FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

An integration middleware for the Internet of Things

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht von

Dipl.-Ing. Markus Jung
Matrikelnummer 0525643

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Dr. techn. Wolfgang Kastner

Diese Dissertation haben begutachtet:

(Ao. Univ. Prof. Dr. techn. Wolfgang Kastner)	(Privatdoz. Dr. techn. Karl Göschka)

Wien, 04.12.2014

(Dipl.-Ing. Markus Jung)

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

An integration middleware for the Internet of Things

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der technischen Wissenschaften

by

Dipl.-Ing. Markus Jung
Registration Number 0525643

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao. Univ. Prof. Dr. techn. Wolfgang Kastner

The dissertation has been reviewed by:

(Ao. Univ. Prof. Dr. techn. Wolfgang Kastner)	(Privatdoz. Dr. techn. Karl Göschka)

Wien, 04.12.2014

(Dipl.-Ing. Markus Jung)

Erklärung zur Verfassung der Arbeit

Dipl.-Ing. Markus Jung
Gattring 11, 3143 Pyhra

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasser)

Acknowledgements

This PhD thesis covers the research work of several years performed at the Institute of Computer Aided Automation, part of Vienna University of Technology. During this time, the Automation Systems Group part of the department was a welcoming working place that made this thesis possible. My deepest gratitude goes to my supervisor Ao. Univ. Prof. Wolfgang Kastner. His feedback and guidance were always valuable and motivating. I also want to thank my colleagues at the Automation Systems Group which provided a friendly working environment and were always open for problem discussions.

Several master- and bachelor-students supported and helped me by performing proof of concept implementations and simulations. Out of the numerous students, I want to thank especially Jürgen Weidinger, Philipp Raich, Daniel Schachinger, Gregor Ryba and Thomas Hofer for providing valuable contributions to the research work.

A great acknowledgement is dedicated to the consortium of the European IoT6 research project. The project cooperation provided me valuable insights into the IoT research community and allowed me to get into contact with leading researchers in the field.

I want to show my appreciation to Prof. Yann Bocchi and Prof. Dominique Guinard for hosting my research visit in at the University of Applied Sciences in Western Switzerland and Prof. Daeyoung Kim for hosting my stay at KAIST in Korea. I always received a warm welcome and strong support at these research locations which made it possible to deepen the research cooperations.

Special thanks go to my family and Sandra for their patience and motivating support. Without them the creation of this thesis would not have been possible.

Abstract

The Internet of Things (IoT) will improve our environment by means of smart objects that are equipped with sensors and actuators, embedded in everyday objects and linked in a ubiquitous way. They enable exciting application scenarios in heterogeneous application domains such as Smart Grids, logistics, factories, traffic management, homes, and buildings. In this context, proprietary communication stacks as well as the heterogeneity of existing home and building automation technologies lead to high integration costs and are an inhibitor for the realization of the IoT.

This thesis presents an integration middleware for the IoT consisting of a communication stack, an integration approach, and a Service-oriented Architecture (SoA) with focus on home and building automation technologies. The main challenge is the optimization of Internet- and Web-technologies to enable energy efficient operation within smart objects. Heterogeneous features, interaction patterns, and semantics of existing technologies need to be supported. Finally, non-functional properties such as scalability, security, and privacy have to be taken into account.

The SoA presented in this thesis includes the design of a centralized core-infrastructure as well as a generic and decentralized access-control concept. It can be used to provide security and fine-grained access control to ensure security and privacy in scenarios where appliances are made available to third-party application providers. Furthermore, an IoT communication stack is presented to offer interoperability between smart objects based on optimized Internet- and Web-technologies. Here, main contributions are a peer-to-peer communication model resting upon Internet Protocol Version 6 (IPv6) multicasting and the concept of a Web-based engineering tool that provides methods to create sophisticated control logic by graphical programming. Finally, an integration concept regarding the most relevant wireless and wired home and building automation technologies, identification technologies, smart meter protocols, and other important information sources in the context of the IoT is presented.

Compared to the state of the art and related work, the datapoint-centric integration approach in combination with a generic information model and the peer-to-peer interaction model provides interoperability without the need for any domain-specific information models. The usability, performance, and availability can be significantly improved by the concept of a graphical logic editor and the avoidance of centralized controllers.

A proof of concept implementation and a case study demonstrate the feasibility and applicability of the concepts. The implementation is released as open source project named IoTSyS. Scalability of the stack and the integration approach is analyzed by means of analytic models and simulation.

Kurzfassung

Das Internet der Dinge (IoT) wird unseren Alltag durch intelligente Objekte verbessern. Diese sind mit Sensoren und Aktuatoren ausgestattet und werden in alltägliche Gegenstände integriert und universell vernetzt. Sie ermöglichen vielfältige Anwendungen in verschiedensten Bereichen wie beispielsweise Smart Grids, Logistik, Produktion, Verkehrsmanagement, Wohnen und Gebäude. Diese Bereiche werden zurzeit von einer Vielzahl proprietärer Heim- und Gebäudeautomationstechnologien beherrscht.

Die vorliegende Arbeit präsentiert eine Integrations-Middleware für das IoT, bestehend aus einem Kommunikationsstack, einem Integrationskonzept und einer Service-orientierten Architektur (SoA) mit Fokus auf Heim- und Gebäudeautomationstechnologien. Eine Herausforderung ist in diesem Zusammenhang die Optimierung von Internet- und Web-Technologien für den energieeffizienten Einsatz innerhalb von intelligenten Objekten. Zahlreiche Funktionen, Interaktionsmuster und die Semantik von bestehenden Technologien müssen integriert und Anforderungen hinsichtlich Skalierbarkeit, Sicherheit und Privatsphäre berücksichtigt werden.

Die vorgestellte SoA beinhaltet das Design einer zentralisierten Infrastruktur sowie einen generischen und dezentralisierten Datenschutzmechanismus. Dadurch können Sicherheit und umfassende Zugriffskontrolle in Szenarien gewährleistet werden, bei denen Geräte für Drittanbieter von Applikationen zur Verfügung gestellt werden. Weiters wird ein IoT-Kommunikationsstack präsentiert, der Interoperabilität zwischen intelligenten Objekten auf Basis von optimierten Internet- und Web-Technologien sicherstellt. Wesentliche Beiträge sind hier ein Peer-to-Peer Kommunikationsmodell aufbauend auf IPv6 Multicasting und das Konzept eines Webbasierten Tools, welches durch graphische Konfiguration die Realisierung von komplexen Kontrollszenarien ermöglicht. Letztlich wird ein Integrationskonzept für relevante Heim- und Gebäudeautomationstechnologien, Identifikationstechnologien, Smart Meter Protokolle und andere wichtige Informationsquellen im IoT-Kontext vorgestellt.

Verglichen mit dem Stand der Technik und anderen verwandten Arbeiten ermöglicht der Datenpunkt-zentrische Integrationsansatz in Kombination mit einem generischen Informationsmodell und dem Peer-to-Peer Interaktionsmodell umfassende Interoperabilität und vermeidet domänenspezifische Informationsmodelle. Benutzerbarkeit, Performance, und Verfügbarkeit werden durch ein graphisches Konfigurationstool und der Vermeidung von zentralen Kontrollinstanzen signifikant verbessert. Zur Demonstration der Anwendbarkeit der vorgestellten Konzepte wird eine Implementierung durchgeführt, die als Open-Source Projekt mit dem Namen IoTSyS veröffentlicht wurde. Die Skalierbarkeit des Kommunikationsstacks und des Integrationskonzepts wird durch analytische Modelle und Simulation untersucht.

Contents

CHAPTER 1

Introduction

The IoT is an emerging paradigm that extends the existing Internet infrastructure to the most constrained devices and the embedded world. Smart objects equipped with Information- and Communication-Technologies (ICT)-intelligence become citizens of a world wide Internet based communication infrastructure that interconnects appliances, sensors and actuators, mobile devices, wearables, identification tags, smart objects and cloud based information systems and services.

The term IoT has been first raised in 2001 [1] by the Auto-ID center of the MIT, which mainly focused on the Electronic Product Code (EPC) based on Radio-Frequency Identification (RFID) tags on objects and later the EPC network which formed an information system to manage this kind of Internet of Things. Within the first use of this terminology, the existing Internet and its communication technologies like the Internet Protocol (IP) were not in particular focus. However, it can be observed that in the recent years the terminology is used for different aspects and seen in a broader context of providing pervasive and ubiquitous interconnection of objects and systems.

In [2], the IoT vision as illustrated in Figure 1.1 is extending the current core and fringe Internet to the IoT for different application domains. Whereas the core Internet is made of servers and routers and the fringe Internet covers currently most of the nodes like personal computers, laptops or smart phones, the IoT will interconnect IP-enabled embedded components such as sensors, machines, active positioning tags, RFID tag readers, and also building automation equipment [2].

Atzori [3] identified three main aspects of the IoT research and development. "Things"-oriented visions focus on the identification and Near Field Communication (NFC) technologies equipping everyday objects with intelligence typically based on embedded micro-controllers, sensors and actuators. Another aspect deals with the "Internet"-oriented visions that mainly focus on the communication related aspects like communication protocols such as IP and the use of IP directly on smart objects. Further, the idea of not only using IP and providing further integration and interoperability by using *Web* technologies such as the Hypertext Transfer Protocol (HTTP) can be summarized in the ongoing activities around establishing a Web of

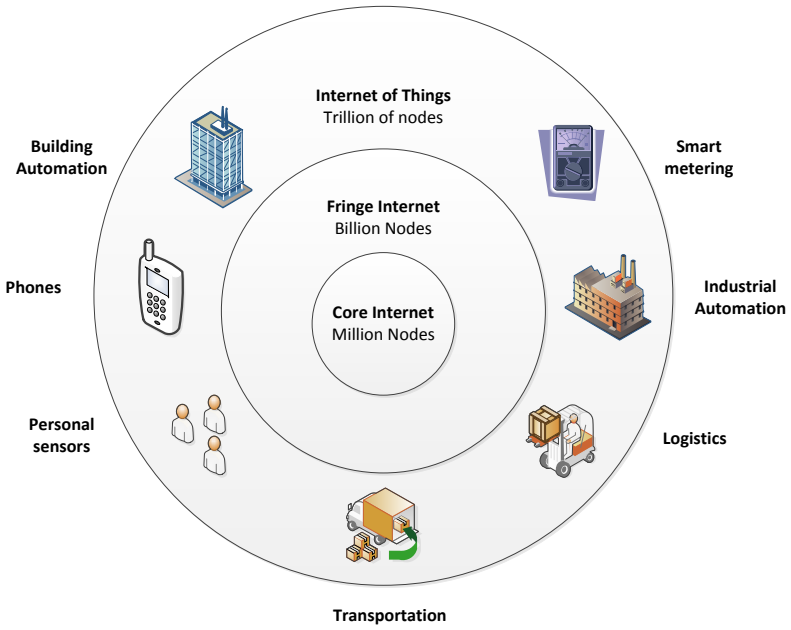


Figure 1.1: IoT Vision [2]

Things (WoT). Finally, it is not only about interconnecting objects and devices. The gathered data need to be processed. Upon the analysis of these data decisions need to be made and applications and services for the smart objects need to be provided. This can be summarized within the more "Semantic"-oriented visions of the Internet of Things which deal with semantic middleware concepts and execution environments, and reasoning and data analysis of the available information. This thesis extends these views [4] to cover also the relevance of existing Building Automation System (BAS) appliances and the widespread use and gaining importance of IPv6 as shown in Figure 1.2.

1.1 The wireless embedded Internet and the Web of Things

A subset of the Internet of Things that is in the scope of this thesis is the wireless embedded Internet based on IPv6 [2]. In this domain, the focus resides on how the Internet especially based on IPv6 can be deployed on wireless operated micro-controllers embedded in objects in order to create a scalable and ubiquitous communication infrastructure. The beginning of this development can be seen in the first work of Adam Dunkels who presented an implementation of a full TCP/IP communication stack fitting into an 8-bit micro controller architecture [5]. With this first step, showing that embedded devices can be connected through the Internet, a first milestone for the IoT was set. In parallel, communication technologies for Wireless Sensor and Actuator Networks (WSANs) became popular. With IEEE 802.15.4, a wireless communication standard has been set designed to interconnect sensors and actuators within a meshed

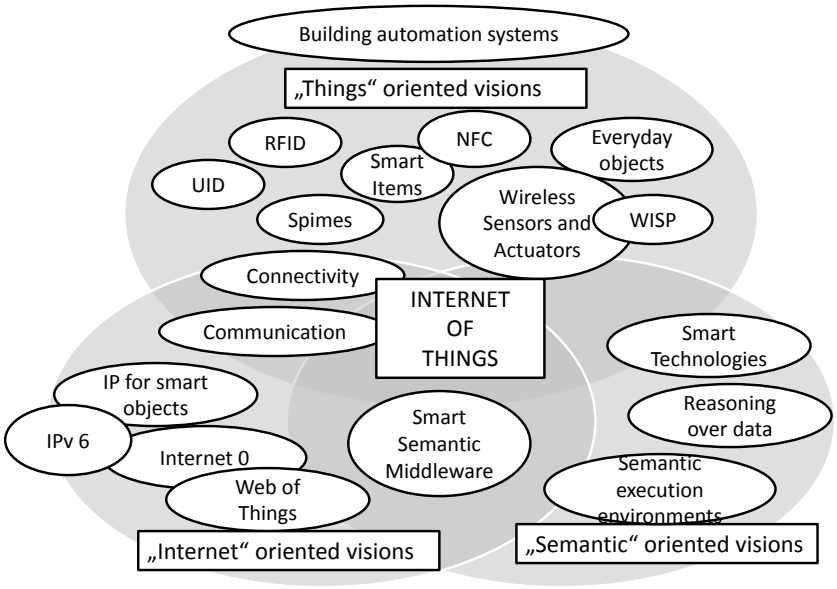


Figure 1.2: Aspects of the Internet of Things ([4] based on [3])

wireless communication network in a cheap and energy efficient way. The first main user of this standard was ZigBee, which up to date did not make a true breakthrough on the market. Soon the idea of using IP communication within WSANs was born, but due to the large number of devices the issue of the exploited Internet Protocol Version 4 (IPv4) address space arose. Therefore, the use of IPv6 became inevitable. The extended addressing space the IPv6 header provides is large enough to identify all possible smart objects, but comes at the cost of a significantly bigger protocol header compared to the IPv4 protocol header making further optimizations necessary. Here, IPv6 over Low Power Wireless Personal Area Networks (6LoWPAN) [6] provides an optimization layer that makes it feasible to use IPv6 within constrained wireless environments, such as within an IEEE 802.15.4 wireless network. Figure 1.3[1] illustrates the type of micro-controllers and dimensions that are capable of hosting an optimized IPv6 communication stack together with an IEEE 802.15.4 radio. Using IPv6 as communication protocol for the IoT provides at least interoperability at the network layer and a potential end-to-end connectivity between globally distributed devices.

Beside having only IPv6 connectivity, there are also developments towards a Web of Things communication architecture which aim at bringing down the Web to embedded devices. In [7], a Web of Things application architecture is presented that allows to integrate physical objects of the real-world into the Web. Web service based communication following the Representational State Transfer (REST) paradigm [8] can be used to provide an interoperable communica-

[1]The image shows the so-called Merkurboard of the Vienna open source domotics user group which is based on an 8-bit 16 MHz AVR micro-controller with 128KB flash and 16KB RAM memory. More details can be found at http://www.osdomotics.com/.

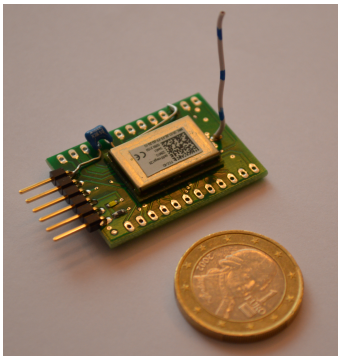


Figure 1.3: IoT device micro-controller hosting a full IPv6 stack and IEEE 802.15.4 radio

tion interface to smart objects and to existing technologies. The main idea behind using IPv6 and *Web* technologies is to solve the interoperability problem that can be found in nowadays home and building automation systems, wireless sensor network technologies, and identification and near field communication technologies. The REST paradigm has been defined by Roy Fielding [8] and is not a standard or technology by itself. Instead, it defines a design pattern for the communication interfaces within a distributed information system, that resides on the basic principles of the Web allowing to create powerful and scalable interfaces that rely on standard Internet and Web technologies. Having HTTP and the underlying Transmission Control Protocol (TCP) as communication protocol makes the use of REST based communication quite expensive for constrained devices and communication networks. This problem has been addressed by Zach Shelby by proposing a User Datagram Protocol (UDP) based equivalent to HTTP called Constrained Application Protocol (CoAP) [9] that allows to efficiently us RESTful Web service communication within constrained devices [10, 11].

1.2 Building automation systems integration and Smart Grids

Commercial buildings being responsible for about 23% of the worldwide energy consumption are receiving high attention within current research on how energy efficiency can be improved and how renewable energy sources can be used to cover the energy demands of buildings [12]. Therefore, building automation systems are a key technology in order to address the ambitious energy goals that have been set by several regional and national governmental institutions.

A further key application domain of the IoT can be found within Smart Grids. The rising costs for energy and the increased usage of renewable energy sources put a huge stress on the current power grid. Smart Grids are the next step in the evolution of the traditional power grid. Information and communication technologies (ICT) are added to support integration of variable and renewable energy sources, electric mobility, to provide customers with feedback on energy consumption and to improve the grid stability. Within the future Smart Grid, numerous applications depend on information provided by multiple stakeholders and gathered from heterogeneous technical infrastructures (cf. Figure 1.4 on page 6).

There are several Smart Grid use cases resting upon an integrated IoT communication in-

frastructure such as providing energy feedback, demand side management, home and building automation and e-mobility.

- **Energy feedback**: Domestic energy consumption is responsible for about 40 % of the overall energy demands of our society [13]. Providing customers with detailed feedback on their energy consumption has been suggested to support them to save energy. Meta-analysis comparing numerous empirical evaluations on the effectiveness of different types of energy feedback showed possible energy savings ranging from 0% to 20% depending on the quality and the level of detail of provided information [14].

- **Demand side management and load management**: Demand side management and load management both aim to reduce load peaks in the power grid and also to shed loads in situations where grid stability is at stake. Several processes, like heating or cooling, have a certain degree of freedom that allows to manage, shift or curtail the consumption of electricity. Interaction of multiple stakeholders is needed to realize this challenging use case within the Smart Grids.

- **Home and building automation**: A requirement for managing energy consumption within buildings is to employ an integration with automation technology, typically providing automation for the domains i) Heating, Ventilation and Air Conditioning (HVAC), ii) security and safety, and iii) consumer electronics. In the residential domain, these technologies add value regarding comfort or ambient assisted living.

- **E-mobility**: E-mobility will further increase load in the Smart Grid that needs to be managed through the usage of smart charging mechanisms, but at the same time the accumulators of electric vehicles can act as a flexible storage if there is a local overproduction from renewable energy sources.

The use cases require an integrated communication infrastructure that extends the local building automation control network. Buildings and appliances need to be managed in an inter-organizational way spanning large geographic areas.

The relevance of home and building automation technologies within the Smart Grid and the problem of lacking standards and interoperability have also been addressed by a NIST framework and roadmap [15] which aims at a guidance for establishing interoperability in a Smart Grid communication infrastructure.

Existing BAS technologies following a multi-tiered architecture with a separation between field communication protocols and management layer interfaces do not fulfil the requirements on such a communication. Numerous technologies exist and some of them focus only on a certain domain of building automation. Interoperability and ease of integration within IT systems is not given.

There are several main international standardized home and building automation technologies in use [16]. Although, a certain regional dominance of standards can be seen. The standards usually have a focus on one or two layers of the multi-tiered automation architecture, which has a *field layer* with sensors and actuators connected to an *automation layer* with

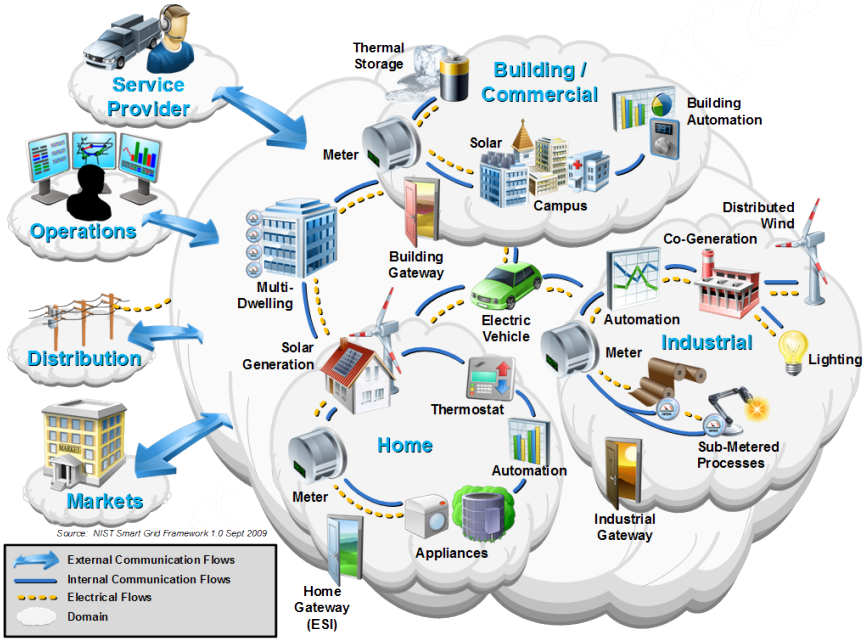


Figure 1.4: Overview over the Smart Grid customer domain [15]

programmable logic controllers, and finally a *management layer* that integrates multiple controllers and connects to the enterprise IT systems. KNX is a major internationally standardized [17] home and building automation system with the highest market share in Europe being the biggest player in German speaking countries. Different physical media can be used by KNX, although the twisted pair bus cable (KNX TP1) is the most dominant solution. Existing power cabling can be reused and a wireless communication option is also available for KNX. KNX defines a custom media access, network and transport layer. Unicast addresses are used for point-to-point communication, but for process communication between appliances a group communication mechanism based on logical addressing is used that relies on multicast-based communication between devices. A strength of this mechanism is that control logic is distributed in the field layer at the devices and no centralized controller is required. A weakness is that no central point of control exists and that no state is persisted that can be used by an external management interface.

A wireless communication standard with gaining popularity is EnOcean [18]. One of the main features is the energy harvesting capability of EnOcean devices. For example, a push button can be operated without wiring or battery just collecting the required energy to transmit a frame through the kinetic energy by pressing the button. Due to the limited energy it uses short telegrams of 14 bytes. A best-effort communication approach is provided through replacing acknowledgement messages through multiple re-transmissions with a random interval. Further, meshed networking is also not supported by EnOcean.

For Smart Grids, and especially smart metering the Metering Bus (M-Bus) provides a European norm to interface with smart meter devices and to collect meter readings. It defines a

wired physical layer based on a master/slave medium access scheme and a wireless communication mechanism. The wireless mechanism uses the 868 MHz ISM band and further specifies security features to protect the transmitted data.

At the automation and management layer, BACnet is one of the currently most important international standardized protocols [19, 20]. BACnet defines a standardized network and application layer. The network layer is in principle agnostic to the used physical layer and medium access technology. The application layer follows an object-oriented approach, with a set of standardized generic *BACnet object types*. To modify the properties of objects, alarming, event handling, file access, remote device management several services are standardized.

A main trend that can be found in nowadays building automation systems is the adoption of the IP protocol and wireless technologies. While IP technology is well-established at the management tier of automation systems, it is still not treated as a first class citizen. State of the art technologies, like for example KNX, are using IP to tunnel their custom network layer, transport layer and application layer technologies. Further, IP technologies ease the integration with enterprise and IT systems and can be one enabler for crossing the borders between different technologies. However, the adoption of IP technologies does not necessarily require an integration with the Internet, although this would allow to fully exploit the potential of having IP in place. IP can also be operated in closed networks, still leaving the main advantage of using mature technologies with several implementations and tools available [12]. Technologies such as 6LoWPAN reflect the trend of using IPv6 in wireless networks, which is also taken into account for recent standardization activities as it can be seen for ZigBee/IP [21].

The advantage of using IP in the field layer can be summarized as [22]:

- **Interoperability:** The IP communication stack is designed to provide interoperability at the network layer of a communication infrastructure. Different link-layer technologies are supported. The protocol is also adopted and available on most platforms and operating systems used on servers, PCs, smart-phones, mobile and embedded devices.
- **Evolvability and versatility:** The IP architecture is evolvable and can be used to run versatile applications. The key to this advantage resides in the end-to-end principle of IP, which allows application layer protocols to evolve independently of the underlying network layer protocols and mechanisms. No application level information is required for routing within the network.
- **Scalability:** The scalability of the IP architecture is field-proven, due to its successful operation of the public Internet in the recent years.
- **Configuration and management:** Because of its maturity and wide-spread use several protocol exist for network configuration and management.
- **Avoiding gateways:** Where gateways seem like a promising alternative to adopting IP in the field layer, they are on the long term a less flexible and inherent complex solution, showing poor scalability.

There are several alternatives to the IP architecture for networking in constrained devices where alternative layering approaches in the network stack and sometimes the strict layering

Region	2013	2014	2015	2016	2017	2018	2019	CAGR% (2014-2019)
NA	1.37	1.94	2.59	3.27	4.02	5.07	6.36	28.6%
LA	0.62	0.85	1.10	1.32	1.57	1.93	2.39	22.8%
Europe	2.05	2.86	3.80	4.76	5.89	7.39	9.09	26.0%
APAC	1.92	2.80	3.87	5.02	6.46	8.45	11.06	31.7%
MEA	0.89	1.35	1.93	2.58	3.32	4.41	5.67	33.2%
Total	6.84	9.81	13.30	16.95	21.26	27.25	34.57	28.6%

Table 1.1: IoT gateway market forecast by region in billion $ [15]

are violated due to optimizations [23–25]. However, the history of the recent years shows a strong trend towards the use of the IP architecture, because of its benefits such as modularity and separation of concerns [26–29].

Meantime, building management follows the cloud computing mainstream and local building maintenance tools are shifted away from a local operation to managed processes provided by third parties. Software-as-a-Service (SaaS) paradigms allow to move the energy management, operation and maintenance of buildings into the cloud.

There is significant research and standardization effort ongoing to solve this interoperability problem and to provide a standardized way of integrating home and building automation systems into the Internet of Things.

The interoperability problem cannot only be solved by using IP and Web communication. Also the application layer and information models need to be standardized. Therefore, a standardized communication stack is required. Further, there are several approaches to define gateways that integrate existing technologies and provide a link from local communication to cloud-based IT services. Here, a seamless integration of IoT devices and existing appliances needs to be found.

The task of gateway components is to act as local data collector concentrator with control, filtering and aggregation capabilities, supporting multiple IoT relevant technologies [30]. Table 1.1 provides a forecast on the IoT gateway market grouped by regions. Significant efforts will be placed in this area of IoT gateways and integration middlewares in order to solve the interoperability problems within the IoT.

1.3 Problem statement and hypothesis

As identified within [15], the use of closed or proprietary standards leads to high integration costs and can be an inhibitor for the realization of the IoT or an application domain like Smart Grids. For true interoperability, not only physical and syntactical interoperability needs to be established, also the semantic interoperability needs to be provided through agreeing on common information models and application services. Use cases requiring this interoperability are, for example, the energy efficient operation of HVAC-systems. Here, the HVAC control system needs to be interfaced which resides on the automation layer of a typical BAS. At the same time room automation and appliances need to be integrated and controlled. In this case, wireless

automation technologies are becoming more relevant due to the ease of installation. Driven through the evolution towards a Smart Grid, smart metering devices also need to be considered, as they provide information about the current energy consumption. In future, information about the power grid, energy market and other information such as weather forecasts have to be taken into account by the BAS to optimally plan the energy usage of a building or residential home. Further use cases can be seen in the home automation domain. A gaining need for the integration of different technologies can be seen in the area of lighting control, safety and security, comfort and convenience, energy management, remote management and assisted living [22]. The integration of these technologies is a challenging technical problem, since most of the protocols and systems are optimized towards their usage domain. They use different physical media and media access protocols, further they provide different capabilities for message exchange and means to structure the exchanged information. Chapter 5 focuses on the analysis of the different technologies and their capabilities. The challenge is here, to identify a concept that is generic enough to provide a common integration layer but at the same time being powerful enough to provide the capabilities required for the different application domains and use cases. This thesis will focus on the problem of finding such an integration concept for providing interoperability amongst the heterogeneous technologies of the IoT. The different levels of interoperability are illustrated in Figure 1.5. This thesis aims at providing technical and informational interoperability towards the level of semantic understanding within the IoT.

Problem statement 1. *Within the IoT application scenarios such as Smart Grids, heterogeneous stakeholders, technologies and data sources need to be integrated. Access to systems needs to be provided to third-party application providers.*

There are several different integration styles that can be used as a basis to tackle the interoperability challenge. From an IT system integration point of view, the main architectural styles are to use either a shared database amongst heterogeneous systems, remote procedure calls, distributed object middlewares, a message bus or file transfer [31]. Most of these architectural approaches can be seen in state of the art IoT integration approaches and middlewares. Centralized cloud-platforms follow the approach of providing a shared database for all stakeholders and appliances. Other technologies such as Message Queueing Telemetry Transport (MQTT) or Extensible Messaging and Presence Protocol (XMPP) follow the message-oriented middleware approach. A detailed overview of state of the art technologies, a survey and comparison are provided within Chapter 2 and Chapter 3. However, it will be shown that a service-oriented architecture based on Web services is the most promising solution for realizing an IoT communication infrastructure. In an inter-enterprise communication context, SOAP-based Web services provide a feature-rich stack that offers strong standardized management, discovery and security protocols.

Hypothesis 1. *A SoA based on Web services offers the most flexible solution for an integration layer that makes it possible to reuse existing information sources for a variety of application scenarios.*

The history of home and building automation systems shows that the adoption and use of these technologies is far behind the expected market growth. This can be explained due to the

large number of closed standards being incompatible to each other [22].

