window size for different tolerance values. IRIS platform (left hand side) and Waspmote Pro platform (right hand side), E_{ES1}, both tolerance and window size are expressed relatively to the mean entity measurement period

that is indicated by the different lines in the plot. The overlapping percentage expresses the percentage of entities that could be measured by n_s sensors. The remaining entities in this series of experiments could only be measured by one sensor. For example, for 50% overlapping, we get the result of an experiment where 50% of the entities can be measured by three different sensors and the rest can only be measured by one. Similarly to when varying the tolerance, considerable improvement in lifetime is observed when the number of sensors eligible to measure the entities is increased and more overlapping between them occurs. The penalty-based sensor allocation (see Chapter 7.6.2) ensures that all sensors are more or less equally utilized.

8.8.4.1 Dynamic Behavior of the System

We now evaluate the dynamic behavior of the system, as explained in Chapter 7.6.4. First, we add an arbitrary incoming measurement that is unknown to the system. Second, we extend our evaluation by assuming the system has some knowledge about the probability distribution of the incoming, otherwise unknown, measurements over time. We use the concept of virtual measurements as explained in Chapter 7.6.4.2.

Dynamic incoming measurements, as introduced in Chapter 7.6.4, are served by adding separate requests that are unknown to the scheduler at the time of scheduling. The scheduler has to dynamically do a best effort integration into its plan or schedule them for the next window. We modeled those incoming measurements, for each $e \in \mathbb{E}$, by adding new measurements reusing the existing experimental framework.

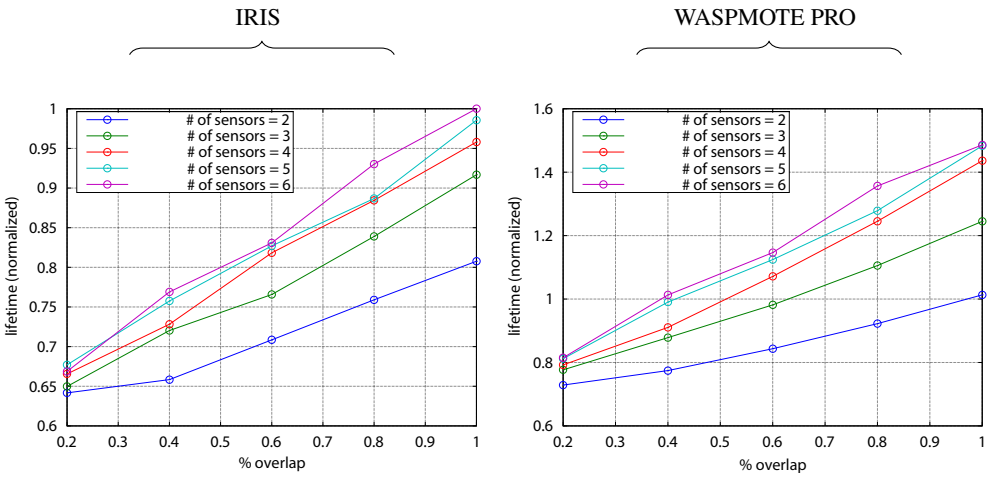


Figure 8.23: Network lifetime vs sensors overlapping percentage. IRIS platform (left hand side) and Waspmote Pro platform (right hand side), E_{ES1}, for different numbers of sensors overlapping. Percentage values are normalized to [0,1]

Figure 8.24 shows the incoming requests response latency for varying window length, as well as the corresponding lifetime. The latency for an incoming measurement is measured as the difference between the time it is requested and the time it is actually returned. Latency, in this case, is always the time until fresh data is delivered as a response to a request. In a real-world scenario, some callers might be satisfied with cached data, if they wish so. We did not further consider caching scenarios, as this measurement would not be visible to the scheduler anyway. It is also expressed relatively to the mean entity measurement period:

$$latency_m = \frac{ts_{returned} - ts_{requested}}{mean(\omega)}$$

Prolonging the windows results in increased latencies. This is because measurements cannot be served in a particular window have to wait until the start of the next window. At the same time, the larger the window size is, the more probable it is that a query able to accommodate an incoming measurement has already been scheduled and has not yet been executed.

Next, we added virtual measurements (Chapter 7.6.4.2) in order to make the system more responsive to dynamic measurements. To derive the time intervals of the virtual measurements, we assume a Gaussian distribution from which we were sampling the timestamps for the simulated dynamic incoming queries. We used the same assumptions as in Chapter 8.8.1 to choose a mean μ, with $\sigma = 0.1\mu$. In an ideal case, this means that if we choose a confidence level of 0.2 we will observe 20% of the incoming measurements to be within our virtual measurements interval. In a real-world scenario

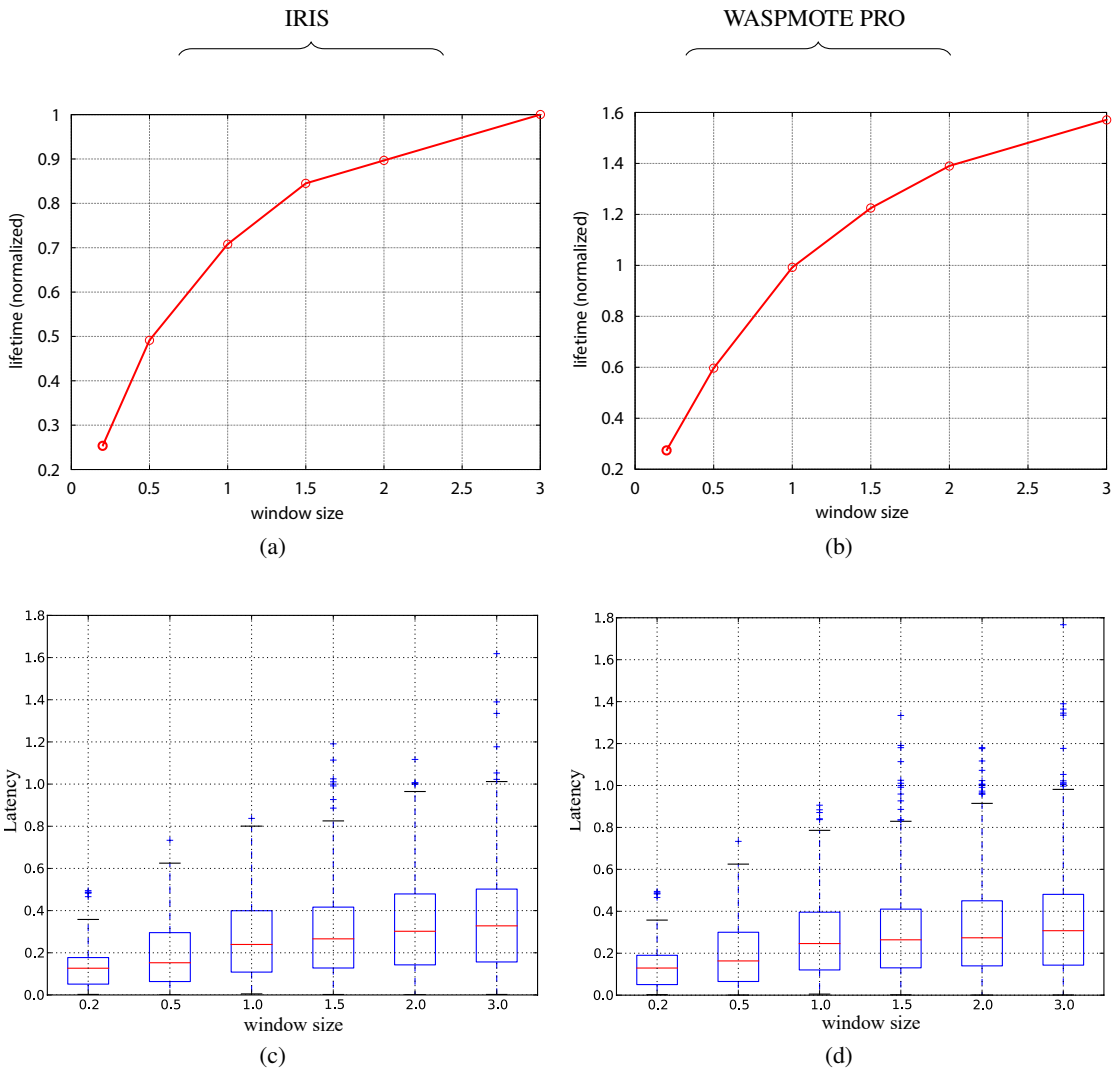


Figure 8.24: Network lifetime and latency vs window size, IRIS platform (left hand side) and Waspmote Pro platform (right hand side), E_{ES1}. The window size is expressed relatively to the mean entity measurement period.

this could be an intelligent container moving along the supply chain that is expected to reach waypoints at a given time, but the actual arrival cannot be precisely detected – for example, because of traffic. Figure 8.25 provides an overview of the change in the network lifetime and the latency of the incoming measurements when virtual measurements with varying confidence levels are used for different window sizes. It is seen that with increasing required confidence levels, the gain from prolonging windows is very small compared to the loss from keeping the nodes awake with the virtual queries. The higher the required confidence level, the longer the node has to be switched on, waiting for the query, resulting in higher energy consumption and thus lower network lifetime. The lower the confidence levels, the higher the latency due to missed wrongly assumed virtual measurements.

This trade-off between network lifetime and the reduction of the incoming measurements latency is shown in Figure 8.26. The effect is quite strong, as high confidence levels lead to high duty-cycles and few sleeping periods. If network lifetime is the key property of a system, then it should be designed in such a way that, as far as these arbitrary incoming messages are concerned, either the system is provided with a very good estimation of when the actual measurements will happen with lower deviations than the one we assumed, or such that latency is not an issue. A system, where the sensor measurement can be predicted with a deviation of only a few seconds or it is not happening at all otherwise, might work well with large confidence levels. Furthermore, a protocol with a better idling energy footprint than MRv6 should be considered.

8.8.4.2 Packet Loss

Packet loss, as mentioned in Chapter 7.6.5, may cause measurements to miss their deadlines and impacts network lifetime. In the experiment presented in Figure 8.27 we varied the percentage of packet loss from low to high for five different tolerances (5s, 10s, 25s, 50s and 100s). The closer the tolerance is to the round-trip time the more likely a deadline miss is to occur. Nonetheless, even small tolerances are still somewhat resilient towards deadline misses for relatively small to moderate packet loss rates. The network lifetime is relatively independent of the tolerance. Naturally, it is very sensitive to packet loss. In harsh and loosy environments the system architect would have to adapt the combination logic of the dynamic partitioning algorithm: measurements could be excluded from the combining algorithm if the remaining time were be smaller than a given threshold, to ensure that a retry is possible within the time constraints.

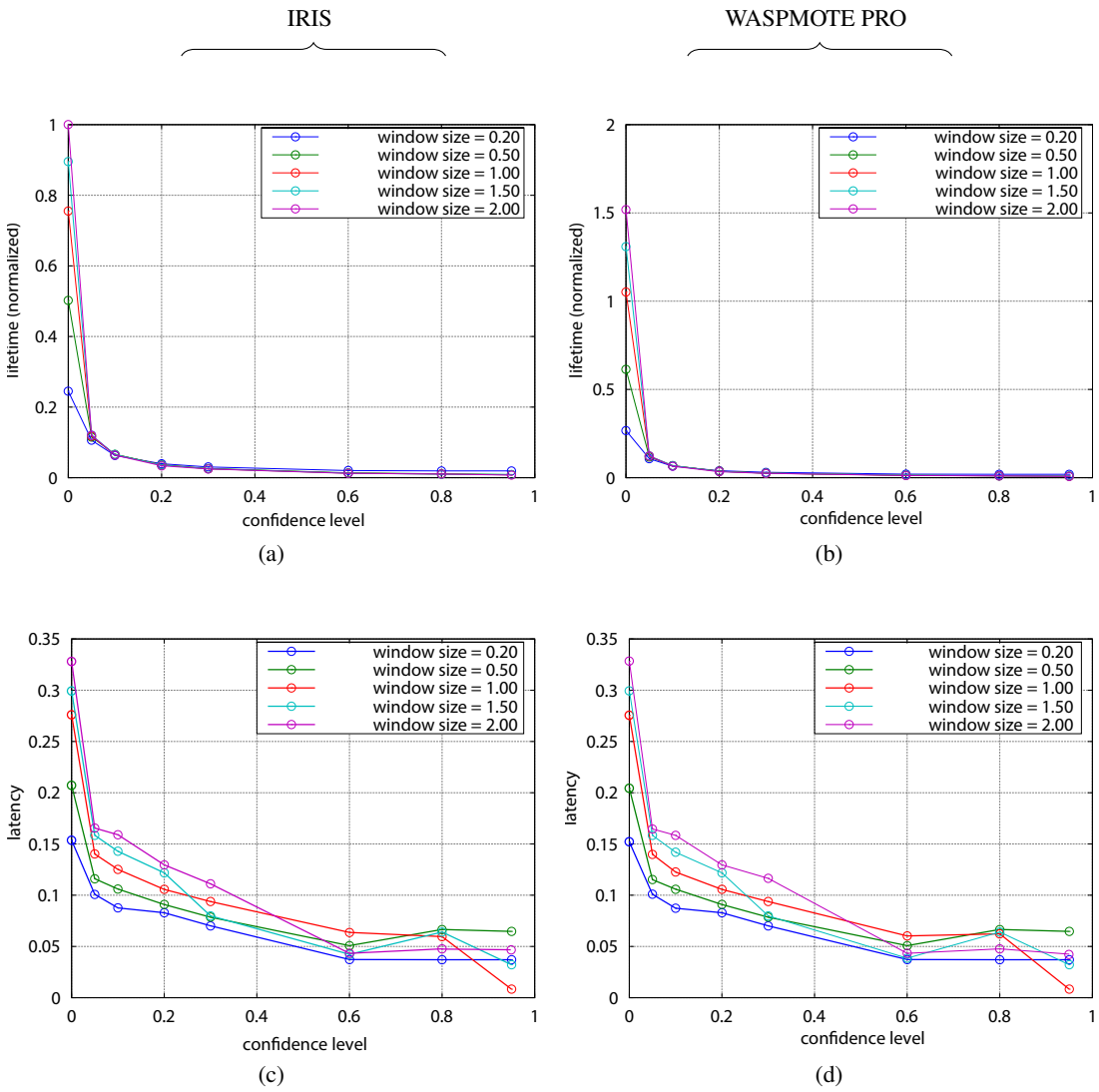


Figure 8.25: Network lifetime and incoming measurements latency vs virtual measurements confidence level, IRIS Platform (left hand side) and Waspmote Pro platform (right hand side), E_{ES2}, for different window lengths

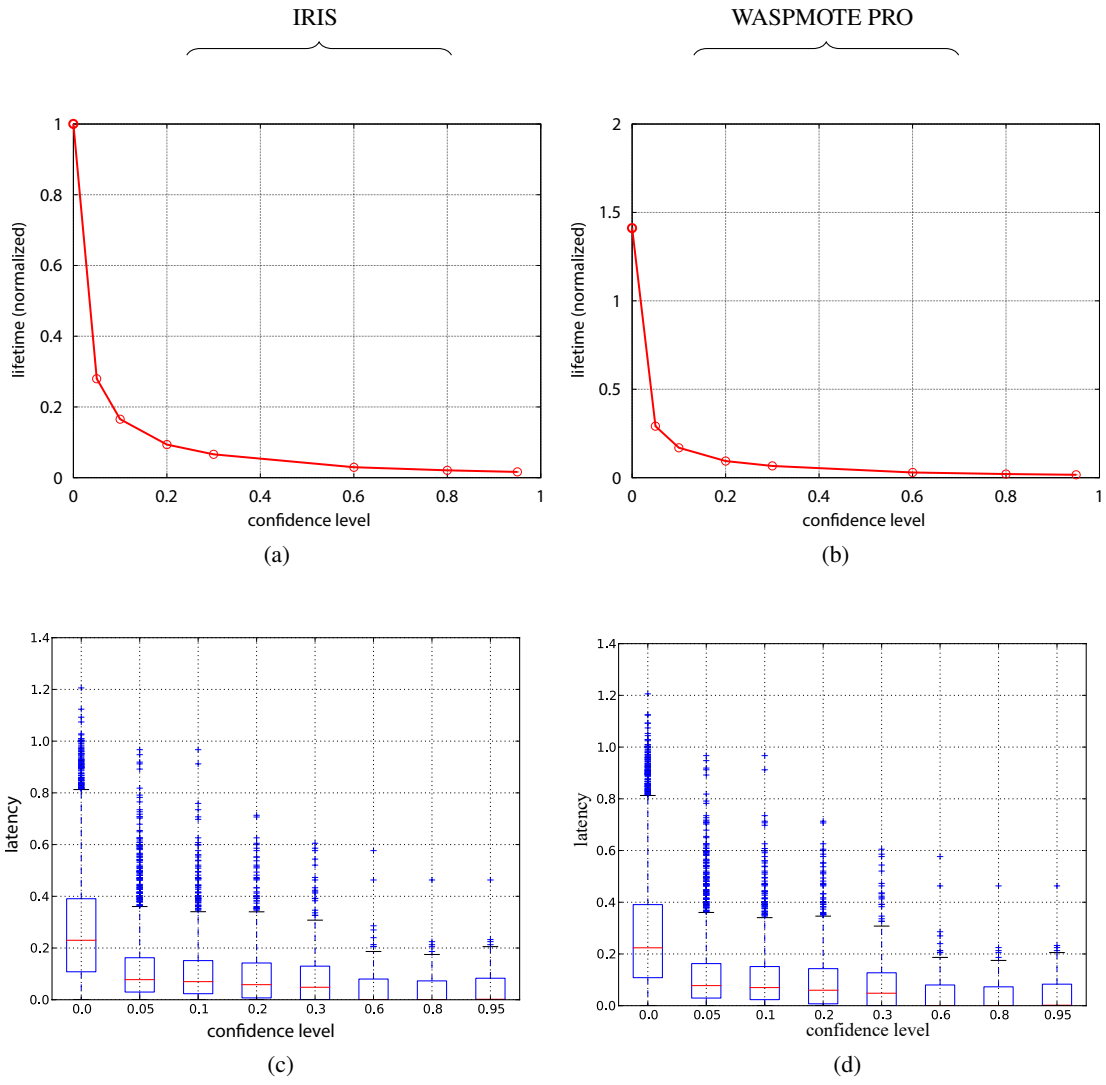


Figure 8.26: Lifetime and latency change, IRIS platform (left hand side) and Waspmote Pro platform (right hand side), E_{ES2}, with varying confidence levels covered by virtual measurements

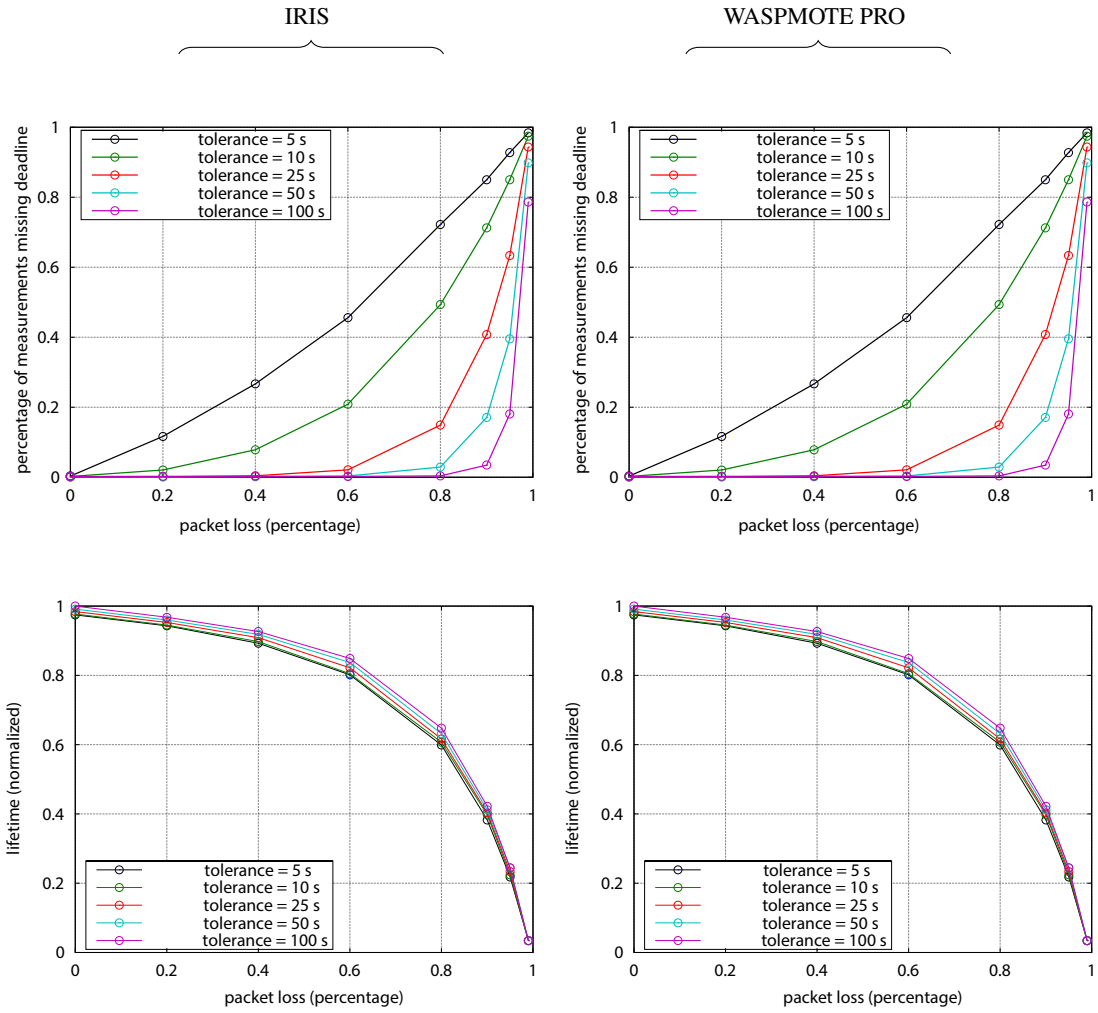


Figure 8.27: Percentage of measurements made outside entities tolerance and network lifetime for varying percentage of packets lost and tolerance values. E_{ES1}. Percentage values are normalized to [0,1]

8.8.5 Conclusions

We presented the evaluation results of our application-layer sleepy node implementation as introduced in Chapter 7. Our experimental platform, Mote Runner, started to support a second platform – Waspmote Pro – with Mote Runner beta13. Based upon our on-mote simulation and evaluation framework (Chapter 8.6) we evaluated our implementation on IRIS and Waspmote Pro motes. A modern platform such as Waspmote Pro provides a huge benefit compared to previous platforms with regard to mid- and longterm sleeping, due to its advanced sleep states.

First, we derived a hybrid energy model that is used to calculate the savings of a deep sleep compared to idling. Within our windowing-based framework (Chapter 7.6) we evaluated three different strategies: a first fit, an exhaustive strategy and a heuristic named dynamic partitioning. The dynamic partitioning heuristic is based on the observation that measurements that can be combined (executed together) are often sequential in time. We have shown that the dynamic partitioning approach does not perform significantly worse than an exhaustive (optimal) approach and considerably better than a simple first fit. It could be shown that the sleepy nodes implementation on Waspmote Pro achieved an enhanced network lifetime that was up to 50% higher than on IRIS motes. Without sleeping, the energy consumption for idling and running is comparable on both platforms. One of the main problems with sleepy nodes is that they cannot respond to any request while sleeping; thus, they, cannot react in a dynamic environment. We also evaluated our system in such dynamic scenarios: first, by just using the next available mote (if any) to serve a request in this window. Second, we assumed that the system knows the probability of a request, but not its exact timing. To cope with them, we introduced *virtual* measurements. The measurements are called virtual, because at the time of scheduling its not yet clear when they will become real measurements. The scheduling algorithm now keeps the mote on for the time period specified by the virtual measurements in order to increase the responsiveness of the system. Our experiments have shown though, that such a strategy – while indeed increasing the responsiveness – decreases the network lifetime significantly. It is important that system designers carefully choose a window size that matches their requirements concerning network lifetime. The less often the system changes, the better and the more soft their real-time requirements are. Higher tolerances allow the algorithm to combine more measurements and, therefore, save energy.

Knowledge about this long term sleeping periods cannot be derived easily by the motes themselves, or by a network-layer protocol. A semantics-aware enterprise can benefit from the knowledge of its entities to maximize the energy savings. Sleepy nodes, in conjunction with a windowing-based framework, allow the system to maintain a degree of flexibility in order to adapt to a changing environment. Sleepy nodes can benefit from a semantic platform in various ways. First, the data stored about entities and positions can be derived easily. In addition, the dynamic behavior can be significantly improved if the system is able to reason about upcoming measurement requests. In combination with learning algorithms that leverage on the semantic data

and gathered data, patterns could be deduced that improve the forecast of measurements and adopt virtual measurements accordingly.

8.9 Conclusions

In this chapter, we presented the evaluation results of our work. We applied empirical and experimental research methodologies. Our empirical evaluation stands on three pillars: first, an internal requirements gathering process (internal stakeholders and domain experts); second, a broader and structured requirements gathering process based upon the IoT-A stakeholder workshops; third, a survey on the use of semantics in the IoT. In addition, we applied an architecture evaluation methodology to gain insight into the strengths and weaknesses of our proposed architecture. The experimental evaluation is based on the Mote Runner platform running JAVA-based applications. We prototyped our CoAP, OData and Sleepy Node implementation and measured, among others, response times and energy.

The proposed architecture has been evaluated using an architecture evaluation methodology based on SAAM [198] and extended with industry best practices. The methodology is described in Chapter 8.1. Our scenario-based architecture analysis showed that the architecture generally fulfills the requirements. It is flexible enough to accommodate a variety of changes. The semantic-awareness approach using Linked Services fits the needs of enterprises well. However, we also identified some risks. The architecture and its components are not well suited for hard real-time requirements. Additional effort is necessary to adopt it accordingly and to introduce mechanisms to support hard real-time requirements.

The stakeholder workshops and the survey generally showed an expected move of IoT-applications towards standardization – mainly towards REST-based systems built upon IP-technology (6LoWPAN) – and an increased interest in semantics. The stakeholder workshops and the survey were conducted in a community that traditionally worked with 802.15.4 networks, we especially did not survey mobile (as in IPad or IPhone) related developments. The use of specialized protocols, which traditionally played an important role in the IoT community, is declining. They are expected to be almost completely replaced by already existing standards. On the application layer the participants were split when asked if today's application-layer protocols sufficiently support the special needs of IoT-applications. All other layers are clearly expected to move towards standards. One explanation could be that the large payloads generated by application-layer protocols and their processing complexity are beyond the constraints of those devices. The energy and processing power needed to use protocols such as SOAP is a magnitude higher than of simple protocols such as CoAP. Furthermore, in the past a lot of research has been done on the network, transport and the MAC-layer. The lack of standardized application-layer protocols could also be because the IoT-market is still yet to be established. Initiatives, for general-purpose application-layer protocols like CoAP are relatively new. Different future solutions – for example, in the

area of home automation, could compete and finally converge towards a standard. Of course, with better battery technology coming up and the availability of more low-power platforms, it is also possible that traditional technologies like HTTP, its successors, or SOAP will play a major role even in future (constrained) IoT-applications. The need for general-purpose specialized IoT-protocols thus might vanish and applications, which are so constrained that they cannot run such a protocol, will then use a specialized protocol and an application gateway solution. The use of semantics is anticipated and some need for it seems to be recognized, mainly driven by the anticipation of machine-learning systems. Of course, the the enterprise community is very much interested in such opportunities. A more traditional "bit-and-byte"-oriented community as, for example, automotive might have a different view. The results might not be easily generalizable to domains other than an enterprise/ERP community. We also looked into a phenomenon called *semaphobia*[225] – the observation that many developers are somewhat reluctant to use semantic technologies, even if they see benefit in it. The top reason, most of our participants assumed to be responsible for this effect, are knowledge and awareness of the development staff, followed by a lack of standardization.

We compared Linked USDL for IoT with related approaches based on a set of criteria. Our criteria-based evaluation shows that Linked USDL for IoT fulfills most requirements. Compared to the current gold standard, – that is, the WS-* family – it lacks infrastructure and tooling support, as well as standardization. In those two aspects WS-* is clearly superior to all other languages, including Linked USDL for IoT. Nonetheless, Linked USDL for IoT is stronger in the area of modeling business aspects. Furthermore, it is — naturally — more tailored towards the needs of Internet of Things applications: it supports distributed descriptions, comes with a Quality of Information vocabulary, and generally offers a a better support for semantic extensions. The WS-* protocol family, representing a non-semantic means of describing services, of course lacks any semantic modeling abilities. In comparison with its direct competitors, OWL-S and SEREDASj, it is either on par (OWL-S) or even more advanced (SEREDASj). Both, OWL-S and SEREDASj, lack support for business aspects. Besides, SEREDASj is currently tightly coupled to JSON. For industry-wide adoption of Linked USDL for IoT it needs to improve on tooling. An open standardization process would prevent users from fearing a vendor lock-in and strengthen their belief in a further development of the language.

As part of our architecture, we prototyped two protocols on the Mote Runner platform: CoAP and OData. OData was chosen as a top-down approach for making semantics available on constrained IoT-devices. Our proof-of-concept implementation showed that OData can be made available to such devices. We compared OData in its two representations JSON and ATOM/XML with a CoAP baseline. The CoAP baseline could be made semantics-aware using Linked USDL. Therefore, it can be seen as a possible data representation in a bottom-down approach. The difference between the baseline and OData representation can be considered the price to be paid for semantics on our IoT-platform. The ATOM/XML representation is possible, but both its power

consumption and its service access times are order of magnitudes larger than with JSON or the CoAP baseline. Compression improves the situation for queries with larger payloads and reduces both the power consumption and the service access times. In the case of the ATOM representation compression reduced large payloads, as the mandatory $metadata, from around 6.0 times the baseline to 3.8 times the baseline.

The JSON representation performed better and was close to the CoAP baseline, but on average also had a relative penalty of between 1.2 and 2.6 times the CoAP SAT value. Compression did not contribute much in the JSON case for small payload sizes. In the worst case it still needed around twice as much time as the CoAP representation. The energy consumption was much lower than the energy consumption of ATOM, and often as good or only slightly worse than the CoAP baseline. The still widely used OData protocol v3 has the drawback that the overall metadata cannot be specified in JSON representation, because it is undefined. This changed with OData v4. The change will make it even more suitable for IoT-applications. Considering that our IoT-platform is on the lower end of the spectrum (a Class 0 device), OData can be expected to be an even more viable alternative on more powerful devices.

We evaluated our sleepy node implementation. The application-layer sleepy node implementation serves as an example of how a semantics-aware enterprise can benefit from the information available at the application layer, for example in a supply chain or warehouse scenario. The sleepy node architecture and implementation integrates well into our proposed platform. Our experiment was based on two IoT-platforms: IRIS and Waspmote Pro. The Waspmote Pro has an advanced sleep state that consumes only very little energy and thus achieved considerably higher energy savings than the IRIS platform. First, we had to extend the existing MRv6 implementation to to allow mid- to longterm sleeping periods. The challenge here was to integrate into the beacon-based protocol and to calculate a wake-up point that is close to the expected next beacon after sleeping. We derived a hybrid energy model for both platforms. The energy model can be used to calculate beneficial sleep times. We were able to show the general benefit of application-layer sleepy nodes. We experimentally evaluated three different strategies: a first fit, an exhaustive approach and a heuristic named dynamic partitioning. We demonstrated that within a time-sliced (windowed) environment this heuristic – based upon combinable subsequent measurements – further reduces the energy consumption. It achieved only slightly worse network lifetimes than the exhaustive approach and generally performs better than first fit. The windowing approach we used has some benefits and some drawbacks. Beneficial for the calculation of the sleeping periods is that no changes to the system are expected. This allows an calculation of the sleeping periods. Nonetheless, this is also one of the major drawbacks: Changes to the system can only be done at the beginning of each window. Larger window sizes have shown to increase the network lifetime, therefore there is a trade-off to be made between window size and the number of possibilities to change the system.

For requests that arrive between two windows one naive approach would be to just schedule it for the next window and inform the requester. Another way of handling such

situations would be to return cached data and information about the age of the data. We experimented with two different scenarios: we try to serve arbitrary incoming requests anyway, and we consider requests that are expected i.e. a known probability distribution exists. In the first case we try combine the request with an already known measurement and schedule the request there, or look into the communication paths for a suitable node. If no such node exists, we resort to the naive strategy of scheduling it starting from the next window. For requests were such a node exists, the data can be served earlier than by waiting for the next window. We also tested scenarios in which we assume to know the probability distribution of some expected requests. We still want to leverage on sleeping, instead of keeping the node on all the time waiting to serve the request. We schedule therefore some virtual measurements i.e. measurements that might become real queries. For the virtual measurements we specified confidence levels. The higher the confidence level the longer we wait for the query. Virtual measurements reduce the time we need to answer the requests that would otherwise be handled like arbitrary requests. Nonetheless, the higher the confidence level the smaller the network lifetime. Therefore, a good estimation of the time frame the requests arrives is necessary for long network lifetimes.

Part V

Conclusions and Outlook

In this part of the thesis we summarize our work and present the main conclusions. We also provide on Outlook presenting opportunities for further work.

Chapter 9

Conclusions and Outlook

Semantics-aware integration and REST-based services are considered major challenges in future enterprise scenarios coined as the *Sensing Enterprise.* The Internet of Things, still being a relatively constrained environment, is currently in the lead with regard to applying semantic technologies. More and more vendors, such as Google, are entering the field with semantic technologies. This thesis explored various aspects of the service-based integration of IoT-devices into a semantics-aware enterprise. We presented conceptual and architectural work on IoT-services, and their integration into semantics-aware enterprises. Based on our integration platform we explored two approaches of semantics-aware enterprise integration: a bottom-up approach and a top-down approach. The bottom-up approach semantically enriches existing protocols. It is based on Linked USDL for IoT, a service description language specifically tailored towards the needs of the IoT. We developed Linked USDL for IoT as an extension of the Linked USDL family of vocabularies. The top-down approach is based on an already existing semantics-aware enterprise protocol which we moved to and evaluated on constrained devices. Furthermore, we discussed possibilities of modeling IoT-specific entities and properties with OData.

We extended our proposed integration platform to support also sleepy nodes. Sleepy nodes are IoT-devices that go to sleep, but still appear as connected to the user. We introduced a windowing-based measurement framework supporting three allocation strategies: a first-fit, an exhaustive approach and a self developed heuristic called dynamic partitioning. We were able to show that dynamic partitioning performs significantly better than first fit and only slightly worse than the exhaustive approach. We also investigated how a window-based framework can react to requests that are unknown at the beginning of a particular window.

9.1 Conclusions

In Chapter 3, we conducted a literature survey on definitions and usages of services in the Internet of Things. At the beginning of our work, we realized that the term IoT-service was often used in an intuitive way and that its relationship to the Internet

of Things in general was unclear. We suggested a definition of Internet of Things services that differs from the more common idea of using service as a synonym for technical interface. Our definition based on the term "transaction" that is consistent with definitions found in service science. The classification and the surrounding concepts are centered on the entity and not that much on the technical representations or means of realization through low-level services. This idea guides the work on Linked USDL for IoT, which we introduced in Chapter 5. We presented conceptual work on the architectural building blocks and design considerations for an Internet of Things service integration framework. We showed how services can be integrated into the Internet of Things-Architecture, which is semantics-aware.