Problem statement 2. *Proprietary communication stacks lead to high integration costs and are an inhibitor for the realization of an Internet of Things.*

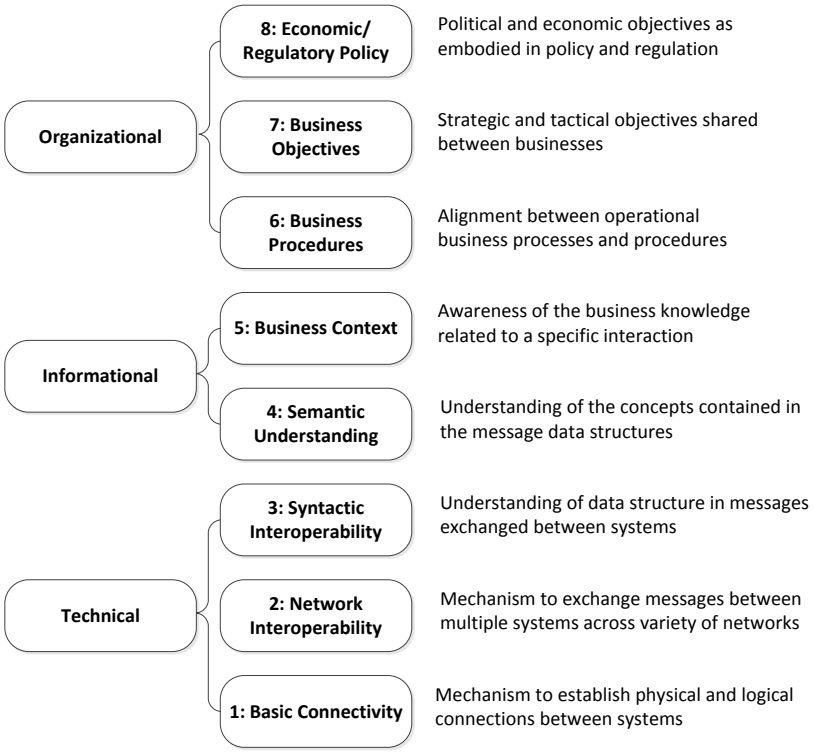


Figure 1.5: The GridWise Architecture Council's eight-layer stack provides a context for determining interoperability requirements [15]

The IP architecture and especially IPv6 can be an enabler due to the enhanced addressing space and the feasibility of use within WSANs thanks to the optimization offered by 6LoWPAN. Further, Web services can provide the required interoperability to realize interoperable end-to-end connectivity between different appliances and also a seamless integration with IT based services. Moreover, a complete system stack is required that also specifies the services, information models and semantics of the application layer. Existing technologies such as OPC Unified Architecture (OPC UA), Open Building Information eXchange (OBIX) or BACnet Web services (BACnet/WS) aim to provide the required abstraction layer for a unified integration and common application layer. Within this thesis, an OBIX based modeling approach is taken. The rationale behind this is explained within Chapter 4 and Chapter 5. OBIX is used to define a generic ontology capable to represent most of the existing BAS which is powerful enough to realize complex application use case scenarios.

Hypothesis 2. *A holistic integration concept taking into account all layers of a communication stack based on IPv6 and Web technologies is the most promising way to provide interoperability at the device level.*

The problem introduced through the heterogeneity of existing home and building automation technologies has been identified within a market analysis for home energy management systems [32], which can be one application within the IoT.

Problem statement 3. *The heterogeneity of existing home and building automation technologies is currently an inhibitor for a successful realization of IoT and Smart Grid applications.*

For a seamless integration of new IoT appliances, gateways offer a transition path to a full deployment of IoT-enabled devices. A gateway concept is required, which maps the existing communication technologies to an IoT communication stack in a component-oriented and modular way. Details of such a gateway concept are covered in Chapter 5.

Hypothesis 3. *A gateway concept including a mapping between different communication principles towards a common communication stack based on Internet and Web technologies can solve the interoperability problem with existing home and building automation technologies, and ease and simplify the engineering process of such systems.*

The use of IP and Web services assumes a certain computational capability of appliances and a certain Quality of Service (QoS) of communication links. Devices are to be considered as always active and links are expected to provide a sufficient bandwidth with a low packet loss to successfully operate IPv6. These assumptions generally do not hold in an IoT context. Further, the message size of Web services and text-based encoding increase the message size significantly and lead to an according demand of bandwidth, memory and CPU power. The future IoT is supposed to host billions of devices, therefore system scalability is of utmost importance.

Problem statement 4. *Existing Web-services-based communication is too heavy weight for the usage within constrained environments like embedded devices or wireless sensor and actuator networks and cannot fully exploit the communication features provided by IPv6. Further, scalability is significantly reduced by the heavy-weight communication and interaction patterns such as polling or unicast-based message exchange.*

There are several performance studies of using traditional HTTP- or Simple Object Access Protocol (SOAP)-based Web services on embedded devices [33] [34]. It can be seen that the resource effort and the energy consumption are significant and the average performance that can be achieved is poor. Although the continuously decreasing costs can be observed for CPUs and memory, the main challenge, especially in the domain of battery operated smart objects, is the energy efficiency. So even if more computing power is becoming available at a reasonable price the energy consumption is the main inhibitor of using more powerful computing resources in the IoT.

Hypothesis 4. *Applying optimizations to Web services and applying communication principles of state of the art building automation technologies allow to create scalable communication interfaces based on IPv6 and Web services that fit even on constrained devices.*

Within Chapter 4, a communication stack is presented, capable to use RESTful Web service communication based on CoAP and OBIX on constrained devices and further standardizing several additional features such as service discovery and security. Compared to other Web service based communication approaches using Web sockets, Device Profile for Web Services (DPWS) or Universal Plug and Play (UPnP) the advantage of Web services are combined with powerful interaction principles such as asynchronous- and group communication. This is especially important since information polling and unicast-based message exchange have a severe performance and scalability impact on WSANs. The advantage of the communication stack, regarding its performance, is shown by simulation and covered in Chapter 4.

Another important aspect for an IoT stack is the ease-of-use and ease-of-configuration. Current approaches mainly rely on script-engines demanding strong software engineering skills.

Problem statement 5. *Script-based IoT application demands high domain and technical knowledge from the end-user and decreases the usability.*

Through a datapoint centric information model and concepts that allow to synchronize the state of datapoints amongst heterogeneous devices, control logic can be configured in a simple way without the need of any scripting logic or data type transformation. More complex application logics like closed loop controllers, such as Proportional-Integral-Derivative (PID)-controllers, can be contained within logic objects offering simple datapoints for interaction.

Hypothesis 5. *Datapoint centric information modeling together with logic objects and state synchronization increase the usability of an IoT communication stack by avoiding any scripting effort for the end-user.*

The IoT stack concept defined in Chapter 4 follows this design approach. A graphical control logic editor tool proves the feasibility of scripting-less control logic editing for selected use cases evaluated in a case study within Chapter 5.

The openness and interoperability of IoT communication stacks come with the problem of exposing sensitive appliances and interfaces in the Internet.

Problem statement 6. *Interoperable and easy to access interfaces impose a security and privacy risk in the IoT context.*

For enforcing security, several state of the art mechanisms can be applied such as asymmetric or symmetric encryption schemes. Challenges arise in the management of keys and maintaining trust in the heterogeneous IoT environment. Security can be introduced at different layers of a communication stack for example at the transport layer being transparent to applications or at the application layer leading to a higher coupling between applications. Privacy issues need to be taken into account and controlled which stakeholders gain access to which data asset.

Hypothesis 6. *A fine-grained access control concept and state of the art security mechanism at the inter-enterprise communication level enable a secure and privacy-aware communication in an IoT communication infrastructure.*

1.4 Methodological approach

This section describes how the stated hypotheses are tested within this thesis. First, the requirements on an integration concept for the IoT are acquired. A survey of state of the art of integration architectures for the IoT is performed and related research work is summarized. Also a survey on the communication principles of existing home and building automation technologies is conducted in order to identify the best-practices for efficient communication for control networks. Based on the findings, a system architecture is derived by evaluating different architectural alternatives. Further, an IoT stack is developed that solely relies on open Internet and Web protocols but comes with optimizations that allow a usage within constrained environments found in WSAN. For this IoT stack, a mapping to existing communication protocols is defined. The feasibility of the integration architecture and the IoT stack is shown through a case study based on a proof of concept implementation. To analyze the scalability properties, analytic and simulation models are used.

Technical requirements engineering methodology

Technical requirements (TR) on an IoT SoA and an IoT integration middleware are identified by interviewing domain experts in the context of a research project [35]. For gathering the requirements, the following methodology is applied. First, the domain experts are recruited including technical experts from an energy utility, transmission and distribution system domain, commercial building field and a large-scale enterprise for electrical engineering. A workshop is conducted with the domain experts to identify functional and non-functional requirements on an IoT ICT infrastructure. Therefore, use cases of several different example applications that should be realized on top of such a communication infrastructure are identified and formalized using UML [36]. The UML use cases served to define the scope of the required ICT infrastructure, the involved stakeholders and the required data assets. Based on the defined UML use cases generalizable non-functional technical requirements are defined.

State of the art survey and BAS technology selection

A state of the art survey for IoT integration architectures is conducted and according technologies and standards are identified. The different approaches are described regarding their communication principles and application use cases. Further, most relevant existing home and building automation technologies are studied, in order to identify the best practices of existing control network protocols. For the evaluation, only protocols based on open standards are taken into consideration.

Software architecture and IoT stack design

The existing identified integration architecture and communication principles led to a collection of architectural decisions [37, 38] that are evaluated and finally decided in order to design an integration software architecture and the IoT communication stack. The rationale behind the decision is explained and the software architecture is described using UML [36] and the 4+1 view model proposed by Kruchten [39].

Proof of concept implementation and case study

To evaluate the integration architecture and the IoT communication stack, a proof of concept implementation is performed and evaluated within a case study that covers selected use cases identified within home and building automation and for which the IoT integration provides an obvious added value. The proof of concept implementation of the IoT SoA is mainly based on the Java platform, partly using a Platform as a Service (PaaS) cloud platform, the JBoss application server and an OSGI-based gateway. For the implementation of the IoT communication stack, the open source operating system Contiki [40] acts as platform.

Scalability analysis methodology

To analyse the scalability of the proposed integration architecture and the IoT communication stack, several quality-of-service metrics can be measured. Besides scalability, typical metrics of interest are response time, throughput, availability, reliability, security, and extensibility [41]. The performance engineering methodology presented in [42] is used to analyze the expected performance characteristics of a system throughout the whole system lifecycle, starting at the early stage of design.

There are different types of models for modeling the workload and performance behavior of a system and usually the difference is how complex and accurate these models are and further how costly it is to create them. The most accurate model is a concrete implementation of the system, which is quite costly while scalability can only be analyzed at a very late state of the system development. Otherwise analytic modeling based on queuing network models or Markov chains [43,44] or simulation models [45] using discrete-event simulation environments can be used. For the analysis of the presented IoT SoA and integration middleware, an analytic approach based on queuing network models is taken, whereas for the analysis of the stack within a WSAN a discrete-event simulation [46] approach is chosen.

1.5 Thesis outline

This work presents a concept for an Internet of Things integration which includes a holistic communication stack based on IP and Web technologies. The stack can be deployed on most constrained devices. An integration middleware for existing home and building automation, smart meters, RFID and Internet information sources offers a similar interface for existing state of the art technologies. Chapter 2 provides an overview of the state of the art and other

IoT-related communication stacks and integration approaches. The Web of Things application architecture of Dominique Guinard [7] is discussed and set into relation with the concept and architecture presented here. Furthermore, relevant standards for IoT integration architectures are analyzed and related research work is compared [47, 48]. Also, different Web service technologies are evaluated and discussed. An IoT system architecture based on the service-oriented architecture principle is presented within Chapter 3. The system architecture contains an overall ICT architecture that identifies several core components that need to be operated within a global IoT system. The main contributions are the composition and architecture of technologies and a fine-grained access control mechanism that is generic enough to be applied to heterogeneous domains [49], addressing the privacy threats within the IoT. A proof of concept implementation is presented and a scalability analysis based on a benchmark and queuing network analysis is used to identify potential bottlenecks. An evaluation of a cloud-based access control component shows that a scalable system setup can be achieved. The IoT communication stack providing an interoperable and unified communication interface for the IoT is presented in Chapter 4. Existing Web and Internet technologies can be used to create a unified communication interface. Domain specific information models can be defined resting upon a common interoperable information meta-model. IPv6 specific features like multicasting can be used to built a communication stack that allows efficient peer-to-peer interaction with Web service based communication [50,51]. The integration middleware concept that provides a seamless integration to existing home and building automation technologies is covered within Chapter 5 [4, 52]. For several technologies out of the home and building automation domain, it is shown how a mapping to the IoT communication stack can be done. The unified integration layer can ease the engineering phase of home and building automation technologies [53]. An evaluation of the developed integration middleware concept regarding interoperability and scalability is included within the chapter. Further, a case study with several use cases found for home and building automation control scenarios spanning heterogeneous technologies and information sources is presented. A proof of concept implementation of the proposed concepts and architectures is described within each of the chapters. It covers an implementation based on Java for the gateway deployed on an embedded PC platform like the Raspberry PI and a Contiki-based implementation for constrained devices [40]. For the evaluation of the scalability, an analytic queuing network model allows to estimate the performance of the gateway and a discrete event simulation is used to analyze the scalability properties of the communication stack within a wireless sensor and actuator network. Within Chapter 6 the thesis is closed with a critical discussion of the results and an outlook on future work.

CHAPTER 2

State of the art and related work

This chapter summarizes the state of the art and related work in context to this thesis. First, an overview of Web service technologies is performed together with a quantitative and qualitative comparison as published in [48]. Secondly, state of the art IoT integration approaches with a focus on home and building automation technologies are analyzed based on the work within [47]. Thirdly, the *Web of Things* research field is defined and related work to the concepts within this thesis is discussed. The chapter is concluded with an overview and comparison of related work.

2.1 Web service technologies

Web services are an important building block of the IoT in order to provide interoperable message exchange between different communication partners. This is also identified within the Smart Grid context by the NIST [15].

Regarding data exchange in the IoT, several standards are identified to be important for the realization, which are SOAP [54], WSDL [55], XML [56], XSD [57, 58], EXI [59] and CIM [60]. Before these standards are put into context with each other a brief overview of the history of Web services is provided.

Web services originated in the area of enterprise IT with the goal of easy integration of heterogeneous software systems also known as enterprise application integration. Platform specific communication middlewares that fall into the Remote Procedure Call (RPC)-based interaction style like Java Remote Method Invocation (RMI), .NET remoting or Distributed Component Object Model (DCOM), make it hard to interact with software systems that run on other platforms. These technologies are typically based on custom protocols with a binary encoding of messages making them hard to be used in cross-enterprise scenarios with multiple stakeholders and a variety of platforms. Web services try to overcome these issues by being based on platform neutral technologies and furthermore by following the principles of *service-oriented computing*. Services can be defined in the following way:

> "Services are self-describing, open components that support rapid, low-cost composition of distributed applications."
>
> *[61]*

Service providers implement the services and offer service descriptions that describe the capabilities and the protocols to interact with a service. In order to provide loose coupling between the service provider and service consumer, a service registry is used to publish service descriptions and to discover services and lookup service provider interfaces. The application of this interaction pattern and the adhering to the requirements of the service design allow to build a SoA that fits well a cross-enterprise application integration scenario that is likely to be found in future Smart Grid ICT infrastructures. Smart metering data is one core data asset that is to be involved in a variety of business processes that span multiple enterprises.

Web services aim at being platform neutral and either allow an RPC- or message-based interaction. In the case of a SoA, reliable asynchronous message-based communication is the typical usage scenario.

Figure 2.1 lists the protocol stacks of the different existing Web service technologies, which are SOAP-based Web services or the so-called WS-* stack and Web services following the REST style. REST Web services can be further divided into HTTP Web services and optimized REST Web services for constrained networks and devices using CoAP as communication protocol.

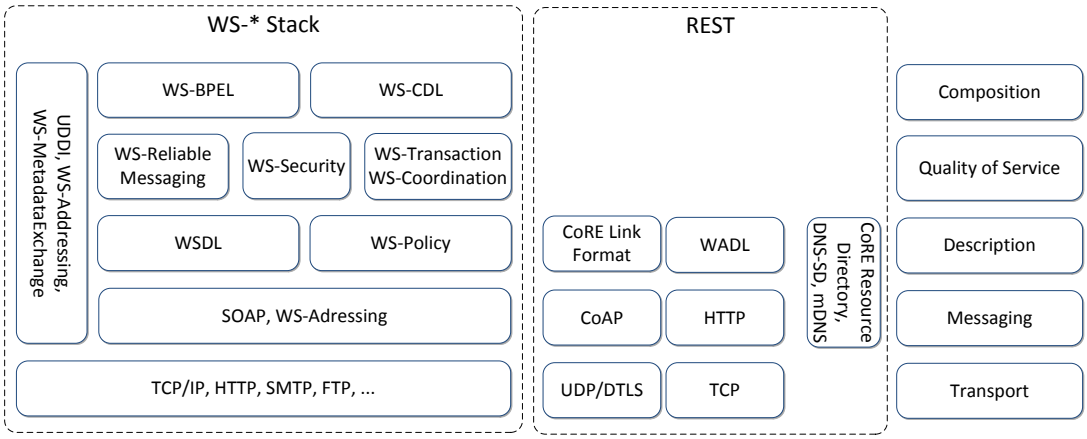


Figure 2.1: WS-* and (Constrained) REST overview

WS-* (SOAP)

Web services based on SOAP can be considered as one industry standard and are widely adopted in enterprise IT system landscapes. SOAP uses eXtensible Markup Language (XML) for message serialization in order to provide a platform neutral representation. This advantage of platform neutrality and human-readability of XML comes at the cost of less efficiency regard-

ing message size and required computational resources for handling SOAP messages. SOAP supports a feature-rich stack building the WS-* stack as presented in Figure 2.1.

The core standards are SOAP and Web Service Description Language (WSDL). They are accompanied by several other standards that provide additional features for specifying policies that allow to describe the QoS attributes and required security levels for the interaction with a service. WS-Reliable Messaging provides mechanisms to reliably deliver a SOAP message in the presence of component, system or network failures. Security at the message level independent of the transport layer is standardized within WS-Security using XML encryption and XML signature. In this way, security mechanisms are interoperable between different vendor implementations. With WS-Transaction and WS-Coordination, the outcome of distributed actions can be coordinated in order to provide distributed atomic transactions.

For the context of enterprise processes, standards such as WS-BPEL and WS-CDL are defined that allow to compose several services to business processes and also to coordinate inter-enterprise interactions. SOAP enables a variety of message exchange patterns. A classic client/-server request/response interaction model is offered, as well as asynchronous messaging or broker-based communication following the publish-subscribe interaction style.

With UDDI, WS-Addressing and WS-MetaDataExchange, the process of service discovery and interfaces for service registries is standardized.

Representational State Transfer (REST)

A different architectural style compared to SOAP Web services are RESTful Web services that follow the representational state transfer architecture style presented by Roy Fielding [62]. HTTP is an architectural style built on top of existing protocols. REST guides how to design the architecture of an application by using resources that are identified by URIs and interaction to these resources through the usage of HTTP in respect to the verbs that are defined by HTTP. The HTTP verbs *GET*, *PUT*, *POST*, *DELETE* need to be properly used to adhere to the REST style. The Web Application Description Language (WADL) is meant to provide comparable service description capabilities like WSDL but does not come to the same maturity, which can also be seen that WADL is still a W3C submission and not a recommendation like WSDL. HTTP does not come with specific security facilities. For security, transport level security by using HTTP Secure (HTTPS) has to be used. This can be considered as point-to-point security which leads to problems if intermediaries or proxies are involved in the communication. The interaction model for REST based services are limited to a request/response protocol. Therefore, asynchronous communication and scenarios where a server pushes updates to a client are only realizable through workarounds based on pull-based approaches. Furthermore, group communication is also not possible with REST based services. REST does not come with separate standards beside existing Web technologies for service discovery or service registries.

Constrained Application Protocol (CoAP)

The Constrained Application Protocol has been recently ratified as IETF RFC [9] handled via the IETF working group on Constrained RESTful Environments (CoRE). As the name suggests it is about bringing the REST architecture to constrained devices like sensor or actuators.

From a layered system view, CoAP resides on the same level as HTTP and provides similar functionality. The main difference is that it uses UDP instead of TCP which comes with several advantages and disadvantages. First, it reduces the computational requirements of a stack implementation compared to TCP. Furthermore, the reliable connection-oriented communication provided by TCP is at the one hand desirable but has severe drawbacks in constrained communication links like wireless networks (e.g., based on IEEE 802.15.4). CoAP complements the idea of having IP-based communication for constrained devices, e.g. within wireless sensor networks, with an application protocol. To achieve having IP addresses on embedded devices, 6LoWPAN provides the required optimizations as layer between the wireless data link layer and IPv6. The available payload on such networks is quite limited making UDP based protocols the preferable choice compared to TCP based protocols. CoAP supports reliable and non-reliable communication either in a request/response interaction model but also asynchronous push-based and multicast-based group communication. Regarding security, CoAP shifts the responsibility to the transport layer using either Datagram Transport Layer Security (DTLS) or IPSecurity (IPSec). For service discovery in Constrained RESTful Environments (CoRE), the domain name system with service discovery capabilities standardized within DNS Service Discovery (DNS-SD) can be used. Beside this, a central resource repository can be provided by a standardized interface based on the CoRE resource directory specification which is at the time being still at IETF draft state [63].

Qualitative comparison

This section provides a qualitative comparison of the Web service technologies.

Criteria

For the comparison the following criteria are used as defined in [48].

- **Open Standard**: In contrast to proprietary protocols, the use of open standards ensures flexibility, maintainability and extensibility in the long term while preventing data/vendor lock-in situations. Hence, a communication technology based on open standards has considerable advantages over proprietary protocols, especially in the field of Smart Grids where long-term planning is required.

- **Complexity**: The more complex a communication technology or standard is, the more effort has to be put into its implementation and maintenance. Therefore, when comparing two technologies that both offer adequate features and have suitable properties the less complex one might be the more advantageous one.

- **Platform neutral**: Communication technologies that are only available on one vendor platform can lead to higher integration costs and definitely offer less flexibility.

- **Interface description**: Technologies featuring machine readable interface descriptions have the advantage that platform specific APIs (e.g., .NET, JAVA) may be generated automatically.

- **Shared library**: A shared library that is required to communicate imposes a tight coupling between a server and a client while a loose coupling in general offers more flexibility and better suits to the service-oriented paradigm.
- **Service discovery**: An important feature of a Web service based SoA is service discovery. Service descriptions are published by service providers in order to be discovery by service consumers. One way to realize this is to create service directories, which can be used by a service provider to register services and by a service consumer to search for certain capabilities and to locate a service endpoint. The discovery can either by done at the design-time of a system by the engineer or at run-time through a dynamic binding to a service endpoint. Directories can be hosted at a central entity or distributed amongst multiple stakeholders in peer-to-peer or hierarchical structure. Further, directory-less approaches that solely rely on local peer-to-peer discovery mechanisms can also be used [64].
- **Message exchange pattern**: Depending on the exact use case, synchronous (request/response) or asynchronous communication (publish-subscribe, eventing) is more convenient.
- **Security**: Some technologies already come with built-in security facilities while other depend on a secure transport layer.
- **Transport**: The transport layer that can be used for the communication technology in question.
- **Application scenario**: A Web service or communication technology can be categorized for certain application scenarios, e.g. enterprise service bus, cross-enterprise SoA, monitoring and control applications, rich client platforms (RCP) vs. thin clients.

Evaluation

Table 2.1 shows the results of a qualitative comparison based on literature research of the various technologies.

Discussion

The results show that every Web service technology comes with specific advantages and features. Where WS-*-based Web services show their strengths in enterprise IT environments due to the rich feature set provided by the stacks, their resource demand is a problem for embedded devices. For protecting the privacy of persons in the IoT, the WS-*-stack offers the required features and technologies to provide a secure access to appliances for third-parties. The features are used within Chapter 3 to create a fine-grained access control mechanism. Certain interaction patterns (e.g., asynchronous communication or group communication) cannot be realized properly with technologies that use HTTP as transport layer. RPC protocols provide more freedom in interaction patterns but tend to be proprietary and platform specific. Recent developments like CoAP and Efficient XML Interchange (EXI) try to combine the advantages

Web service technology	WS-*	REST (HTTP)	REST (CoAP)
Open standard	Yes	No	Draft
Complexity	Medium	Low	Low
Platform neutral	Yes	Yes	Yes
Interface description	Yes	No	Yes
Shared library	No	No	No
Service discovery	Yes	No	Yes
Security	Yes	No (HTTPS)	No (IPsec)
Message exchange pattern	Sync	Sync	Sync/Async
Transport layers	HTTP	TCP	UDP
Target application platform	Enterprise App.	Thin clients	M2M

Table 2.1: Qualitative comparison [48]

of Web services, e.g. ease of integration and standardization, with the efficiency of binary RPC protocols. This makes the use of Web service technologies feasible and allows to use IP based communication throughout the whole IoT communication infrastructure directly on embedded devices. Especially, the opportunity for group communication of CoAP enables a peer-to-peer interaction of IoT devices.

2.2 Integration in the IoT and semantic interoperability

This section describes which state of the art approaches for the integration of heterogeneous technologies in the IoT exist and how semantic interoperability can be provided for communication stacks directly deployed on embedded devices [47].

For BAS technologies, existing standards deal with the problem of interoperability. The main standards found in this area are OPC UA, OBIX and BACnet/WS. Other approaches that aimed to bring Web services down to home automation devices are DPWS and UPnP.

The different standards have their strengths and weaknesses for different application use cases. However, all of them might be a potential candidate to realize semantic interoperability within an IoT SoA.

IPSO application framework

The IPSO application framework defines the application layer based on RESTful Web services to be used on constrained IP smart objects found in home automation, building automation and general M2M applications. The information model is structured in function sets. A function set is specific to a type of resource and defines a human readable name, a path template, a resource type used for discovery, the interface definition, and the data type and the allowed value range. Several function sets are specified for example to represent management capabilities of the device or more domain specific such as light control or sensors. The message encoding is using either a plain text encoding or the Sensor Markup Language (SenML). For SenML as default encoding Javascript Object Notation (JSON) is suggested. The framework aims at being complementary to existing standards such as the ZigBee smart energy profile 2 or OBIX. Since the framework does not intend to be an IoT standard the achievable interoperability is questionable. Further, no standardization body is taking care for the maintenance of the application framework. Furthermore, the capabilities for information modeling are rather puristic. For these reasons, the IPSO application framework is not considered for the definition of an IoT stack within this thesis.

Message Queue Telemetry Transport (MQTT)

MQTT is an open message protocol for Machine to Machine (M2M) communication that enables the transferring of telemetry-style data in the form of messages from pervasive devices or constrained networks to a server or small message broker. Pervasive devices may range from sensors and actuators, to mobile phones, embedded systems on vehicles, or laptops and full scale computers.

There are a couple of specifications for the MQTT protocol. The MQTT v3.1 specification enables a publish/subscribe messaging model in an extremely lightweight way. It is useful for connections with remote locations where a small code footprint is required and/or network bandwidth is at a premium. Based on the MQTT v3.1 specification, an Organization for the Advancement of Structured Information Standards (OASIS) standardization process was started in March 2013 to make MQTT an open, simple and lightweight standard protocol for M2M telemetry data communication.

MQTT for Sensor Networks (MQTT-S) v1.2 specification for sensors is aimed at embedded devices on non-TCP/IP networks, such as ZigBee. MQTT-S is a publish/subscribe messaging protocol for Wireless Sensor Network (WSN), with the aim of extending the MQTT protocol beyond the reach of TCP/IP infrastructures for sensor and actuator solutions.

MQTT is not fully overlapping with CoAP and Lightweight Open Mobile Alliance (OMA) Device Management (DM), since MQTT is a telemetry protocol. CoAP is a resource access protocol, while OMA Lightweight M2M (LWM2M) is a device management protocol. Therefore, they are not designed to satisfy the same requirements and use cases.

MQTT is more focused on publishing events with a subscribe mechanism. In contrast, CoAP is designed to integrate RESTFul architectures in constrained environments.

In conclusion, due to the centralized architecture of MQTT, even when considering that it will play a key role in the telemetry market, it is not likely to reach a critical mass in the rest of the use cases and application scenarios where the IoT and M2M are involved. The message-oriented middleware approach with a central message broker works well for sensor networks, where the information distribution to interested client is within the problem scope. However, for actuation the round-trip to the message broker is suboptimal. Datapoint-centric systems suit better the RESTful Web service design paradigm. Therefore, MQTT is not considered as communication protocol for the IoT communication stack defined in Chapter 4.

OBIX

The OBIX specification is maintained by OASIS. The main goal of OBIX is to provide an open Web service interface for accessing any kind of building automation systems. As a platform independent technology, OBIX can be used on top of any existing technology. To exchange data, OBIX defines a small set of Web services that can be used over SOAP or HTTP binding. The object model is concise but very flexible due to its support for object-oriented concepts.

The basic elements within OBIX are *objects*. Each object is of a certain object type. Currently, 17 standard object types have been specified where each type directly maps to an XML element name. Typical examples are objects that model single data items like *bool*, *int*, *real*, and *str* as well as objects types for time and date representations (e.g., *date* and *time*). In addition to model normal data (e.g., datapoints of type *real* or *bool*), OBIX provides the concept of operations (*op*). Operations are methods that can be invoked by a client.

An important concept of OBIX is the use of so called *contracts*. Contracts can be compared to templates. They are used to define new object types but also provide a possibility to specify default values.

Objects are identified by either a name, a hyper-reference, or both. Names are used to define the role of an object and used as programmatic identifiers. The name of an object is

represented by the *name* attribute and should not be used for display purposes. Instead the *displayname* facet shall be used.

In addition to these basic information modeling concepts, the OBIX specification provides a *Core Contract Library*. Within this library, basic objects like *Units* and *Weekdays* are already defined. Furthermore, OBIX defines the following concepts:

- *Points*: Points provide an abstraction of the datapoint concept within the automation system.
- *Watches*: Watches are used by clients to register to dedicated objects for monitoring object changes.
- *Alarming*: Concepts for querying, watching, and acknowledging of alarms are also part of OBIX.
- *History*: OBIX supports mechanisms to retrieve the history of object values.

OBIX is based on a service-oriented client/server architecture where OBIX clients can access the data of an OBIX server using Web services. It follows more strictly the RESTful design paradigm and provides a *read*, *write* and *invoke* service to be used for interacting with OBIX objects identified through a Uniform Resource Identifier (URI).