Chapter 4 built on the ideas of Internet of Things-services in general and described how they can be embedded into an enterprise environment. We presented an architecture for an IoT-aware enterprise, typically named *the sensing enterprise*. Its novelty is the combination of two concepts – linked services and distribution – which, when combined, serve the needs of both enterprises and small constrained IoT-devices. We argue that (distributed) linked services are especially well suited for IoT-applications given their limited battery power, as well as storage and processing constraints. They allow only a minimal subset of the service description to be stored on the mote. More information can be accessed on-demand. We identified the key drivers that will drive enterprises in the coming years, namely: interoperability, use of standardized technologies, enablement of sense-making, and realtime business decision support. Especially interoperability, sense making, and real-time business support would benefit from semantic service descriptions and linked services. We suggest an abstraction named semantic physical business entity, which can be used in enterprise architectures and especially in modeling environments.

Linked USDL for IoT, our extension of Linked USDL for supporting the Internet of Things, is presented in Chapter 5. It has the advantage of being a service description language that goes beyond the technical interface. It consists of different modules that cover functional, operational and business aspects. In addition, it allows the usage of already existing domain specific vocabularies, because it is modeled in RDF. We extended Linked USDL to support REST-based IoT technical interfaces and properties that are IoT-specific. We contribute four new vocabularies to Linked USDL to support Internet of Things applications. Each of this vocabularies targets a specific aspect of the Internet of Things. The covered aspects are events, quality of information, technical endpoints and the REST paradigm. Furthermore, we embedded it into related ontologies. We evaluated Linked USDL for IoT by performing a criteria-based evaluation and a multivariate comparison. We identified nine key properties a service description for the IoT should have and their importance. The criteria-based evaluation compares Linked USDL for IoT to related approaches, namely WS-*, hRESTS, WADL, OWL-S and SEREDASj. The multivariate comparison also takes the importance of the properties into account. In the IoT-domain, we showed – based on our nine properties – an advantage of RDF-based languages over traditional XML-based technologies. Linked

USDL for IoT is one of the few approaches that have light-weight design principles in mind. To some degree, however, it lacks tool support and standardization. In these two aspects (tool support and standardization) WS-* can be seen as gold standard and Linked USDL for IoT and similar approaches will have to catch up.

Our empirical results are supported by a survey, stakeholder workshops, and internal experts. In particular, we were interested in the acceptance of semantics in the IoT, and its relationship to IoT-technologies. We can conclude that, within the surveyed group, there is a strong expectation towards the future enterprise IoT to be based on RESTful paradigms supporting standards known from the Internet community. On the application-layer the participants were split when asked if today's application-layer protocols sufficiently support the special needs of IoT-applications. One explanation could be that the large payloads generated by application-layer protocols and their processing complexity are beyond the capabilities of those devices. HTTP, for example, is quite verbose. It remains open in our opinion if advances in hardware and new developments on the web (like HTTP/2 or SPDY) will reduce the need for a general-purpose IoT application-layer protocol, or if the IoT-community will have its own protocol stack. It is interesting that Enterprises move towards IoT-protocols, instead of moving enterprise protocols (as SOAP) towards IoT. The observation that semantics while being advocated for many years now still did not gained widespread use could be explained by the lack of training and the fact that the first generations of computer scientist educated in semantic web just entered the enterprises. In general, most people see potential for semantics in the Internet of Things and expect usages starting from describing things and services up to reasoning on a semantic layer.

As explained in Chapter 1.4 our IoT-stack is based on CoAP as application-layer protocol. We implemented a CoAP solution on the Mote Runner platform. In particular, we implemented the base specification [353] and two extensions: block-wise transfer [352] and observe [162]. We implemented the CoAP protocol (like all our software) in Java. On one hand, it was quite an advantage because the ramp-up time was quite low, compared to – for example – nesC. On the other hand, however, it was counter-intuitive to work with the CoAP specification which is based on unsigned datatypes. The use of Java in low-level and protocol programming would, in our opinion, greatly benefit from unsigned types. For allowing (distributed) linked services some kind of redirection possibility is desirable, similar to what HTTP offers. CoAP does not have this property at the moment. We therefore discussed for possible solutions: 3.xx message codes, CoAP options, Content Type and Data encoding. We concluded that, while the 3.xx family might be the most desirable, currently the most promising solution is to move the problem to the application itself. In our opinion this is a gap in the standard that needs to be addressed in future CoAP releases.

We evaluated a top-down approach by using an existing semantics-aware enterprise protocol and applied it to Class 0 IoT-motes. We propose the OData protocol as an end-to-end solution for the integration of REST-based sensor networks into enterprise IT systems. We concentrated on direct communication with the Mote. Interoperability

with enterprise systems is one of main arguments for using OData. On the downside, one has to stay within the modeling capabilities of OData: OData currently offers fewer semantic modeling abilities than description languages based on RDF. It was coined ”semi-semantic” by some people, because of its lack of integration into the semantic web. This seemed to be definitely justified for OData v2, already less justified for OData v3. With OData v4, semantic annotations are fully supported. Notably, the OData entity abstraction fits a typical sensor network usage very well, including abstracting the topology and the sensors (or actors) as entities. This makes OData a very good abstraction for systems that can be seen as sensor network databases. Querying the sensor network becomes easily possible, as well as accessing temporal data. Moreover, these can be carried out in a standardized RESTful way that is immediately accessible by other applications. Compared to Linked USDL for IoT OData follows a close-world assumption. Linked USDL for IoT is better integrated into the semantic world and leverages on semantic technologies and linking well known vocabularies together. A combination or integration of Linked USDL and OData, or more general RDF-based languages and OData, is a current research field.

We implemented a proof-of-concept OData-enabled prototype on the IRIS platform. The difference in resource usage and transmission time between the baseline and OData representation can be considered the price to be paid for semantics on our IoT-platform. The ATOM/XML representation is possible, but both its power consumption and its service access times are order of magnitudes larger than with JSON or the CoAP baseline. Compression improves the situation and reduces the power consumption and the service access times. In the case of the ATOM representation compression reduced large payloads, as the mandatory $metadata, from around 6.0 times the baseline to 3.8 times the baseline. Nonetheless, in most cases it is still considerably higher than the JSON or CoAP representation. For a single temperature query with a minimal ATOM representation, it needed 40% more time than JSON and around twice (93% more) the time of the baseline. The JSON representation for this single temperature call was around one third slower than the CoAP baseline.

The JSON representation, in general, performed better and was close to the CoAP baseline, but on average still was in the range of 1.2 to 2.6 times the CoAP SAT value depending on the query. Compression did not contribute much in the JSON case for queries resulting in small payload sizes. This is to be expected though, as (dictionary-based) compression needs more and repetitive data to perform better and significantly reduce the payload. For example $Q7$, a query with a lot of payload where the amount of data could reduced to three thirds of the original: the SAT time immediately decreased by around the same and also power consumption was reduced significantly. The effect is expected. As shown in Chapter 2.8, a strength of general-purpose compression algorithms are large amounts of sensor data [110]. Therefore, OData as a sensor database with queries that return a lot of data, for example collected over a day, will almost certainly benefit strongly from the general purpose compression. Within a very constrained environment, as provided by the IRIS platform,

the differences between the resource usage in terms of memory, processing time and energy consumption of ODATA/JSON, compared to a pure CoAP solution, could be considered small enough to justify the benefits of a full semantic integration, while the ATOM representation is clearly not feasible unless for small use-cases. ODATA/JSON proved to be a worthwhile alternative to pure CoAP solutions, providing enterprises and other systems direct access to data-driven IoT-devices. In contrast to the CoAP baseline solution, direct and standardized access the to semantic information is possible. Moreover, for an enterprise system a sensor node or a wireless sensor network is just another datasource that can be used in business processes, just like any other datasource without the need for adapters or special low-level information. XML processing should be avoided. JSON was superior to ATOM in almost every aspect. Transmission of large ATOM files should be, if possible, redirected to a system with more performance capabilities.

In addition to interoperability, energy efficiency is one of the major issues in wireless sensor system. The concept of sleepy nodes, as defined by IETF [313], is relatively new in the community. Sleepy nodes are sensor nodes that might be in an energy saving mode for some time and thus not available for any communication, but, by definition, a sleepy node should appear as being connected to the network to the user. In Chapter 7 we present an implementation of a Sleepy Nodes extension to our IoT and semantics-aware enterprise architecture. Sleepy nodes can leverage on the semantic knowledge stored in enterprise repositories to save energy. Our prototype system was running on two hardware platforms: MEMSIC Iris and Waspmote Pro. The Waspmote Pro has a more precise clock and a hardware deep sleep mode. Compared to the IRIS platform, which does not have such a deep sleep mode, the Waspmote Pro platform had a significantly longer lifetime. We derived a hybrid energy model, combining time-based aspects and event-based aspects, for both hardware platforms to calculate beneficial sleeping times. We were able to show that our model predicts the actual energy consumption quite accurately. The MRv6 implementation had to be modified to support mid- and longterm sleeping periods. The wakeup time had to be calculated to be as close to a beacon as possible, thus reducing the time a node has to keep his radio on.

We compared three different allocation strategies: exhaustive, first fit and a self-developed heuristic, which we named dynamic partitioning. As expected, the exhaustive approach performed best, but is only feasible for a small number of sensors and entities. Within the constraints of the windowing-based framework the dynamic partitioning approach performed significantly better than the first-fit algorithm, and only slightly worse than the exhaustive approach. We also adopted our windowing-based system to cope with requests that are either unknown at the beginning of a window or only the probability of a request is known in advance. We introduced the concept of a virtual measurement, allowing the system to *guess* the arrival of a request. This increases the responsiveness of the system, but sacrifices a lot of energy. Keeping the mote on in order to wait for a virtual measurement to become a real measurement has a large

impact on the system lifetime and thus should preferably only be used if the period in which a virtual measurement becomes a real measurement is small.

9.2 Outlook

The Internet of Things is expected to grow tremendously over the next decade. Gartner predicts $2.5M per minute in IoT spending and 1M new IoT devices sold every hour by the year 2021. We expect significant growth not only in the consumer sector, but especially in the enterprise sector.

Linked USDL for IoT currently is capable to describe services in the IoT following a REST-based input/output concept. Having semantically-enriched services is a precondition for many upcoming applications. We hope that the vocabulary is taken up by the semantics community and further integrated into reasoning and machine learning scenarios. Furthermore, non REST-based endpoints could be added to increase the compatibility with legacy enterprise systems. Interoperability between systems could be further improved by automatically generate endpoints depending on the application. It would then be possible to generate, for example, a SOAP/WSDL endpoint in addition to a REST endpoint.

We proposed a semantics-aware enterprise architecture and some modeling concepts that center around semantics. We would encourage further work on modeling IoT with semantics in a BPMN-context and its automatic execution. Engines and concepts (such as SPBQL) need to be fully specified and integrated into an business process execution framework. We would like to see experimental results of such a platform. Further work on the distribution of service descriptions, versioning and the combination of distributed service descriptions would also be worthwhile.

OData is an upcoming semantics-aware protocol, which can also play a major role in IoT-applications. Our experiments were conducted with OData v3 and should be moved to OData v4 and its JSON representation. With more advanced hardware platforms new energy measurements with better compression schemes would be interesting for both JSON and also XML. Our modeling of services and sensors for the IoT was based on the capabilities of OData v3. More work on how to model the IoT on OData v4 would be interesting. For a better integration of OData into the Internet of Things new vocabularies are required. Thus, work on the specifications of those vocabularies and their standardization is necessary. Recent work on the integration of OData and the traditional SPARQL/RDF-based semantic web looks quite promising. This would also allow to merge and combine Linked USDL for IoT and OData-based services.

The sleepy node implementation could be enhanced in several ways. The current implementation is very much bound to the capabilities of the Mote Runner Platform and its MRv6 protocol. We suggest to further investigate its behavior on different platforms and with different protocol stacks, with cross-layer approaches in mind. We expect further gains in network lifetime when all layers are implemented in an energy efficient way. Not only the network lifetime as a whole would improve, but especially

our concepts for reacting to dynamic behavior would benefit. For example, virtual measurements would probability perform much better with a more energy efficient MAC and network protocol that is consuming less energy while idling. The dynamic partitioning strategy could be extended. For example, it currently always just takes the current window into consideration. A look-ahead could be implemented that combines measurements from two windows, if the tolerances allow it. The window sizes are currently fixed and chosen by the designer. Dynamic window sizes are an option that would add further flexibility to the system. The virtual measurement concept could also be applied to different scenarios with different underlying probability distributions.

To summarize, interoperability and integration on all layers, including the semantic layer, will be the key for enabling a magnitude of new applications. Compared to the semantic web, which was suggested to be the next big game changer, but did not keep these promises, we suggest a more pragmatic approach. Instead of describing the whole world we go step-by-step. The reluctance to use semantics, termed as ”semaphobia” by Lanthaler and Gütl [225, 223] will shrink over time when semantics are used sense-fully and are gradually introduced into software products. Google for example, announced in 2015 a dedicated operating system for the Internet of Things, and in this context even more important a project called Weave that can be used to semantically describe things using JSON based semantics. Widespread use of semantics, and especially semantic descriptions of things and services, is in our believe only a matter of time. The future will be based upon semantic interoperability, but it will be different than the semantic web movement predicted.

Part VI

Appendix

Acronyms

6LoWPAN	IPv6 over Low power Wireless Personal Area Networks (IETF specification).
A	Ampere.
ACK	acknowledgement.
API	application programming interface.
BPMN	Business Process Model and Notation.
CoAP	Constrained Application Protocol.
CON	confirmable message (CoAP).
CoRE	Constrained RESTful Environments.
CSMA	carrier sense multiple access.
ERP	Enterprise Resource Planning.
EXI	efficient XML interchange.
GPIO	General Purpose Input Output.
HTML	hypertext markup language.
HTTP	hypertext transfer protocol.
IEEE	Institute of Electrical and Electronics Engineers.
IETF	Internet Engineering Task Force.
IoT	Internet of Things.
IoT-A	Internet of Things-Architecture.

IP	Internet Protocol / Integration Platform.
IPI	Integration Platform instance.
IPv4	Internet Protocol version 4.
IPv6	Internet Protocol version 6.
J	Joule.
JSON	JavaScript Object Notation.
JSON-LD	JavaScript Object Notation - Linked Data.
LAN	Local area network.
M2M	machine-to-machine.
MAC	medium access control.
MOTE	Wireless Sensor Node.
MR	Mote Runner.
MRv6	Mote Runner Implementation of IPv6.
OWL	Web Ontology Language.
PBE	Physical Business Entity.
PDU	protocol data unit.
PHY	physical layer.
PRA	Physical Resource Adapter.
PSU	Power Supply Unit.
RDF	Resource Description Framework.
REST	Representational State Transfer.

RX	receiving.
SPARQL	SPARQL Protocol and RDF Query Language (recursive acronym.
SPBE	Semantic Physical Business Entity.
TCXO	Temperature Compensated Crystal Oscillator.
TX	transmit, sending data.
UART	universal asynchronous receiver/transmitter.
UDP	User Datagram Protocol.
URI	Uniform Resource Identifier.
USDL	Unified Service Description Language.
V	Volt.
VM	Virtual Machine.
W	Watt.
WSN	Wireless Sensor Network.

Bibliography

[1] Gregory Abowd, Len Bass, Paul Clements, Rick Kazman, and Linda Northrop, "Recommended Best Industrial Practice for Software Architecture Evaluation." DTIC Document, Tech. Rep., 1997.

[2] Paul Adamczyk, Patrick H Smith, Ralph E Johnson, and Munawar Hafiz, "Rest and web services: In theory and in practice," in *REST: from research to practice*. Springer, 2011, pp. 35–57.

[3] Ben Adida and Mark Birbeck, "RDFa primer: Bridging the human and data webs," *Retrieved June*, vol. 20, p. 2008, 2008.

[4] Kal Ahmed and Graham Moore, "SPARQL / OData Interop," Position Paper for the W3C Open Data Web Workshop, 2013.

[5] Hans Akkermans, Ziv Baida, Jab Gordijn, Nieves Peiia, Ander Altuna, and Inaki Laresgoiti, "Value Webs: using ontologies to bundle real-world services," *Intelligent Systems, IEEE*, vol. 19, no. 4, pp. 57 – 66, jul-aug 2004.

[6] Rama Akkiraju, Joel Farrell, John A Miller, Meenakshi Nagarajan, Amit P Sheth, and Kunal Verma, "Web Service Semantics - WSDL-S," IBM and University Of Georgia, 2005, W3C Member Submission.

[7] Rosa Alarcon and Erik Wilde, "Linking data from restful services," in *Third Workshop on Linked Data on the Web, Raleigh, North Carolina (April 2010)*, 2010.

[8] Rosa Alarcón and Erik Wilde, "RESTler: crawling RESTful services," in *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 1051–1052.

[9] Rosa Alarcon, Erik Wilde, and Jesus Bellido, "Hypermedia-driven RESTful service composition," in *Service-Oriented Computing*. Springer, 2011, pp. 111–120.

[10] Cesare Alippi and Cristian Galperti, "An adaptive system for optimal solar energy harvesting in wireless sensor network nodes," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 55, no. 6, pp. 1742–1750, 2008.

[11] Dean Allemang and James Hendler, *Semantic Web for the Working Ontologist, Second Edition: Effective Modeling in RDFS and OWL*, 2nd ed. Morgan Kaufmann, 6 2011. ISBN 978-0123859655

[12] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu, "Web services agreement specification (WS-Agreement)," in *Open Grid Forum*, vol. 128, 2007, p. 216.

[13] Markus Anwander, Gerald Wagenknecht, Torsten Braun, and Kirsten Dolfus, "Beam: A burst-aware energy-efficient adaptive mac protocol for wireless sensor networks," in *Networked Sensing Systems (INSS), 2010 Seventh International Conference on*. IEEE, 2010, pp. 195–202.

[14] Kevin Ashton, "That internet of things thing," *RFiD Journal*, vol. 22, no. 7, pp. 97–114, 2009.

[15] Atmel Corporation, "ARM920T-based Microcontroller Datasheet 1768MSATARM09-Jul-09," 2009. [Online]. Available: http://www.atmel.com/Images/1768s.pdf

[16] Atmel Corporation, "ATmega 128/ATmega128L 8-bit Atmel Microcontroller with 128KBytes In-System Programmable Flash Datasheet 2467XAVR06/11," 2011. [Online]. Available: http://www.atmel.com/Images/doc2467.pdf

[17] Atmel Corporation, "ATmega640/1280/1281/2560/2561 Datasheet 2549PAVR10/2012," 2012. [Online]. Available: http://www.atmel.com/images/doc2549.pdf

[18] Luigi Atzori, Antonio Iera, and Giacomo Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, Oct. 2010.

[19] Auto ID Center, "Auto ID Labs," http://www.autoidlabs.org/.

[20] Franz Baader, Ian Horrocks, and Ulrike Sattler, "Description logics as ontology languages for the semantic web," in *Mechanizing Mathematical Reasoning*. Springer, 2005, pp. 228–248.

[21] Michael Baar, Heiko Will, Bastian Blywis, Thomas Hillebrandt, Achim Liers, Georg Wittenburg, and Jochen Schiller, "The scatterweb msb-a2 platform for wireless sensor networks," *Freie Universität Berlin, Berlin, Germany, Tech. Rep*, 2008.

[22] Muhammad Ali Babar, Len Bass, and Ian Gorton, "Factors influencing industrial practices of software architecture evaluation: an empirical investigation," in *Software Architectures, Components, and Applications*. Springer, 2007, pp. 90–107.

[23] Muhammad Ali Babar and Ian Gorton, "Comparison of scenario-based software architecture evaluation methods," in *Software Engineering Conference, 2004. 11th Asia-Pacific*. IEEE, 2004, pp. 600–607.

[24] Muhammad Ali Babar, Liming Zhu, and Ross Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Software Engineering Conference, 2004. Proceedings. 2004 Australian*. IEEE, 2004, pp. 309–318.

[25] Siddharth Bajaj, Don Box, Dave Chappell, Francisco Curbera, Glen Daniels, Phillip Hallam-Baker, Maryann Hondo, Chris Kaler, Dave Langworthy, Anthony Nadalin *et al.*, "Web Services policy 1.2-framework (WS-policy)," *W3C Member Submission*, vol. 25, p. 12, 2006.

[26] Mario R Barbacci, Robert J Ellison, Anthony Lattanze, Judith Stafford, Charles B Weinstock, and William Wood, "Quality attribute workshops," Carnegie Mellon University, Tech. Rep. CMU/SEI-2002-TR-019, 2002.

[27] Kenneth C Barr and Krste Asanović, "Energy-aware lossless data compression," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 3, pp. 250–291, 2006.

[28] Alistair Barros, Uwe Kylau, and Daniel Oberle, "Unified Service Description Language 3.0 (USDL) Overview," *SAP Research*, 2011.

[29] Alistair Barros and Daniel Oberle, *Handbook of service description USDL and its methods*. New York: Springer, 2012. ISBN 978-1461418641

[30] Len Bass, *Software architecture in practice*. Addison-Wesley Professional, 2012. ISBN 978-0321815736

[31] C. Basu, J.J. Caubel, Kyunam Kim, E. Cheng, A. Dhinakaran, A.M. Agogino, and R.A. Martin, "Sensor-Based Predictive Modeling for Smart Lighting in Grid-Integrated Buildings," *Sensors Journal, IEEE*, vol. 14, no. 12, pp. 4216–4229, Dec 2014.

[32] Robert Battle and Edward Benson, "Bridging the semantic Web and Web 2.0 with representational state transfer (REST)," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, no. 1, pp. 61–69, 2008.

[33] M. Bauer, M. Boussard, N. Bui, F. Carrez, C. Jardak, J. De Loof, C. Magerkurth, S. Meissner, A. Nettstrater, A. Olivereau, M. Thoma, J. Walewski, J Stefa, and A. Salinas, "Final architectural reference model for the IoT v3.0, Internet of Things Architecture, Deliverable D1.5," 2013. [Online]. Available: http://www.iot-a.eu

[34] Martin Bauer, Alessandro Bassi, Matthieu Boussard, Nicola Bui, Martin Fiedler, Edward Ho, Thorsten Kramp, Sebastian Lange, Karsten Magerkurth, Stefan Meissner, Sonja Meyer, Andreas Nettsträtter, Alexis Olivereau, Alexander Salinas, Matthias Thoma, Rob van Kranenburg, and Joachim Walewski, *Enabling things to talk*, Alessandro Bassi, Martin Bauer, Martin Fiedler, Thorsten Kramp, Rob Van Kranenburg, Sebastian Lange, and Stefan Meissner, Eds. Springer, 2013.

[35] Martin Bauer, Mathieu Boussard, Nicola Bui, Jourik De Loof, Carsten Magerkurth, Stefan Meissner, Andreas Nettsträter, Julinda Stefa, Matthias Thoma, and Joachim W Walewski, "Iot reference architecture," in *Enabling Things to Talk*. Springer, 2013, pp. 163–211.

[36] Markus Becker, Kepeng Li, Koojana Kuladinithi, and Thomas Poetsch, "Transport of CoAP over SMS," 2014. [Online]. Available: http://tools.ietf.org/html/draft-becker-core-coap-sms-gprs-05

[37] David Beckett and Tim Berners-Lee, Eds., *RDF/XML Syntax Specification (Revised)*, ser. W3C Recommendation. W3C, Feb. 2004, http://www.w3.org/TR/rdf-syntax-grammar/.

[38] David Beckett and Tim Berners-Lee, Eds., *Turtle - Terse RDF Triple Language*, ser. W3C Team Submission. W3C, Mar. 2011, http://www.w3.org/TeamSubmission/turtle/.

[39] Mark Beckner and Triston Arisawa, *Microsoft Dynamics CRM API: Development for Online and On-Premise Environments*. Apress, 2013. ISBN 978-1430263463

[40] Mike Belshe, Martin Thomson, and Roberto Peon, "Rfc7540: Hypertext transfer protocol version 2 (http/2)," Internet Engineering Task Force, Tech. Rep., 2015.

[41] Maria Bengtsson and Sören Kock, " coopetition in business networksto cooperate and compete simultaneously," *Industrial marketing management*, vol. 29, no. 5, pp. 411–426, 2000.

[42] Tim Berners-Lee, "Linked data - design issues," W3C, Tech. Rep., 2006. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[43] Tim Berners-Lee and Tim Connolly, Eds., *Notation3 (N3): A readable RDF syntax*, ser. W3C Team Submission. W3C, Mar. 2011, http://www.w3.org/TeamSubmission/n3/.

[44] Jan Beutel, "Fast-prototyping using the BTnode platform," in *Design, Automation and Test in Europe, 2006. DATE'06. Proceedings*, vol. 1. IEEE, 2006, pp. 1–6.

[45] Jan Beutel, Matthias Dyer, Martin Hinz, Lennart Meier, and Matthias Ringwald, "Next-generation prototyping of sensor networks," in *Proceedings of the 2nd*

international conference on Embedded networked sensor systems. ACM, 2004, pp. 291–292.

[46] Jan Beutel, Oliver Kasten, and Matthias Ringwald, "BTnodes–a distributed platform for sensor nodes," in *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, 2003, pp. 292–293.

[47] Matthias Biehl, Jad El-Khoury, Frédéric Loiret, and Martin Törngren, "On the modeling and generation of service-oriented tool chains," *Software & Systems Modeling*, vol. 13, no. 2, pp. 461–480, 2014.

[48] Dennis JA Bijwaard, Wouter AP van Kleunen, Paul JM Havinga, Leon Kleiboer, and Mark JJ Bijl, "Industry: Using dynamic wsns in smart logistics for fruits and pharmacy," in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2011, pp. 218–231.

[49] Christian Bizer, Kai Eckert, Robert Meusel, Hannes Mühleisen, Michael Schuhmacher, and Johanna Völker, "Deployment of rdfa, microdata, and microformats on the web–a quantitative analysis," in *The Semantic Web–ISWC 2013*. Springer, 2013, pp. 17–32.

[50] Christian Bizer, Tom Heath, and Tim Berners-Lee, "Linked data-the story so far," *Semantic Services, Interoperability and Web Applications: Emerging Concepts*, pp. 205–227, 2009.

[51] Hendrik Bohn, Andreas Bobek, and Frank Golatowski, "SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains," in *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, april 2006. doi: 10.1109/ICNICONSMCL.2006.196 p. 43.

[52] Thomas Michael Bohnert, "FI-WARE: The Future Internet Technology Foundation," 2011.

[53] Carsten Bormann, "A TCP transport for CoAP," 2014. [Online]. Available: https://tools.ietf.org/html/draft-bormann-core-coap-tcp-01

[54] Carsten Bormann, Mehmet Ersue, and A Keranen, "Terminology for constrained node networks," *RFC 7228*, 2013. [Online]. Available: http://tools.ietf.org/html/rfc7228

[55] Matthias Born, Florian Dörr, and Ingo Weber, "User-friendly semantic annotation in business process modeling," in *Web Information Systems Engineering–WISE 2007 Workshops*. Springer, 2007, pp. 260–271.

[56] Mathieu Boussard, Stefan Meissner, Andreas Nettsträter, Alexis Olivereau, Alexander Salinas Segura, Matthias Thoma, and Joachim W Walewski, "A process for generating concrete architectures," in *Enabling Things to Talk*. Springer, 2013, pp. 45–111.

[57] Andrés Boza, MME Alemany, Llanos Cuenca, and Angel Ortiz, "Event Management for Sensing Enterprises with Decision Support Systems," *Annals of Data Science*, vol. 2, no. 1, pp. 103–109, 2015.

[58] Adam Brandenburger and Barry Nalebuff, *Co-opetition*. Broadway Business, 2011. ISBN 978-0385479493

[59] Torsten Braun, Thiemo Voigt, and Adam Dunkels, "Energy-efficient TCP operation in wireless sensor networks," *Praxis der Informationsverarbeitung und Kommunikation*, vol. 28, no. 2, pp. 93–100, 2005.

[60] Tim Bray, "RFC7159: The JavaScript Object Notation (JSON) Data Interchange Format," *Internet Engineering Task Force (IETF)*, 2014.

[61] Dan Brickley and R. V. Guha, Eds., *RDF Vocabulary Description Language 1.0: RDF Schema*, ser. W3C Recommendation. W3C, Feb. 2004, http://www.w3.org/TR/rdf-schema/.

[62] S Brown and CJ Sreenan, "Updating software in wireless sensor networks: A survey," *Dept. of Computer Science, National Univ. of Ireland, Maynooth, Tech. Rep*, 2006.

[63] Marcin Brzozowski, Hendrik Salomon, and Peter Langendoerfer, "On efficient clock drift prediction means and their applicability to IEEE 802.15.4," in *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*. IEEE, 2010, pp. 216–223.

[64] Thomas Buchholz, A Küpper, and Michael Schiffers, "Quality of context information: What it is and why we need it," in *Proceedings of the 10th HP–OVUA Workshop*, vol. 2003, 2003.

[65] Nicola Bui and Michele Zorzi, "Health care applications: a solution based on the internet of things," in *Proceedings of the 4th International Symposium on Applied Sciences in Biomedical and Communication Technologies*. ACM, 2011, p. 131.

[66] Eyuphan Bulut and Ibrahim Korpeoglu, "DSSP: a dynamic sleep scheduling protocol for prolonging the lifetime of wireless sensor networks," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 2. IEEE, 2007, pp. 725–730.

[67] Jean-Paul Calbimonte, Ho Young Jeung, Oscar Corcho, and Karl Aberer, "Enabling query technologies for the semantic sensor web," *International Journal on Semantic Web and Information Systems*, vol. 8, no. EPFL-ARTICLE-183971, pp. 43–63, 2012.

[68] Domenico Caputo, Luca Mainetti, Luigi Patrono, and Antonio Vilei, "Implementation of the EXI schema on wireless sensor nodes using Contiki," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 770–774.

[69] A Caracaş, Clemens Lombriser, Yvonne Anne Pignolet, Thorsten Kramp, Thomas Eirich, Rolf Adelsberger, and Urs Hunkeler, "Energy-efficiency through micro-managing communication and optimizing sleep," in *Sensor, Mesh and Ad Hoc Communications and Networks (SECON), 2011 8th Annual IEEE Communications Society Conference on*. IEEE, 2011, pp. 55–63.

[70] Alexandru Caracas, "From business process models to pervasive applications: Synchronization and optimization," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*. IEEE, 2012, pp. 320–325.

[71] Alexandru Caracaş and Alexander Bernauer, "Compiling business process models for sensor networks," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. IEEE, 2011, pp. 1–8.

[72] Alexandru Caracaş and Thorsten Kramp, "On the expressiveness of BPMN for modeling wireless sensor networks applications," in *Business Process Model and Notation*. Springer, 2011, pp. 16–30.

[73] Alexandru Caracas, Thorsten Kramp, Michael Baentsch, Marcus Oestreicher, Thomas Eirich, and Ivan Romanov, "Mote runner: A multi-language virtual machine for small embedded devices," in *Sensor Technologies and Applications, 2009. SENSORCOMM'09. Third International Conference on*. IEEE, 2009, pp. 117–125.

[74] Alexandru Mircea Caracaş, "Modeling, compiling, and efficiently executing business processes on resource-constrained wireless sensor networks," Ph.D. dissertation, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20220, 2012, 2012.

[75] Jorge Cardoso, "Modeling service relationships for service networks," in *Exploring Services Science*. Springer, 2013, pp. 114–128.

[76] Jorge Cardoso and Carlos Pedrinaci, "Evolution and Overview of Linked USDL," in *6th International Conference Exploring Services Science, IESS 2015, Porto, Portugal*. Lecture Notes in Business Information Processing, 2015.

[77] Jorge Cardoso, Carlos Pedrinaci, and Pieter De Leenheer, "Open semantic service networks: modeling and analysis," in *Exploring Services Science*. Springer, 2013, pp. 141–154.

[78] Angelo P Castellani, Mattia Gheda, Nicola Bui, Michele Rossi, and Michele Zorzi, "Web Services for the Internet of Things through CoAP and EXI," in *Communications Workshops (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[79] Angelo P Castellani, Akbar Rahman, Esko Dijk, Thomas Fossati, and Salvatore Loreto, "Best practices for HTTP-CoAP mapping implementation," 2013. [Online]. Available: http://tools.ietf.org/html/draft-castellani-core-http-mapping-07

[80] Angelo Paolo Castellani, Nicola Bui, Paolo Casari, Michele Rossi, Zach Shelby, and Michele Zorzi, "Architecture and protocols for the internet of things: A case study," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*. IEEE, 2010, pp. 678–683.

[81] Lei Chang, "A psychometric evaluation of 4-point and 6-point Likert-type scales in relation to reliability and validity," *Applied psychological measurement*, vol. 18, no. 3, pp. 205–215, 1994.

[82] David Chappell, "Introducing OData — Data Access for the web, the cloud, mobile devices, and more," Chappell and Associates, Tech. Rep., 2011.

[83] Dazhi Chen and Pramod K Varshney, "QoS Support in Wireless Sensor Networks: A Survey." in *International Conference on Wireless Networks*, vol. 13244, 2004, pp. 227–233.

[84] Nanxing Chen and César Viho, "Passive Interoperability Testing for Request-Response Protocols: Method, Tool and Application on CoAP Protocol," in *Testing Software and Systems*. Springer, 2012, pp. 87–102.

[85] Roberto Chinnici, Hugo Haas, Amelia A Lewis, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana, "Web services description language (WSDL) version 2.0 part 2: Adjuncts," *W3C Recommendation*, vol. 6, 2007.

[86] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana, "Web services description language (wsdl) version 2.0 part 1: Core language," *W3C recommendation*, vol. 26, p. 19, 2007.

[87] Michele Chinosi and Alberto Trombetta, "BPMN: An introduction to the standard," *Computer Standards & Interfaces*, vol. 34, no. 1, pp. 124–134, 2012.

[88] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana *et al.*, "Web services description language (WSDL) 1.1," 2001.

[89] Jason H Christensen, "Using RESTful web-services and cloud computing to create next generation mobile applications," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*. ACM, 2009, pp. 627–634.

[90] Charles Hutchison Clark, *Brainstorming, the Dynamic Way to Create Successful Ideas*. Doubleday, 1958.

[91] Mike Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004. ISBN 978-0321205681

[92] Mike Cokus and Santiago Pericas-Geertsen, "XML binary characterization properties," *W3C XBC Working Group Note*, 2005.

[93] Walter Colitti, Kris Steenhaut, and Niccolò De Caro, "Integrating wireless sensor networks with the web," *Extending the Internet to Low powerand Lossy Networks (IP+ SN 2011)*, 2011.

[94] Walter Colitti, Kris Steenhaut, Niccolò De Caro, Bogdan Buta, and Virgil Dobrota, "Evaluation of constrained application protocol for wireless sensor networks," in *Local & Metropolitan Area Networks (LANMAN), 2011 18th IEEE Workshop on*. IEEE, 2011, pp. 1–6.

[95] Oscar Corcho and Raúl García-Castro, "Five challenges for the semantic sensor web," *Semantic Web*, vol. 1, no. 1, 2, pp. 121–125, 2010.

[96] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 2. ISBN 978-0262531962

[97] Paolo Costa, Geoff Coulson, Cecillia Mascolo, Gian Pietro Picco, and Sivahuran Zachariadis, "The RUNES middleware: a reconfigurable component-based approach to networked embedded systems," in *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on* , vol. 2, sept. 2005. doi: 10.1109/PIMRC.2005.1651554 pp. 806 –810 Vol. 2.

[98] Douglas Crockford, "The application/json media type for javascript object notation (json)," 2006. [Online]. Available: http://tools.ietf.org/html/rfc4627

[99] Hui Dai and Richard Han, "TSync: a lightweight bidirectional time synchronization service for wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 8, no. 1, pp. 125–139, 2004.

[100] Nana Dankwa, "An evaluation of transmit power levels for node localization on the mica2 sensor node," *signal*, vol. 5, p. 03, 2004.