Within this thesis, OBIX is identified as the most promising technology to provide a base for an IoT stack. The features are suitable for the integration of state of the art communication technologies. Together with an optimized protocol binding the technology can be used within constrained embedded devices [2]. The communication stack defined in Chapter 4 uses OBIX for defining an IoT information model and extends the standard with several optimizations and a peer-to-peer interaction model.

Lightweight OMA device management

Lightweight OMA Device Management is a protocol for device management of M2M devices in 3GPP LTE-A networks.

This represents a stable take off point for device management in cellular networks, due to the considerations to support IoT/M2M communications in the release 10 of the LTE-A standard. The application of this protocol in the M2M-domain requires efficient message formats and transport replacement such as CoAP and the CoRE link format. For that reason, Lightweight OMA DM has chosen CoAP as message exchange protocol.

In addition, it focuses on providing mechanisms for asynchronous and synchronous communication, store, forward and caching mechanisms for optimizing the communication, and security features mechanisms to provide two way authentication and secure communication channels. Lightweight OMA DM is supported by oneM2M, which provides an international initiative that will play a relevant role to propose the standards for the syntactic and semantic information.

oneM2M defines an abstraction layers using a common format. This will ease the creation of the higher-layers for the IoT and M2M that enables a high-level modeling of real world entities, development of applications, and finally huge quantities of data collection. oneM2M will also offer support and solutions to facilitate the development of vertical industries and new markets.

The oneM2M standard follows a similar approach as presented within this thesis, however custom information models are defined and important features such as peer-to-peer interaction are not provided.

ETSI M2M

ETSI M2M is a service-oriented architecture to build the Service Capabilities Layer (SCL) for M2M/IoT devices, M2M/IoT Gateways, and M2M/IoT servers.

ETSI M2M standardizes the resource tree structure that resides on the M2M SCL from each one of the components. These components exchange information by means of reference points. The reference points enable the interoperability between the mentioned components such as devices, gateways and servers [65].

ETSI M2M interfaces are being implemented following the RESTful architectures style over HTTP and CoAP. The information is represented by a tree of resources, that uses XML-based or JSON-based representations for information interchange.

The *dIa* interface specifies the communication between the devices and the gateways (M2M Gateway Service Capability - GSCL), the *mId* interface between the gateways and the servers (M2M Network Service Capabilities Layer - M2M NSCL), and the *mIa* interface between the M2M NSCL and the network applications.

These interfaces provide the functionality for the registration of devices/gateways to the backend, the authorization mechanism to read or write a resource, subscription and notifications for specific events, and device management operations.

In addition to the interfaces, ETSI M2M offers the identification of the application and devices requirements for asynchronous and synchronous communications, quality of service mechanisms based on policies for optimizing the communication, and security for mutual authentication between M2M NSCL and device/gateways and secure channel establishment for data transportation.

ETSI M2M is re-using existing and well-defined standards for the device management. Finally, ETSI M2M implementations are being deployed by projects such as FI-WARE which has developed preliminary instances of the M2M interface [66], and by companies such as Radisys, Grid2Home, Intecs, Intel, InterDigital, Sensinode and Telecom Italia [67]. They have tested several types of devices for different applications, and the integration with technologies such as ZigBee, WiFi and cellular networks.

For integrating of existing technologies, ETSI M2M proposes a gateway concept residing on OBIX. However, within this thesis a different device modeling approach is taken and further features are defined.

BACnet Web Services (BACnet/WS)

BACnet/WS is an international standard maintained by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE). It is an extension to the Data Communication Protocol for Building Automation and Control Networks (BACnet) specifications.

Although the intention of BACnet/WS was to enrich the BACnet protocol with a Web service interface, BACnet/WS is not limited to the BACnet protocol. Due to its generic design, BACnet/WS can in principle be used for any kind of technology.

The fundamental elements of the data model in BACnet/WS are *nodes*. Except the root node which represents the entry point of the underlying data model, each node has exactly one parent node and may have an arbitrary number of children. Using this way of arranging nodes, a *hierarchy* can be defined which structures the underlying data model.

A node represents a data primitive within the data model. The network visible data of a node is represented as a set of *attributes*. Attributes are used to describe nodes and may themselves have attributes. While it is possible to specify proprietary attributes, BACnet/WS defines a fixed set of standard attributes.

The communication concept of BACnet/WS is based on the client/server model. In BACnet/WS a set of services is defined to provide read and write access to the node hierarchy. Currently, 11 different Web services are defined which are mainly used to read and write the attributes of nodes or to retrieve information about the attributes (e.g., the array size). Typical examples are the *getValue()* and *getValues()* services to read the value of an attribute as well as the *setValue()* and *setValues()* services that are used to change the value of an attribute. Other means like services to change the data model (e.g., add or delete nodes) or to query the node hierarchy are not supported (browsing the node hierarchy has to be done by reading the *Children* attribute).

Although the BACnet/WS specification also follows a RESTful Web service approach as OBIX, the specification was at the time of defining the IoT stack and system concept presented in Chapter 4 less mature. Also the, extensibility provided by OBIX through the contract mechanism cannot be found.

OPC unified architecture (OPC UA)

OPC UA is the successor of the popular OLE for process control (OPC). OPC provides a unified interface to automation systems and uses the Windows specific Component Object Model (COM)/DCOM for data exchange, which introduces an interoperability problem if other platforms than Windows should be used. By OPC UA the interoperability of OPC is addressed through specifying a platform neutral binary message format for exchanging messages and a SOAP-based Web service protocol binding. OPC UA is an international standard that consists of multiple sub-parts, such as a part that describes the information modeling capabilities, a part of the specification taking care of the data access interfaces, a part for alarming and conditions and parts for historical access and data programs.

To model information in OPC UA, a so-called *address space model* is introduced that provides the following features [68]:

- The address space model supports object-oriented mechanisms allowing the definition of type hierarchies and inheritance.
- The modeling concept allows a definition of models that consist of full-meshed nodes.
- Information models are defined server-side and so, clients do not need to be aware of the models since they are able to retrieve them from the server.
- Type information and type hierarchies can be accessed by clients since they are exposed like normal data.

- A key concept of OPC UA information modeling is that the OPC-specific information model can be extended by defining own models on top of existing ones.

Information modeling in OPC UA is based on defining *nodes* and *references* between them. Nodes are used to model any kind of information. This includes the representation of data instances or the definition of data types. Depending on the node class, a node has several attributes that describe it. Some attributes are common to all nodes like the *NodeId* (uniquely identifying a node within the address space), the *NodeClass* (defining the node class), the *DisplayName* (localized text that can be used by a client for displaying the name), and the *BrowseName* (identifying a node when browsing the address space). Other attributes are only available for certain node classes. For example, variables have the attribute *Value* that represents the value of the variable and the attribute *DataType* that indicates the data type of the *Value* attribute.

OPC UA is completely based on the client/server model. The communication is performed in *sessions*. To gain access to data, an OPC UA client establishes a connection to one or more OPC UA servers. In OPC UA, devices can choose one of two different transport protocols for communication. To secure the communication, a *secured channel* is set up during session establishment. This secured channel is provided by the OPC UA communication layer which is located between the transport layer and the application layer. Based on these secure transport protocols, OPC UA defines different services that are used by OPC UA clients to communicate with the servers and vice versa. These services are grouped into so called *services sets*. Typical service sets are the *Attribute Service Set* which contains services to read and write attributes of nodes as well as the *Session Service Set* that consists of services to open and close a session.

OPC UA provides a powerful and feature rich system specification. However, it follows more the approach of heavy-weight SOAP Web services which have not seen a broad acceptance in the area of WSANs, due to the high resource demands. The OPC UA information modeling and communication services can be considered as incompatible to the design paradigm of RESTful Web services. Therefore, a definition of OPC UA over CoAP is not possible in a meaningful way, which leads to the conclusion of not considering OPC UA for the definition of the IoT stack within this thesis.

Comparison of standard capabilities

This section provides a qualitative comparison of different application layer IoT standards based on a literature research. The used criteria are explained in the following paragraphs and the comparison is provided in a tabular form.

The *information modeling* criterion refers to the capabilities of the used meta-modeling approach that can be used to represent different concepts and their relationship and to express information. This includes *comprehensiveness*, *flexibility*, *extensibility*, *semantic capabilities* and *complexity* which refer to the fact of how many concepts are provided by the meta-model. This concerns, for example, if an object-oriented-modeling approach is available and whether only generic concepts are provided or the meta-model is already aligned to certain domains. Furthermore, it is addressed if the meta-model can be modified or extended and which semantics can be provided for human beings. Finally, a criteria states how complex it is to use the technology in practice. These aspects are chosen, since they are heavily influencing whether a

technology can be applied in different domains and if information models can be extended and customized for certain application scenarios.

For the provided *communication services*, the amount of services (e.g., data access, device management and configuration, discovery) are accounted and the possible transport mechanisms and encodings are outlined. The built-in security capabilities are also used as a criterion. These aspects are especially important if a technology needs to be applied in a constrained environment. If a technology only provides a rudimentary set of communication services, custom extensions are required leading to a degradation of the interoperability of a technology.

Finally, the maturity is evaluated by comparing the amount of *available implementations*, *industry adoptions* and *standardization status*. Table 2.2 summarizes the assessment results. These criteria are important if a technology is selected for a product, since the development costs are strongly depending on them.

Comparison of resource representation and data formats

In general, the *information encoding* could either be text-based or binary-based. Whereas text-based encodings are desirable for human interaction and allow for investigating exchanged messages with standard tools, binary encodings are far more efficient for machine-to-machine communication. The *encoding efficiency* reflects the ratio between the pure information payload and the overhead introduced with the encoding. For example, some encodings (e.g., XML-based) are rather verbose, since meta-information might be provided in a redundant way within a message. If meta information related to exchanged messages is separately exchanged in order to keep the message format small, a strong *communication partner coupling* is introduced, since the message formats have to be kept synchronized between all communicating entities. Encodings should be *standardized* in order to provide long-term interoperability and should provide *platform independence* by not being limited to a specific platform. Table 2.3 summarizes the assessment results.

		IPSO App. Frame.	MQTT(-S)	OBIX	oneM2M	ETSI M2M	BACnet/WS	OPC UA
Inf. modeling	Comprehensiveness	Low	Medium	Medium	Medium	Medium	Low	High
	Flexibility	Low	Medium	High	High	High	Medium	High
	Extensibility	Low	Medium	High	Medium	Medium	Medium	High
	Semantic capabilities	Low	Medium	Medium	High	High	Medium	Medium
	Simplicity	High	Medium	Medium	Medium	Low	Medium	Low
Comm. services	Amount	Low	Medium	Low	High	High	Medium	High
	Transport	HTTP, CoAP	TCP, UDP	HTTP, CoAP	HTTP, CoAP	HTTP, CoAP	HTTP	TCP
	Encoding	Plaintext, JSON	MQTT Binary	XML, OBIX Binary EXI, JSON	XML, JSON	XML, JSON	XML	XML, OPC UA Bin.
	Security	None (Transport)	Weak	None (Transport)	High (Application)	High (Application)	None (Transport)	High (Application)
Maturity	Avail. implementations	Low	Medium	Medium	Medium	Medium	Low	Medium
	Industry adoption	Medium	Medium	Low	High	High	Low	Medium
	Standardization status	Not standardized	Upcoming OASIS standard	OASIS standard	OMA, 3GPP and ETSI standards	ETSI, OMA and Broadband Forum standards	ISO standard	IEC standard

Table 2.2: Comparison of integration standards

	Plain Text	**JSON**	**XML**	**RDF**	**EXI**	**EXI(schema-informed)**	**Custom binary**
Information encoding	Text	Text	Text	Text	Binary	Binary	Binary
Encoding efficiency	Medium	Medium	Low	Low	High	Highest	Highest
Communication partner coupling	No	No	No	No	No	Yes	Yes
Standardized	No	No	Yes	Yes	Yes	Yes	No
Platform independence	High	High	High	Medium	Medium	Medium	Low

Table 2.3: Message encoding evaluation

2.3 Web of Things

The term *Web of Things (WoT)* refers to the integration of physical and every-day objects into the *World Wide Web*. The WoT is the next evolution step of the IoT. Whereas the IoT mainly focuses on interconnecting objects using IP, the WoT goes further and uses Web technologies to interconnect objects. The term was first discussed at the Internet of Things conference in 2008, by researchers like Dominique Guinard [69, 70], Vlad Trifa [71], Erik Wilde [72], Dave Ragett [73] and others.

Within his PhD thesis, Dominique Guinard presents *"A Web of Things Application Architecture - Integrating the Real World into the Web"* [7]. Figure 2.2 gives an overview of the four layers of the WoT architecture, which are i) accessibility, ii) findability, iii) sharing and iv) composition. Above these layers the applications reside and are offered to end users.

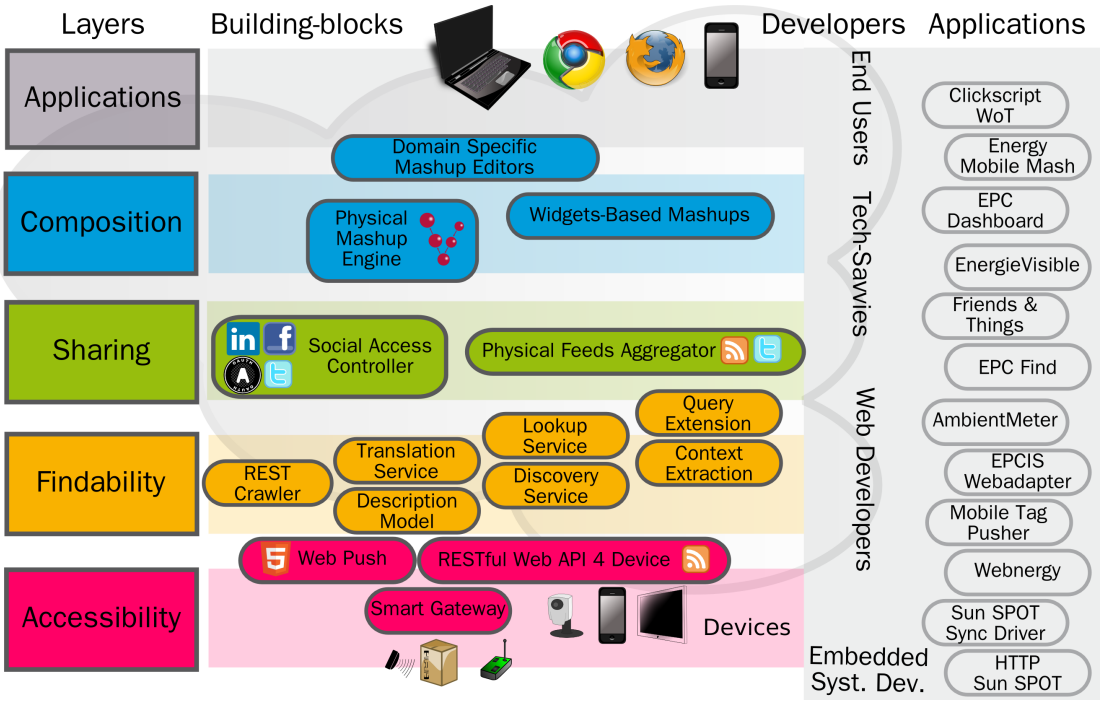


Figure 2.2: A Web of Things Application Architecture [7]

The *device accessibility layer* defines how a consistent and uniform access to all kinds of connected objects can be provided. Guinard uses RESTful Web services based on HTTP 1.1 to create such a uniform access API. The advantages of using Web technologies for the IoT and a first realization of a Web of Things are presented in [74] and in [74].

The main two requirements for a device to participate in the Web of Things are:

1. Implementation of the TCP/IP protocols ideally over an IEEE 802.3 (Ethernet) or IEEE 802.11 (WiFi) network.

2. Implementation of a Web server supporting HTTP 1.1 protocol.

[7]

For machine-to-machine communication, JSON is used to encode the device and object representation. Within his thesis, he already identified the problem of using heavy-weight TCP/IP and HTTP 1.1 communication on constrained nodes. Therefore, two mechanisms are presented that allow a real-time notification of Web clients if devices update their status. The first option is to use so-called *Web hooks*, which are simply HTTP call-back addresses that can be used by a device to notify clients. The other approach is to use Web Sockets, which allows to switch an HTTPs connection to a full duplex communication using TCP/IP. Guinard also identifies the need of integrating existing technologies not capable of Web communication through *smart gateways* also called *reverse proxies* [75]. The overall integration architecture is given in Figure 2.3.

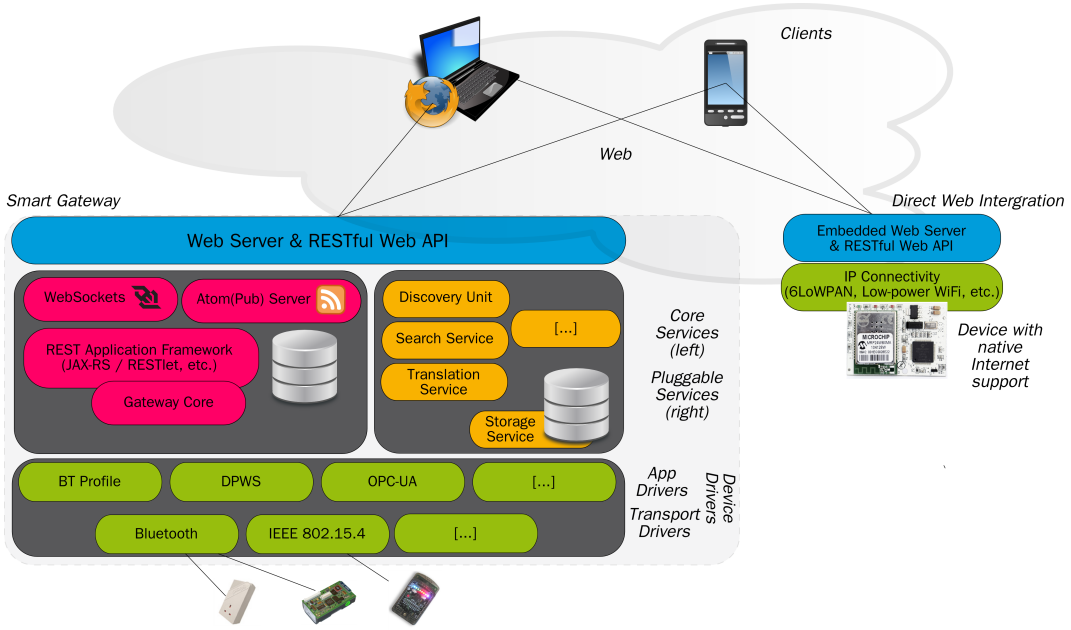


Figure 2.3: Web and Internet integration with Smart Gateways (left) and direct integration (right) [7]

The integration architecture foresees existing devices or devices which are very unlikely to become TCP/IP and HTTP enabled to be integrated using the smart gateway which has a modular design. It is consisting of core modules providing pure device access and additional optional modules that can be used for discovery, storage and other translation services. Further, there are devices that directly operate TCP/IP and HTTP. For Web clients, the integration architecture is transparent.

The *findability* layer, as described by Guinard, is related to the local and global discovery services described in Chapter 3. Guinard puts more emphasize on how to describe devices and according meta-models and how a search can be realized upon such a semantic description layer. The *sharing* layer deals with how an access to the services of devices and things can be passed to third parties. The concept mainly relies on using existing social network infrastructures and an authorization protocol such as the OAuth 2.0 Authorization Framework (OAuth). Finally, the composition layer deals with the fact how applications can be built based on Web-mash-ups using the discovery devices. Therefore, two approaches are presented. The first approach is based on a physical mash-up engine where a script-based approach allows to build application logic. The second approach is based on Widgets which are components typically based on HTML and JavaScript code.

Within the present thesis, a similar but more advanced approach to the device accessibility layer is presented compared to the work of Guinard. The IoT stack in Chapter 4 describes how a uniform communication interface for the IoT can look like. However, the approach does not stop at the message exchange or message encoding. It takes further into account a standardized meta-model for information modeling and standard application layer services that are common in the context of the IoT. In this way, a major contribution towards more interoperability within the IoT is reached. Further, the communication efficiency is increased by providing a peer-to-peer interaction mechanism based on group communication. Chapter 5, with the concept of an IoT integration middleware, is related to the *smart gateway* approach of Guinard. However, more focus is put on how existing BAS communication stacks can be integrated and information sources like the EPC Information System (EPCIS).

Similar concepts to the *findability* and *sharing* layer are part of this thesis, although there are significant differences regarding the technical mechanisms and protocols to achieve the functionality. Finally, the concept for creating control logic through a Web-based commissioning tool provides a script-less approach to build applications and further allows to decentralize the application logic on multiple devices in order to make the system more reactive and fault-tolerant. This provides a significant contribution compared to the architecture of Guinard which can be considered as the state of the art of a Web of Things architecture.

2.4 Related work

IoT gateways

In [10], Shelby presents how RESTful Web services can be adopted for a deployment on constrained devices connected through wireless networks. The paper introduces the standardization work done by the IETF CoRE working group, with CoAP as a main contribution. The interaction through CoAP with native IoT devices is sketched and the possible architecture is outlined. [2] deals with the implications of constrained data links that operate IPv6 on the upper application layer protocols. Furthermore, this book provides first ideas how to integrate various application layer protocols in the IoT.

How CoAP and EXI can be used to deploy Web services on constrained devices is described in [11]. The experimental evaluation gives first insights in the footprint of the CoAP stack

implementation and packet size improvements that can be gained by the use of EXI. In [76], it is shown how CoAP can be used as transport binding for SOAP Web services using DPWS on top of the implementation. Both papers present possible protocol stacks for native IoT devices. In contrast, this thesis presents how existing BAS devices can be integrated through a transparent gateway in such novel systems.

State of the art integration of BAS using Web service technologies like OPC UA, BACnet/WS and OBIX are presented in [77], [78] and [79]. The work influenced the architectural decisions within this thesis. The implications of using IPv6 as network layer for the integration of BAS in the IoT are analyzed in [4], which is also incorporated into the design of the proposed IoT stack.

Semantic problems that arise when different application layer protocols are integrated, and a solution to these problems based on ontologies and semantic Web technologies are addressed by e.g. [80] and [81]. The defined approaches there show how different systems can be modeled through a generic abstraction layer. Semantic Web technologies can be considered as future work to extend the IoT stack. However, due to the computational effort such an integration must consider the use of gateway devices with the according computational resources.

Group communication

In [82], a RESTful runtime script container for IoT applications is presented. The concept is based on a JavaScript execution engine that is extended with CoAP capabilities to perform asynchronous communication. For creating control logic, the runtime container acts in a client/-server model with the CoAP enabled devices. For control logic that involves the interaction between two different devices, all the information has to be transported to the application server from the one device and back to the other device. Although it is shown that using asynchronous communication this can be done quite efficiently, still a single-point of failure is introduced. The group communication model presented in this thesis avoids this single point of failure and the additional round trip of information.

A first standard draft on using IPv6 multicast for CoAP group communication is provided in [83]. The document guides how to use CoAP in the context of group communication. Furthermore, it includes different use cases and protocol flows that illustrate the group communication. The standard proposes the use of a specific group communication resource that should be offered by a CoAP endpoint. A group may consist of one or more CoAP endpoints. A CoAP endpoint can join multiple groups. This can be identified as misconception that limits the use of group communication scenarios, because in the concept of this thesis a group is not based on CoAP endpoints instead it is based on datapoints, where one CoAP endpoint can provide multiple datapoints. Having CoAP as message exchange protocol does not provide a standardized information model of resources, no normalized datapoint semantics are specified. This is covered by the application layer service specified in this stack through OBIX, enhanced with a novel group communication extension.

An alternative solution for group communication in CoAP is analyzed in [84]. There, virtual entities are introduced that provide a group endpoint for multiple resources within the wireless sensor network. The mechanism is realized through the use of CoAP unicast messages. As will

be shown in this thesis, the latency for synchronizing a state amongst a group of devices can be strongly reduced if IPv6 multicasting is used.

The relevance of multicast communication for peer-to-peer interaction in wireless networks for home and building automation has already been identified in [85]. The authors present how dynamic core based multicast routing protocol for ad-hoc wireless networks can improve the routing of multicast requests in ZigBee networks. Similarly, [86] also presents an efficient multicast routing mechanism. The principles can be seen related to the concept presented in this thesis. However, for the most efficient solutions also the upper layers above the network layer need to be considered to provide a meaningful way of group communication.

In [87], a new protocol named stateless multicast RPL forwarding (SMRF) is described. The proposed concept optimizes the multicast forwarding defined by RPL. By simulation the authors show, that compared to Trickle [88] based multicasting, the delay and the energy efficiency can be improved at the cost of increased packet loss. This multicast routing protocol is also considered for the group communication model shown in Chapter 4 and its performance is evaluated within the simulation.

Control approaches in the Internet of Things

A centralized control approach based on a RESTful runtime script container is presented in [82] where a JavaScript execution engine is used as runtime container and acts a client/server-model with CoAP enabled devices. This introduces a single point of failure and a further source of delay since for all control interactions a round-trip to the central control unit is required.

WoTKit [89] provides a framework to create IoT mash-ups. Its architecture is based on a Java Web application using the Spring Framework. For exchanging sensor data between components, a Java Messaging Service (JMS) broker is used. A graphical editor allows wiring logic modules similar to the popular Yahoo Pipe Web mash-up editor. A user can use Python as scripting language to create new control modules. Within this architecture, also a centralized component is responsible to execute the control logic which is a disadvantage compared to the approach presented in this thesis. The graphical wiring of modules provides a more convenient way to create control logic but due to the high flexibility of the data model of exchanged messages, it is not as flexible as the datapoint-centric approach in this thesis.

There are several Internet of Things platforms like Paraimpu [90], Xively[1] (known as COSM or Pachube) or ThingSpeak[2]. What they have in common is a centralized cloud platform that is used to collect sensor data and information about devices. Usually control logic is executed at the server side and more or less user-friendly scripting approaches are available. In contrast, this thesis presents a concept that avoids the use of a centralized cloud platform for executing control logic. Furthermore, it presents the concept of a graphical control logic editor that avoids the use of scripting and increases the usability for the end user.

[1] https://xively.com/

[2] https://www.thingspeak.com/

Smart grids and service-oriented architectures

Interoperability among heterogeneous technologies and stakeholders within the Smart Grid is a key issue which has been addressed by the NIST Framework and Roadmap for Smart Grid Interoperability Standards [15]. Web services using SOAP for message exchange with HTTP used as transport layer and XML for message encoding are key technologies to provide interoperability. Open and interoperable message interfaces are one aspect, but for true interoperability also the information models and semantics on the application layer need to be addressed. For this purpose, technologies and standards like OPC UA and OBIX provide standardized service interfaces, information models and data representations. OPC UA provides a core information model but can be extended with custom information models for certain domains or to map other technologies. The focus of OPC UA resides in the area of industrial automation systems. Its importance for Smart Grids is outlined in [15]. OPC UA provides beside a custom binary protocol also a protocol binding to SOAP for message exchange and XML for data encoding. OBIX can be seen as an alternative to OPC UA which focuses on the domain of building automation system. It favours a RESTful protocol design and comes with a standard object model that can be used to represent appliances found within home and building automation systems. It supports also a powerful way to extend the existing object model with custom types using the concept of contracts.

For interoperable information models in the Smart Grid, the Common Information Model (CIM) [91] provides a common vocabulary and basic ontology using UML for the electric power industry. In theory, the CIM would be applicable within the IoT context, however the current focus is more on the energy distribution network domain and more IoT specific models would be required to provide a true benefit within the IoT context.

In [92], different views on service-oriented architectures in the context of Smart Grids are outlined. Within the inter-enterprise view, a SoA based on OPC UA, CIM and semantic Web services is presented. More details on the created SoA are provided within [93]. This concept can be seen as alternative architecture compared to the service-oriented architecture developed within this thesis. In an inter-enterprise context the proposed technologies have their advantage and especially the semantic Web technologies open the field for interesting application scenarios. However, the applicability for embedded devices is not taken into account and can be considered as nearly infeasible with the proposed technology stack.

[94] presents use cases for the application of OPC UA in the Smart Grid and shows the feasibility of using OPC UA within an ICT infrastructure for the Smart Grid. Also, implementation details regarding the integration with business process engines are described. The interesting aspect is the close integration with business processes. This can also be achieved with the technology stack proposed in this thesis. There are several business process engines that allow a direct integration of RESTful Web services.

Access control in federated service-oriented architectures and Smart Grids

A Web Service Architecture for Enforcing Access Control Policies is described in [95]. Within this paper, a concept is presented how authorization for Web service requests can be handled. It is

based on user access rights and uses the Extensible Access Control Markup Language (XACML) and WS-Policy.

How access control can be realized in a cross-organizational context is shown in [96], where Web services of different service providers are composed in order to realize business functionality. Different access control frameworks for a SoA are evaluated and a 2-level access control architecture is proposed that overcomes the limitations of the evaluated frameworks. The architecture is in general similar to the presented access control architecture within Chapter 3, but details on the policy structure are not stated. Within this thesis, a concrete and generic policy structure is given and also the integration of BAS technologies is considered within this generic policy structure. Furthermore, a performance evaluation is conducted within this thesis and provides also insights into the scalability of such an architecture.

Challenges regarding the *authentication and authorization for Smart Grid application interfaces* are addressed by [97]. Key questions are outlined regarding the possibility of losing the authenticated user identity if various application interfaces are in use, privilege escalation and the challenge of defining and enforcing consistent authorization policies. Furthermore, the paper describes interoperable standards that can be used in a reference architecture to address these challenges. Similar technologies have been identified as applicable for the SoA defined in this thesis, but the paper does not come with any implementation or evaluation details.

A Public Key Infrastructure (PKI) trust model is evaluated and a Smart Grid PKI is presented in [98]. The presented PKI model can complement the SoA access control concept described within this thesis.

CHAPTER 3

A service-oriented architecture for the Internet of Things

In the future Smart Grid, numerous stakeholders within the electric power grid need to exchange information in order to realize applications like customer energy feedback, billing and invoicing of variable tariffs, demand side management and efficient charging of electric vehicles. This chapter presents a SoA for the IoT with a concrete instantiation for Smart Grids. Using a SoA based on Web services promises a convenient way to provide a data infrastructure capable to realize the required interactions in a flexible way. A main concern in such an architecture is how access to data can be controlled in order to prevent security or privacy violations. Access control depends strongly on authentication and authorization mechanisms. Therefore, this chapter presents firstly a SoA for the Smart Grid, and secondly an access control mechanism taken into consideration from the early beginning of the system design. A proof of concept implementation and a scalability analysis are used to investigate the requirements on computational resources in a large scale deployment.