[101] Mohsen Darianian and Martin Peter Michael, "Smart home mobile RFID-based Internet-of-Things systems and services," in *Advanced Computer Theory and*

Engineering, 2008. ICACTE'08. International Conference on. IEEE, 2008, pp. 116–120.

[102] Ian Davis, Thomas Steiner, and Arnaud Hors, Eds., *RDF 1.1 JSON Alternate Serialization (RDF/JSON)*, ser. W3C Editor's Draft. W3C, Aug. 2013, https://dvcs.w3.org/hg/rdf/raw-file/default/rdf-json/index.html.

[103] Suparna De, Tarek Elsaleh, Payam Barnaghi, and Stefan Meissner, "An internet of things platform for real-world and digital objects," *Scalable Computing: Practice and Experience*, vol. 13, no. 1, 2012.

[104] Surpana De, Payam Barnaghi, Martin Bauer, and Stefan Meissner, "Service modelling for the Internet of Things," in *Computer Science and Information Systems (FedCSIS), 2011 Federated Conference on*, sept. 2011, pp. 949 –955.

[105] Mauricio De Simone and Rick Kazman, "Software architectural analysis: an experience report," in *Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research*. IBM Press, 1995, p. 18.

[106] Philippe Debaty, Patrick Goddi, and Alex Vorbar, "Integrating the physical world with the web to enable context-enhanced services," Mobile and Media Systems Laboratory, HP Laboratories, Technical Report, Sep. 2003.

[107] Francesco Di Cerbo, Michele Bezzi, Samuel Paul Kaluvuri, Antonino Sabetta, Slim Trabelsi, and Volkmar Lotz, "Towards a trustworthy service marketplace for the future internet," in *The Future Internet*. Springer, 2012, pp. 105–116.

[108] Patrick Dittmer, Marius Veigt, Bernd Scholz-Reiter, Nils Heidmann, and Steffen Paul, "The intelligent container as a part of the internet of things," in *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2012 IEEE International Conference on*. IEEE, 2012, pp. 209–214.

[109] Glen Dobson, Russell Lock, and Ian Sommerville, "QoSOnt: a QoS ontology for service-centric systems," in *Software Engineering and Advanced Applications, 2005. 31st EUROMICRO Conference on*. IEEE, 2005, pp. 80–87.

[110] Kirsten Dolfus and Torsten Braun, "An evaluation of compression schemes for wireless networks," in *Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT), 2010 International Congress on*. IEEE, 2010, pp. 1183–1188.

[111] John Domingue, Dumitru Roman, and Michael Stollberg, "Web service modeling ontology (WSMO)-An ontology for semantic web services," in *Position paper at the W3C Workshop on Frameworks for Semantics in Web Services*, 2005, pp. 9–10.

[112] Olivier Dousse, Petteri Mannersalo, and Patrick Thiran, "Latency of wireless sensor networks with uncoordinated power saving mechanisms," in *Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2004, pp. 109–120.

[113] Mathilde Durvy, Julien Abeillé, Patrick Wetterwald, Colin O'Flynn, Blake Leverett, Eric Gnoske, Michael Vidales, Geoff Mulligan, Nicolas Tsiftes, Niclas Finne *et al.*, "Making sensor networks IPv6 ready," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*. ACM, 2008, pp. 421–422.

[114] ECMA, *The JSON Data Interchange Format*. ECMA International, October 2013. [Online]. Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

[115] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, and Rukman Senanayake, "The OWL-S editor–a development tool for semantic web services," in *The Semantic Web: Research and Applications*. Springer, 2005, pp. 78–92.

[116] Jeffrey Erman, Vijay Gopalakrishnan, Rittwik Jana, and Kadangode K Ramakrishnan, "Towards a spdyier mobile web?" *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 2010–2023, 2015.

[117] ETH Zurich, "BTnode Rev3 Sensor Guide." [Online]. Available: btnode.ethz.ch

[118] Franz Färber, Sang Kyun Cha, Jürgen Primsch, Christof Bornhövd, Stefan Sigg, and Wolfgang Lehner, "SAP HANA database: data management for modern business applications," *ACM Sigmod Record*, vol. 40, no. 4, pp. 45–51, 2012.

[119] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees, "The SAP HANA Database–An Architecture Overview." *IEEE Data Eng. Bull.*, vol. 35, no. 1, pp. 28–33, 2012.

[120] J Farrell and H Lausen, "Semantic Annotations for WSDL and XML Schema (SAWSDL), W3C Recommendation (2007)."

[121] Laura Marie Feeney, Christian Rohner, Per Gunningberg, Anders Lindgren, and Lars Andersson, "How do the dynamics of battery discharge affect sensor lifetime?" in *Wireless On-demand Network Systems and Services (WONS), 2014 11th Annual Conference on*. IEEE, 2014, pp. 49–56.

[122] Christina Feier, Axel Polleres, Roman Dumitru, John Domingue, Michael Stollberg, and Dieter Fensel, "Towards intelligent web services: The web service modeling ontology (WSMO)," *2005 International Conference on Intelligent Computing (ICIC05), 23-26 Aug 2005, Hefei, Chin*, 2005.

[123] Youyi Feng and Baichun Xiao, "Integration of pricing and capacity allocation for perishable products," *European Journal of Operational Research*, vol. 168, no. 1, pp. 17–34, 2006.

[124] Javier D Fernández, Miguel A Martínez-Prieto, Claudio Gutiérrez, Axel Polleres, and Mario Arias, "Binary RDF representation for publication and exchange (HDT)," *Web Semantics: Science, Services and Agents on the World Wide Web* , vol. 19, pp. 22–41, 2013.

[125] Roberta Ferrario and Nicola Guarino, "Towards an Ontological Foundation for Services Science," in *Future Internet FIS 2008*, ser. Lecture Notes in Computer Science, John Domingue, Dieter Fensel, and Paolo Traverso, Eds. Springer Berlin / Heidelberg, 2009, vol. 5468, pp. 152–169.

[126] Nicolas Ferry, Sylvain Ducloyer, Nathalie Julien, and Dominique Jutel, "Power/energy estimator for designing WSN nodes with ambient energy harvesting feature," *EURASIP Journal on Embedded Systems*, vol. 2011, p. 6, 2011.

[127] Roy T Fielding and Richard N Taylor, "Principled design of the modern Web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

[128] Roy Thomas Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[129] Klaus Finkenzeller, *RFID-Handbuch*. Carl Hanser Verlag GmbH & Co. KG, 2002, vol. 4. ISBN 978-3446429925

[130] Florian Fischer, Barry Norton, and Patrick Un, "D3. 4.6 MicroWSMO v2– Defining the second version of MicroWSMO as a systematic approach for rich tagging," *Soa4all project deliverable*, 2010.

[131] Marios Fokaefs and Eleni Stroulia, "Using WADL Specifications to Develop and Maintain REST Client Applications," in *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 81–88.

[132] Paul Fremantle, Sanjiva Weerawarana, and Rania Khalaf, "Enterprise services," *Communications of the ACM*, vol. 45, no. 10, pp. 77–82, 2002.

[133] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[134] Frieder Ganz, Payam Barnaghi, and Francois Carrez, "Automated semantic knowledge acquisition from sensor data," 2014.

[135] Jose Maria Garcia, Carlos Pedrinaci, Manuel Resinas, Jorge Cardoso, Pablo Fernández, and Antonio Ruiz-Cortés, "Linked USDL agreement: effectively

sharing semantic service level agreements on the Web," in *Web Services (ICWS), 2015 IEEE International Conference on*. IEEE, 2015, pp. 137–144.

[136] Jan D. Gehrke, Christian Behrens, Reiner Jedermann, and Ernesto Morales, "The Intelligent Container- Toward Autonomous Logistic Processes," 2006.

[137] James M Gilbert and Farooq Balouchi, "Comparison of energy harvesting systems for wireless sensor networks," *International Journal of Automation and Computing*, vol. 5, no. 4, pp. 334–347, 2008.

[138] Nils Glombitza, Martin Lipphardt, Christian Werner, and Stefan Fischer, "Using graphical process modeling for realizing SOA programming paradigms in sensor networks," in *Wireless On-Demand Network Systems and Services, 2009. WONS 2009. Sixth International Conference on*. IEEE, 2009, pp. 61–70.

[139] Nils Glombitza, Richard Mietz, Kay Romer, Stefan Fischer, and Dennis Pfisterer, "Self-description and protocol conversion for a web of things," in *Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 229–236.

[140] Nils Glombitza, Dennis Pfisterer, and Stefan Fischer, "Integrating wireless sensor networks into web service-based business processes," in *Proceedings of the 4th international Workshop on Middleware Tools, Services and Run-Time Support For Sensor Networks*, ser. MidSens '09. ACM, 2009. ISBN 978-1-60558-851-3 pp. 25–30.

[141] Nils Glombitza, Dennis Pfisterer, and Stefan Fischer, "Ltp: An efficient web service transport protocol for resource constrained devices," in *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*. IEEE, 2010, pp. 1–9.

[142] Nils Glombitza, Dennis Pfisterer, and Stefan Fischer, "Using state machines for a model driven development of web service-based sensor network applications," in *Proceedings of the 2010 ICSE Workshop on Software Engineering for Sensor Network Applications*. ACM, 2010, pp. 2–7.

[143] Jorge Granjal, J Sa Silva, Edmundo Monteiro, and F Boavida, "Why is IPSec a viable option for wireless sensor networks," in *Mobile Ad Hoc and Sensor Systems, 2008. MASS 2008. 5th IEEE International Conference on*. IEEE, 2008, pp. 802–807.

[144] Joe Gregorio, Roy Fielding, Marc Hadley, Mark Nottingham, and David Orchard, "RFC6570: URI Template," *Internet Engineering Task Force (IETF) Request for Comments*, 2012.

[145] Adrian Groza and Ioan Alfred Letia, "Plausible description logic programs for stream reasoning," *Future Internet*, vol. 4, no. 4, pp. 865–881, 2012.

[146] Tao Gu, Xiao Hang Wang, Hung Keng Pung, and Da Qing Zhang, "An ontology-based context model in intelligent environments," in *Proceedings of communication networks and distributed systems modeling and simulation conference*, vol. 2004, 2004, pp. 270–275.

[147] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon, "SOAP Version 1.2," *W3C recommendation*, vol. 24, 2003.

[148] Dominique Guinard, "A web of things application architecture-Integrating the real-world into the web," Ph.D. dissertation, ETH Zurich, 2011.

[149] Dominique Guinard, Iulia Ion, and Simon Mayer, "In search of an internet of things service architecture: REST or WS-*? A developers perspective," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 2012, pp. 326–337.

[150] Dominique Guinard and Vlad Trifa, "Towards the web of things: Web mashups for embedded devices," in *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, 2009, p. 15.

[151] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio, "Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services," *Services Computing, IEEE Transactions on*, vol. 3, no. 3, pp. 223–235, 2010.

[152] Dominique Guinard, Vlad Trifa, Staminis Karnouskos, Patrick Spiess, and Domnic Savio, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," *Services Computing, IEEE Transactions on*, vol. 3, no. 3, pp. 223 –235, july-sept. 2010.

[153] Dominique Guinard, Vlad Trifa, and Erik Wilde, "A resource oriented architecture for the web of things," *Proc. of IoT*, 2010.

[154] Marc J. Hadley, "Web Application Description Language (WADL)," Sun Microsystems, Inc., Mountain View, CA, USA, Tech. Rep., 2006.

[155] Stephan Haller, "The things in the internet of things," *Poster at the (IoT 2010). Tokyo, Japan, November*, vol. 5, p. 26, 2010.

[156] Stephan Haller, Martin Bauer, Franois Carrez, Richard Egan, Levent Gürgen, Jan Höller, Jiménez Holgado, Jose Antonio, Bernard Hunt, and Gunter Woysch, "First reference model white paper," IoT-I, White Paper, Sep. 2011.

[157] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth, "The Internet of Things in an Enterprise Context," in *Future Internet FIS 2008*, ser. Lecture

Notes in Computer Science, John Domingue, Dieter Fensel, and Paolo Traverso, Eds. Springer Berlin / Heidelberg, 2009, vol. 5468, pp. 14–28.

[158] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth, *The internet of things in an enterprise context*. Springer, 2009.

[159] Stephan Haller and Carsten Magerkurth, "The real-time enterprise: Iot-enabled business processes," in *IETF IAB Workshop on Interconnecting Smart Objects with the Internet*. Citeseer, 2011.

[160] Stephan Haller, Alexandru Serbanati, Martin Bauer, and Francois Carrez, "A domain model for the internet of things," in *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*. IEEE, 2013, pp. 411–417.

[161] Ivaylo Haratcherev, Gertjan Halkes, Tom Parker, Otto Visser, and Koen Langendoen, "PowerBench: A scalable testbed infrastructure for benchmarking power consumption," in *Int. Workshop on Sensor Network Engineering (IWSNE)*, 2008, pp. 37–44.

[162] Klaus Hartke, "Observing Resources in CoAP," 2014. [Online]. Available: hhttps://datatracker.ietf.org/doc/draft-ietf-core-observe/

[163] Michael Hausenblas, Michael Kerrin, Michael Pizzo, Evelyne Viegas, and Neil Wilson, "Linking Structured Data — Using OData to access an RDF triple store through a semi-structured conceptual mode," Microsoft Research, Tech. Rep. MSR-TR-2013-57, May 2013. [Online]. Available: http://research.microsoft.com/apps/pubs/default.aspx?id=193076

[164] Tian He, Sudha Krishnamurthy, John A Stankovic, Tarek Abdelzaher, Liqian Luo, Radu Stoleru, Ting Yan, Lin Gu, Jonathan Hui, and Bruce Krogh, "Energy-efficient surveillance system using wireless sensor networks," in*Proceedings of the 2nd international conference on Mobile systems, applications, and services.* ACM, 2004, pp. 270–283.

[165] R. Heras and Santos N., "Concepts and Solutions for Identification and Lookup of IoT Resources," 2012.

[166] Jason Hill, Mike Horton, Ralph Kling, and Lakshman Krishnamurthy, "The platforms enabling wireless sensor networks," *Communications of the ACM*, vol. 47, no. 6, pp. 41–46, 2004.

[167] Jason L Hill and David E Culler, "Mica: A wireless platform for deeply embedded networks," *Micro, IEEE*, vol. 22, no. 6, pp. 12–24, 2002.

[168] Pascal Hitzler, Markus Krtzsch, and Sebastian Rudolph, *Foundations of Semantic Web Technologies*, 1st ed. Chapman and Hall/CRC, 8 2009. ISBN 978-1420090505

[169] Hanh H Hoang, Jason J Jung, and Chi P Tran, "Ontology-based approaches for cross-enterprise collaboration: a literature review on semantic business process management," *Enterprise Information Systems*, vol. 8, no. 6, pp. 648–664, 2014.

[170] Richard J Hodges, Jean C Buzby, and Ben Bennett, "Postharvest losses and waste in developed and less developed countries: opportunities to improve resource use," *The Journal of Agricultural Science*, vol. 149, no. S1, pp. 37–45, 2011.

[171] Ralph Hodgson and Paul J Keller, "Qudt-quantities, units, dimensions and data types in owl and xml," *Online (September 2011) http://www. qudt. org*, 2011.

[172] YongGeun Hongm, JooSang Young, and MyungKi Shin, "Analysis of the communications of sleepy nodes in constrained node networks," in *2014 The International Industrial Information Systems Conference*, 2014, pp. 121–124.

[173] Vincent Huang, Vlasios Tsiatsis, Martin Bauer, Martin Strohbach, Claudia Villalonga, Stephan Haller, Frdric Montagut, Jesus Bernat Vercher, Bernd Tessendorf, and Stefan Meissner, "State of the art - sensor information services," Sensei Project, Report, Jan. 2008.

[174] Yi Huang and Dennis Gannon, "A comparative study of web services-based event notification specifications," in *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*. IEEE, 2006, pp. 8–pp.

[175] David A Huffman *et al.*, "A method for the construction of minimum redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[176] Jonathan Hui and Pascal Thubert, "Compression format for IPv6 datagrams over IEEE 802.15. 4-based networks," *RFC4944*, 2011. [Online]. Available: https://tools.ietf.org/html/rfc4944

[177] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark, "MQTT-SA publish/subscribe protocol for Wireless Sensor Networks," in *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*. IEEE, 2008, pp. 791–798.

[178] Philipp Hurni and Torsten Braun, "Energy-efficient multi-path routing in wireless sensor networks," in *Ad-hoc, Mobile and Wireless Networks*. Springer, 2008, pp. 72–85.

[179] Philipp Hurni, Benjamin Nyffenegger, Torsten Braun, and Anton Hergenroeder, "On the accuracy of software-based energy estimation techniques," in *Wireless Sensor Networks*. Springer, 2011, pp. 49–64.

[180] IBM Corporation, "Mote Runner MRv6 Protocol," available as part of the Mote Runner distribution, 2014.

[181] IBM Corporation, "Mote Runner Online Documentation," available as part of the Mote Runner v14 distribution, 2014.

[182] IBM Corporation, "IBM Research: IBM Mote Runner Homepage," 2015. [Online]. Available: http://www.zurich.ibm.com/moterunner/

[183] IEEE, "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2003*, pp. 1–320, Sept 2006.

[184] IEEE, "IEEE Standard for Information technology– Local and metropolitan area networks– Specific requirements– Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low Rate Wireless Personal Area Networks (WPANs)," *IEEE Std 802.15.4-2006 (Revision of IEEE Std 802.15.4-2003)*, pp. 1–320, Sept 2006.

[185] IEEE, "IEEE Standard for Information technology," *IEEE Std 802.15.4a*, Aug 2007.

[186] Tobias Jacobs, Markus Joos, Carsten Magerkurth, Stefan Meissner, Sonja Meyer, Klaus Sperner, Matthias Thoma, and Gerd Voelkensen, "Adaptive, fault-tolerant orchestration of distributed IoT service interactions," IOTA-A, Tech. Rep., 2012.