3.1 Requirements

For gathering the requirements on a SoA for Smart Grids, a user survey has been conducted in the scope of a research project (Smart Web Grid [35]) and several consumer-driven requirements on such a communication infrastructure have been identified. Further, technical experts have been interviewed to identify technical and non-functional requirements. The following subsections provide an overview of the results whereas more details on the conducted study and methodology can be found within [35].

Consumer-driven requirements

- The trustworthiness of an authority responsible for storing energy data is a key requirement of consumers (CR1).

- Private consumers dislike a centralized storage and management of data due to the potential privacy threats. The data should be kept local and stored as decentralized as possible (CR2).
- Also the consumer wants to know what happens with the data, which third-party has access to the data stored within a trustworthy platform and for which purpose the access is granted. Therefore, it has to be transparent to the user, which data are collected (CR3) and how they could be used (CR4). This requirement has also been identified within [99] and [100] for context aware homes.
- The fears of private consumers and the concern of business consumers about unauthorized access to their data require strong access control mechanisms. Here, the requirement is that not only it is visible who has access to the data. Thus, the user needs to be in control if the access is granted or not and also in which level of detail (CR5).
- For an IoT SoA, secure data transmission (CR6) and adjustable data granularity (CR7) of transmitted data need to be provided.

The privacy threat within the IoT is immanent. Having numerous sensors deployed, such as smart meters, enables the detection of user patterns and behaviours.

Technical requirements

- Interoperable interfaces (TR1): The infrastructure shall be based on interoperable interfaces providing platform and vendor neutrality in order to create an ICT infrastructure that can be easily extended and maintained to avoid the risk of a vendor lock-in situation.
- Internet protocol based connectivity (TR2): The Internet Protocol should be used to interconnect data sources and applications within the Smart Grid ICT infrastructure. As outlined in [15], the advantages of using an IP-based network infrastructure are the maturity of a large number of IP standards, the availability of tools and applications, and the widespread use of IP technologies in private and public communication networks.
- Security (TR3): Communication between different stakeholders over the Internet should be based on strong security mechanisms regarding encryption and authentication in order to avoid eavesdropping of exchanged data and to protect interfaces from unauthorized usage (cf. [101]).
- Single-sign on user authentication (TR4): For end users, only a single set of credentials should be required for applications based on the Smart Grid ICT infrastructure, otherwise the user is required to memorize multiple account credentials. This leads usually to a simplified set of predictable user identifiers and passwords, which can be exploited through brute force attacks as stated in [101].
- Application portal (TR5): Smart Grid applications shall be bundled within an application portal to provide a trustworthy entry point to the users.

- Third-party applications (TR6): The ICT infrastructure should create applications and allow third-parties to provide applications based on the data sources and systems of the Smart Grid environment. This is a main requirement also identified in [101], since the utility companies will not take over all service provider responsibilities that will be available in the future Smart Grid.

3.2 Architecture overview and alternatives

Figure 3.1 presents an overview on the different stakeholders within an IoT SoA and their provided services. The fundamental building blocks of the IoT SoA are the services and capabilities offered by the different stakeholders. Beside these interfaces, several auxiliary functionalities are required. Due to the central position in a Web service based SoA we call it *IoT SoA core*.

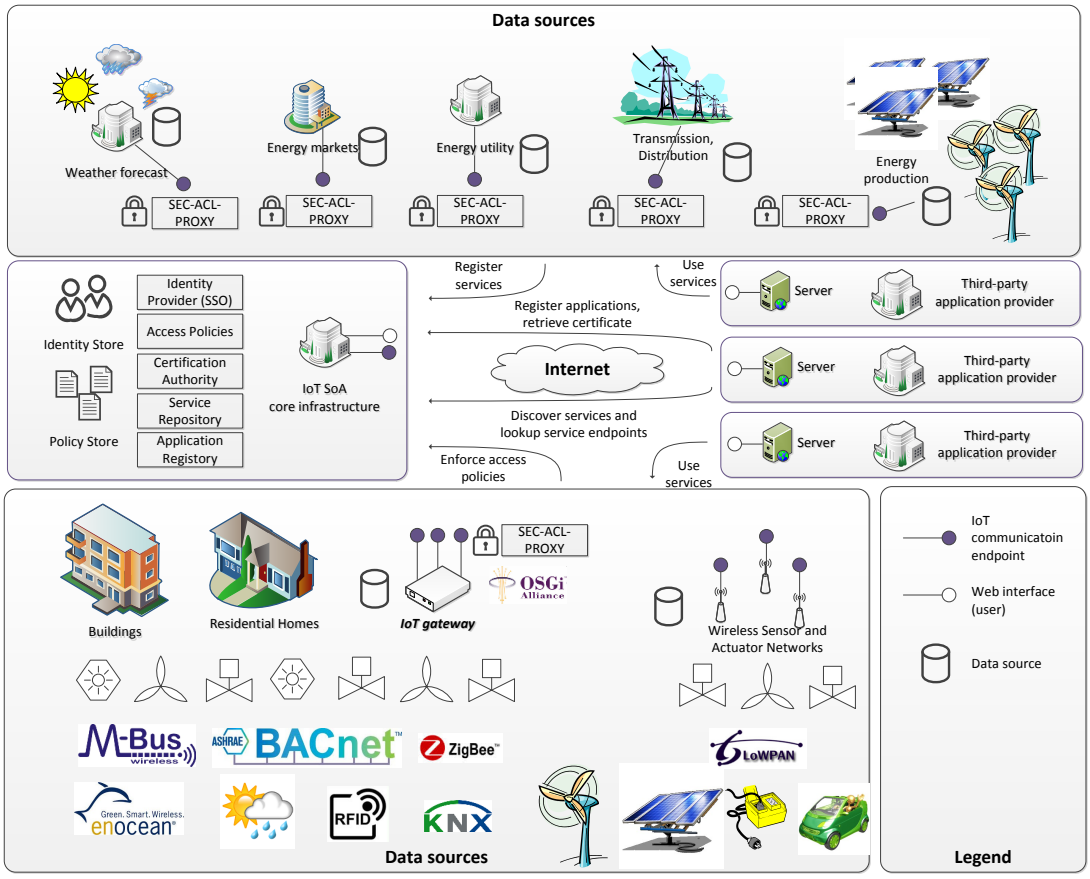


Figure 3.1: Architecture overview

The architecture follows the paradigm of a service-oriented architecture for integrating the various data sources and systems (e.g., smart meter readings and histories, home and building automation appliances, electric vehicle charging infrastructures) that are characteristic for the

Smart Grid environment. As illustrated in Figure 3.1, there is a huge heterogeneity on existing technologies and protocols (e.g., KNX, BACnet, ZigBee, 6LoWPAN) used within building automation systems. Also for smart metering various protocols and technologies already exist (e.g., Wireless M-Bus). A similar situation can be found for other Smart Grid data sources like weather forecasts, energy market prices or information about the current distribution grid status. For the integration of these heterogeneous data sources, SOAP-based Web services can be used. For existing interfaces, a gateway concept provides a mapping to a unified communication standard which can further be complemented with a generic security and access control proxy element (*SEC-ACL-PROXY*) that can be placed transparently between SOAP service consumers and providers. The Web service endpoints for the data sources are registered at a centralized infrastructure named *IoT core*, which provides features such as identity management for communication partners and users, a service repository, data access policies and a Web platform that provides access to applications. The architecture allows to integrate third party applications which can use the core infrastructure to discover and access the data sources in a secure way. A detailed description of the architecture and its components is provided in the following subsections. First, an overview of possible technologies to realize such components is given, taking into account security and privacy from the early beginning of the design phase and several architectural decisions and technology alternatives.

Data storage

A central architectural decision is where the data within an IoT architecture should be stored. One architecture is to provide a centralized typically cloud-based data platform, to which all the data are transferred. Xively [102] is such a public IoT cloud platform to which a private or commercial customer can push its sensor data. The platform provides several APIs based on REST Web services, raw sockets or MQTT to store data and to retrieve data. The customer can decide which data streams should be kept private and which should be available to the public. Based on the centralized data store, data analysis and business intelligence can be operated and third-party application providers can provide additional services. The centralized data architecture benefits regarding ease of data processing and analysis. Resting upon a central cloud-based component might lead to failures or times at which the service becomes unavailable either due to a network connectivity issue. A different approach is to use a decentralized data storage architecture, to keep the data of a private or commercial customer stored locally at a gateway component and to offer a per-application based access to third-party service providers. The decentralized architecture holds benefits regarding scalability and privacy but comes with higher costs regarding maintenance, commissioning and organization effort to manage the interaction between the different components.

Building automation systems integration

For the integration of building automation systems, different standardized approaches exist that offer also a standardized application layer for interfacing in a homogeneous way different building automation technologies. The main existing standards for this integration are OBIX,

OPC UA, BACnet/WS, MQTT, ETSI M2M (ETSI M2M), Internet Protocol for Smart Objects (IPSO) profile as outlined in Chapter 2.

Message exchange technology

There are several Web service technologies available ranging from the feature rich but complex WS-* stack based on SOAP Web services and the accompanying standards or basic RESTful Web services relying plainly on HTTP and XML. For SOAP-based Web services, DPWS provides optimizations that allow to use this type of technology also within constrained environments. Further, recently RESTful Web services for constrained environments based on CoAP are a convenient alternative to HTTP based Web services. The simplicity of RESTful Web services comes however with shortcomings regarding meta-data exchange about interfaces, security and enterprise related features like business process orchestration. Web services are an alternative to other communication technologies like platform specific object middlewares (.NET, Java RMI) or message-oriented middleware (JavaMQ, MQTT, XMPP) approaches. Finally, still a popular way of providing message exchange is to define a custom socket-based protocol either using TCP or UDP for message exchange.

Message encoding

Encoding of exchanged messages can be handled in different ways within a distributed information system. In general, two approaches can be identified. Either a binary information representation or a text-based information representation can be used. Binary information representation provides the most efficient way of representing information at the cost of requiring message parsers and encoders available at all involved platforms that deal with the specific message formats. Most technologies provide a custom encoding format. Some approaches also aim at a framework for any type of message exchange such as for example Google Protocol Buffers [103]. For text-based encoding, the most prominent technology is to use XML or more recently JSON. The main advantage is that not only the data itself can be conveyed within the message but also additional meta-data and structure information can be included. This comes however at the cost of increased message sizes and increased resource demands, which can be addressed by compressed message encodings such as EXI.

Networking

For networking, IPv4 or IPv6 are the main available choices, since inter-network communication in a global way is required for the IoT. Some BAS technologies provide a custom network layer adjusted for the very domain specific needs of the according protocol. For internetworking reasons, these types of network layer protocols are not an optimal choice.

Wireless communication technology

For the wireless communication technology of IoT devices, several alternatives exist. The most prominent technologies are either to use WiFi, Bluetooth Low Energy (BluetoothLE) or meshed

networks based on IEEE 802.15.4. To use IPv6 in such a constrained environment, optimizations are required like offered by 6LoWPAN.

Global and local service discovery

For the service discovery, several different technologies are available depending on the used Web service technology. For WS-* Web services, Universal Description, Discovery and Integration (UDDI) or Electronic Business using XML (ebXML) provide means to realize a discovery infrastructure. For RESTful based approaches, Domain Name System (DNS) or the modified variant DNS-SD can be used. DPWS can provide a customized discovery mechanism applying the WS-Discovery standard. Beside these standardized solutions, custom mechanisms can be defined to mediate the interaction between service providers and service consumers.

Application platform

The client application platform could be a Web-based application hosted on a centralized Web application server, a rich-client application or a mobile application. The architecture of the IoT SoA has strong implications on the software development effort. Some technology choices might lead to features that are not available for certain platforms.

IoT SoA core platform

Different platforms and technologies can be used to realize a core infrastructure for the IoT. The main platforms to realize such an infrastructure are the Java or .NET platform which host the required communication stacks for Web services and Web applications. Also the technologies for authentication and authorization need to be available. Alternatives like C or C++ are not suitable for such centralized platforms since the introduced software development effort would be too high and the resource constraints are not that high to mandate for a native application development. Further, the support for Web services, Web development and enterprise features is limited for these types of platforms.

IoT SoA cloud paradigm

It is possible to operate such an infrastructure on premise, or base it on a cloud deployment model. Cloud computing supports technologies like virtualization, Web services and distributed databases and allows to dynamically allocate required computational resources for applications. According to [104], there are different delivery models available which identify five characteristics, three service models and four deployments models.

The five characteristics are on-demand self-service, broad network access, resource pooling, rapid elasticity and measured service. For the service models either Software as a Service (SaaS), PaaS or Infrastructure as a Service (IaaS) are available. Figure 3.2 illustrates the different levels of user control when it comes to the deployment according to one of the service models. Within the IaaS service model, the customer keeps control of most of the layers of the platform, whereas for the SaaS service model the customer is only a user of the applications hosted by the cloud provider. Finally, as deployment model either a *private cloud*, *community*

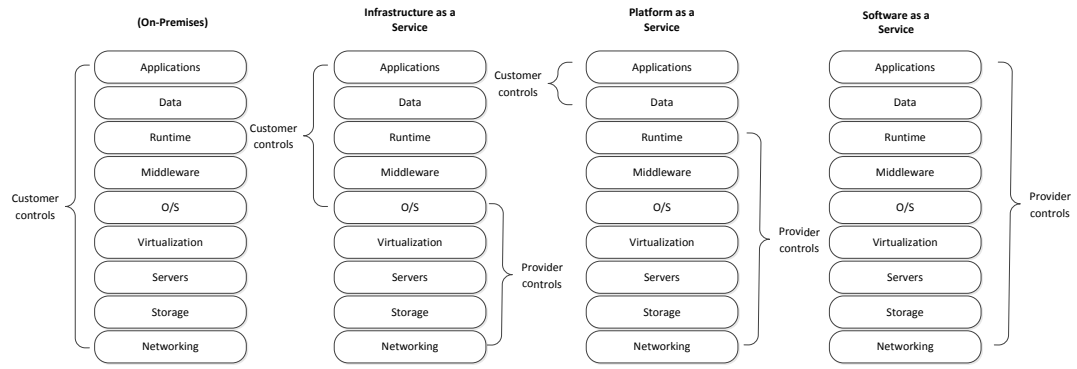


Figure 3.2: Cloud delivery models

cloud, *public cloud* or *hybrid cloud* can be used. Within a private cloud deployment, the whole computational resources are provided by the customer and the only benefits are internal sharing of resources between different business units and flexibility for assigning the computational resources. Within a community cloud, the cloud infrastructure is shared between consumers of different organizations having a common concern. A public cloud provides open access to any type of organizations. A hybrid cloud deployment model is a mixture of two or more cloud deployment models.

Security

Security related architectural decisions deal with requirements on providing the communication security primitives like *confidentiality*, *non-repudiation*, *integrity* and *authenticity* between communication partners. Further, the access to data needs be controlled so only authorized communication partners get access to sensitive data. Therefore, authorization and fine-grained access control can also be considered as a security relevant topic.

For ensuring the security primitives, symmetric and asymmetric cryptography mechanisms are one of the most relevant fundamental technologies. Symmetric cryptography relies on a shared key for encrypting exchanged messages. This can also be used for authentication purpose to only allow communication between parties that share the key. There are several shared key encryption algorithms like DES, 3DES or AES which have adequate security for an IoT communication infrastructure. Shared key encryption is mainly used to ensure confidentiality between two communication partners. To ensure integrity, non-repudiation and authenticity, asymmetric key encryption based on a private and public key pair is used. Here, the public key is shared between communication partners and can be used to decrypt information which is encrypted using the private key. There are several asymmetric encryption algorithms available although the most popular ones are RSA, DSA or ECC. Since asymmetric encryption is quite resource intensive it is not suitable to encrypt large data sets or complete messages. Instead hashing functions like MDA or SHA1 are used to create a digital finger print of a large message and only this fingerprint is encrypted using asymmetric cryptography algorithms. A key chal-

lenge of using public-/private-key-based encryption is the maintenance and the commissioning of the keys and their exchange between communication partners. Therefore, a public-key infrastructure is required with a root certification authority that establishes a trust relationship between all the communication partners. Alternatively, a Web of Trust approach can be used to realize such a trust relationship in a decentralized way. The main standard to exchange public keys is by using an X.509v3 based certificate that can be passed to another communication partner.

System authentication

For the system authentication, several different alternatives are available. The system authentication refers to the validation of the identity of communicating devices like servers, computers, mobile devices, or sensors and actuators. Here, the most basic way to achieve such an authentication is to use user-name and password based authentication or to rely on properties of the transport level like network or medium address (e.g., IP address or Ethernet MAC address) of the device. Stronger authentication mechanisms either rely on symmetric or asymmetric cryptographic mechanisms.

User authentication

For the user authentication in general, the same means as for the system authentication can be used, although user-name and password are the most popular means of providing user credentials.

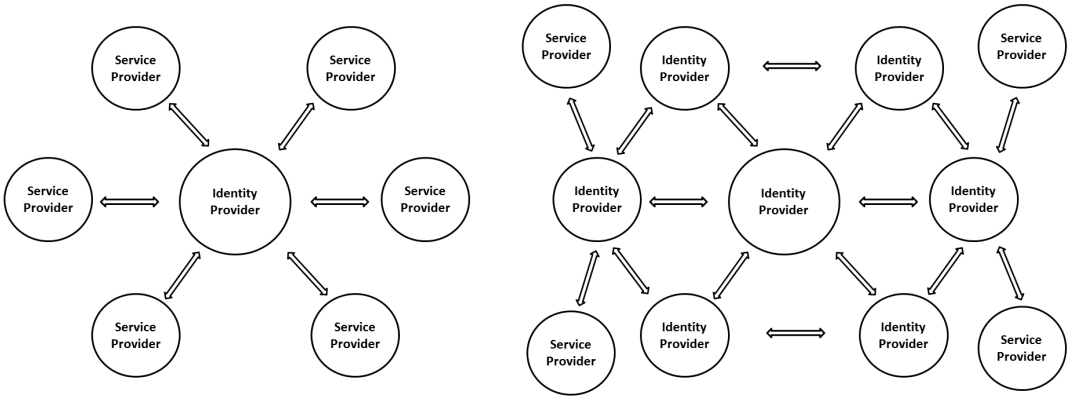
Distribution of authentication logic

The authentication of a communication entity is usually done within a security domain, in which an identity provider is responsible for managing user accounts or system identification and according credentials to authenticate an entity. For Single-Sign On (SSO) solutions, an identity provider acts as central authentication instance and all service providers (e.g., applications) trust the identity provider. If multiple security domains of different enterprises and stakeholders need to be integrated, a cross-domain federation of identity providers can be used to realize a global SSO. The two alternatives are illustrated in Figure 3.3a and Figure 3.3b. Authenticated entities are mutually accepted if they are authenticated at one identity provider of an arbitrary security domain [105].

For issuing assertion about authentication at a certain trust domain, the Security Assertion Markup Language (SAML) provides a standardized way of exchanging such trust assertions between different security domains.

Authorization technology

The authorization technology deals with the way how access policies are structured and evaluated. Technologies in this domain are the Enterprise Privacy Authorization Language (EPAL) or XACML. EPAL is proposed by IBM and still in standardization. XACML is an OASIS standard and provides more functional capabilities than EPAL. XACML supports at the one side a



(a) Federation of service providers with one identity provider

(b) Federation of cross-domain identity providers

Figure 3.3: Federation of security domains [105]

language to define access policies and on the other side a reference architecture and a data flow model which specify how to evaluate access policies within a distributed information system. In general, both technologies provide the same features, although EPAL is for most features, such as data types, functions, decision requests, combining algorithms for policy rules, subjects with multiple attributes and target description and error handling, a functional subset of XACML. Other features, such as nested policies, policy references and XML attribute values within policies and multiple responses, are completely missing within EPAL [105]. The core components of an XACML architecture as presented in Figure 3.4 are a Policy Enforcement Point (PEP), context handler, Policy Decision Point (PDP), Policy Information Point (PIP), and a Policy Administration Point (PAP).

- **Policy Enforcement Point:** This component is responsible for the enforcement of the access policies at the data source or next to the service provider. It translates domain specific terminology and request types into XACML decision requests.
- **Policy Decision Point:** Access policies are evaluated at this component. XACML based decision requests are accepted, evaluated by checking all policies and finally the PDP either returns a *permit*, *deny* or *not applicable* message.
- **Policy Administration Point:** The PAP is used to create XACML policies and to put them into a policy store which is provided to the PDP.

Centralized or decentralized policy evaluation

The access control policies could either be kept centralized or also distributed within the communication architecture. If only a central component is used scalability, availability and also performance issues might arise. Alternatively, policies could be distributed, for example at

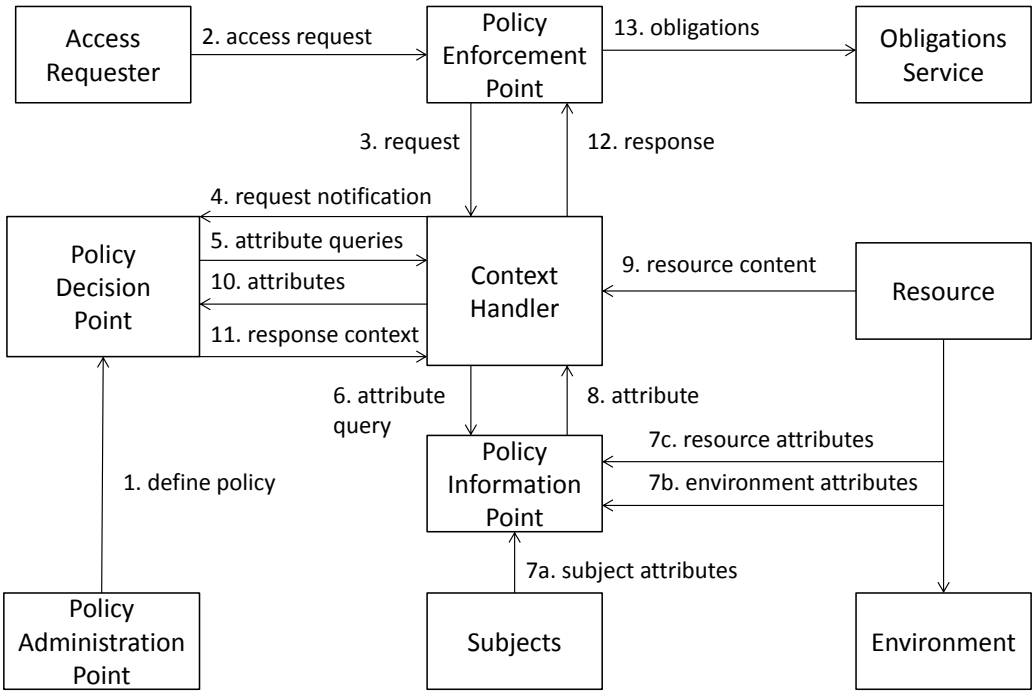


Figure 3.4: XACML architecture [106]

gateway components. This however, introduces the threat of potential local manipulations and might annul the security concept.

Authorization policy administration

The policy configuration could either be kept in a static way or just in-time when a third-party requires access to data assets within the IoT SoA.

Summary

Table 3.2 on page 52 summarizes the different architectural alternatives for realizing an IoT SoA. The architectural choices made for a proof of concept implementation within the scope of this thesis are highlighted in the table, and discussed within the following subsections.

The results of a comparison of REST vs. WS-* Web services for the Internet of Things based on a survey [107] are given in Table 3.1. The overview can act as guideline to choose a service platform.

Requirement	REST	WS-*	Justification
Mobile & embedded	+	-	Lightweight, IP/HTTP support
Ease of use	++	-	Ease to learn
Foster third-party adoption	++	-	Ease to prototype
Scalability	++	+	Web mechanisms
Web integration	+++	+	Web is RESTful
Business	+	++	QoS & security
Service contracts	+	++	WSDL
Advanced security	-	+++	WS-Security

Table 3.1: Guidelines for choosing a service architecture for IoT platform [107]

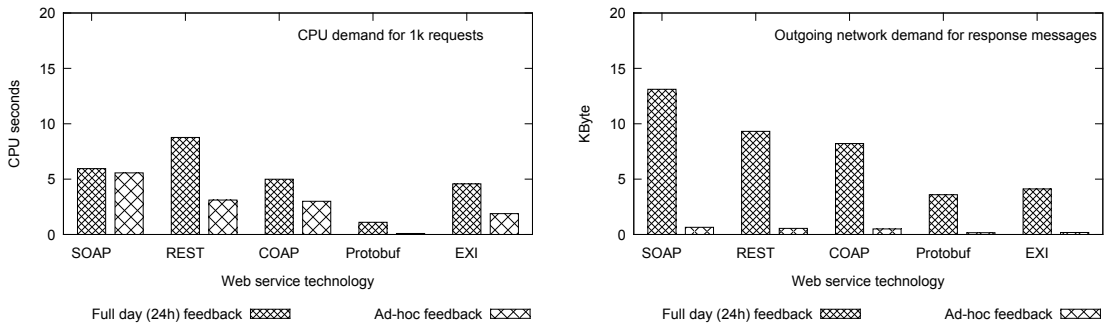
Benchmark

For a quantitative comparison of the various Web service technologies, a Web service interface for each technology is implemented and evaluated within a testbed regarding two scenarios. For the benchmark, a SOAP Web service and RESTful Web services based on HTTP and CoAP are implemented. As baseline for comparison, two TCP socket-based binary protocols are implemented using Google Protocol Buffers and EXI for message encoding. The performance metrics that are of interest are i) *CPU demand per request*, ii) *I/O demand per request*, and iii) *costs per request*. These performance metrics allow to determine the required computational resources for certain scenarios. As methodological approach for the benchmark the operational analysis [108] is taken.

Scenario: There are two smart gird scenarios used for the evaluation which are i) passing a full day smart meter history of fifteen minute intervals' readings to an energy consumer, and ii) returning a single ad-hoc smart meter measurement for live feedback. End user feedback typically contains the current load and the consumed energy. This scenario is used since the national law of Austria [109] that implements the European Union directive [110] for smart metering demands that a daily energy feedback should be provided to the customer using a fifteen minute interval for the provided meter readings.

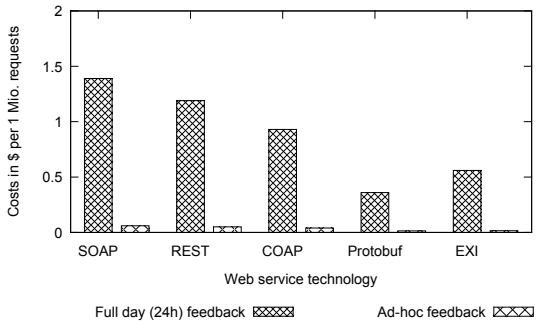
In the taken scenario, a client requests smart meter readings throughout a period of 10 minutes from a server offering a Web service interface. The Web service interface provides randomized data and a simple interface to query meter readings for a certain meter device and given period. It returns a list of meter measurements containing the average load and energy consumption.

Results: The Central Processing Unit (CPU) demands are represented in CPU seconds required to process a certain amount of requests. Figure 3.5a lists the CPU demands for each Web service technology. As expected, Web service technologies that are based on an XML-based message encoding show worse performance compared to binary RPC approaches. The differences regarding CPU demands show that the difference between SOAP and REST is neglectable, since most of the effort resides in the XML serialization. Since both technologies



(a) CPU demand

(b) I/O demand



(c) Costs

Figure 3.5: Benchmark of Web service technologies

are based on HTTP they show comparable performance. The slightly higher CPU demands of REST can be explained with implementation specific issues, since the reference SOAP stacks have a higher maturity. CoAP shows better performance due to the usage of UDP and the low packet-loss in the local network of the testbed. If a higher-rate of packet loss or failure could be expected the performance gains of using UDP would be reduced. Protobuffer and EXI have been compared using the same custom defined TCP/IP protocol. It can be seen that EXI plays in the same performance class.

The I/O demands represent the required outgoing network bandwidth to send the response to a requesting client. This is of main interest since here the most traffic occurs and the message encoding shows big differences. The significant difference of XML based and binary or compressed encodings becomes evident if time series of a full-day energy feedback are exchanged. However, for ad-hoc access the differences are low. This can be explained that XML compression is not effective if only a small amount of data needs to be exchanged. Also the overhead of SOAP and XML becomes neglectable.

For the cost estimation, a simple linear cost model is used:

- C_{CPU}: Costs for a CPU second.
- C_{IO-OUT}: Costs for one KByte of outgoing network traffic.

- C_{IO-IN}: Costs for one KByte of incoming network traffic.
- C_R: Total costs per request.
- D_{CPU}: Total average time spent of a request at the CPU.

$$C_R = C_{CPU} * D_{CPU} + C_{IO-OUT} * D_{IO-OUT}{}^{1} + C_{IO-IN} * D_{IO-IN} \tag{3.1}$$

The costs are calculated using the simple cost model presented above. For the full-day energy feedback, the costs are mainly driven by the I/O costs, for ad-hoc requests the CPU costs take a larger proportion of the costs. These costs represent a server-side infrastructure. As a simple calibration of the cost model, a snapshot of the pricing of the Amazon EC2 cloud is taken. The presented numbers only show the costs that occur due to the interface technology which is responsible for marshalling and un-marshalling network packets. The number is relatively small but within a system with millions of requests the costs can become significantly higher.

Discussion

The results prove the expectation that technologies based on an XML-based encoding show a poor performance compared to binary RPC based protocols. EXI seems to make heavy weight enterprise IT technologies feasible for direct communication with smart meters allowing to use the same interface technologies throughout the whole Smart Grid ICT infrastructure and therefore easing integration of metering and control networks with IT systems of different stakeholders. The presented performance benchmarks are based on a prototype implementation model, therefore the results may vary for different stack implementations but the magnitude of different resource demands of the different technologies can be seen. Furthermore, this evaluation provides a methodology that can be reused for concrete implementations in order to estimate the resource demand and the subsequent costs. The calibration of the cost model may differ between scenarios where data needs to be exchanged between enterprises through the Internet where network I/O is reasonable cheap and expensive connectivities (e.g., a GSM modem for a multi-utility controller).

Table 3.3 illustrates which requirements are satisfied through the architectural decisions and which technology is used to implement this decision. More details about the implementation will be provided throughout this chapter.

3.3 A service-oriented architecture for the Internet of Things

In order to interconnect different systems (cf. TR1), a service-oriented architecture (SoA) is a state of the art approach. Interoperable interfaces are required due to the large heterogeneity of technologies found within the Smart Grid and at the different stakeholders. As an architectural pattern, it is possible to realize a SoA using different technologies, but the most

[1]The demand for IO is represented in KByte.

Architectural decision		Alternatives					
Data & Communication	Data storage	Centralized	Decentralized				
	Building automation systems integration	OBIX	OPC UA	BACnet/WS	MQTT	ETSI	IPSO
	Message exchange protocol	SOAP	HTTP	CoAP			
	Message encoding	XML	JSON	EXI	Protobuf	Binary	
	Networking	IPv6	IPv4				
	Data and media access layer	IEEE 802.15.4	Bluetooth LE	WiFi			
	Global and local service discovery	UDDI	ebXML	DNS-SD	Customized		
Platform	Target application platform	Web application	Rich Client Platform	Native mobile app			
	IoT core platform	Java	.NET	C	C++		
	IoT core cloud paradigm	On premise	IaaS	PaaS	SaaS		
Security	System authentication	PKI	Usern. & Pass.	Transport level	Token based		
	User authentication	PKI	Usern. & Pass.	Transport level	Token based		
	Distribution of authentication logic	Centralized	Federated				
Authorization	Authorization technology	XACML	EPAL	Customized			
	Authorization distribution	Centralized	Decentralized				
	Authorization policy administration	Static admin.	On-demand auth.	None			

Table 3.2: IoT SoA architectural alternatives

IoT SoA Requirement	Architecture Decision	Technology Decision
CR1 trustworthy authority TR5 application portal TR6 third-party applications	Central portal and application registry	Java, JBoss application server
TR1 interoperable interfaces TR2 IP based connectivity	Service-oriented architecture based on Web services and open standards	SOAP Web services, OBIX
CR2 decentralized as possible	Gateway approach and IoT integration middleware	Java, Raspberry PI board with technology connectors
CR3 what data is collected CR4 how it is used CR5 access control mechanism for IoT data CR7 adjustable granularity	Access control policies Policy decision point	XACML
CR6 secure data transmission TR3 security	Security and access control proxy, Web service message level security, encryption and signatures, PKI certificates	WS-Security; shared key encryption: 3DES, AES public key encryption: RSA, DSA
TR5 SSO user authentication	SAML v2 Web SSO, Identity provider	Java, JBoss Picketlink XACMLv2 framework

Table 3.3: IoT SoA architectural decision fulfilling consumer-driven and technical requirements

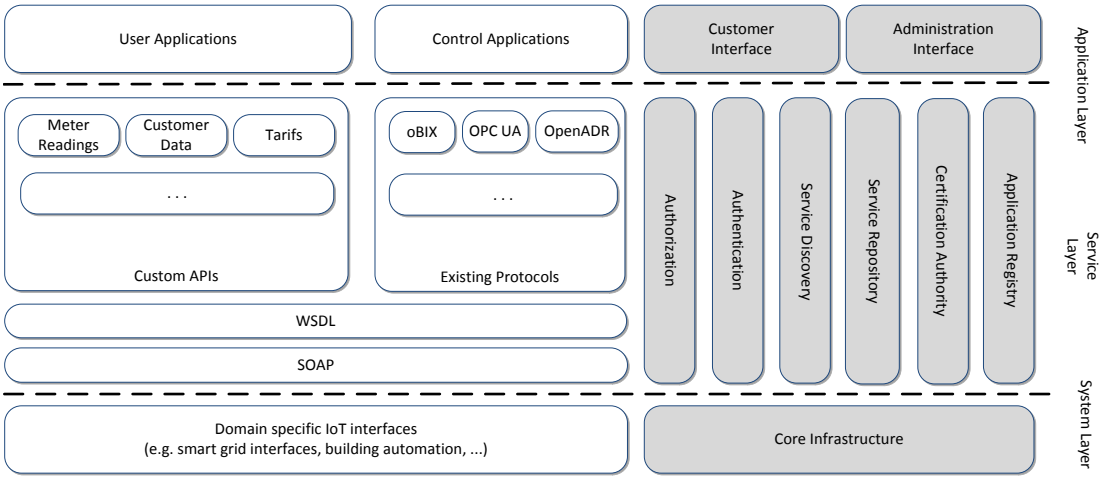


Figure 3.6: Layered functional overview

reasonable choice is to build a SoA based on Web services using the WS-* stack with SOAP and the Web Service Description Language (WSDL) at the core of the protocol stack. Using HTTP for message transfer and XML for message encoding, Web services provide a platform neutral way of providing interfaces to data sources and systems. The selection of SOAP-based Web services directly affects the security and privacy preserving mechanisms. The WS-* stack has a feature-rich technology set for enforcing strong security and privacy requirements. Furthermore, WSDL allows to have well-defined interfaces and type specifications of exchanged messages which is of advantage if it comes to technically specifying access control policies and enforcing them.

Figure 3.6 illustrates a layered functional overview of the architecture. Existing Smart Grid and IT infrastructure are integrated based on SOAP Web service endpoints including a WSDL description. The centralized services that are required to operate the Smart Grid ICT infrastructure are covered within the core infrastructure. On top of the services applications, user interfaces and application interfaces are provided.

For the SOAP service definitions, existing protocol and data representations have been used. For the integration of building automation systems, sensor networks or more general machine-to-machine (M2M) information into a service-oriented architecture based on Web services, OBIX is the approach that is used within this IoT SoA. It offers an XML language for representing information provided by appliances using further enterprise friendly technologies HTTP and URIs. Another goal is to provide a standardized representation for common M2M features like datapoints, histories and alarms and extensibility for custom enhancements. OBIX can be used on embedded devices or on gateway devices that offer access to building automation systems or smart meter infrastructures. For building automation systems, like KNX, BACnet, EnOcean, ZigBee and smart meters using Wireless M-Bus, a gateway concept is presented in Chapter 5.

The service endpoints are either hosted at centralized infrastructures, e.g. the meter data management systems or an enterprise service bus, or in a decentralized manner at a Web ser-

vice gateway at the residential or commercial customer. By using the Internet and IP as network infrastructure, interworking is provided at the network layer and new data sources of stakeholders and application providers can be easily connected and integrated (cf. TR2). This approach allows keeping the data as decentralized as possible and under the control of the data owner (cf. CR2).

A data owner represents a natural person or a legal entity to which several data assets (e.g., smart meter readings) can be linked. A so-called data owner identifier which is a random generated identifier is linked to all data assets and used to register service endpoints at the service repository. It is then possible for an application to lookup the endpoint location of a certain service type for a data owner. Taking smart meter data as an example, for some customers the data might be provided by gateway devices operating at the site of the customer. For other customers, the data might be provided from central data stores like a meter data management system at the energy utility. The data owner identifier is also used within access control policies that specify which application is allowed to access certain data sets linked to the data owner (cf. CR5). Finally, the data owner identifier is also used as pseudo-identifier for a centralized user single-sign on. In this way, a user can authenticate at a third-party application without exposing any further identity information. If an application accesses SOAP endpoints it can link only the data owner identifier to the retrieved data, e.g. smart meter data, which further allows to protect the privacy of a data owner.

3.4 IoT SoA core components and roles

This section provides a closer look on the core components of an IoT SoA and its respective roles. A motivating use case for the discussion of the roles and components is taken out of the Smart Grid application domain. The chosen use case is the sharing of smart meter readings collected in a residential home to a third-party energy advisor providing consulting services.

Within this use case, the data owner, smart meter gateway, energy utility, core platform, service registry, application registry, identity provider, policy administration point and the third-party energy advisor are involved. The details of the roles and components are discussed in the following subsections. In the chosen example, the data owner wants to use an energy advisor application offered through the IoT SoA application platform. Therefore, the local smart meter reading's Web service endpoint needs to be registered at the service registry. The energy advisor needs to register as application provider in order to receive a certificate that can be used for further authentication purposes. The core platform provides an identity provider that can be used for user authentication purposes. The core platform further comes with a policy administration point and policy decision point, which are used by the data owner to configure the access rights to data assets and to check the authorization of a service consumer at request time.

Data owners

A central entity in the IoT SoA concept is the *data owner*. The data owner is a natural person or legal entity where certain information (e.g., smart meter readings) or interfaces (e.g., home

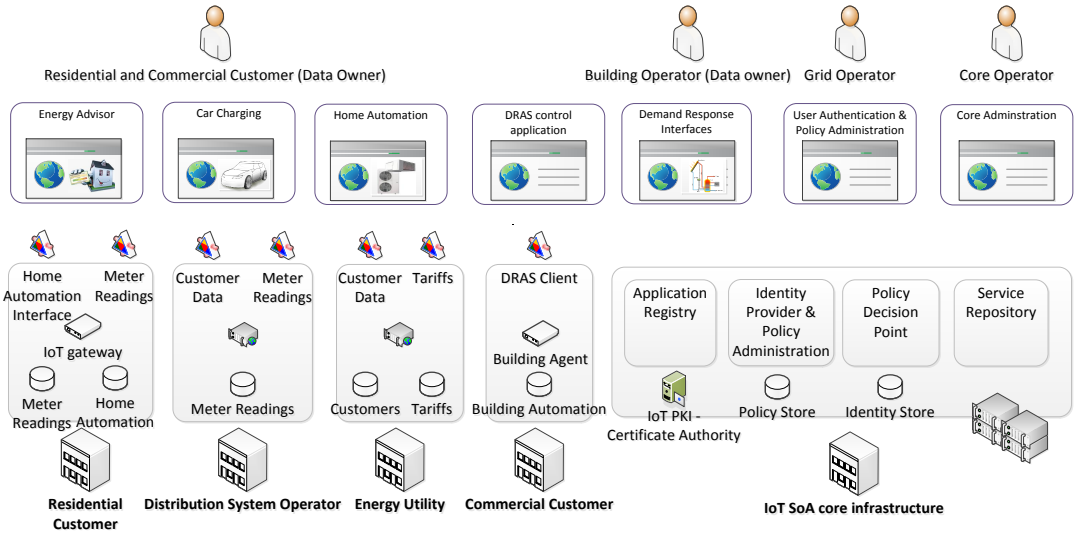


Figure 3.7: Overview of core IoT SoA components

automation) are linked to. Web service endpoints are registered for a data owner. Applications (service consumers) can look up the service endpoint for a certain data owner at runtime. Data owners can specify access policies that define which application is allowed to access data related to the data owner. The data owner concept is important in the scenario where third-party applications are used by a customer. Another scenario within the SoA is the machine-to-machine communication, e.g. based on OpenADR communication.

Identity provider & policy administration point

The identity provider allows the various users of the SoA to authenticate themselves. There are several user roles ranging from the residential customer acting as data owner, to system and grid operators over to SoA administrators. The identity provider is based on an identity store (e.g., Lightweight Directory Access Protocol (LDAP) or relational database) that holds the user credentials. The SoA provides user-name and password authentication as well as user authentication based on signed public keys stored within X.509 certificates, likely to be equipped on a smart card.

The identity provider can be used to establish on a SSO context between the IoT core infrastructure and third-party application providers. In that way, a customer only needs one set of credentials for authentication in the IoT context. Since the user authentication happens at the IoT core infrastructure it is secure to let the data owner administrate access control polices. For this reason, the policy administration point provides a Web-based user interface.

Global and local service discovery

A central component within a SoA is the global service discovery component. It acts as service registry and allows a service provider to register a Web service. Service consumers may use the service registry to discover services and service descriptions at the design time of a system, and furthermore use the registry to look up service endpoints (e.g., URLs) at runtime. For the realization of a service registry, several alternatives are available like UDDI, ebXML or DNS-SD. Service discovery might also be realized in a local way using WS-Discovery, UPnP or Multicast DNS (mDNS). The service repository in the presented architecture is a custom implementation and acts as a lightweight variant of UDDI taking some concepts of WS-Discovery as input. The reason is to align the registry to the specific needs of access control and privacy protection.

Application registry

The application registry is used to register applications, which works in a similar way to the registration of service consumers. Each application provider creates a private and public key pair and let the public key be signed by the core certification authority to receive an X.509 certificate. The X.509 certificate contains the signed public key and can be used for authentication of application requests. These certificates are used for authentication at the system level.

Certification authority

The certification authority is the core of the IoT SoA PKI and provides certificates containing signed public keys to users and systems for the purpose of authentication.

3.5 Access control in the IoT SoA

This section contains a detailed view on how access control is handled by the IoT SOA. To illustrate the interactions and the details of the involved components, the handover of meter readings to a third-party energy advisor application is taken as use case. The access control concept applies SAML and XACML as state of the art technology, but extends these technologies for the specific requirements of this SoA.

Component overview

Figure 3.8 provides an overview of the components involved in the access control mechanism and at which node they are located. In this generalized overview, only a single service provider and service consumer are presented as placeholder for multiple concrete data interfaces and applications. The service provider in this case can be the IoT gateway offering an interface to an energy advisor application provided by a third party.

The Service Provider (SP) offers a SOAP based Web service to access data or an interface to some capabilities. The interface does not need to be adapted for the access control concept. Instead, a generic PEP is used as SOAP intermediary between the Service Consumer (SC) and the SP following the XACML data flow model. The task of this component is to interpret the

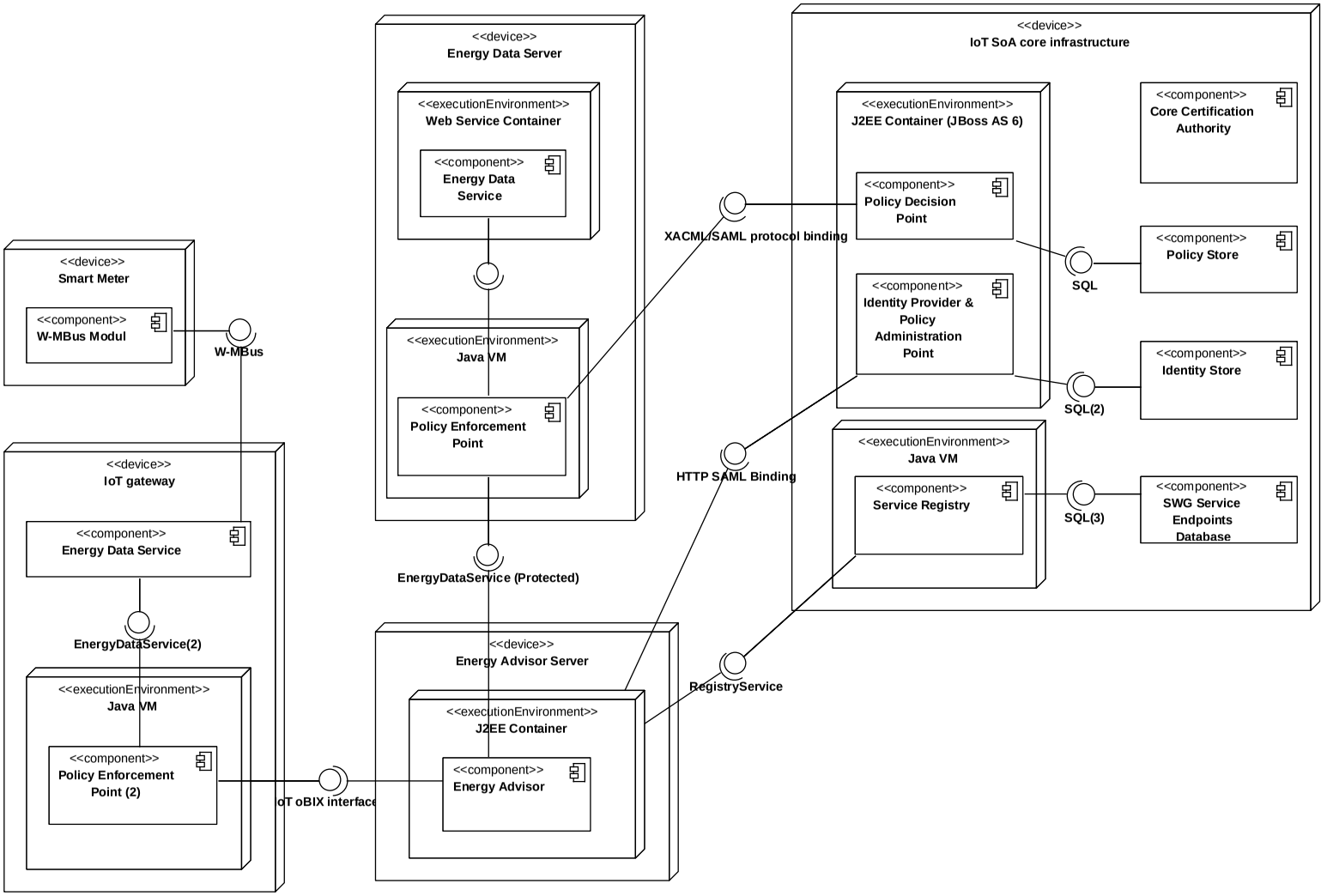


Figure 3.8: IoT SoA access control components

incoming SOAP request and send an authorization decision request to the PDP. The PDP offers an XACML SAML protocol binding based on SOAP for such decision requests. In the backend, it uses a policy store that holds the XACML policies used to evaluate the access control request. The policies are maintained by the data owner using a third party application. The third party application has to be registered at the IoT SoA core infrastructure. The third party creates a public-private key pair and the public key is signed by the core infrastructure in order to create a trust relationship. For authorizing third party service consumers' access, the data owner is redirected to the Identity Provider (IDP) that authenticates at the user level. The IDP also acts as PAP allowing a user to administrate the access control policies. The IDP provides a SSO interface for the third party application provider using the SAML Web Browser SSO Profile with the HTTP redirect protocol binding for the exchange of SAML assertions. The communication interfaces between the various components are either based on HTTP if a human actor is involved. For all other machine to machine communications, SOAP is used. The SOAP interface offered by the service provider is domain specific and does not need to be altered since a generic access control mechanism is employed.

Generic access control concept

XACML policy language: The XACML policy language as outlined in Figure 3.9 provides a powerful way for specifying policies. An XACML policy structure consists of one or many policy sets that can be recursively encapsulated and contain one or multiple policies. Within a policy set, the policy combining algorithm defines the final result of the evaluation. A policy set or a single policy may address a certain target which can be identified through specifying the resource, the subject, the action or the environment. Within a policy, rules can be specified. A rule consists of a target, a condition and an effect. The condition element allows refining the applicability of the rule based on XACML built-in functions. The effect in this use case is simple: either *permit* or *deny*.

Policy structure: To have a generic access control mechanism, the policies are based on the information that is available within a SOAP request without referring to any domain or protocol specific vocabulary. The policy model is structured as follows. The various data sources in the Smart Grid for each data owner represent the resource. The resource identifier has therefore two attributes: firstly, the data owner identifier which is a random-generated UUID pseudo-identifier; secondly, the service name which is based on a standard terminology used within the IoT SoA. The service name could be represented through a URI that identifies a certain SOAP port type. A subject is identified through the available public key information which is represented through a hash identifier based on the used public key. In this way, the access control concept can be kept generic and independent of which public key encryption algorithm is used for signing messages. The action is represented through the according SOAP action that a service consumer wants to call on the Web service. For each data owner identified, a policy set is used that targets the certain data owner identifier. This policy set contains multiple children policy sets, where each policy set targets the available data sources represented through the service name. Then for each application represented through the hash identifier of the public

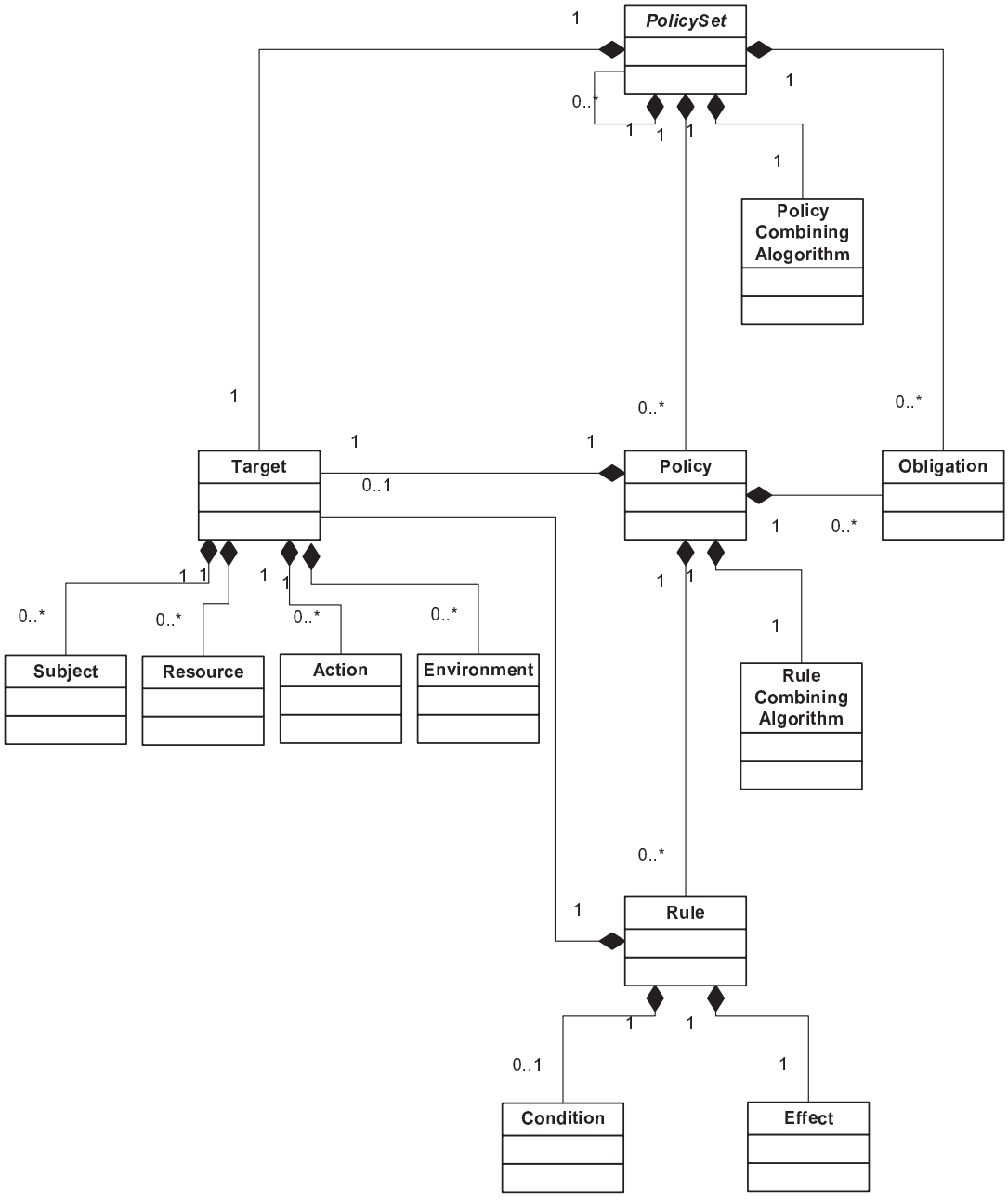


Figure 3.9: XACML language model [106]

key, a policy file resides that contains a rule specifying which SOAP operation is allowed to be used. In this way, a fine grained and generic authorization mechanism based on XACML is possible. For SOAP Web services that are rather generic like for example the OBIX read and write operation the policy structure can be extended to also cover the URI of a certain OBIX object, e.g. the smart meter object or the power history object.

Authorization decision request: The policy enforcement point that resides locally at the Web service endpoint is responsible to create an according XACML authorization decision request and constructs it in the following way. Take as an example the SOAP request provided in Listing 3.1. The request queries the smart meter object as given in Listing 3.2 using the OBIX SOAP Web service. For simplicity and illustration, the example request is not encrypted.

Listing 3.1: Smart meter request

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/" xmlns:sec="http
    ://schemas.xmlsoap.org/soap/security/2000-12">
 <s:Header>
  <sec:Signature>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <ds:SignedInfo>...</ds:SignedInfo>
    <ds:SignatureValue>...</ds:SignatureValue>
    <ds:KeyInfo>
     <ds:KeyValue>...</ds:KeyValue>
    </ds:KeyInfo>
   </ds:Signature>
  </sec:Signature>
 </s:Header>
 <s:Body>
 <read xmlns="http://obix.org/ns/wsdl/1.1"
     href="http://iotgateway/smartmeter" />
</s:Body>
</s:Envelope>
```

Listing 3.2: OBIX object smart meter

```
<obj href="/simMeter" is="iot:SmartMeter">
  <real name="power" href="power" val="125.0"
      unit="obix:units/watt"/>
  <real name="energy" href="energy" val="3215.0"
      unit="obix:units/kilowatthours"/>
  <ref name="power history" href="power/history"
      is="obix:History"/>
  <ref name="energy history" href="energy/history"
      is="obix:History"/>
</obj>
```

The request provides all information to decide if a request is allowed or denied. The subject identifier is represented through generating a hash value of the public key which can be found in the *ds:KeyInfo* field of the WS-Security header. The signature is used to ensure the integrity of the request and that it was created by the owner of the according application that owns the public and private key pair. The resource is identified through the SOAP port type which is in this case *http://obix.org/ns/wsdl/1.1* and the data owner identifier which in this case has to be configured in the OBIX gateway. If a Web service offers data for multiple data owners, it needs to be extracted either out of the SOAP header or of an argument within the called SOAP

operation. The resulting XACML decision request is provided in Listing 3.3 and illustrates how the information is used to generically create an XACML decision request. This request is sent to the policy decision point using the SOAP binding. This request is also secured through a signature and by encrypting the payload.

Listing 3.3: XACML decision request

```
<samlp:RequestAbstract
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:xacml-samlp="urn:oasis:xacml:2.0:saml:protocol:schema:os"
  xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os" ID="ID_a9bd16e8-bfa2-450d
      -b131-2d953adae395"
  Version="2.0"
  IssueInstant="2012-04-23T16:36:50.764+02:00"
  xacml-samlp:InputContextOnly="true"
  xacml-samlp:ReturnContext="true"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.
      org/2001/XMLSchema"
  xsi:type="xacml-samlp:XACMLAuthzDecisionQueryType">
  <Request
    xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
 xmlns:ns2="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
   <Subject xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
SubjectCategory="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject">
     <Attribute xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id" DataType="http://
    www.w3.org/2001/XMLSchema#string" Issuer="smartwebgrid">
       <AttributeValue xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
           PUBLIC_KEY_HASH</AttributeValue>
     </Attribute>
   </Subject>
   <Resource xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
     <Attribute xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id" DataType="http://
    www.w3.org/2001/XMLSchema#string" Issuer="smartwebgrid">
       <AttributeValue xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
           DATAOWNERIDENTIFIER</AttributeValue>
     </Attribute>
     <Attribute xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
AttributeId="urn:tuwien:auto:smartwebgrid:resource:servicetype" DataType="http://
    www.w3.org/2001/XMLSchema#string" Issuer="smartwebgrid">
       <AttributeValue xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">http
           ://obix.org/ns/wsdl/1.1</AttributeValue>
     </Attribute>
     <Attribute xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
AttributeId="urn:tuwien:auto:smartwebgrid:resource:resource-object" DataType="
    http://www.w3.org/2001/XMLSchema#string" Issuer="smartwebgrid">
       <AttributeValue xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">/
           smartmeter</AttributeValue>
     </Attribute>
   </Resource>
   <Action xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
     <Attribute xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
      DataType="http://www.w3.org/2001/XMLSchema#string" Issuer="smartwebgrid">
       <AttributeValue xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">read<
           /AttributeValue>
     </Attribute>
   </Action>
  </Request>
```