[187] Tobias Jacobs, Markus Joos, Carsten Magerkurth, Stefan Meissner, Sonja Meyer, Klaus Sperner, Matthias Thoma, and Gerd Voelksen, "Internet-of-things architecture iot-a project deliverable d2. 5-adaptive, fault-tolerant orchestration of distributed iot service interactions," *IoT-A Project Deliverable*, vol. 5, pp. 06–11, 2012.

[188] Antonio J Jara, Miguel A Zamora, and Antonio F Skarmeta, "An internet of things—based personal device for diabetes therapy management in ambient assisted living (AAL)," *Personal and Ubiquitous Computing*, vol. 15, no. 4, pp. 431–440, 2011.

[189] Xiaolin Jia, Quanyuan Feng, Taihua Fan, and Quanshui Lei, "RFID technology and its applications in Internet of Things (IoT)," in*Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*. IEEE, 2012, pp. 1282–1285.

[190] Steve Jones, "Towards an acceptable definition of service," *IEEE Software*, 2005.

[191] T Kamiya and J Schneider, "Efficient XML Interchange (EXI) Format 1.0," *World Wide Web Consortium Recommendation REC-exi-20110310*, 2011.

[192] Eirini Karapistoli, F-N Pavlidou, Ioannis Gragopoulos, and Ioannis Tsetsinas, "An overview of the IEEE 802.15. 4a standard," *Communications Magazine, IEEE*, vol. 48, no. 1, pp. 47–53, 2010.

[193] Stamatis Karnouskos, "The cooperative internet of things enabled smart grid," in *Proceedings of the 14th IEEE international symposium on consumer electronics (ISCE2010), June*, 2010, pp. 07–10.

[194] Staminis Karnouskos, Domnic Savio, Patrick Spiess, Dominique Guinard, Vlad Trifa, and Oliver Baecker, "Real-world Service Interaction with Enterprise Systems in Dynamic Manufacturing Environments," in *Artificial Intelligence Techniques for Networked Manufacturing Enterprises Management*. Springer, 2010.

[195] Stephan Karpischek, Florian Michahelles, Florian Resatsch, and Elgar Fleisch, "Mobile sales assistant-an nfc-based product information system for retailers," in *Near Field Communication, 2009. NFC'09. First International Workshop on.* IEEE, 2009, pp. 20–23.

[196] Kay Kadner, "W3C Unified Service Description Language Incubator Group," 2010. [Online]. Available: http://www.w3.org/2005/Incubator/usdl/

[197] Rick Kazman, Gregory Abowd, Len Bass, and Paul Clements, "Scenario-based analysis of software architecture," *Software, IEEE*, vol. 13, no. 6, pp. 47–55, 1996.

[198] Rick Kazman, Len Bass, Mike Webb, and Gregory Abowd, "SAAM: A method for analyzing the properties of software architectures," in *Proceedings of the 16th international conference on Software engineering*. IEEE Computer Society Press, 1994, pp. 81–90.

[199] Rick Kazman, Mark Klein, and Paul Clements, "Evaluating Software Architectures-Methods and Case Studies," 2001.

[200] Simon Kellner, Mario Pink, Detlev Meier, and E-O Blass, "Towards a realistic energy model for wireless sensor networks," in *Wireless on Demand Network Systems and Services, 2008. WONS 2008. Fifth Annual Conference on*. IEEE, 2008, pp. 97–100.

[201] Yunseop Kim, Robert G Evans, and William M Iversen, "Remote sensing and control of an irrigation system using a distributed wireless sensor network," *Instrumentation and Measurement, IEEE Transactions on*, vol. 57, no. 7, pp. 1379–1387, 2008.

[202] Marc Kirchhoff, "Eine SPARQL-Schnittstelle fr OData-Services," Ph.D. dissertation, University of Kassel, Electrical Engineering and Computer Science, 2014.

[203] Marc Kirchhoff and Kurt Geihs, "Semantic description of OData services," in *Proceedings of the Fifth Workshop on Semantic Web Information Management* . ACM, 2013, p. 2.

[204] Marc Kirchhoff and Kurt Geihs, "Integrating OData services into the semantic web: a SPARQL interface for OData," in *Proceedings of the 14th International Conference on Knowledge Technologies and Data-driven Business*. ACM, 2014, p. 2.

[205] Scott Klein and Herve Roggero, "OData with SQL Azure," in *Pro Sql Azure*. Springer, 2010, pp. 147–167.

[206] Ralph Kling, Robert Adler, Jonathan Huang, Vincent Hummel, and Lama Nachman, "Intel mote: Using Bluetooth in sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems* . ACM, 2004, pp. 318–318.

[207] Donald E Knuth, "Dynamic huffman coding," *Journal of algorithms*, vol. 6, no. 2, pp. 163–180, 1985.

[208] Michael Koch and Alexander Richter, *Enterprise 2.0*. Oldenbourg Verlag, 2009.

[209] J Kopecky, T Vitvar, and K Gomadam, "MicroWSMO," *Deliverable, Conceptual Models for Services Working Group, URL: http://cmswg. sti2. org/TR/d12/v0*, vol. 1, no. 20090310, p. d12v01_20090310, 2008.

[210] Jacek Kopeckỳ, Tomas Vitvar, Dieter Fensel, and Karthik Gomadam, "hRESTS & MicroWSMO," *CMS WG Working Draft*, 2009.

[211] Jonathon Kopecky, Karthik Gomadam, and Tomas Vitvar, "hrests: An html microformat for describing restful web services," in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, vol. 1. IEEE, 2008, pp. 619–625.

[212] Jonathon Kopecky, Tomas Vitvar, Carine Bournez, and Joel Farrell, "Sawsdl: Semantic annotations for wsdl and xml schema," *Internet Computing, IEEE*, vol. 11, no. 6, pp. 60–67, 2007.

[213] Hans-Jörg Körber, Housam Wattar, and Gerd Scholl, "Modular wireless real-time sensor/actuator network for factory automation applications," *Industrial Informatics, IEEE Transactions on*, vol. 3, no. 2, pp. 111–119, 2007.

[214] Gerd Kortuem, Fahim Kawsar, Daniel Fitton, and Vasughi Sundramoorthy, "Smart objects as building blocks for the internet of things," *Internet Computing, IEEE*, vol. 14, no. 1, pp. 44–51, 2010.

[215] Joel Koshy and Raju Pandey, "Remote incremental linking for energy-efficient reprogramming of sensor networks," in *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*. IEEE, 2005, pp. 354–365.

[216] Matthias Kovatsch, "Demo abstract: Human-coap interaction with copper," in *Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011 International Conference on*. IEEE, 2011, pp. 1–2.

[217] Matthias Kovatsch, Yassin N. Hassan, and Simon Mayer, "Practical semantics for the internet of things: Physical states, device mashups, and open questions," in *Proceedings of the 5th International Conference on the Internet of Things (IoT 2015)*, Seoul, South Korea, Oct. 2015.

[218] Matthias Kovatsch, Martin Lanter, and Zach Shelby, "Californium: Scalable cloud services for the internet of things with coap," in *Proceedings of the 4th International Conference on the Internet of Things (IoT 2014)*, 2014.

[219] Jayaram Krishnaswamy, *Microsoft SQL Azure Enterprise Application Development*. Packt Publishing Ltd, 2010. ISBN 978-1849680806

[220] Mauri Kuorilehto, Marko Hä, Timo D Hä *et al.*, "A survey of application distribution in wireless sensor networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2005, no. 5, pp. 774–788, 1900.

[221] Nandakishore Kushalnagar, Gabriel Montenegro, C Schumacher *et al.*, "IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals," *RFC4919, August*, vol. 10, 2007. [Online]. Available: https://tools.ietf.org/html/rfc4919

[222] Walter Lang, Reiner Jedermann, Damian Mrugala, Amir Jabbari, Bernd Krieg-Bruckner, and Kerstin Schill, "The intelligent containera cognitive sensor network for transport management," *Sensors Journal, IEEE*, vol. 11, no. 3, pp. 688–698, 2011.

[223] Markus Lanthaler, "Leveraging Linked Data to Build Hypermedia-Driven Web APIs," in *REST: Advanced Research Topics and Practical Applications*. Springer, 2014, pp. 107–123.

[224] Markus Lanthaler, "Third Generation Web APIs," Ph.D. dissertation, Institute of Information Systems and Computer Media, Graz University of Technology, Austria, 2014.

[225] Markus Lanthaler and Christian Gütl, "A semantic description language for RESTful data services to combat Semaphobia," in *Digital Ecosystems and Technologies Conference (DEST), 2011 Proceedings of the 5th IEEE International Conference on*. IEEE, 2011, pp. 47–53.

[226] Markus Lanthaler and Christian Gütl, "On using JSON-LD to create evolvable RESTful services," in *Proceedings of the Third International Workshop on RESTful Design*. ACM, 2012, pp. 25–32.

[227] Markus Lanthaler and Christian Gütl, "Seamless integration of RESTful services into the web of data," *Advances in Multimedia*, vol. 2012, p. 1, 2012.

[228] Kenneth J Laskey and Kathryn B Laskey, "Uncertainty Reasoning for the World Wide Web: Report on the URW3-XG Incubator Group." in *URSW*. Citeseer, 2008.

[229] Jon Lathem, Karthik Gomadam, and Amit P Sheth, "Sa-rest and (s) mashups: Adding semantics to restful services," in *Semantic Computing, 2007. ICSC 2007. International Conference on*. IEEE, 2007, pp. 469–476.

[230] Edward A. Lee, "Cyber Physical Systems: Design Challenges," *Object-Oriented Real-Time Distributed Computing, IEEE International Symposium on*, vol. 0, pp. 363–369, 2008.

[231] Torsten Leidig and Carlos Pedrinaci, "Linked usdl," 2012. [Online]. Available: http://www.linked-usdl.org

[232] Christian Lerche, Klaus Hartke, and Matthias Kovatsch, "Industry adoption of the internet of things: a constrained application protocol survey," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, 2012, pp. 1–6.

[233] Libelium Comunicaciones Distribuidas S.L., "WASMOTE Datasheet, v11," 2014. [Online]. Available: http://www.libelium.com/v11-files/documentation/waspmote/waspmote/datasheet_eng.pdf

[234] Libelium Comunicaciones Distribuidas S.L., "WASMOTE Overview, Hardware Datasheet," 2014. [Online]. Available: http://www.libelium.com/v12-files/documentation/waspmote/waspmote/datasheet_eng.pdf

[235] Libelium Comunicaciones Distribuidas S.L., "WASPMOTE Technical Guide - Document version: v5.6," 2014. [Online]. Available: http://www.libelium.com/downloads/documentation/waspmote_technical_guide.pdf

[236] Rensis Likert, "A technique for the measurement of attitudes." *Archives of psychology*, 1932.

[237] Dirk Lill and Axel Sikora, "Wireless technologies for safe automation-insights in protocol development," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. IEEE, 2008, pp. 1181–1184.

[238] Mikael Lindvall, Roseanne Tesoriero Tvedt, and Patricia Costa, "An empirically-based process for software architecture evaluation," *Empirical Software Engineering*, vol. 8, no. 1, pp. 83–108, 2003.

[239] Seng W Loke, "Facing uncertainty and consequence in context-aware systems: towards an argumentation approach," *Proceedings of the Workshop on Advanced Context Modelling, Reasoning and, Management*, 2004.

[240] Alessandro Ludovici, Piergiuseppe Di Marco, Anna Calveras, and Karl H Johansson, "Analytical model of large data transactions in coap networks," *Sensors*, vol. 14, no. 8, pp. 15 610–15 638, 2014.

[241] Michael Lütjen, Patrick Dittmer, and Marius Veigt, "Quality driven distribution of intelligent containers in cold chain logistics networks," *Production Engineering*, vol. 7, no. 2-3, pp. 291–297, 2013.

[242] Carsten Magerkurth, Stephan Haller, and Pascal Hagedorn, "Intersecting the architecture of the internet of things with the future retail industry," in *International Joint Conference on Ambient Intelligence*. Springer, 2010, pp. 315–319.

[243] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson, "Wireless sensor networks for habitat monitoring," in *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. ACM, 2002, pp. 88–97.

[244] Lawrence Mandel, "Describe REST Web services with WSDL 2.0," *Rational Software Developer, IBM*, 2008.

[245] J Maranzano *et al.*, "Best Current Practices: Software Architecture Validation," 1993.

[246] Francesco Marcelloni and Massimo Vecchio, "A simple algorithm for data compression in wireless sensor networks," *Communications Letters, IEEE*, vol. 12, no. 6, pp. 411–413, 2008.

[247] Mike Marin, Richard Hull, and Roman Vaculín, "Data centric bpm and the emerging case management standard: A short survey," in*International Conference on Business Process Management*. Springer, 2012, pp. 24–30.

[248] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi, "The flooding time synchronization protocol," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 39–49.

[249] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne *et al.*, "OWL-S: Semantic markup for web services," *W3C member submission*, vol. 22, pp. 2007–04, 2004.

[250] David Martin, Mark Burstein, Drew Mcdermott, Sheila Mcilraith, Massimo Paolucci, Katia Sycara, Deborah L Mcguinness, Evren Sirin, and Naveen Srinivasan, "Bringing semantics to web services with OWL-S," *World Wide Web*, vol. 10, no. 3, pp. 243–277, 2007.

[251] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki *et al.*, "Bringing semantics to web services: The OWL-S approach," in *Semantic Web Services and Web Process Composition*. Springer, 2005, pp. 26–42.

[252] David Martin, Massimo Paolucci, and Matthias Wagner, "Towards semantic annotations of web services: Owl-s from the sawsdl perspective," in *Proceedings of the OWL-S Experiences and Future Developments Workshop at ESWC*. Citeseer, 2007.

[253] Michael S Matell and Jacob Jacoby, "Is There an Optimal Number of Alternatives for Likert Scale Items? Study," *Educational and psychological measurement*, vol. 31, pp. 657–674, 1971.

[254] Frank Maurer and Sebastien Martel, "Extreme programming: Rapid development for Web-based applications," *IEEE Internet computing*, no. 1, pp. 86–90, 2002.

[255] Maxim Integrated, "DS3231 Datasheet - Extremely Accurate I2C-Integrated RTC/TCXO/Crystal, 19-5170; Rev 9; 1/13," 2013. [Online]. Available: http://datasheets.maximintegrated.com/en/ds/DS3231.pdf

[256] Duncan McFarlane, Vaggelis Giannikas, Alex CY Wong, and Mark Harrison, "Product intelligence in industrial control: Theory and practice," *Annual Reviews in Control*, vol. 37, no. 1, pp. 69–88, 2013.

[257] Deborah McGuinness and Frank van Harmelen, Eds., *OWL Web Ontology Language — Overview*, ser. W3C Recommendation. W3C, Feb. 2004, http://www.w3.org/TR/owl-features/.

[258] James D McLurkin, "Algorithms for distributed sensor networks," Ph.D. dissertation, Department of Electrical Engineering and Computer Sciences, University of California, 1999.

[259] Stefan Meissner, Dan Dobre, Matthias Thoma, and Gregorio Martin, "Internet of things architecture iot-a project deliverable d2. 1–resource description specification," *IoT-A Project Deliverable*, 2012.

[260] MEMSIC, "MICAz wireless measurement system - 6020-0065-05 Rev A," 2004. [Online]. Available: hhttp://www.memsic.com/userfiles/files/DataSheets/WSN/micaz_datasheet-t.pdf

[261] MEMSIC, "IRIS WIRELESS MEASUREMENT SYSTEM Datasheet 6020-0124-02 Rev A," 2009. [Online]. Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf

[262] MEMSIC Inc., "MICAz Wireless Measurement Syste Datasheet 6020-0065-05 Rev A." [Online]. Available: http://www.memsic.com/userfiles/files/DataSheets/WSN/micaz_datasheet-t.pdf

[263] MEMSIC Inc., "TelosB - TelosB Mote Platform Datasheet 6020-0094-03 Rev A." [Online]. Available: http://www.memsic.com/userfiles/files/Datasheets/WSN/telosb_datasheet.pdf

[264] S. Meyer, K. Sperner, C. Magerkurth, and J. Pasquier, "Towards modeling real-world aware business processes," in *Second International Workshop on Web of Things*. ACM, 2011, p. 8.

[265] Sonja Meyer, "Modeling Real-World Business Processes — Design and Implementation of a Business Process Meta-Model for the Internet of Things," Ph.D. dissertation, Faculty of Science of the University of Fribourg, 2014.

[266] Sonja Meyer, Andreas Ruppen, and Lorenz Hilty, "The Things of the Internet of Things in BPMN," in *Advanced Information Systems Engineering Workshops*. Springer, 2015, pp. 285–297.

[267] Sonja Meyer, Andreas Ruppen, and Carsten Magerkurth, "Internet of things-aware process modeling: integrating IoT devices as business process resources," in *Advanced Information Systems Engineering*. Springer, 2013, pp. 84–98.

[268] Sonja Meyer, Klaus Sperner, Carsten Magerkurth, and Jacques Pasquier, "Towards modeling real-world aware business processes," in *Proceedings of the Second International Workshop on Web of Things*. ACM, 2011, p. 8.

[269] *Open Data Protocol (OData), Version 2.0, Core Protocol*, Microsoft Corporation, 2011.

[270] *Open Data Protocol (OData) Version 3.0 — Atom Format*, Microsoft Corporation, 2013.

[271] *Open Data Protocol (OData) Version 3.0 — Common Schema Definition Language (CSDL)*, Microsoft Corporation, 2013.

[272] *Open Data Protocol (OData) Version 3.0 — JSON Verbose Format*, Microsoft Corporation, 2013.

[273] *Open Data Protocol (OData) Version 3.0 — URL Conventions*, Microsoft Corporation, 2013.

[274] *Open Data Protocol (OData), Version 3.0, Core Protocol*, Microsoft Corporation, 2013.

[275] Nilo Mitra, Yves Lafon *et al.*, "Soap version 1.2 part 0: Primer," *W3C recommendation*, vol. 24, p. 12, 2003.

[276] Gordon E Moore *et al.*, "Cramming more components onto integrated circuits," 1965.

[277] Lama Nachman, Jonathan Huang, Junaith Shahabdeen, Robert Adler, and Ralph Kling, "Imote2: Serious computation at the edge," in *Wireless Communications and Mobile Computing Conference, 2008. IWCMC'08. International*. IEEE, 2008, pp. 1118–1123.

[278] Mark Nelson and Jean-Loup Gailly, *The data compression book*. M&T Books New York, 1996, vol. 2. ISBN 978-1558514348

[279] Aaron Newman and Jeremy Thomas, *Enterprise 2.0 implementation: integrate Web 2.0 services into your enterprise*. McGraw Hill Professional, 2008.

[280] Mark Nottingham, "Rfc5988: Web linking," *Internet Engineering Task Force (IETF) Request for Comments*, 2010.

[281] Mark Nottingham, "Web linking," 2010. [Online]. Available: http://tools.ietf.org/html/rfc5988?chocaid=397

[282] Mark Nottingham and Robert Sayre, "The atom syndication format," 2005. [Online]. Available: http://tools.ietf.org/html/rfc4287

[283] Natalya Noy and Deborah McGuinness, "Ontology development 101: A guide to creating your first ontology," Stanford University, Knowledge Systems Laboratory, Tech. Rep., 2001, kSL-01-05.

[284] OASIS, "OASIS Project," 2008. [Online]. Available: http://www.oasis-project.eu/

[285] WS-BPEL OASIS, "Web Services Business Process Execution Language version 2.0, 2007."

[286] *Open Data Protocol (OData) Version 4.0 — Common Schema Definition Language (CSDL)*, OASIS Standard, 2014.

[287] *Open Data Protocol (OData), Version 4.0, Core Protocol*, OASIS Standard, 2014.

[288] Object Management Group, "Business Process Model Notation (BPMN) Version 2.0," *Object Management Group specification*, 2011.

[289] Benedikt Ostermaier, Matthias Kovatsch, and Silvia Santini, "Connecting things to the web using programmable low-power wifi modules," in *Proceedings of the Second International Workshop on Web of Things*. ACM, 2011, p. 2.

[290] Marko Paavola and Kauko Leiviska, *Wireless sensor networks in industrial automation*. INTECH Open Access Publisher, 2010.

[291] Krishna V Palem, Lakshmi NB Chakrapani, Zvi M Kedem, Avinash Lingamneni, and Kirthi Krishna Muntimadugu, "Sustaining moore's law in embedded computing through probabilistic and approximate design: retrospects and prospects," in *Proceedings of the 2009 international conference on Compilers, architecture, and synthesis for embedded systems*. ACM, 2009, pp. 1–10.

[292] C Ravindranath Pandian and Murali Kumar, *Simple Statistical Methods for Software Engineering: Data and Patterns*. CRC Press, 2015.

[293] David L Parnas, "On the design and development of program families," *Software Engineering, IEEE Transactions on*, no. 1, pp. 1–9, 1976.

[294] David Lorge Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[295] Cesare Pautasso, Olaf Zimmermann, and Frank Leymann, "Restful web services vs. big'web services: making the right architectural decision," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 805–814.

[296] Carlos Pedrinaci, Jorge Cardoso, and Torsten Leidig, "Linked USDL: a vocabulary for web-scale service trading," in *The Semantic Web: Trends and Challenges*. Springer, 2014, pp. 68–82.

[297] Carlos Pedrinaci and John Domingue, "Toward the next wave of services: Linked services for the web of data." *Journal of Universal Computer Science*, vol. 16, no. 13, pp. 1694–1719, 2010.

[298] Carlos Pedrinaci, Dong Liu, Maria Maleshkova, David Lambert, Jacek Kopecky, and John Domingue, "iServe: a linked services publishing platform," in *CEUR workshop proceedings*, vol. 596, 2010.

[299] Carlos Pedrinaci, Dong Liu, Maria Maleshkova, David Lambert, Jacek Kopecky, and John Domingue, "iServe: a Linked Services Publishing Platform," in *Workshop on Ontology Repositories and Editors for the Semantic Web*, ser. ORES 2010, 2010. ISSN 1613-0073

[300] Padmanabhan Pillai and Kang G Shin, "Real-time dynamic voltage scaling for low-power embedded operating systems," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001, pp. 89–102.

[301] SV Pokraev, MU Reichert, Maarten WA Steen, and Roel J Wieringa, "Semantic and pragmatic interoperability: a model for understanding," 2005.

[302] Joseph Polastre, Jason Hill, and David Culler, "Versatile low power media access for wireless sensor networks," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*. ACM, 2004, pp. 95–107.

[303] Joseph Polastre, Robert Szewczyk, and David Culler, "Telos: enabling ultra-low power wireless research," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*. IEEE, 2005, pp. 364–369.

[304] Johan Pouwelse, Koen Langendoen, and Henk Sips, "Dynamic voltage scaling on a low-power microprocessor," in *Proceedings of the 7th annual international conference on Mobile computing and networking*. ACM, 2001, pp. 251–259.

[305] Carolyn C Preston and Andrew M Colman, "Optimal number of response categories in rating scales: reliability, validity, discriminating power, and respondent preferences," *Acta psychologica*, vol. 104, no. 1, pp. 1–15, 2000.

[306] Nissanka B. Priyantha, Aman Kansal, Michel Goraczko, and Feng Zhao, "Tiny web services: design and implementation of interoperable and evolvable sensor networks," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, ser. SenSys '08. New York, NY, USA: ACM, 2008. doi: http://doi.acm.org/10.1145/1460412.1460438. ISBN 978-1-59593-990-6 pp. 253–266. [Online]. Available: http://doi.acm.org/10.1145/1460412.1460438

[307] Programmable Web, "Programmable Web — APIs by Protocol," 2015. [Online]. Available: http://www.programmableweb.com/protocol-api

[308] Laure Quintin and Anta Theel, "Internet-of-Things Architecture," Tech. Rep., 2011, D8.1.

[309] Vijay Raghunathan, Curt Schurgers, Sung Park, and Mani B Srivastava, "Energy-aware wireless microsensor networks," *Signal Processing Magazine, IEEE*, vol. 19, no. 2, pp. 40–50, 2002.

[310] Cliff Ragsdale, *Spreadsheet Modeling & Decision Analysis: A Practical Introduction to Management Science, 6. ed.* South-Western, 2011.

[311] Rahman, Akbar, "Enhanced Sleepy Node Support for Coap, Internet-Draft, Version 05," 2014. [Online]. Available: {http://tools.ietf.org/html/draft-rahman-core-sleepy-05}

[312] Rahman, Akbar, "Sleepy Devices: Do we need to Support them in CORE?, Version 01," 2014. [Online]. Available: https://tools.ietf.org/html/draft-rahman-core-sleepy-nodes-do-we-need-01

[313] Rahman, Akbar and Fossati, Thomas and Loreto, Salvatore and Vial, Matthieu, "Sleepy Devices in CoAP - Problem Statement, Version 01," 2013. [Online]. Available: https://tools.ietf.org/html/draft-rahman-core-sleepy-problem-statement-01

[314] Calyampudi Radhakrishna Rao and Helge Toutenburg, *Linear Models and Generalizations: Least Squares and Alternatives*, ser. Springer Series in Statistics. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-74226-5

[315] Ivan S Razo-Zapata, Pieter De Leenheer, Jaap Gordijn, and Hans Akkermans, "Service network approaches," in *Handbook of service description*. Springer, 2012, pp. 45–74.

[316] Abdelmounaam Rezgui and Mohamed Eltoweissy, "Service-Oriented Sensor-Actuator Networks," *Communications Magazine*, vol. 45, no. 12, Dec. 2007.

[317] Leonard Richardson, Mike Amundsen, and Sam Ruby, *RESTful Web APIs*. OReilly Media, 2013.

[318] Leonard Richardson and Sam Ruby, *RESTful web services.* " O'Reilly Media, Inc.", 2008.

[319] Dominik Riemer, Ljiljana Stojanovic, and Benedikt Kmpgenn, "ProaSense: The Proactive Sensing Enterprise," in *12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 – June 4*, 2015.

[320] Jeremy Rifkin, "The third industrial revolution," *Engineering & Technology*, vol. 3, no. 7, pp. 26–27, 2008.

[321] Mikko Rinne, Eva Blomqvist, Robin Keskisärkkä, and Esko Nuutila, "Event processing in rdf," in *Proceedings of the 4th International Conference on Ontology and Semantic Web Patterns-Volume 1188*. CEUR-WS. org, 2013, pp. 52–64.

[322] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Rubén Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Cristoph Bussler, and Dieter Fensel, "Web service modeling ontology," *Applied ontology*, vol. 1, no. 1, pp. 77–106, 2005.

[323] Kay Römer, Friedemann Mattern, Thomas Dübendorfer, and Jürg Senn, "Infrastructure for virtual counterparts of real world objects," *ETH Zurich*, 2002.

[324] Michele Rossi, Alessandro Bassi, Francois Carrez, and Michele Zorzi, "The EU IoT-A Project Toward a Common Language for the Internet of Things," in *Ad Hoc and Sensor Networks Technical Committee (AHSN TC) Newsletter*, vol. 1, no. 6. IEEE Communications Society, 2011.

[325] Mirko Rossini, Thomas Fossati, Salvatore Loreto, and Pierpaolo Giacomin, "Sleepy option for coap," 2012.

[326] Luis Ruiz-Garcia, P Barreiro, and JI Robla, "Performance of zigbee-based wireless sensor nodes for real-time monitoring of fruit logistics," *Journal of Food Engineering*, vol. 87, no. 3, pp. 405–415, 2008.

[327] Manuel Ruiz-Sandoval, Tomonori Nagayama, and BF Spencer Jr, "Sensor development using Berkeley Mote platform," *Journal of Earthquake Engineering*, vol. 10, no. 2, pp. 289–309, 2006.

[328] Patrick Runow, Michael Clasen, and Hendrik Lock, "Anbindung mobiler Anwendungen an komplexe IT-Systeme." in *GIL Jahrestagung*, 2012, pp. 263–266.

[329] Andreas Ruppen and Sonja Meyer, "An Approach for a Mutual Integration of the Web of Things with Business Processes," in *Enterprise and Organizational Modeling and Simulation*. Springer, 2013, pp. 42–56.

[330] Andres Ruppen, "A Model-Driven, Component Generation Approach for the Web of Things," Ph.D. dissertation, Univesity of Fribourg, 2015.

[331] Christopher M Sadler and Margaret Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in *Proceedings of the 4th international conference on Embedded networked sensor systems*. ACM, 2006, pp. 265–278.

[332] Sherif Sakr, "XML compression techniques: A survey and comparison," *Journal of Computer and System Sciences*, vol. 75, no. 5, pp. 303–322, 2009.

[333] Shahnaz Saleem, Sana Ullah, and Kyung Sup Kwak, "A study of IEEE 802.15.4 security framework for wireless body area networks," *Sensors*, vol. 11, no. 2, pp. 1383–1395, 2011.

[334] Michael Sambeth, Peter Heinckiens, and Co KG, "SOA in Large-scale SAP Templates." in *GI Jahrestagung (1)*, 2010, pp. 384–390.

[335] Gérald Santucci, Cristina Martinez, and Diana Vlad-Câlcic, "The sensing enterprise," in *FInES workshop at FIA*, 2012.

[336] SAP AG, "BAPI — Introduction and Overview," 1997.

[337] Naveen Sastry and David Wagner, "Security considerations for IEEE 802.15. 4 networks," in *Proceedings of the 3rd ACM workshop on Wireless security*. ACM, 2004, pp. 32–42.

[338] Robert R Schaller, "Moore's law: past, present and future," *Spectrum, IEEE*, vol. 34, no. 6, pp. 52–59, 1997.

[339] Jochen Schiller, Achim Liers, and Hartmut Ritter, "ScatterWeb: A wireless sensornet platform for research and teaching," *Computer Communications*, vol. 28, no. 13, pp. 1545–1551, 2005.

[340] Jochen Schiller, Achim Liers, Hartmut Ritter, Rolf Winter, and Thiemo Voigt, "Scatterweb-low power sensor nodes and energy aware routing," in *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*. IEEE, 2005, pp. 286c–286c.

[341] Daniel Schmidt, Marc Krämer, Thomas Kuhn, and Norbert Wehn, "Energy modelling in sensor networks," *Advances in Radio Science*, vol. 5, no. 12, pp. 347–351, 2007.

[342] Winston KG Seah, Zhi Ang Eu, and Hwee-Pink Tan, "Wireless sensor networks powered by ambient energy harvesting (WSN-HEAP)-Survey and challenges," in *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology, 2009. Wireless VITAE 2009. 1st International Conference on*. IEEE, 2009, pp. 1–5.

[343] Toby Segaran, Evans Colin, and Jamie Taylor, *Programming the Semantic Web*, 1st ed. O'Reilly Media, 7 2009. ISBN 978-0596153816

[344] Ulrich Sendler, "Industrie 4.0," *Berlin/Heidelberg*, 2013.

[345] Rahul C Shah and Jan M Rabaey, "Energy aware routing for low energy ad hoc sensor networks," in *Wireless Communications and Networking Conference, 2002. WCNC2002. 2002 IEEE*, vol. 1. IEEE, 2002, pp. 350–355.

[346] Ali Shareef and Yifeng Zhu, "Energy modeling of processors in wireless sensor networks based on petri nets," in *Parallel Processing-Workshops, 2008. ICPP-W'08. International Conference on*. IEEE, 2008, pp. 129–134.

[347] Mary Shaw and David Garlan, *Software architecture: perspectives on an emerging discipline*. Prentice Hall Englewood Cliffs, 1996, vol. 1. ISBN 978-0131829572

[348] Kamran Sheikh, Maarten Wegdam, and Marten Van Sinderen, "Middleware support for quality of context in pervasive context-aware systems," in *Pervasive Computing and Communications Workshops, 2007. PerCom Workshops' 07. Fifth Annual IEEE International Conference on*. IEEE, 2007, pp. 461–466.

[349] Zach Shelby, "Constrained RESTful Environments (CoRE) Link Format," 2012. [Online]. Available: http://tools.ietf.org/html/rfc6690

[350] Zach Shelby, "RFC6690: Constrained RESTful Environments (CoRE) Link Format," *IETF standards, CoRE working group*, 2012.

[351] Zach Shelby and Carsten Bormann, *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, 2011, vol. 43. ISBN 978-0-470-74799-5

[352] Zach Shelby and Carsten Bormann, "Blockwise transfers in CoAP," *Internet Engineering Task Force*, 2014. [Online]. Available: https://tools.ietf.org/html/draft-ietf-core-block-15

[353] Zach Shelby, Klaus Hartke, Carsten Bormann, and B Frank, "RFC 7252: The Constrained Application Protocol (CoAP)," *Internet Engineering Task Force*, 2014. [Online]. Available: https://tools.ietf.org/html/rfc7252

[354] Zach Shelby, Klaus Hartke, Carsten Bormann, and Brian Frank, "Constrained application protocol (coap), internet-draft," 2011.

[355] Amit P Sheth, Karthik Gomadam, and Jon Lathem, "SA-REST: Semantically interoperable and easier-to-use services and mashups," *IEEE Internet Computing*, no. 6, pp. 91–94, 2007.

[356] Kang G Shin and Parameswaran Ramanathan, "Real-time computing: A new discipline of computer science and engineering," *Proceedings of the IEEE*, vol. 82, no. 1, pp. 6–24, 1994.

[357] Jon Siegel, *CORBA 3 fundamentals and programming*. John Wiley & Sons New York, NY, USA:, 2000, vol. 2.

[358] Doug Simon and Cristina Cifuentes, "The squawk virtual machine: Java on the bare metal," in *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. ACM, 2005, pp. 150–151.

[359] Doug Simon, Cristina Cifuentes, Dave Cleal, John Daniels, and Derek White, "Java on the bare metal of wireless sensor devices: the squawk Java virtual machine," in *Proceedings of the 2nd international conference on Virtual execution environments*. ACM, 2006, pp. 78–88.

[360] Pavle Skocir, Stjepko Zrncic, Damjan Katusic, Mario Kusek, and Gordan Jezic, "Energy consumption model for devices in machine-to-machine system," in *Telecommunications (ConTEL), 2015 13th International Conference on*. IEEE, 2015, pp. 1–8.

[361] Randall B Smith, Bernard Horan, John Daniels, and Dave Cleal, "Programming the world with sun SPOTs," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 2006, pp. 706–707.

[362] Klaus Sperner, Sonja Meyer, and Carsten Magerkurth, "Introducing entity-based concepts to business process modeling," in *3rd International Workshop and Practitioner Day on BPML*, 2011.

[363] Patrik Spiess, Stamatis Karnouskos, Dominique Guinard, Domnic Savio, Oliver Baecker, LMSD Souza, and Vlad Trifa, "SOA-based integration of the internet of things in enterprise services," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*. IEEE, 2009, pp. 968–975.

[364] Thomas Staub, Simon Morgenthaler, Daniel Balsiger, Paul Kim Goode, and Torsten Braun, "Adam: Administration and deployment of adhoc mesh networks," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2011 IEEE International Symposium on a*. IEEE, 2011, pp. 1–6.

[365] Girts Strazdins, Atis Elsts, and Leo Selavo, "Mansos: easy to use, portable and resource efficient operating system for networked embedded devices," in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems* . ACM, 2010, pp. 427–428.

[366] C Timurhan Sungur, Patrik Spiess, Nina Oertel, and Oliver Kopp, "Extending BPMN for wireless sensor networks," in *Business Informatics (CBI), 2013 IEEE 15th Conference on*. IEEE, 2013, pp. 109–116.

[367] Microsoft OData Team, "Open Data Protocol Ecosystem overview." [Online]. Available: http://www.odata.org/ecosystem

[368] OASIS OData technical committee, "Whats new in odata version 4.0," Tech. Rep., 2013.