```
</samlp:RequestAbstract>
```

Interaction

Figure 3.10 illustrates the interaction between the various stakeholders in the IoT SoA in respect to the access control concept. The first step happens when a user wants to access a Web-based application. The browser of the customer is redirected to the central identity provider where the user is authenticated using either a user-name and password or mutual authentication based on signed public keys (X.509 certificate). On the first usage of an application, the user is redirected to the policy administration point where the access rights of the application on the user data sources can be administrated and transformed from a domain specific terminology into the generic XACML access policy structure. After the user is authenticated an SAML assertion based on the SAML Web single sign-on profile is returned to the browser of the customer including the data owner identifier as identity attribute. The application server receives this SAML assertion and creates an authentication context for the customer. Based on the data owner identifier it queries the service repository for the required Web service endpoints. The application server then issues the according SOAP requests, e.g. a query to the smart meter object offered by the OBIX interface. The SOAP request is encrypted and signed by the application. The PEP acts as a SOAP intermediary in front of the Web service endpoint that provides the interface to the data source. It extracts all information to create an XACML decision request and sends it to the central PDP. The PDP evaluates the policies and decides whether access is permitted or denied and returns the decision. Based on the decision, the original SOAP request is forwarded by the PEP and processed. Otherwise a SOAP fault message is returned to the application server.

3.6 Implementation

Within this section a proof of concept implementation of the IoT SoA is presented which demonstrates the feasibility of the proposed concepts.

The IoT SoA core infrastructure is implemented within a testbed using Java 6 and the JBoss application server. The IDP and PDP are based on the open source JBoss Picketlink framework, but customized for the specific needs of the IoT SoA core. The description of the implementation follows the 4+1 view model proposed by Kruchten [39]. Further, a cloud-based implementation of the PDP is also performed in order to analyze the increased scalability. Therefore, the Appscale PaaS framework is used [111].

The data assets stored by the core database concern information about the user (e.g., *data owner*) including contact information, name, address and organization and a data owner acting as a unique pseudo identifier. The user data is linked to a user role which identifies the user as data owner, service provider or application provider. Further, the database stores a service context which identifies the relationship between a data owner, service provider and service consumer. The service context allows to register certain types of services and further specialize the target scope to certain service providers and consumers. For each service context, a service description holds information about the type of the service, a version and a link to a WSDL

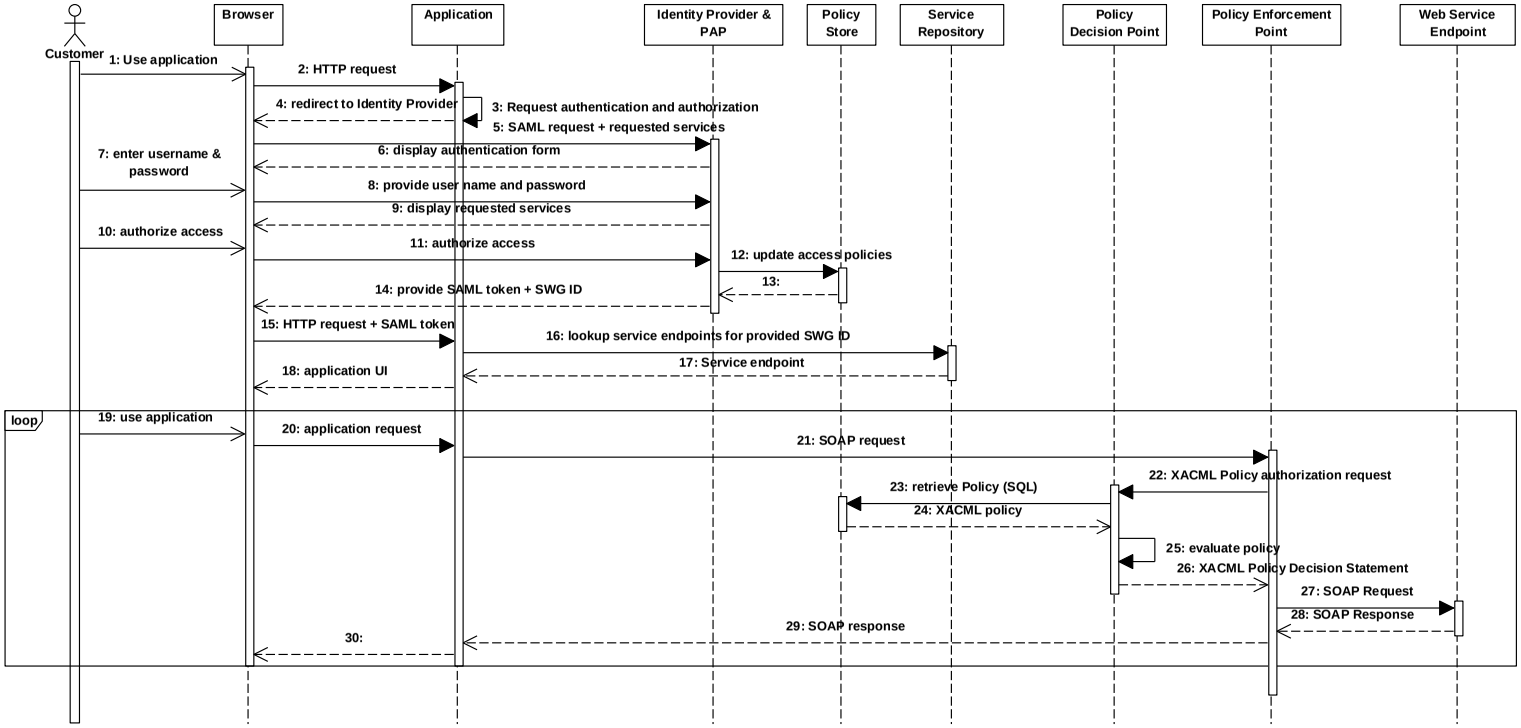


Figure 3.10: Access control interaction

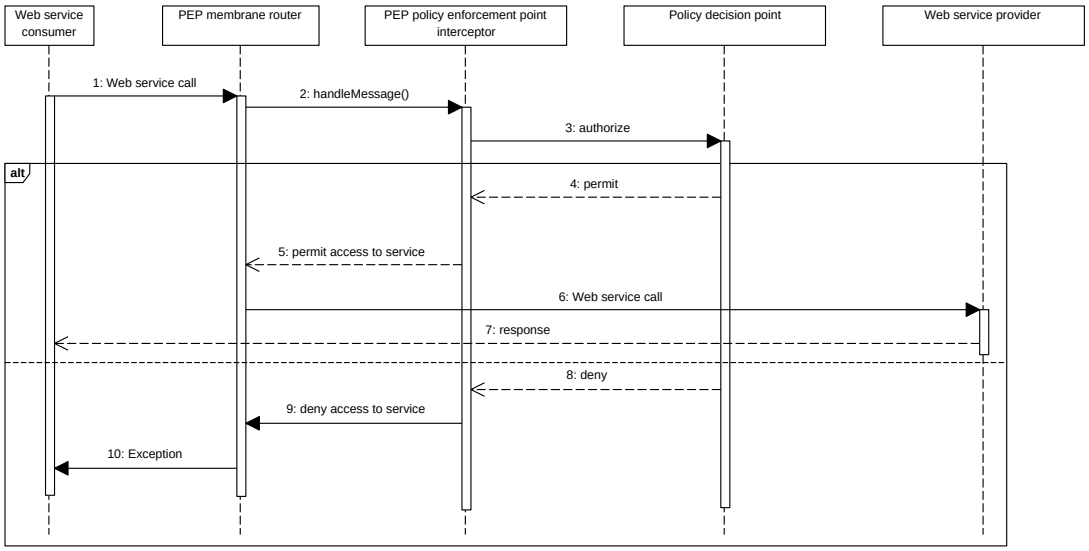


Figure 3.11: Sequence diagram describing PEP‘s mechanism [112]

document describing the interface in a machine-interpretable way. Furthermore, a public key is provided that can be used to send encrypted requests to the service endpoint. Finally, the database holds a policy set for each data owner in order to restrict the access to the provided data sources of the data owner.

For the discovery of services and mediation of the connection phase between a service consumer and service provider, the service registry plays an essential role. The Web service registry provides a single point of information about where the service Uniform Resource Locator (URL) and the WSDL are located. The main purpose of the registry is to offer features for registration, de-registration and lookup of services providing certain capabilities.

The sequence diagram in Figure 3.11 illustrates the behavior of the PEP in more detail. The PEP is another key component of the IoT SoA core and acts as a router between the service provider, service consumer and the policy decision point.

When a request fits the specification, the *dataOwnerId* and the function name get extracted and enveloped in an XACML decision request that is sent with an *authorize()* method to the PDP awaiting the evaluation result. If the requester is permitted to retrieve the Web service‘s return values, the router forwards the request. If the request does not fulfil the requirements, an exception is thrown.

Figure 3.12 shows the abstraction of the policy decision point work flow when an XACML request reaches the service and what happens while processing the request. A speciality of the PDP is that no exceptions reach the outside. Due to the specification, the PDP could reply with four answers.

- Deny

- Permit
- NotApplicable
- Indeterminate

Deny and *Permit* are the answers that are normally expected by the requesters. According to [113], *Not Applicable* is sent when no policy or policy set is applicable to the incoming request. Moreover, *Indeterminate* means that no exact solution for the request could be evaluated for a given policy which often indicates an error in the policy or the request. This means to the requester that no matter what error occurs the access is denied. The message that can be interpreted from the possible error-like states is that either something went wrong or the access gets denied.

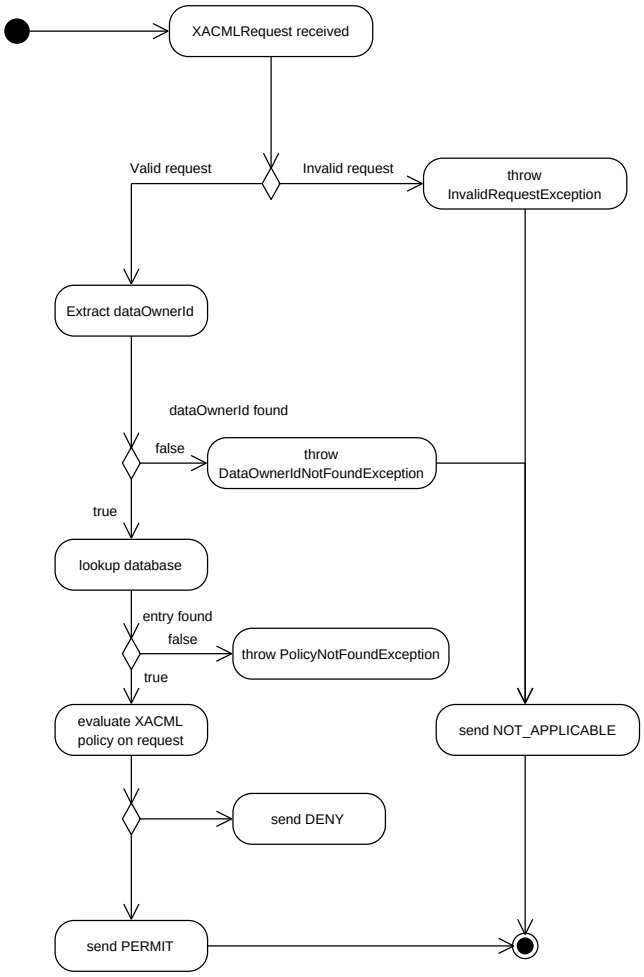


Figure 3.12: Activity diagram of the PDP [112]

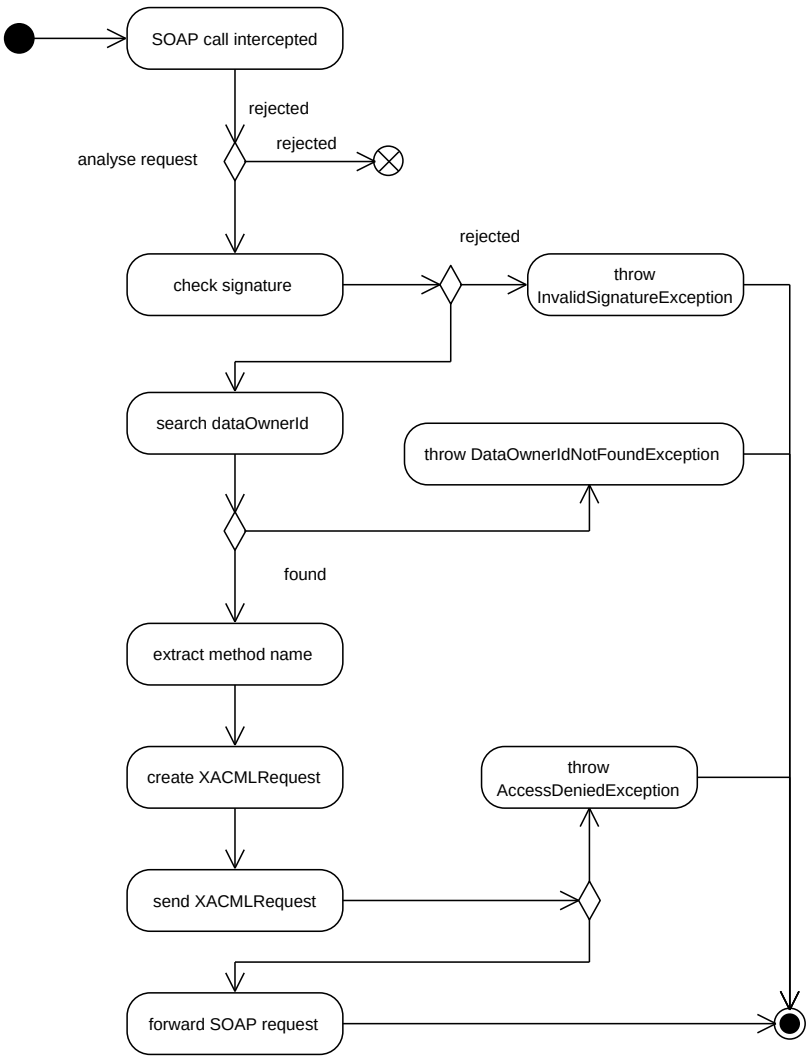


Figure 3.13: Activity diagram of the PEP [112]

Related to Figure 3.11, Figure 3.13 shows the activity diagram of the PEP describing the flow of recognizing an incoming request, validating and processing SOAP messages. A configured listener awaits incoming SOAP messages that comply with the parameters of the configuration. If the request is directed to the service the PEP is sitting in front of, the assigned signature is evaluated. After extracting the *dataOwnerId* and the *method name* an XACML request is created and synchronously sent to the given policy decision point. When the decision request gets permitted the PEP forwards the original request to the original Web service and finishes the processing of this request.

Besides the centralized policy decision point, the policy enforcement point is the interface

to request the PDP. Each service provider has to put the PEP in front of the service.

If the PEP resides on the same device that facilitates the service or if the device is an embedded system equipped with low memory and a weak CPU, the enforcement point shall be put on a gateway that communicates with the service-providing gadget or tool. In the given figure, only the scenario with a smart meter device and a separated PEP is presented. The service registry is used by every provider and client. Therefore, the deployment of the policy enforcement point could be realized with three different scenarios.

1. The policy enforcement point is deployed on a single machine for one service if the underlying service causes heavy traffic.

2. A set of policy enforcement points is deployed on a single machine for a set of corresponding small services with low traffic.

3. The policy enforcement point is installed on the same machine as the protected service if the machine meets the requirements for the expected demands.

Cloud-based implementation

In order to increase the scalability of the central IoT SoA core components, a cloud-based implementation of certain components is performed. Realization of the proof of concept prototype consisted of configuration and deployment of Appscale and the implementation of the PDP and PAP components deployable in Appscale. Code creation was conducted locally with the use of the Google App Engine (GAE) plug-in[2] for Eclipse. This environment enables functional tests to be undertaken locally. A Dell Power EDGE R715 server was set up with Xen hypervisor[3], an open source standard for virtualization. The Appscale virtual machine images were downloaded and configured according to the instructions given on the Appscale homepage.

Implementation of the PDP component

In order to communicate with the outside world, the PDP component publishes a Web service, that implements the interfaces specified in the XACML-SAML profile. The parsed decision requests evaluated against a policy according to the rules defined in the XACML standard. Since there are already existing implementations, that have shown their compliance with the standard, it was decided to port such a component to Appscale. After screening the available implementations, the Picketlink project from the JBoss Community[4] was identified as the most appropriate one. It offers full compliance with the XACML-SAML profile 2.0 and is published as open source project. Most of the Picketlink code could be adopted one-to-one although several components had to be rebuilt to missing support of certain Java API classes which are not available in the GAE. Parsing of the messages had to be adapted to comply with the whitelist of allowed Java classes used by GAE. The mechanism of the whitelist was introduced to prevent the usage of unsafe classes, that could break the sandbox environment of GAE.

[2] https://developers.google.com/appengine/docs/java/tools/eclipse
[3] http://www.xen.org/products/xenhyp.html
[4] http://www.jboss.org/picketlink

During parsing the key attributes *resourceId* and *serviceName* are extracted. Each policy can be identified distinctively by the use of these attributes. Therefore, they are necessary for determining the correct policy for evaluation of the received request later on.

Apart from just porting Picketlink to Appscale the policy lookup mechanism was changed significantly. In its original version, Picketlink bootstraps its PDP with all available policies. These are loaded and kept in memory through the whole runtime of the program. The *PolicyLookup* mechanism was adjusted to conform with the mechanism proposed in the designed architecture. When evaluating a received request, a *PolicyFinder* is responsible to return the right policy. Normally, Picketlink would search its cached policies for the desired policy by matching the targets. In this implementation, the database based PAP is queried for the right policy by supplying the key attributes, that have been extracted from the request. The resulting policy is then returned to the *PolicyFinder*.

Implementation of the PAP component

The PAP component was developed based on the distributed database support of Appscale. For data exchange, a Web service and a Web form were implemented. The *DatastoreService* offered by GAE is used for storing and retrieving the policies. The key attributes as well as the policy in plain text are added as properties for storage of the policy. For retrieving a policy, the key attributes have to be supplied. These are then used to create a query, that returns the appropriate policies. After completing the first round of tests, a second implementation for policy retrieval was created incorporating the *memcache* service. It is used to cache policy data retrieved from the database to save processing time in case the same policy is requested in the near future.

Authorization as a service

The centralized administration, storage and evaluation of policies introduces a bottleneck in the system. A cloud-based deployment of a central component like the XACML PDP and PAP can address the scalability requirements [114]. For the IoT SoA, a deployment of core components based on the PaaS cloud service model is used. Figure 3.14 illustrates the cloud based deployment. An IaaS framework provides a flexible layer of computational resources that can be used by the PaaS framework which provides a load balancer, worker nodes and database nodes. Finally, on top of these frameworks the PDP and PAP reside.

3.7 Evaluation

For the evaluation of the IoT SoA, a scalability analysis of the core components is performed for an on-premise and a cloud-based implementation.

Performance evaluation

Deploying a central component like the IoT SoA core requires high performance and scalable architecture. In order to evaluate the required computational resources, a performance evalu-

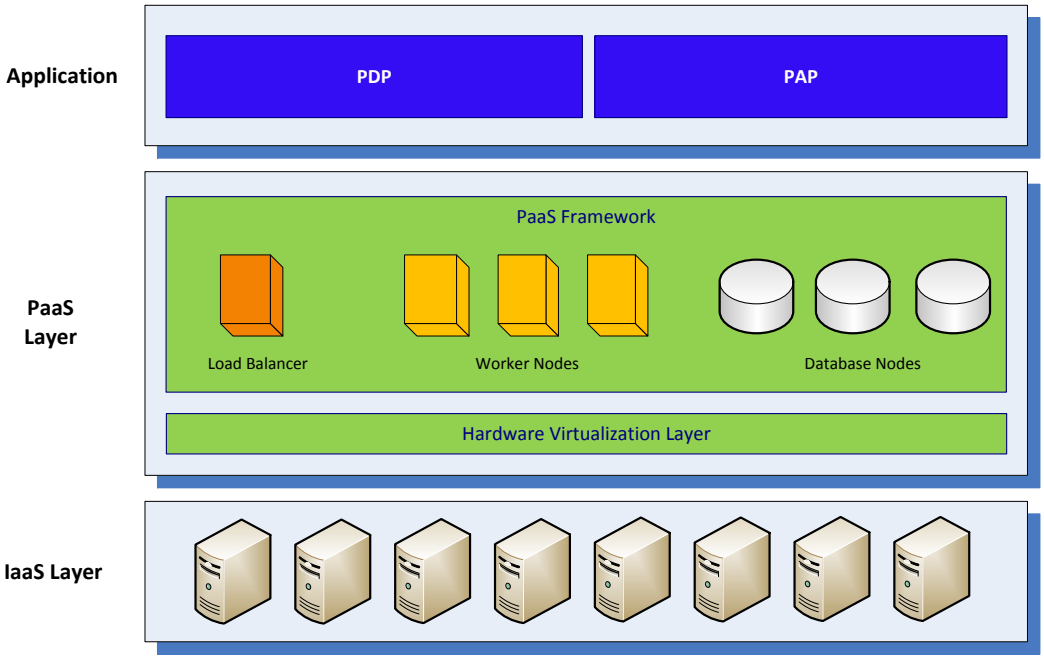


Figure 3.14: Authorization as a Service [114]

ation based on a testbed is done and an analytic Queueing Network (QN) is setup.

Especially the access control and policy evaluation engine is identified as bottleneck in the core infrastructure. For a performance evaluation, a workload and a QN model are required. Figure 3.15 shows the used QN model of the PDP. The model consists of two servers, one application server and one database server. For establishing a QN, the resources of interest and request types need to be identified.

In case of the PDP, this includes the different policy decision requests that might lead to a *Permit*, *Deny* or *Not Applicable* result. Following the operational analysis methodology by Denning [108], the service demands on resources imposed by different request classes can be obtained through a benchmark on a concrete implementation.

Operational analysis

For the operational analysis, different request types are executed against a concrete system implementation for a given time period and based on system monitoring the utilization of resources is measured. Based on the measured utilization, the service demands for one request can be derived.

The following notation is used for this analysis.

- $\mathcal{T}$: observation period
- A_0: total number of requests sent to the system

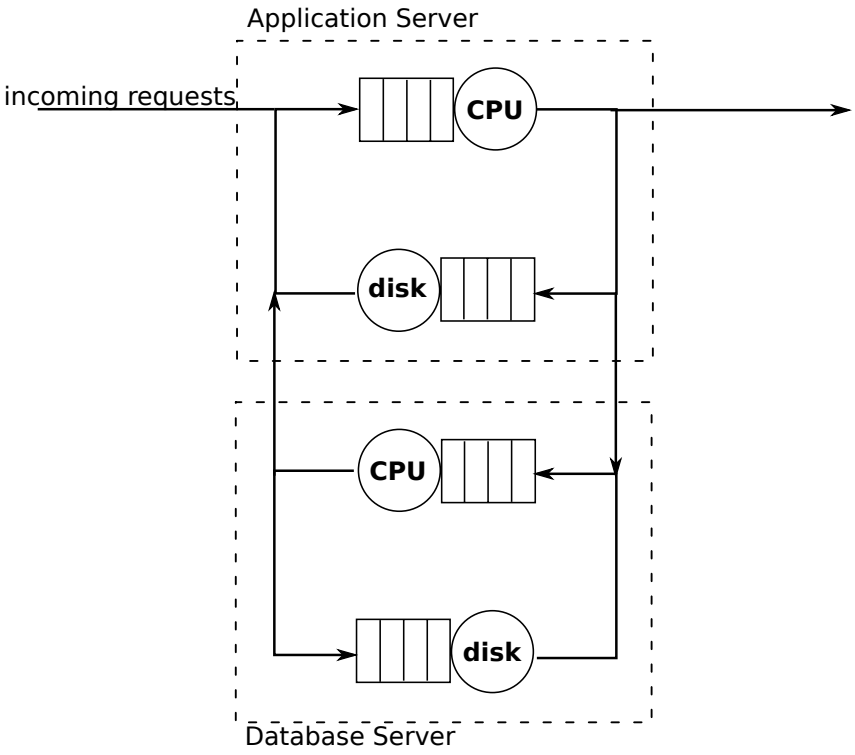


Figure 3.15: Queuing network model

- C_0: total number of requests completed by the system
- ρ: utilization
- λ: average arrival rate
- T: response time

With the *service demand law* the service demand D_i can be calculated based on the system utilization ρ and the throughput X_0.

$$D_i = \frac{\rho_i}{X_0} \tag{3.2}$$

$$X_i = \frac{C_i}{\mathcal{T}} \tag{3.3}$$

$$D_i = \frac{\mathcal{T} \times \rho_i}{C_i} \tag{3.4}$$

For performing this analysis, the core infrastructure is deployed on a virtualized server using a 64-bit Linux 3.0.0 operating system with 4 GB of RAM and a 2 GHz AMD Opteron 6128

	Permit	Deny	Not Applicable
CPU	0.1070	0.1062	0.1034

Table 3.4: Service demands in CPU seconds of the three classes for a CPU at the application server requesting the PDP.

CPU core. A representative set of 100,000 policies is used for the operational analysis. Table 3.4 lists the service demands identified within the analysis. The demands on the database server are negligible compared to the effort for processing the XML-based policies at the application server. Further, it can be seen that the initial assumption that the different types of policy evaluation results lead to different service demands does not hold true and a quite similar service demand can be measured.

Analytic model

For a QN model the utilization the average response time can be calculated based on Equation 3.5 using the measured service demands. The service demands D_i can also be used to calculate the device utilization U_i for a given request arrival rate (cf. Equation 3.6).

$$R_r = \sum_{i=1}^{K} \frac{D_{i,r}}{1 - U_i} \tag{3.5}$$

$$U_i = \sum_{r=1}^{R} \lambda * D_{i,r} \tag{3.6}$$

$$\tag{3.7}$$

The results in Figure 3.16 present the calculated response times for different resource configurations. A simple approximation is done for multi-core setups. Instead of adding more resources and more queues to the model, the service demands are simply divided depending on the number of cores to reflect the impact of additional computational resources.

Benchmark

With a benchmark, the validity of the used analytic model is tested. Therefore, the impact of increasing load on the average response time is investigated for different computational resource scenarios. Multiple instances of the JMeter benchmarking tool are used to execute the requests. The setup deployment is shown in Figure 3.17. Randomized requests leading to different policy evaluation results are sent to the PDP.

The results are given in Figure 3.18. In the figure, the benchmark results are compared to the calculated expected response times. It can be seen the analytic model provides a good estimation for the scalability of the system.

These results allow to state a lower bound on the required computational resources depending on the expected workload of the access control mechanism.

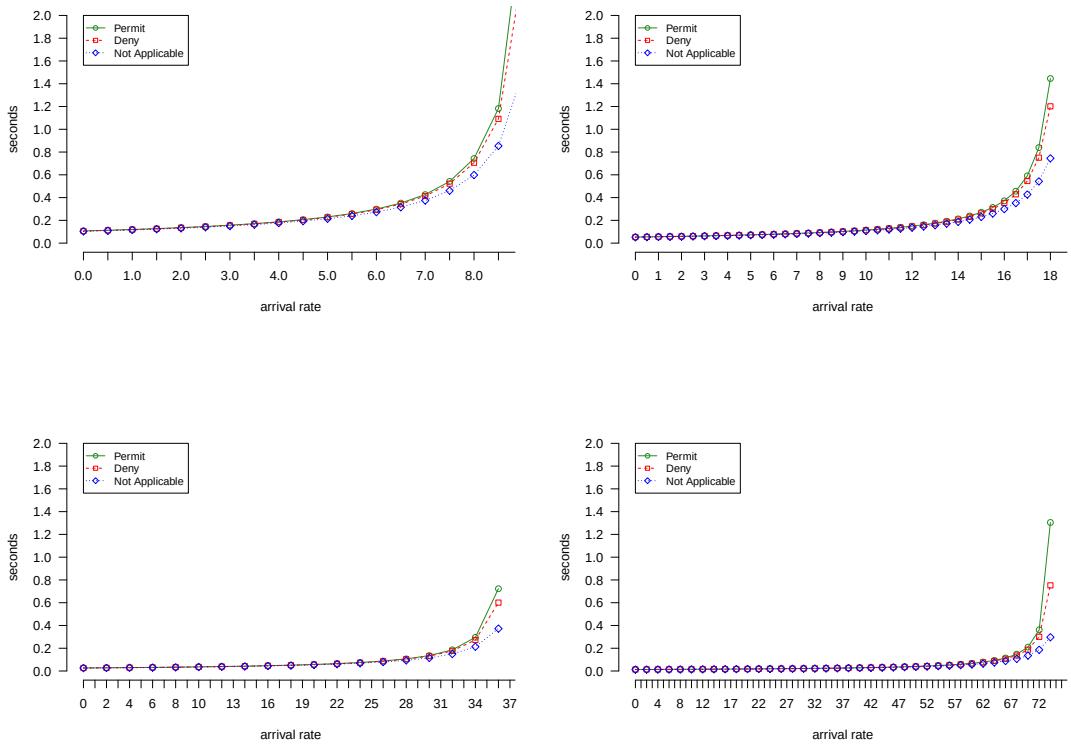


Figure 3.16: Calculated response time based on the QN model for different computational resources [112]

Cloud-based components evaluation

For evaluation of the cloud-based implementation, a setup of Appscale was used together with a JMeter client sending continuous requests. During a test routine, this machine communicates with the Appscale application and executes a defined testplan. Appscale was executed on an Xen server capable of running multiple virtual machines. Each database offered by Appscale, except MySQL Cluster, was tested with multiple Appscale deployments ranging from one node up to a maximum of four nodes. A MySQL Cluster configuration requires the mode of replication to be divisible by the number of nodes (e.g., with six nodes, two or three times replication). Therefore, a deployment for MySQL Cluster in a four nodes' Appscale setup was not possible and test runs were only conducted for one, two and three nodes. Table 3.5 lists the details of the used test setup.

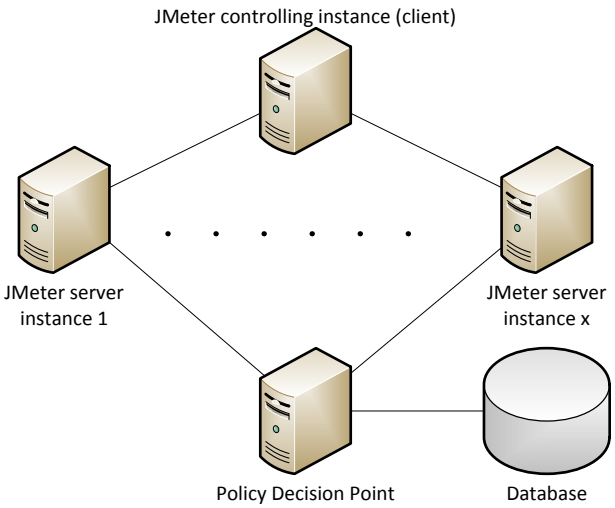


Figure 3.17: Setup of the benchmarking environment.

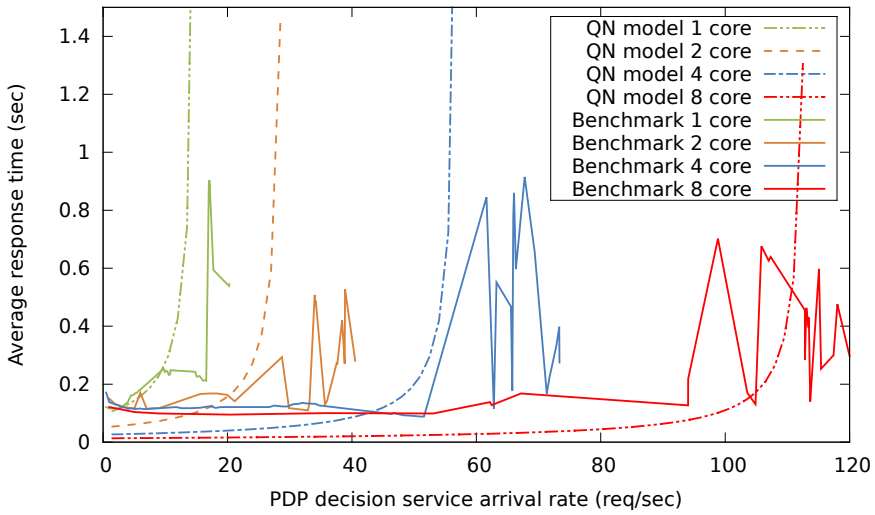


Figure 3.18: Comparing analytic calculations with benchmark results

Closed workload test

The closed workload test was designed to issue decision requests to the PDP interface of the access control system. The requests are created based on a CSV file containing 10.000 randomly generated unique IDs. The test runs alternately send requests that shall result in a *Permit* and a *Deny* decision. The load induced by the tests is controlled through the amount of threads

Xen server	Dell Power EDGE R715 16 x 2GHz AMD Opteron 6128 CPUs 2 processor sockets, 8 cores per socket 16374, 14 MB RAM Linux (GNU/Linux 2.6.32-5-xen-amd64 x86_64)
Appscale instances	2 vCPUs 3500 MB RAM Ubuntu 10.04.3 (GNU/Linux 2.6.32-305-ec2 x86_64)
JMeter server	2 vCPUs 2048 MB RAM Ubuntu 11.10 (GNU/Linux 3.0.0-12-generic x86_64)
On-premise server	2 vCPUs 2048 MB RAM Ubuntu 12.04.1 (GNU/Linux 3.2.0-32-generic x86_64)

Table 3.5: Cloud hardware test setup

issuing requests. The count of threads is increased by one every five minutes until a maximum count of 36 is reached. The maximum load is maintained for another 5 minutes until the test run is finished. Because of the closed workload characteristic of this testplan the number of requests issued to the system is partly controlled by the response times generated by the system under test.

Table 3.6 displays the results gathered from the test runs. When comparing the test runs of a one node Appscale deployment with multi-node deployments, a clear decrease of performance can be observed. This behavior can be explained by the fact, that a one node setup does not have to process any load balancing tasks. All Appscale components are placed on one node and there is no need for complex supervision logic. Hypertable and Cassandra showed the appearance of "Service Unavailable" responses. These are triggered as soon as a request is queued and hits the mark of 60 seconds. MySQL Cluster showed the best performance from all databases tested with Appscale. The load created by the test runs did not lead to the appearance of "Service Unavailable" responses as observed with the other databases. The increase in performance from two to three nodes' MySQL Cluster deployments already results in a higher performance than in a one node deployment. The replication tasks performed by MySQL Cluster are seemingly not as costly as the ones performed by the other databases. With MySQL Cluster, Appscale seems to deliver the scalability at a far better ratio than with the other databases tested.

After changing the target of the testplans the same test runs were executed against the on-premises solution developed for comparison. The maximum capacity of the on-premises system yields to about 630 positive responses per minute. In these test runs, no errors of any

Node(s)	Hypertable	Cassandra	MySQL Cluster	On-Premises
1	22 tpm	44 tpm	185 tpm	630 tpm
2	5 tpm	19 tpm	178 tpm	-
3	11 tpm	22 tpm	325 tpm	-
4	16 tpm	27 tpm	-	-

Table 3.6: Estimated maximum transactions per minute for the tested databases

type were recorded.

Open workload test

After analysis of the closed workload test, an open workload test was configured. This test induced a constant load on the system for ten minutes to produce a desired arrival rate. After each of these phases, a cool down period of five minutes followed. For the next load phases, the desired arrival rate was increased in an interval step size of 0.25. This test was executed against the system with 10.000 respectively 30.000 stored policies.

Figure 3.19 illustrates the comparison between an Appscale deployment concerning one, two and three nodes with an on-premises solution. The upper diagram is based on 10.000 stored policies whereas the lower diagram is based on 30.000 stored policies. All graphs show a slow increase in the beginning that turns into an exponential growth.

The upper boundary of the graph for 10.000 policies is reached at the following numbers:

- 206 tpm (transactions per minute) for Appscale with one node
- 203 tpm (transactions per minute) for Appscale with two nodes
- 364 tpm (transactions per minute) for Appscale with three nodes
- 706 tpm (transactions per minute) for the on-premises solution

The upper boundary of the curves for 30.000 policies is reached at the following numbers:

- 155 tpm (transactions per minute) for Appscale with one node
- 70 tpm (transactions per minute) for Appscale with two nodes
- 230 tpm (transactions per minute) for Appscale with three nodes
- 75 tpm (transactions per minute) for the on-premises solution

Processing times at the application server

In order to gain more knowledge about the numbers received in the conducted tests, the actual processing times at the *AppServer* component were analyzed. For this purpose, statements logging a timestamp were placed at relevant points in the program code. This approach enabled the measurement of the time needed to execute the Java code in order to process a request. To further highlight the differences between the databases used, the timespan for querying the database was investigated. In Table 3.7, the aggregated data for the databases MySQL Cluster and Cassandra are shown. Two major differences are clearly visible when comparing the numbers of MySQL Cluster with the numbers of Cassandra. The processing times logged for MySQL Cluster are nearly identical in each of the configurations, whereas the processing times for Cassandra are increasing significantly if more than one node is used in the Appscale deployment. More importantly, the times measured for database access are below one second for MySQL Cluster in contrast to the times measured for Cassandra that are ranging from seven to fourteen seconds. These high numbers are leading to the bad performance results that have been observed in all of the Cassandra test runs compared to the test runs for MySQL Cluster.

Node(s)	MySQL Cluster DB Access	MySQL Cluster Full Request	Cassandra DB Access	Cassandra Full Request
1	0,2 sec	0,96 sec	7,26 sec	8,13 sec
2	0,32 sec	0,99 sec	8,67 sec	9,56 sec
3	0,23 sec	0,99 sec	14,66 sec	15,85 sec
4	-	-	14,27sec	15,44sec

Table 3.7: Processing times measured at the AppServer

Considering the time for database access plays such a crucial role in the gathered results, a PAP component using *memcache* for caching purposes was analyzed as well. To test the caching mechanism, the initialization phase of the test runs was extended to load 100 policies into the cache. Since a *memcache* service is instantiated for each AppServer instance, the test repeated requesting the same policy until each instance had cached the policy. The actual test workload was limited to requests concerning policies that were already loaded into the cache. The tests were conducted with a Cassandra deployment. The results are shown in Table 3.8. Accessing the cached policies is only taking three milliseconds. This performance increase is decreasing the overall processing time for all deployments.

Node(s)	Cached DB Access	Cached Full Request
1	0,003 sec	0,75 sec
2	0,003 sec	0,69 sec
3	0,003 sec	0,67 sec
4	0,003 sec	0,65 sec

Table 3.8: Processing times with caching mechanism

Conclusion

This section provided an evaluation of the proposed IoT SoA that can be used for application use cases such as a Smart Grid scenario. It has been shown how an architecture can be created that provides a platform which mediates the interaction between different partners and stakeholders in an IoT SoA in a secured, trustworthy and interoperable way. A focus is put on a generic access control mechanism that protects the data access on the arbitrary data sources and services to be found within an IoT SoA. The proposed access control mechanism is analyzed regarding its scalability using an analytic model and experimental evaluation based on proof of concept implementations.

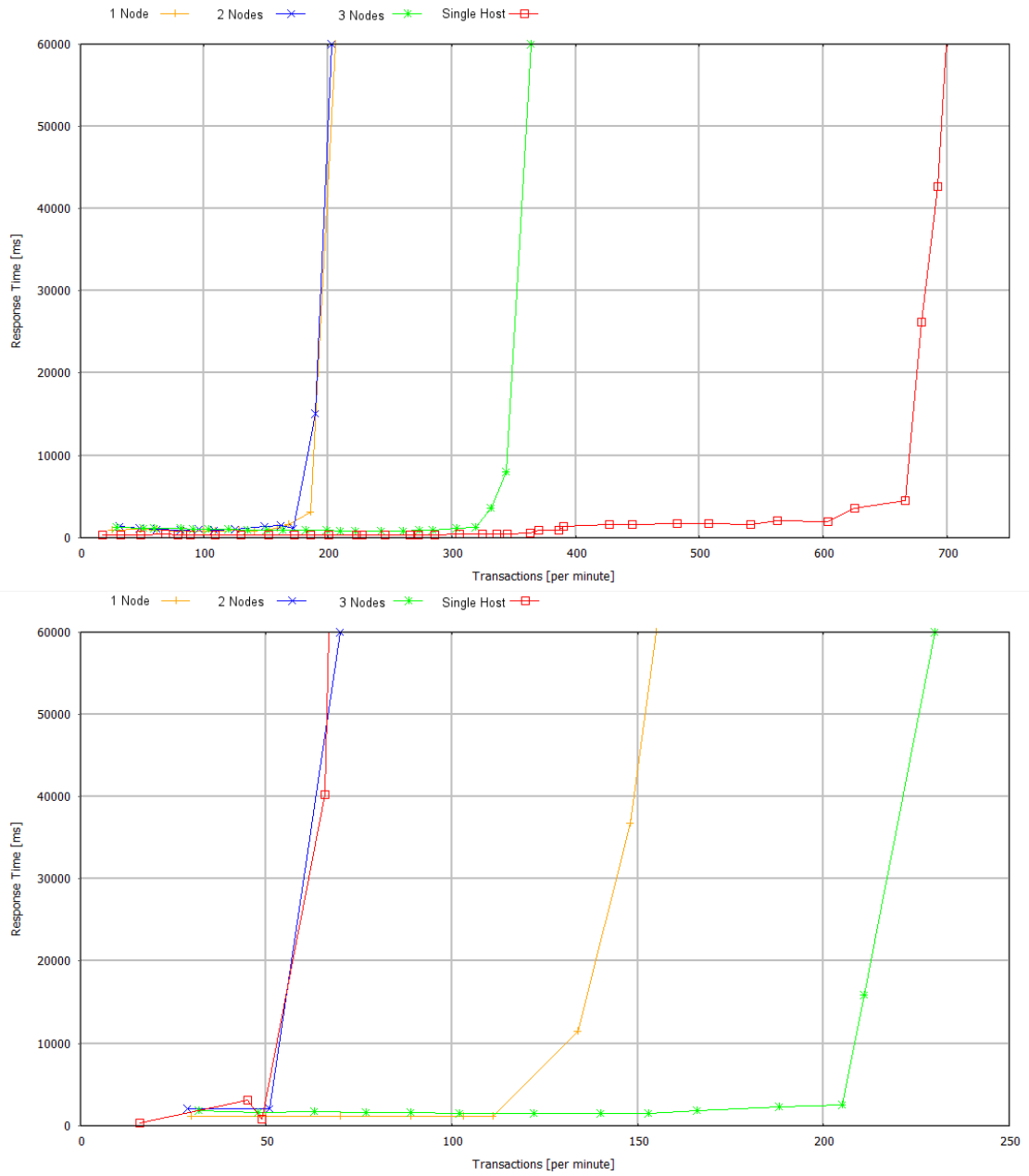


Figure 3.19: Response time with increasing transactions per minute for 10.000 and 30.000 policies

CHAPTER 4

An IoT communication stack

This chapter covers an IoT communication stack that provides interoperability amongst smart objects.

Further, the requirements on a system for creating easy control logic without the need for developing specific applications or control logic scripts avoiding the use of a centralized controller are defined. A set of communication protocols and interaction principles that enable an efficient message exchange for sensors and actuators within a WSAN are identified. It is shown how the requirements for a scripting-free distributed control logic execution can be reached.

The main contribution is an IoT communication stack. It reuses existing standards and technologies such as 6LoWPAN, CoAP and OBIX. The main contributions are the definition of a standardized set of object definition based on OBIX for the IoT, a peer-to-peer communication model resting upon IPv6 multicasting and the concept of a Web-based engineering tool that provides means to create sophisticated control logic through a graphical programming way. Further, a proof of concept implementation is presented and by simulation the scalability, the performance and energy-consumption impact of the communication stack are analyzed.

A *stack* might refer to the set of layered communication protocols as well as to a concrete implementation as a piece of software as stated in the following definition.

> "The smart object software must implement the communication protocols used by the smart objects. Because these protocols are designed in a layered style, where each layer is stacked on top of each other, communication protocols are typically known as a *stack*. The software that implements the protocol is also called a stack."
>
> *[22]*

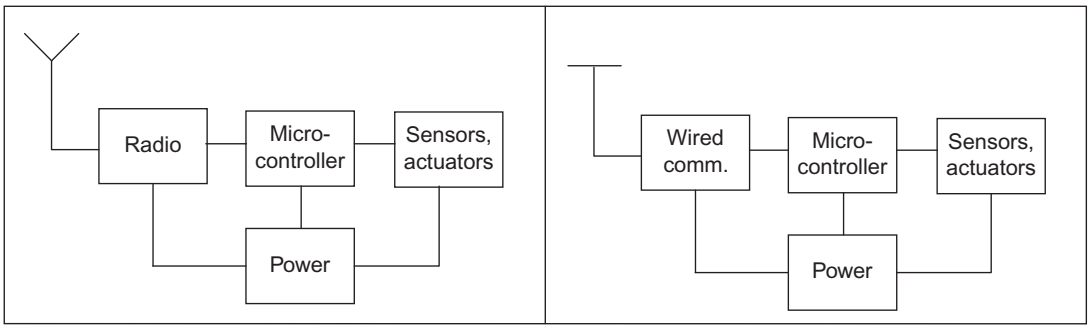


Figure 4.1: Smart object hardware architecture [22]

4.1 Smart objects

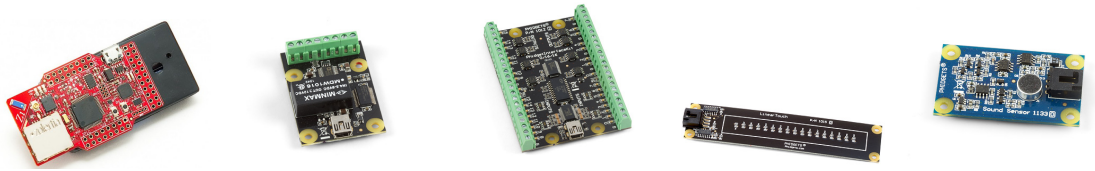
Within this section the definition of a smart object is clarified and also an impression of the hardware category is given that constrains the capabilities of an IoT stack and its implementation. The following technical definition can be used to describe smart objects.

> "A smart object is an item equipped with a form of sensor or actuator, a tiny microprocessor, a communication device, and a power source. The sensor or actuator gives the smart object the ability to interact with the physical world. The microprocessor enables the smart object to transform the data captured from the sensors, albeit at a limited speed and at limited complexity. The communication device enables the smart object to communicate its sensor readings to the outside world and receive input from other smart objects. The power source provides the electrical energy for the smart object to do its work. For smart objects, size matters. They are significantly smaller than both laptops and cell phones. For smart objects to be embedded in everyday objects, their physical size cannot exceed a few cubic centimetres."
>
> *[22]*

The exact meaning of smart objects depends on the concrete application scenario and the behavior of a smart object. A smart object within transportation or logistics might look very differently from a smart object deployed within smart cities or building automation. What is common to all these objects is i) the interaction with the physical world, and ii) the communication capabilities. A typical hardware architecture for the design of smart objects is given in Figure 4.1. The hardware consists of a power source, a micro-controller, sensors or actuators, and wired or wireless communication modules.

An example platform following this architecture is the Zolertia Z1 platform (cf. Figure 4.2a), used for proof of concept implementations within this thesis. The platform provides a micro-controller, radio and an on-board temperature sensor and accelerometer. Further, it provides additional bus interfaces to connect further IO-modules as illustrated in Figure 5.22. Such I/O

modules range over a variety of sensors and actuators, customizable for a concrete application scenario.



(a) Zolertia Z1 platform (b) 4 Analog Outputs (c) 16 Digital Inputs (d) Touch element (e) Sound sensor

Figure 4.2: Example I/O modules from www.phidgets.com

The most common micro-controllers used for smart objects are listed in Table 4.1. The list includes just a few examples and is quite likely to change over time.

Name	Manufacturer	RAM (kB)	ROM (kB)	Current consumption (active/sleep) mA
MSP430xF168	Texas Instruments	10	48	2/0.001
AVR ATmega128	Atmel	8	128	8/0.02
8051	Intel	0.5	32	30/0.005
PIC18	Microchip	4	128	2.2/0.001

Table 4.1: Microcontroller in smart objects [22]

4.2 Requirements

The aim of the IoT stack is to provide a common communication interface amongst all technologies found in the Internet of Things environment. This section presents requirements on an IoT stack and an overview of the stack and how the identified requirements are addressed.

Datapoint-centric information representation

The *information modeling* facilities of an automation system define how devices are represented. This includes concepts to represent entities, their capabilities and relations and further defines low-level means to define data types and their representation. *Datapoint*-centric information modeling can be seen as an approach that relies on simple base input and output datapoints that are offered by devices or functional elements. All the device interaction mainly relies on reads and writes on these datapoints which change the current state of the underlying I/O signal or memory value. In this way, the device interaction can be kept *stateless*. Each operation is idempotent and self-contained. A different approach would be an *object-oriented* or *operation-oriented* where the capabilities of devices are expressed using a more functional orientation. In such systems, commands are exchanged between communication partners and usually APIs provide a defined way to interact with the devices. The operation-oriented modeling of device interface tends to introduce a *state* between different calls. *Stateful* interfaces

increase the complexity and performance requirements since a session needs to be maintained between the client and a server. Furthermore, an *object-oriented* approach can also lead to a mixture in which devices are represented using basic properties as datapoints and more sophisticated functionality as operation. However, to support interoperability and to provide a scripting-free way of specifying control logic, a datapoint based interaction is required (R1).

Generic base information model

For a system to support interoperability, a standard base information model needs to be defined that specifies the meta-model used to represent entities (R2). The most important aspect is to have a set of common data types that specify the format and encoding of information datapoints, since all the process communication between different devices will rely upon this specification. Furthermore, this leads to the convenient situation that only the definition of the base data types needs to be standardized to allow interworking between different devices. It enables the opportunity to create generic user interfaces by providing input elements aligned to the available base data types. Optionally, it is desirable to have standardized means to express the capabilities of entities like devices and to standardize the set of input and output datapoints offered. This can be done, for example, in a domain-specific way and would ease the creation of domain specific applications.