[369] Texas Instruments, "MSP430F15x, MSP430F16x, MSP430F161x MIXED SIGNAL MICROCONTROLLER Datasheet," 2002. [Online]. Available: http://www.ti.com/lit/ds/symlink/msp430f1611.pdf

[370] Texas Instruments, "CC1000 ingle Chip Very Low Power RF Transceiver Datasheet SWRS048A January 2007," 2007. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc1000.pdf

[371] Texas Instruments, "CC2420 2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver Datasheet SWRS041c 28.10.2014," 2014. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc2420.pdf

[372] Matthias Thoma, Triantafyllos Afouras, and Torsten Braun, "An application-layer restful sleepy nodes implementation for internet of things systems," in *2015 8th IFIP Wireless and Mobile Networking Conference (WMNC)*. IEEE, 2015, pp. 16–23.

[373] Matthias Thoma, Alexandru-Florian Antonescu, Theano Mintsi, and Torsten Braun, "Linked services for enabling interoperability in the sensing enterprise," in *International IFIP Working Conference on Enterprise Interoperability*. Springer, 2013, pp. 131–144.

[374] Matthias Thoma, Alexandru-Florian Antonescu, Theano Mintsi, and Torsten Braun, "Linked services for m2m communication with enterprise it systems," in *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC)*. IEEE, 2013, pp. 1212–1216.

[375] Matthias Thoma, Torsten Braun, and Carsten Magerkurth, "Enterprise Integration of Smart Objects using Semantic Service Descriptions," in *Proc. of IEEE Wireless Communication and Networking Conference (WCNC 2014), Istanbul, Turkey*, 2014.

[376] Matthias Thoma, Torsten Braun, Carsten Magerkurth, and Alexandru-Florian Antonescu, "Managing things and services with semantics: A survey," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–5.

[377] Matthias Thoma, Torsten Braun, Carsten Magerkurth, and Alexandru-Florian Antonescu, "On the use of IoT-technologies and semantics: Final Report," Tech. Rep., Mar. 2015.

[378] Matthias Thoma, Torsten Braun, Carsten Magerkurth, and Klaus Sperner, "Linked USDL for IoT," Tech. Rep., Dec. 2014.

[379] Matthias Thoma, Theofilos Kakantousis, and Torsten Braun, "Rest-based sensor networks with odata," in *Wireless On-demand Network Systems and Services (WONS), 2014 11th Annual Conference on*. IEEE, 2014, pp. 33–40.

[380] Matthias Thoma, Sonja Meyer, Klaus Sperner, Stefan Meissner, and Torsten Braun, "On iot-services: Survey, classification and enterprise integration," in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*. IEEE, 2012, pp. 257–260.

[381] Matthias Thoma, Klaus Sperner, and Torsten Braun, "Service descriptions and linked data for integrating wsns into enterprise it," in *Software Engineering for Sensor Network Applications (SESENA), 2012 Third International Workshop on*. IEEE, 2012, pp. 43–48.

[382] Matthias Thoma, Klaus Sperner, Torsten Braun, and Carsten Magerkurth, "Integration of wsns into enterprise systems based on semantic physical business entities," in *Wireless Days (WD), 2013 IFIP*. IEEE, 2013, pp. 1–8.

[383] Di Tian and Nicolas D Georganas, "A node scheduling scheme for energy conservation in large wireless sensor networks," *Wireless Communications and Mobile Computing*, vol. 3, no. 2, pp. 271–290, 2003.

[384] Damon Tyman, Nirupama Bulusu, and Jens Mache, "An activity-based sensor networks course for undergraduates with sun spot devices," in *ACM SIGCSE Bulletin*, vol. 41, no. 1. ACM, 2009, pp. 34–38.

[385] Lodewijk FW Van Hoesel and Paul JM Havinga, "A lightweight medium access protocol (LMAC) for wireless sensor networks: Reducing preamble transmissions and transceiver state switches," 2004.

[386] B-J Van Putten, Markus Kuestner, and Martin Rosjat, "The Future Factory initiative at SAP research," in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*. IEEE, 2009, pp. 1–4.

[387] V Vanitha, V Palanisamy, N Johnson, and G Aravindhbabu, "Liteos based extended service orientedarchitecture for wireless sensor networks," *International Journal of Computer and Electrical Engineering*, vol. 2, no. 3, p. 432, 2010.

[388] Ruben Verborgh, Thomas Steiner, DV Deursen, Rik Van de Walle, and Joaquim Gabarró Vallés, "Efficient runtime service discovery and consumption with hyperlinked RESTdesc," in *Next Generation Web Services Practices (NWeSP), 2011 7th International Conference on*. IEEE, 2011, pp. 373–379.

[389] Ruben Verborgh, Thomas Steiner, Rik Van de Walle, and Joaquim Gabarró Vallés, "The Missing Links - -How the Description Format RESTdesc applies the Linked Data Vision to connect hypermedia APIs," in *Proceedings of the First Linked APIs workshop at the Ninth Extended Semantic Web Conference*, 2012.

[390] Steve Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, no. 6, pp. 87–89, 2006.

[391] Tomas Vitvar, Jacek Kopeckỳ, Jana Viskova, and Dieter Fensel, *Wsmo-lite annotations for web services*. Springer, 2008.

[392] W3C OWL Working Group, Ed., *OWL 2 Web Ontology Language*, ser. W3C Recommendation. W3C, Dec. 2012, http://www.w3.org/TR/owl2-overview/.

[393] Gerald Wagenknecht, Markus Anwander, Torsten Braun, Thomas Staub, James Matheka, and Simon Morgenthaler, "MARWIS: a management architecture for heterogeneous wireless sensor networks," in *Wired/Wireless Internet Communications*. Springer, 2008, pp. 177–188.

[394] Wei Wang, Suparna De, Ralf Toenjes, Eike Reetz, and Klaus Moessner, "A comprehensive ontology for knowledge representation in the internet of things," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on*. IEEE, 2012, pp. 1793–1798.

[395] Lucas Francisco Wanner, Arliones Stevert Hoeller, Augusto Born. de Oliveira, and Antonio Augusto Frohlich, "Operating System Support for Data Acquisition in Sensor Networks," in *Emerging Technologies and Factory Automation, 2006. ETFA '06. IEEE Conference on*, sept. 2006. doi: 10.1109/ETFA.2006.355355 pp. 582 –585.

[396] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F Ferguson, *Web services platform architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-reliable messaging and more.* Prentice Hall PTR, 2005.

[397] Stephen A White, "Introduction to BPMN," *IBM Cooperation*, vol. 2, no. 0, p. 0, 2004.

[398] Ian H Witten, Radford M Neal, and John G Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[399] Li Da Xu, "Enterprise Systems: State-of-the-Art and Future Trends," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 4, pp. 630 –640, nov. 2011.

[400] Wei Ye, John Heidemann, and Deborah Estrin, "An energy-efficient MAC protocol for wireless sensor networks," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2002, pp. 1567–1576.

[401] DENG Yi-tao, "The Internet of Things Technology Application in the Supply Chain Management [J]," *Logistics Sci-Tech*, vol. 9, pp. 92–94, 2010.

[402] Miao Yun and Bu Yuxin, "Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid," in *Advances in Energy Engineering (ICAEE), 2010 International Conference on*. IEEE, 2010, pp. 69–72.

[403] Wen Zhao and Yu-Sheng Zheng, "Optimal dynamic pricing for perishable assets with nonhomogeneous demand," *Management science*, vol. 46, no. 3, pp. 375–388, 2000.

[404] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee, "DAML-QoS ontology for web services," in *Web Services, 2004. Proceedings. IEEE International Conference on*. IEEE, 2004, pp. 472–479.

[405] Junjie Zhu, Bo Chai, and Jiming Chen, "Adaptive channel assignment testbed on MICAz," in *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*. IEEE, 2014, pp. 205–206.

[406] Hubert Zimmermann, "Osi reference model–the iso model of architecture for open systems interconnection," *Communications, IEEE Transactions on*, vol. 28, no. 4, pp. 425–432, 1980.

[407] Zolteria, "Zolertia Z1 - Datasheet," Zolertia, Tech. Rep., 2010.

[408] Detlef Zuehlke, "SmartFactoryTowards a factory-of-things," *Annual Reviews in Control*, vol. 34, no. 1, pp. 129–138, 2010.

Publications

Refereed Publications (Conferences and Workshops)

- Thoma Matthias; Afouras, Triantafyllos; Braun, Torsten: *An Application-Layer RESTful Sleepy Node Implementation For Internet of Things Systems*, Wireless and Mobile Networking Conference (WMNC), Munich, Germany, October 5-7, 2015. IFIP/IEEE, ISBN 978-1-5090-0351-8

- Thoma, Matthias; Kakantousis, Theofilos; Braun, Torsten: *REST-based Sensor Networks with OData*, 2014 11th Annual Conference on Wireless On-demand Network Systems and Services (WONS). Obergurgl, Austria, April 2-4, 2014. IEEE, ISBN 978-1-4799-4937-3.

- Thoma, Matthias; Braun, Torsten; Magerkurth, Carsten: *Enterprise Integration of Smart Objects using Semantic Service Descriptions.*, Wireless Communications and Networking Conference (WCNC), Istanbul, Turkey, May 6-9, 2014. IEEE, ISBN 978-1-4799-3083-8

- Thoma, Matthias; Braun, Torsten; Magerkurth, Carsten; Antonescu, Alexandru-Florian: *Managing Things and Services with Semantics: A survey*, Network Operations and Management Symposium (NOMS), Krakow, Poland, May 5-9, 2014. IEEE, ISBN: 978-1-4799-0911-7

- Thoma, Matthias; Sperner, Klaus; Braun, Torsten; Magerkurth, Carsten: *Integration of WSNs into Enterprise Systems Based on Semantic Physical Business Entities*, Wireless Days (WD), Valencia, Spain, November 13-15, 2013. IFIP/IEEE, ISBN 978-1-4799-0542-3

- Thoma, Matthias; Antonescu, Alexandru-Florian; Mintsi, Theano; Braun, Torsten: *Linked Services for M2M Communication with Enterprise IT Systems*, Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International Conference on, Sardinia, Italy, July 1-5, 2013. IEEE, ISBN 978-1-4673-2479-3

- Thoma, Matthias; Antonescu, Alexandru-Florian; Mintsi, Theano; Braun, Torsten: *Linked Services for Enabling Interoperability in the Sensing Enterprise.*, Enterprise Interoperability: 5th International IFIP Working Conference, IWEI 2013, Enschede, Netherlands, March 27-28, 2013. ISBN 978-3-642-36795-3

- Thoma, Matthias; Meyer, Sonja; Sperner, Klaus; Meissner, Stefan; Braun, Torsten: *On IoT-services: Survey, Classification and Enterprise Integration*, 2012 IEEE International Conference on Internet of Things (iThings), Besancon, France, November 20-23, 2012. IEEE, ISBN: 978-1-4673-5146-1

- Thoma, Matthias; Sperner, Klaus; Braun, Torsten: *Service descriptions and linked data for integrating WSNs into enterprise IT*, Third International Workshop on Software Engineering for Sensor Network Applications, SESENA 2012, Zurich, Switzerland, 2 June 2012. IEEE, ISBN 978-1-4673-1793-1.

- Antonescu, Alexandru-Florian; Thoma, Matthias; Robinson, Philip: *Service Level Management Convergence for Future Network Enterprise Platforms*, 2012 Future Network & Mobile Summit, Berlin, Germany, July 4-6, 2012. IEEE, ISBN 978-1-4673-0320-0

Book Chapters

- Bauer, Martin; Boussard, Mathieu; Bui, Nicola; De Loof, Jourik; Magerkurth, Carsten; Meissner, Stefan; Nettstrter, Andreas; Stefa, Julianda; Thoma, Matthias; Walewski, Joachim W. (2013) (in alphabetical order) *IoT Reference Architecture*. In Enabling Things to Talk — Designing IoT solutions with the IoT Architectural Reference Model (pp. 163-211), ISBN 978-3-642-40403-0, Springer Berlin Heidelberg.

- Boussard, Mathieu; Meissner, Stefan; Nettstrter, Andreas; Olivereau, Alexis; Salinas, Alexander; Thoma, Matthias; Walewski, Joachim W. (2013) (in alphabetical order) *A Process for Generating Concrete Architectures*. In Enabling Things to Talk — Designing IoT solutions with the IoT Architectural Reference Model (pp. 163-211), ISBN 978-3-642-40403-0, Springer Berlin Heidelberg.

Unrefereed Publications (Technical Reports, Project Deliverables, Workshops)

- Mercier, Huges; Braun, Torsten; Felber, Pascal; Kropf, Peter; Kuonen,Pierre. *2015 Doctoral Workshop on Distributed Systems Bern*, Le Louverain, Switzerland, June 21-23, 2015. Institut für Informatik und angewandte Mathematik, Technical Report, INF-15-002, 2015

- Thoma, Matthias; Braun, Torsten; Magerkurth, Carsten; Antonescu, Alexandru-Florian: *On the use of IoT-technologies and Semantics: Final Report, Technical Report*, Technical Report, Switzerland, 2015. Available online: http://www.iot4bpm.de/bpm4iot_survey_tr.pdf

- Mercier, Huges; Braun, Torsten; Felber, Pascal; Kropf, Peter; Kuonen,Pierre. *2014 Doctoral Workshop on Distributed Systems Bern*, Kandersteg, Switzerland, June 3-5, 2014. Institut für Informatik und angewandte Mathematik, Technical Report, IAM-14-001, DOI 10.7892/boris.55030

- Thoma, Matthias; Braun, Torsten; Magerkurth, Carsten; Sperner, Klaus: *Linked USDL for IoT*, Technical Report, Switzerland, 2014. Available online: http://www.iot4bpm.de/bpm4iot_lusdl4iot_tr.pdf

- Bauer, Martin; Boussard, Mathieu; Bui, Nicola; Carrez, Francois; Jardack, Christine; De Loof, Jourik; Magerkurth, Carsten; Meissner, Stefan; Nettstrter, Andreas; Olivereau, Alexis; Thoma, Matthias; Walewski, Joachim W. (in alphabetical order) *Final architectural reference model for the IoT v3.0*, IoT-A, The Internet of Things Architecture. 2013.

- Mercier, Hugues; Braun, Torsten; Felber, Pascal; Kropf, Peter; Kuonen, Pierre. *2013 Doctoral Workshop on Distributed Systems* (Technical Report IAM-13-002). Bern: Institut für Informatik und angewandte Mathematik, Technical Report IAM-13-002, 2013. DOI 10.7892/boris.44127

- Bauer, Martin; Boussard, Mathieu; Bui, Nicola; Carrez, Francois; Jardack, Christine; De Loof, Jourik; Magerkurth, Carsten; Meissner, Stefan; Nettstrter, Andreas; Olivereau, Alexis; Thoma, Matthias; Walewski, Joachim W. (in alphabetical order) *Converged architectural reference model for the IoT*, IoT-A, The Internet of Things Architecture. 2012.

- Bauer, Martin; Boussard, Mathieu; Bui, Nicola; Carrez, Francois; Jardack, Christine; De Loof, Jourik; Magerkurth, Carsten; Meissner, Stefan; Nettstrter, Andreas; Olivereau, Alexis; Thoma, Matthias; Walewski, Joachim W. (in alphabetical order) *Updated reference model for IoT*, IoT-A, The Internet of Things Architecture. 2012.

- Meissner, Stefan; Dobre, Dan; Thoma, Matthias; Martin, Gregorio: *Resource Description Specification*, IoT-A, The Internet of Things Architecture. 2012.

- Jacobs, Tobias; Joos, Markus; Magerkurth, Carsten; Meissner, Stefan; Meyer, Sonja; Sperner, Klaus; Thoma, Matthias; Voelkensen, Gerd: *Adaptive, faulttolerant orchestration of distributed IoT service interactions* , IoT-A, The Internet of Things Architecture. 2012.

- Bauer, Martin; Bui, Nicola; Carrez, Francois; Giacomin, Pierpaolo; Haller, Stephan; Ho, Edward; Jardack, Christine; De Loof, Jourik; Magerkurth, Carsten; Nettstrter, Andreas; Thoma, Matthias; Walewski, Joachim W.: *Introduction to the Architectural Reference Model for the Internet of Things*, IoT-A, The Internet of Things Architecture. 2012.

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname:	Thoma, Matthias
Matrikelnummer:	10-130-524
Studiengang:	Doktorat PHIL NAT, Reglement 2008, Informatik
	Bachelor ☐ Master ☐ Dissertation ☑
Titel der Arbeit:	REST-based Internet of Things-Services and Applications in an semantics-aware Enterprise
LeiterIn der Arbeit:	Prof. Dr. Torsten Braun Communication and Distributed Systems Institut für Informatik

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe r des Gesetzes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist. Ich gewähre hiermit Einsicht in diese Arbeit.

Zürich, 09.11.2016

Ort/Datum

Unterschrift

Curriculum Vitae

Personal Details

Name	Matthias Thoma
Address	Rümlangstrasse 55
	8052 Zürich
Date of Birth	28.03.1982
Place of Birth	Emmendingen
Nationality	German

Education

2011-2016 **University of Bern**
Communication and Distributed Systems Group
PhD-Student
Supervisor: Prof. Dr. Torsten Braun

2002-2010 **Karlsruhe Institute of Technology (KIT)**
Diplom Informatiker (equiv. Master of Science in Computer Science)
Diploma Thesis: Modellierung und optimierte Abbildung von eingebetteter Software auf Multi-/Many-Core Systeme
Supervisors: Prof. Dr.-Ing. Jörg Henkel; Prof. Dr. Wolfgang Rosenstiel

Work Experience

2015- **Avaloq Evolution AG**
Software-Engineer

2011-2015 **SAP (Switzerland) Inc.**
Strategic Innovation Enablement
Research Associate
Internet of Things, Mobile Systems and Human-Computer Interaction

2007-2010 **Forschungszentrum Informatik (FZI)**
Systementwurf in der Mikroelektronik - Prof. Dr. Wolfgang Rosenstiel
Research Assistant
Code analysis
Modeling of parallel embedded systems

2006 **Infosys Limited**
Research Intern
Static code analysis of C++ systems