Simple application layer communication services

The communication services at the application layer need to be standardized and fixed. For datapoint centric systems, this requires the definition of services to read and write a datapoint (R3). The exchanged payload for this service only carries values encoded to the basic data types defined in the generic base information model. For building control logic, these services are sufficient and allow wiring output and input datapoints together into one communication relationship. The communication model here follows the semantics that if the state of an output datapoint is changed the wired input datapoints are also updated. The whole functionality is encapsulated within the communicating entities which are either representing physical devices or virtual entities that represent functionality solely. A system might provide more sophisticated services, which can be supported by a graphical control engineering tool. But if these services are extensible, then a control editor must support some kind of scripting for the engineering process.

Group communication interaction model

A group communication interaction model is essential for realizing distributed control logic that avoids the use of a central controller (R4). Aligned to the datapoint semantics, a communication group consists of a set of datapoints that share the same group communication identifier. If a datapoint is changed, e.g. through a physical sensor signal, it transmits the update to all other datapoints in the group based on the simple write service with the datapoint payload on a group address identifier. The complete control logic in such a system can be based on the group tables that map the group address to datapoints. Group tables are directly main-

tained on involved devices. No centralized controller with further application logic is required. Although it would be possible to realize such interaction model also with unicast-based messaging it would be more complicated to maintain the according addressing information and keep it synchronized. The communication effort is significant if a high number of entities participate within a single group.

Logic and virtual entities

Logic entities are essential in a datapoint centric system. More sophisticated application logic is contained in these entities but the interaction with them is based solely on reads and writes on input or output datapoints (R5). Logic entities may reside on centralized controllers but can also be deployed on constrained physical devices. Virtual entities are quite similar to logic entities and can represent interfaces to other systems.

Communication principles

Within RFC 6568, the design and application spaces for 6LoWPAN are defined. Similarly, RFC 6606 provides a problem statement and requirements for 6LoWPAN routing. The use cases taken in these RFCs can be used to define some generalized requirements for the required communication principles of an IoT stack. The analyzed use cases are industrial monitoring, structural monitoring, connected home, healthcare, vehicle telematics and agricultural monitoring, which all come with specific characteristics regarding the required communication principles. In general, a communication stack needs to support point-to-multipoint, point-to-point, and multipoint-to-point interaction (R6). The point-to-multipoint use case is especially for building automation use case of relevance, whereas the multipoint-to-point interaction is required for use cases that involve the data gathering of a large number of sensors towards a single data sink.

Ease of use

State of the art automation usually requires the installation of dedicated software applications in order to commission the devices and to engineer control logic. Typically, this type of software comes with high licensing costs and might impose complicated installation steps. Further, the software usually only runs on a certain platform and operating system making it hard to migrate to other systems. Therefore, it is desirable to have a Web-based application for configuring devices and the creation of control logic (R7).

Energy and memory efficiency

Since battery-operated and constrained nodes might be in use, energy and memory efficiency are important aspects. At the one side this puts constraints on the computational resources in use (CPU, memory), and on the other hand on the design of the communication protocols. The main requirement here is to optimally design the communication stack regarding stack implementation and memory size (R8) and to use radio-duty cycling mechanisms in order to

keep the radio of the node switched off most of the time. The right balance between energy efficiency and achievable quality-of-service needs to be found and depends on the concrete application domain.

Scalability

A communication stack for the IoT needs to be capable of supporting billions of devices. Scalability is a crucial requirement that needs to be supported by the system architecture and the communication stack. This includes features such as a sufficient large addressing space (R9) and an efficient communication and architecture that is able to provide a certain quality of service if a large numbers of nodes are added under the assumption that enough computational resources are available.

Security

The IoT reaches out to various application domains including private homes and critical system infrastructure. Security is an essential aspect that needs to be considered. Appliances need to be protected from unauthorized external access in order prevent fraud or critical damages (R10).

4.3 Stack overview

Smart objects shall directly provide communication interfaces based on this stack. For existing technologies, a gateway can offer a similar interface. Figure 4.3 illustrates the horizontal layers of the stack and the vertical services offered by the stack. The stack includes the capabilities for service discovery of smart objects, service description of the provided communication interfaces, a generic and homogeneous data access service for reading and writing object properties and querying time series information. Finally, security needs to be taken into consideration for the overall communication concept.

The following sections will describe the details of the horizontal layers and the provided functionality. Section 4.4, Section 4.5 and Section 4.7 are intended to provide an overview of the constraints imposed through the data-link layer, the details of the required features of IPv6 and an overview of message exchange protocols and information encoding technologies. The main contributions can be found in the following subsections describing the OBIX-based information models, the peer-to-peer interaction model and the graphical scripting-less control logic editor concept. Furthermore, the overall composition can be considered as contribution. Table 4.2 summarizes the requirements on the system stack and how they are addressed by the proposed IoT stack.

4.4 Media and data link

For IoT devices as long as IPv6 communication is provided any type of link can be used. Most prominent are technologies such as IEEE 802.15.4, Bluetooth Low Energy and WiFi. The main

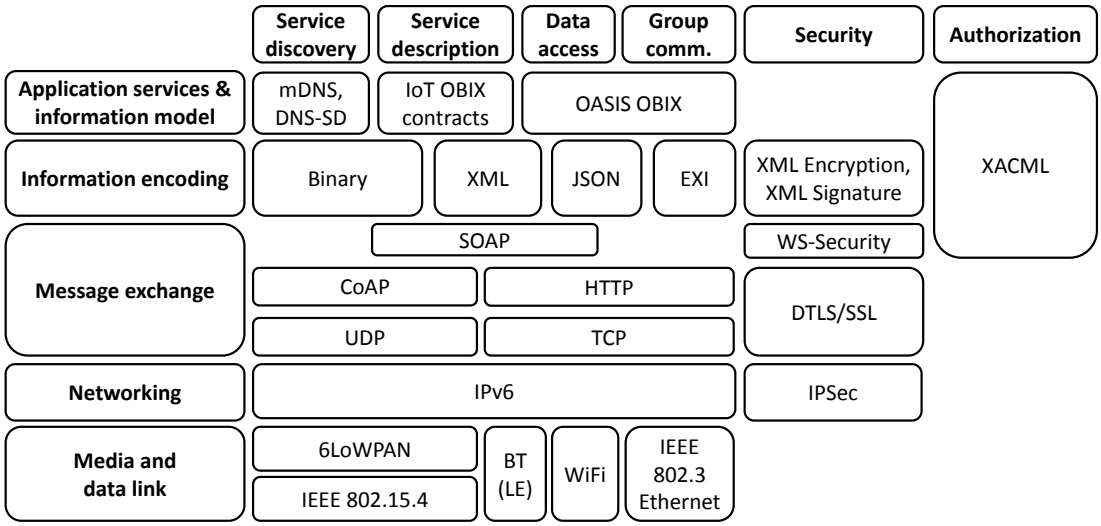


Figure 4.3: IoT stack overview

IoT stack requirement	Design decision
R1 Interoperability Datapoint centric information representation R2 Generic base information model R3 Standardized application layer services	OASIS OBIX CoAP message exchange (REST)
R4 Group communication interaction model	CoAP and IPv6 multicast
R5 Logic and virtual entities	OBIX contracts for logic blocks
R6 Interaction styles Point-to-point Point-to-multipoint Multipoint-to-point	IPv6 unicast, IPv6 multicast
R7 Ease of use	Web based user interface and commissioning tool, graphical control logic editor
R8 Energy efficiency	Radio-duty cycling Optimized message encoding Optimized application layer
R9 Scalability and addressing space	IPv6 networking, decentralized control logic
R10 Security	AES encryption, XACML-based authorization

Table 4.2: IoT stack design decision fulfilling requirements

Technology	Range	Speed	Power Use	Cost
WiFi	100m	nn Mbit/s	high	$$$
Bluetooth	10-100m	n Mbit/s	medium	$$
802.15.4	10-100m	0.n Mbit/s	low	$

Table 4.3: Comparing wireless link layer technologies

difference between these technologies is the energy consumption, range, throughput and costs for communication components as summarized in Table 4.3.

For the IoT stack, IEEE 802.15.4 is further considered. IEEE 802.15.4 is a standard for low-power, low-data-rate and low-cost wireless sensor networks. It specifies the physical and medium access control layer. Several higher layer standards such as ZigBee, ISA100a, WirelessHART and 6LoWPAN rely on it for the physical and data access layer. The maximum packet size in IEEE 802.15.4 is 127 bytes. Depending on the MAC header only 86 or 116 bytes are available for the upper layer protocols depending on the MAC security options. This limit imposes strong constraints of the higher layer of the IoT stack that follow in the next subsections.

Regarding the transmission power, it is important to identify the major energy consumers within smart objects. For battery operated objects, this is a very critical property that needs to be as much optimized as possible. Taking into account the measurements given in Figure 4.4 it is important to note that listening consumes as much as power as transmitting.

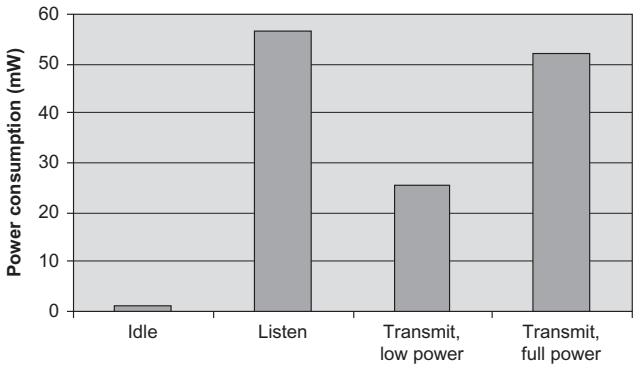


Figure 4.4: Power consumption of a IEEE 802.15.4 radio (TI CC2420) [22]

Therefore, for energy optimization it is not enough to simply reduce the amount of data that is transmitted. Instead, the radio needs to be set in idle mode as often as possible. However, since this would mean no messages could be received a mechanism is required to coordinate all senders and receivers in a wireless network to turn the radio on at the right time in order to receive all transmitted messages. This mechanism is called *radio duty cycling*. There are several ways to realize radio duty cycling. One approach is to use a time synchronization in order to synchronize senders and receivers. There are synchronous and asynchronous radio

Mechanism	Type of mechanism	Typical radio on-time (%)
X-MAC	Asynchronous	1.4
Arch Rock	Asynchronous	0.65
ContikiMAC	Asynchronous	0.45
TSMP	Synchronous	0.32
Dozer	Synchronous	0.16

Table 4.4: Comparing radio duty cycle mechanisms [22]

duty cycling mechanisms. The asynchronous low-power listening protocol is used in the X-MAC protocol [115]. Within this procedure, the radio is switched off most of the time and periodically switched on for a short time duration to receive potential transmissions. The configuration of the off-time depends on the type of traffic pattern to expect and the average delay that is acceptable for a transmission to be received at the destination node. If a sender wants to transmit a message it first sends a sequence of short packages, so-called strobe packets. These packets need to be sent for a full idle duration of a potential receiver. Once the receiver wakes up and recognizes the strobe packages it sends a strobe acknowledgement which leads then to the full transmission of the message. In this way, the energy consumption can be significantly reduced at the receiver side and most of the effort is put at the sender side. The main advantage of asynchronous radio duty cycling mechanisms is their simplicity. They can be improved by synchronous mechanisms which rely on time synchronization. An example for this is TSMP used by WirelessHART and ISA100a. In TSMP, all nodes are synchronized within 50 μs. The time is divided into slots in which a node can be listening, transmitting or sleeping. A node that wants to transmit a packet sends first a short time synchronization byte, that determines if a receiver has to stay on-line. By doing so, the energy consumption can be significantly reduced.

4.5 IPv6

IPv6 is a standard specified by the IETF in 1998. According to the specification, the primary changes concerning its antecessor IPv4 fall into the categories i) expanded addressing capabilities, ii) header format simplification, iii) improved support for extension and options, iv) flow labeling capability, and v) authentication and privacy.

- **Expanded addressing capabilities:** The address space of IPv6 is extended from 32 to 128 bit. The calculation provided in [13], 2^{128} /(4 * π * 6378137^2) = $6.6 * 10^{23}$ indicates the number of possible devices per square meter using such a large address space. This allows providing each device with an IPv6 address, to use it for identification in the future IoT and to avoid any overlay mechanism with custom addressing schemes and overlay routing mechanisms. With the extended address space, additional improvements regarding auto-configuration of nodes and the scalability of multicast routing are defined. Furthermore, a new address type called anycast is introduced, which offers to send a packet to any one of a group of nodes.

- **Header format simplification:** The header simplifications introduced in IPv6 make the protocol more attractive for deployments on embedded or constrained devices.
- **Flow labeling capability:** The new flow labeling supports a sender to label the packets of particular flows and to request certain QoS properties. For example, real-time in the sense of non-interrupted video streaming or voice over-IP are use cases for the flow label. However, in this context, the term real-time should not be confused with real-time constraints known from certain control network applications where timing guarantees need to be met by control devices. In the area of building automation, the flow label capability might be useful.
- **Authentication and privacy:** IPv6 comes with an extension to provide authentication, data integrity and data confidentiality, all features that are definitely required in the future IoT.

4.6 6LoWPAN

6LoWPAN is an acronym for IPv6 over Low-power Wireless Personal Area networks (LoWPAN). This subsection summarizes the most relevant information about 6LoWPAN in the context of this thesis based on [16]. Its idea is to deploy IP even on most constrained devices with limits regarding memory and processing power which are wirelessly connected with narrow bandwidth and strong requirements on energy efficiency [116]. 6LoWPAN is not a fully specified protocol stack, it is more a framework of standards that respectively address several issues of the IoT. As stated in [6], several problems have to be addressed if IPv6 is used on top of IEEE 802.15.4 networks.

Since devices within a LoWPAN have limited resources, usually no user interfaces are available. Thus, it is required to keep the configuration and management efforts as low as possible. In the best case, the protocols provide bootstrap mechanisms without manual configuration.

6LoWPAN provides an adaptation layer that makes it possible to use IPv6 on constrained links. A popular link for 6LoWPAN is for example provided by IEEE 802.15.4 networks. Additionally, efficient mechanisms for multicast and unicast routing are specified which facilitate a special neighbor discovery approach.

Routing

Routing is a crucial aspect of IP-based network communication. For LoWPANs, approaches for unicast and multicast routing in IPv6 based wireless networks are defined.

For unicast communication, two different routing approaches are applicable. On one hand, routing can be performed at the data link layer by the underlying mesh-network. In this case, the adoption layer ensures that a pairwise communication is possible. However, it is not specified which routing mechanism is applied. On the other hand, a routing protocol at the network layer can be taken that addresses the special demands of wireless sensor and actuator networks.

For this purpose, the Routing Protocol for Low-Power and Lossy Networks (RPL) as specified in RFC 6550 [117] is used. The routing mechanism is based on a Destination Oriented

Directed Acyclic Graph (DODAG). Every graph has exactly one destination node (root), where every path in the graph finally leads to. The graph is stored in a distributed way, whereas the building process is initiated by the root node. For the computation and maintenance of the graph, ICMPv6 messages are used. Generally, messages are sent upwards in the graph, but it is also possible that the root node sends messages to its children. RPL operates either in a non-storing mode or storing mode. In the case of P2P traffic all downward traffic in the non-storing mode has first to reach the DODAG root. For the storing mode, a routing node processes the Destination Advertisement Object (DAO) messages and constructs downward routes. This improves the P2P traffic by allowing a packet to be transmitted via a common ancestor between the source and the destination before reaching the DODAG root.

Immediate communication among nodes in different paths of the tree is not possible. Thus, some routes are far more expensive in terms of the used metric than the optimal one. If such links are frequently used, the overall performance drops. In order to overcome this issue, multiple instances of RPL can be defined. Each instance constructs its own DODAG using a different metric or a different root node, however an RPL node has to belong to at most one DODAG within an RPL instance.

IPv6 multicasting in 6LoWPAN

Within the informational RFC 6568 [118], the design and application spaces for IPv6 over Low-Power Wireless Personal Area Networks are investigated. One aspect of the design space is the traffic pattern which can be roughly categorized into point-to-multipoint (P2MP), multipoint-to-point (MP2P) and point-to-point (P2P) communication. Within traditional wireless sensor networks, P2MP traffic typically occurs for updating firmware on devices. Otherwise, the main traffic at the operation time is MP2P traffic, in which collected sensor readings are transmitted to a data sink behind the border router of a 6LoWPAN. However, for the home automation use case, multicasting and multiple P2MP connections within the network play an important role.

For 6LoWPAN, there are different options to realize multicasting. The most dominant network routing protocol up to now is the RPL. RPL is based on a DODAG. The graph is stored in a distributed way amongst the participating nodes and ICMPv6 messages are used to maintain the graph. RPL operates either in a non-storing mode or storing mode. In the case of P2P traffic, all downward traffic in the non-storing mode has first to reach the DODAG root. The storing mode can be operated with multicast or without multicast support. With multicast support, DAOs are used to relay group registrations up towards the DODAG root. In contrast to unicast traffic in which a packet is forwarded only to a single child node multicast traffic is forwarded to all children nodes that have registered to a multicast group. Multicasting routing states are therefore installed on each router between the listeners and the DODAG root.

A different approach is provided by the Multicast Protocol for Low power and Lossy Networks (MPL) [119] which, at the time being, is still in draft state. MPL avoids the creation and maintenance of a multicast forwarding topology. It uses the Trickle algorithm for controlling packet and control information transmissions. Trickle optimizes the propagation of information efficiently and mainly specifies when messages need to be transmitted, without defining a concrete frame format or even the overall purpose the Trickle algorithm is used for. The basic idea is that when two neighbors are consistent (e.g., share the same state information),

Bits	8	4	4	32
Field	prefix	flags (transient)	scope	group ID

Table 4.5: Optimized 6LoWPAN multicast address format

the message exchange rate between them is slowed down exponentially. For this purpose, the so called trickle timer is used. If a node receives consistent information, it increases this timer. If an inconsistency is detected, the trickle timer is reset. This dynamic behavior prevents from "wasting" messages and allows to propagate changes within milliseconds.

The main difference between RPL and MPL based multicasting is that in the case of RPL a multicasting topology is maintained while in case of MPL only a per packet state is preserved. This leads to several advantages and disadvantages of the mechanisms regarding scalability, performance and complexity. A thorough evaluation is provided within [87]. Since MPL is still in a draft state the protocol can be expected to be improved.

Oikonomou and Philipps [87] also propose a new protocol named stateless multicast RPL forwarding (SMRF). The concept optimizes the multicast forwarding defined by RPL. By simulation the authors show that compared to Trickle based multicasting the delay and the energy efficiency can be improved at the cost of increased packet loss.

Alternatively, the most simple way to realize multicasting in 6LoWPAN networks is to use flooding, by having each node that receives a multicast message retransmitting it once. Therefore, a sequence number stored in the *LOWPAN_BC0* field of the 6LoWPAN header can be used to avoid multiple retransmission. Although, flooding seems to be the worst option regarding network congestion and energy efficiency, the main advantage of this mechanism resides in the simplicity of its implementation.

For LLNs it is not optimal to transmit the full 128 bits IPv6 multicast address therefore for 6LoWPAN the LOWPAN_IPHC provides stateless multicast address compression allowing to compress the address to either 48 bits or 32 bits carrying inline the flag and scope field or to 8 bits without the flag and scope field fixed to the link-local scope. In order to avoid conflicts with well-known addresses[1] the transient flag must be set. In that case, it is possible to use the 48 or 32 bits representation with an address space of either 32 bits (cf. Table 4.5) or 16 bits for group communication relationships within local networks.

4.7 Message exchange and information encoding

For message exchange, either HTTP, CoAP or SOAP can be used. The SOAP binding is appropriate if enterprise systems need to interact with IoT devices. HTTP and CoAP can be used for RESTful interaction. Both are similar, but HTTP uses TCP as underlying transport layer protocol. This provides reliability but limits the communication to a connection-oriented point-to-point communication and can be considered as heavy-weight communication protocol for sensor networks regarding bandwidth and computational resources of nodes. Furthermore,

[1] http://www.iana.org/assignments/ipv6-multicast-addresses/ipv6-multicast-addresses.xml

HTTP only allows a request/response interaction using a client/server communication model. In contrast, CoAP uses UDP as transport layer. This provides unreliable packet-oriented communication with group communication and asynchronous interaction within the client/server communication model. Due to these differences, CoAP provides means for non-confirmed and confirmed message exchange and furthermore extends the regular HTTP protocol with an observe mechanism. The enhancement supports observing a resource and avoids frequently polling of resources such as event streams or alarms. Further, it is possible to rely on IPv6 multicasting for a group communication mechanism.

For the IoT stack, the application payloads can be encoded using different technologies. XML is a dominant encoding for message encoding due to the broad platform support and the interoperability. XML is a text-based encoding conveying semi-structured information. Beside the information itself also some meta-data about the structure are included in a message. The strong advantage of XML is the possibility to define a message structure and custom language based on XML schema. This meta-information can be used to create platform specific message parsers and encoders. However, the encoding based on XML is quite verbose due to being based on a text-based encoding. A similar but slightly more efficient encoding is JSON, which is becoming more popular in the WoT context. The main advantage of JSON is in the area of Web-based applications involving JavaScript. JSON-encoded objects can be directly parsed to JavaScript objects without the need of any message encoding or decoding libraries. A nice alternative to XML is EXI, which can be used to binary encode XML based documents. It does this by building up a grammar that introduces binary codes for literals found within an XML document. It can be operated in a schema-informed or non-schema information way. If a schema can be provided the information encoding becomes most efficient.

The message encoding has to be aligned with the design of the application layer services and information models. If OBIX (cf. Section 4.8) is used there are several alternative message exchange and information encodings defined. The standard and information modeling capabilities of OBIX are strongly aligned to XML since the OBIX object model is defined through an XML schema document. The need for optimized encodings for the requirements of constrained environments such as 6LoWPAN has been identified by the OBIX technical committee and a binary encoding for OBIX has been specified. The OBIX binary encoding provides a very efficient way of exchanging messages, but comes with the disadvantage of requiring message encoders and parsers implemented on every platform that should be supported.

Figure 4.5 and Figure 4.6 provide a comparison regarding message size, and throughput [50, 120]. For the evaluation of the protocol binding message size, a benchmark is used. All available stack objects are instantiated and a client performs a read and write request on all available objects and datapoints with the various protocol bindings and information encodings. As server, a Java based implementation, as described in Section 4.10, hosts the different objects and implements the different protocol bindings and encodings. The message encodings are evaluated by measuring the CPU demands of decoding message payloads. For this evaluation, direct message parsers for the basic OBIX value types *bool*, *int*, *real* and *str* (with 10 characters) are used.

Regarding message size, it can be seen that CoAP clearly outperforms HTTP. However, the main difference resides between text-based and binary message encodings. Here, binary

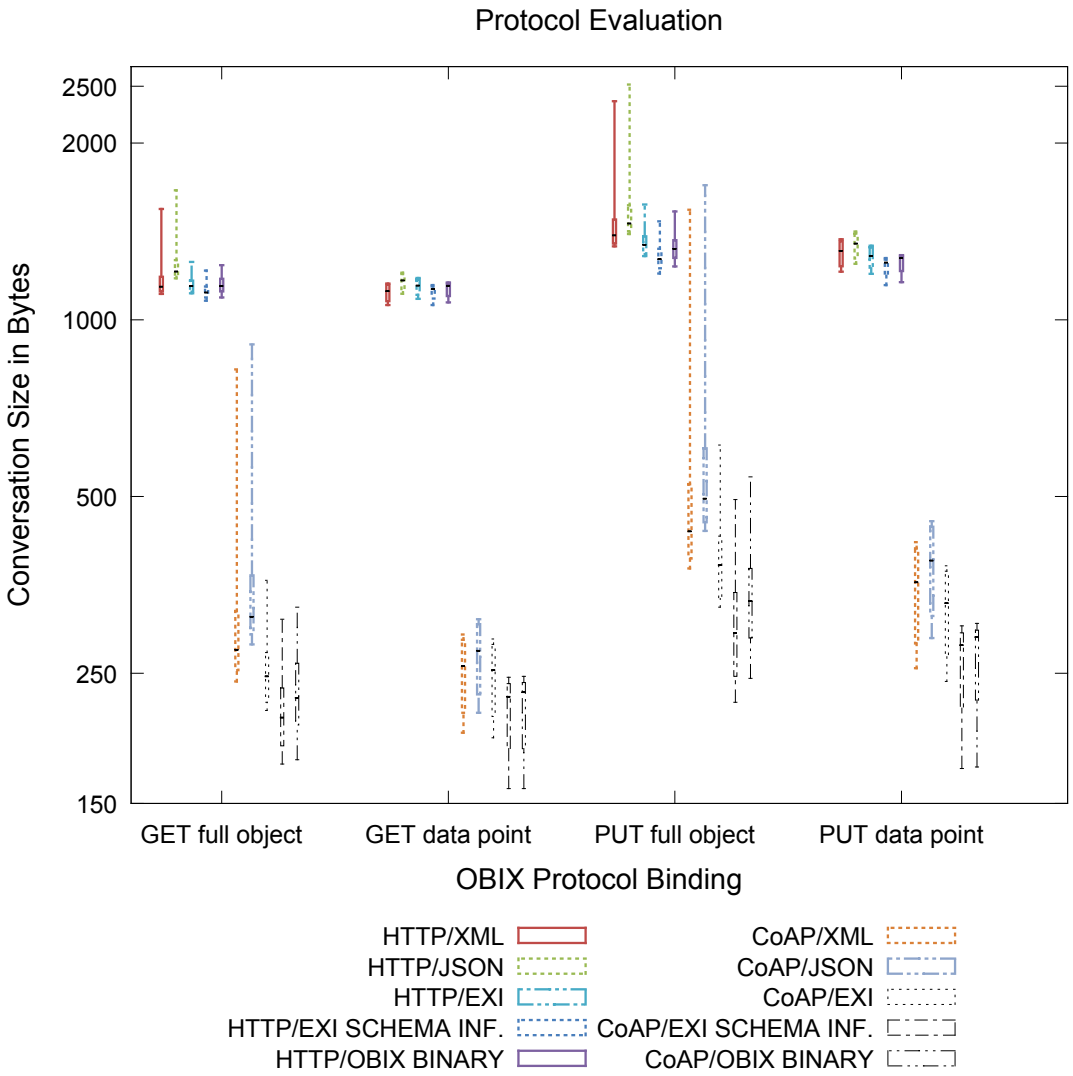


Figure 4.5: Protocol binding message size evaluation [120]

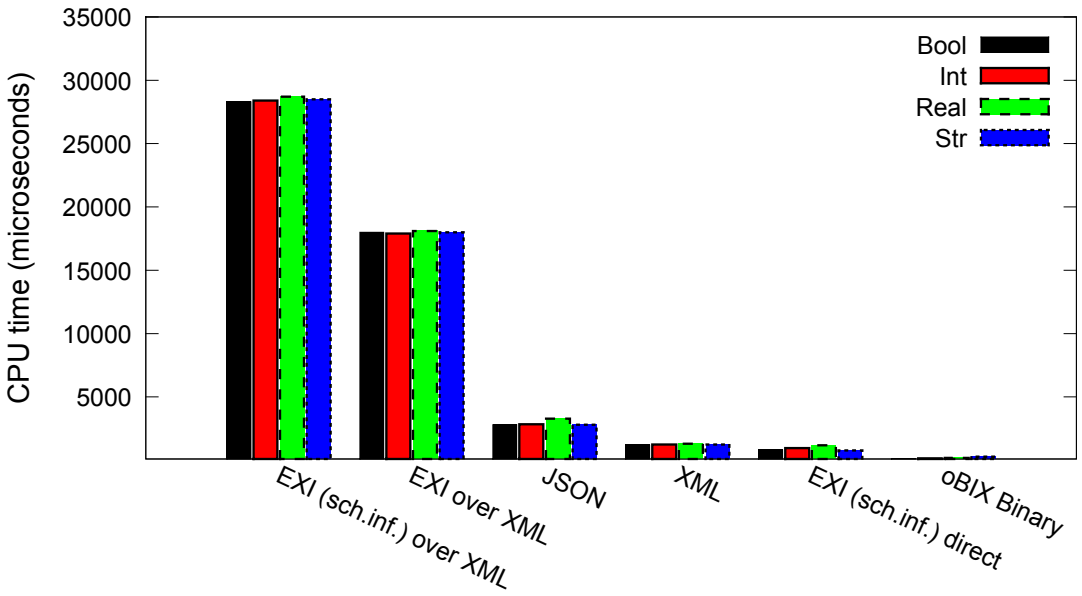


Figure 4.6: Message encoding throughput evaluation [50]

Message exchange	Encoding	Use Case
SOAP	XML	Enterprise IT integration
CoAP	EXI, XML, JSON	M2M communication
HTTP	JSON	Browser based access

Table 4.6: Protocol binding use case

message encodings provide the most relevant improvements regarding message size efficiency. Surprisingly, the schema-informed EXI encoding for OBIX performs slightly better than the custom OBIX binary encoding.

The computational resources required for the different encodings are shown in Figure 4.6. For arbitrary EXI payloads, first a conversion to XML has to be done, therefore the computational effort for decoding these payloads is the highest. Unexpectedly, the JSON decoder performs slightly worse than the XML decoder. This can be explained by the fact of better platform support and more mature implementations for XML parsers. The best performance is achieved by the direct schema-informed EXI decoder and the OBIX binary decoder.

4.8 Application services & information model

Within the Internet, HTTP is the most prominent application layer protocol used for the Web and also for Web services. However, beside HTTP numerous other protocols play an important role such as Simple Network Management Protocol (SNMP), File Transfer Protocol (FTP),

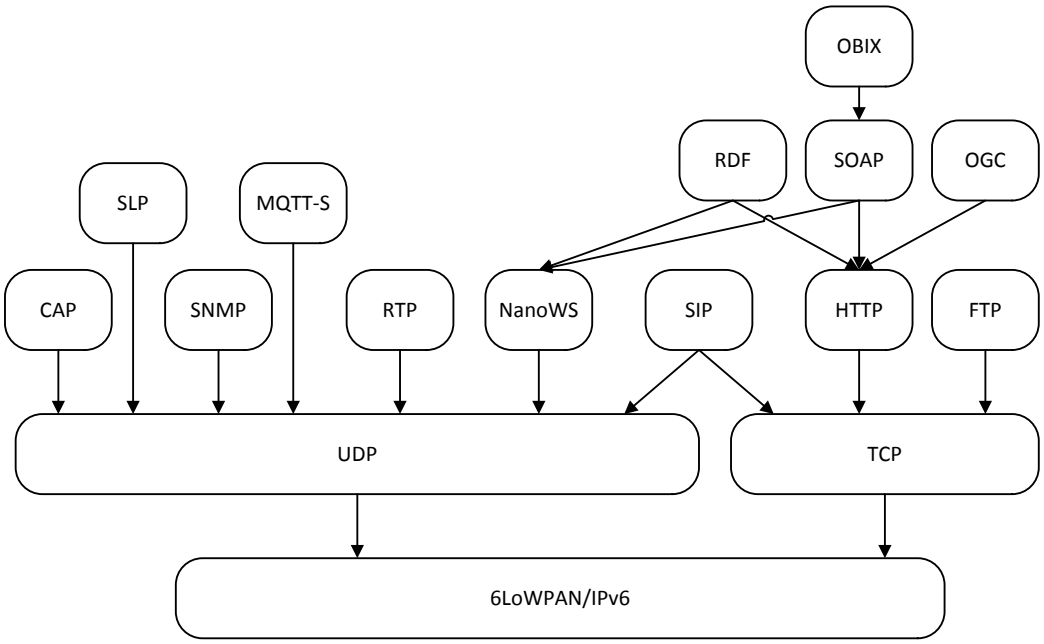


Figure 4.7: Potential application layer protocols for the Internet of Things [2]

Session Initiation Protocol (SIP) or Real Time Protocol (RTP). For the Internet of Things, the same protocols should be used. Due to the resource demands some of them might not be applicable to be used within constrained environments like micro-controllers and 6LoWPAN based wireless networks. The overview presented in Figure 4.7 discusses possible application layer protocol candidates for the IoT identified by [2]. Most suitable candidates for the IoT are protocols such as the ZigBee Compact Application Protocol (CAP), MQTT-S and OBIX.

Further, other industry protocols are mentioned such as BACnet, KNX, ANSI C12.19 or DLSM/COSEM. Shelby et al. [2] identify that OBIX with an efficient communication binding could be an alternative to run industry or domain specific protocols within the IoT. The reason for that lies in RESTful design of OBIX and the generic meta-information-model that can be used for a variety of different domains.

> "Instead of running a control network specific building automation protocol such as BACnet/IP or KNX over 6LoWPAN, OBIX together with compression and UDP/IP binding may be a solution. Careful design of the OBIX objects and elements used would be important to keep packet sizes reasonable."
>
> *[2]*

This approach is followed within this thesis. The proposed IoT communication stack is based on OBIX. The OBIX architecture contains concepts for information modeling of M2M

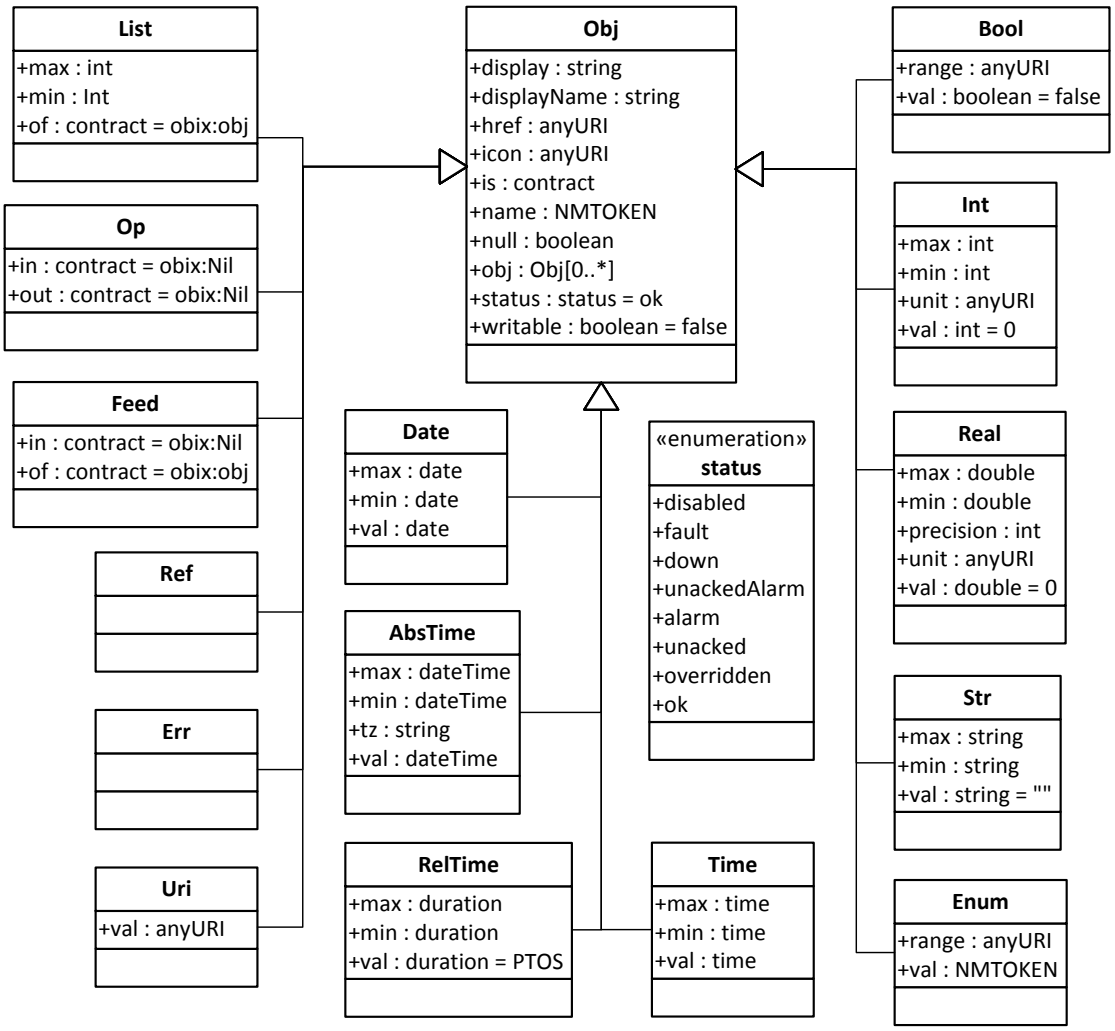


Figure 4.8: OBIX object model

information using a standardized syntax, for interaction in order to transfer M2M information within a network and finally, a normalized representation of typical BAS features such as datapoints, histories, and alarms. OBIX is in line with RESTful design and provides a REST-binding to HTTP since the very early beginning. In the first version only an XML based information encoding is provided but has recently been extended to JSON and EXI. Further, there are now protocol bindings for CoAP and Web Sockets which make OBIX more suitable for the IoT. The CoAP protocol binding and the message encodings for JSON and EXI for OBIX have been proposed within the work performed in this thesis [52] and also contributed to the standard [121, 122].

The OBIX meta-information model or so-called *object model* can be used to represent devices and entities out of the domain of building automation as OBIX objects. Everything in

OBIX is an *object*. According to the RESTful design every object is identified through a URI and an object may contain multiple child objects and reference other objects. The standard further distinguishes between basic value objects for representing simple boolean, numeric, character or time information and more complex structures as lists or feeds. Finally, complex objects can be created based on an arbitrary aggregation of the other object types. The object model is kept generic so that any type of BAS can be represented and beside this also other information sources. This generic approach is on the one side interoperable, since a simple meta-model is common to all objects, but on the other side no semantic information for specific BAS domains such as lighting, HVAC, shading, safety or security is present.

To create a standardized structure for objects and to convey some contract information between communication partners, so-called *OBIX contracts* provide a template mechanism that allows to define requirements on certain properties or data structures. Contracts contain type information that can be used by a client to rely on the presence of object properties or operations.

To illustrate this concept, Listing 4.1 shows an XML representation of an OBIX object for a room thermostat taken from the specification [123]. The room thermostat is identified through the URI *http://myhome/thermostat* and contains three nested child objects which are represented through basic value objects. With the *is* attribute the contracts of objects can be specified. In this case for all the child objects a datapoint semantic is given. Following the RESTful design, the object can be queried with an HTTP *get* request and modified through an HTTP *put* request. The payload of these requests is the full XML encoded object representation. If only a single datapoint is of interest, HTTP *get* or *put* requests on *http://myhome/thermostat/setpoint* could be used to modify only the current desired room temperature setpoint. As it can be seen in the example, further information about the unit can be represented as an additional attribute and the current value is provided within the *val* attribute.

Listing 4.1: OBIX example [123]

```
<obj href="http://myhome/thermostat/">

  <!-- spaceTemp point -->
  <real name="spaceTemp" is="obix:Point"
      val="70.0"
      unit="obix:units/fahrenheit"/>

  <!-- setpoint point -->
  <real name="setpoint" is="obix:Point"
      val="72.0"
      unit="obix:units/fahrenheit"/>

  <!-- furnaceOn point -->
  <bool name="furnaceOn" is="obix:Point" val="true"/>

</obj>
```

Information models

The OBIX meta-model is kept quite generic and does not offer any domain specific information models. For an IoT communication stack, it is desirable to have further standardized semantics

such as the capabilities of a smart object. A capability might be a temperature sensor or a switching actuator. For a client, it is of interest, for example, how to query the datapoint object representing the temperature and to know which unit will be used. Similar for the switching actuator it is required to know how to trigger a state switch.

A datapoint is an atomic unit of information within a building automation system and can either be a hard datapoint or a soft datapoint. A hard datapoint represents either a physical input or output signal. In contrast, a soft datapoint represents a software variable within the control system, such as for example a setpoint within a PID controller.

For modeling device capabilities, either a command-oriented or datapoint-oriented approach can be taken. A light switch can be represented either through a boolean field representing the switch state or by two operations like *turnOn* and *turnOff*. Both approaches are valid, however the datapoint centric approach is superior since firstly, it fits better to a resource centric design, and secondly, the interoperability problem can be relaxed. Communication partners only need to agree on a basic set of write and read operations and a standardized type system for representing datapoint information. For the command-oriented approach, any kind of operation needs to be standardized and agreed between all vendors. By having vendors defining their own operations, it would be quite unlikely to achieve a common level of interoperability.

For representing devices, two approaches can be identified. Firstly, a generic device representation, where a device offers an arbitrary list of input and output datapoints. Secondly, a capabilities-oriented approach where the device can provide one or multiple capabilities and a capability exactly defines the number and types of datapoints. A capability could be for example a 4-fold push button. Figure 4.9 illustrates the two approaches.

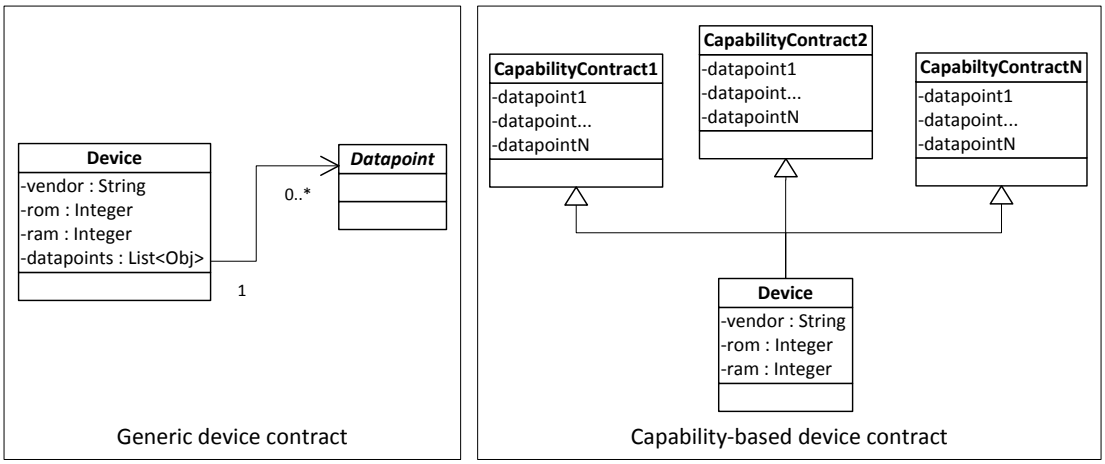


Figure 4.9: Device modeling approaches

Generic device abstraction

Within the generic device abstraction a device is represented through a basic set of properties with information about the device capabilities and the vendor. A list of available datapoints provides access to the device capabilities. This representation can be applied on any type of device but does not offer much semantic information about the capabilities although syntactical interoperability can be guaranteed. Only the datapoint type information is available and might be enriched with semantic annotations.

Domain specific device abstraction

For the IoT stack, a set of domain specific OBIX contracts have been defined. OBIX contracts allow a common abstraction amongst heterogeneous technologies and devices. Further, they can be used to define platform specific types in an object-oriented type system. A thing is the most basic abstraction and further sub-divided into a sensor and actuator, which provides a capability in a certain domain (e.g., light switch actuator).

Figure 4.10 provides an overview of a possible capability hierarchy for an IoT communication stack.

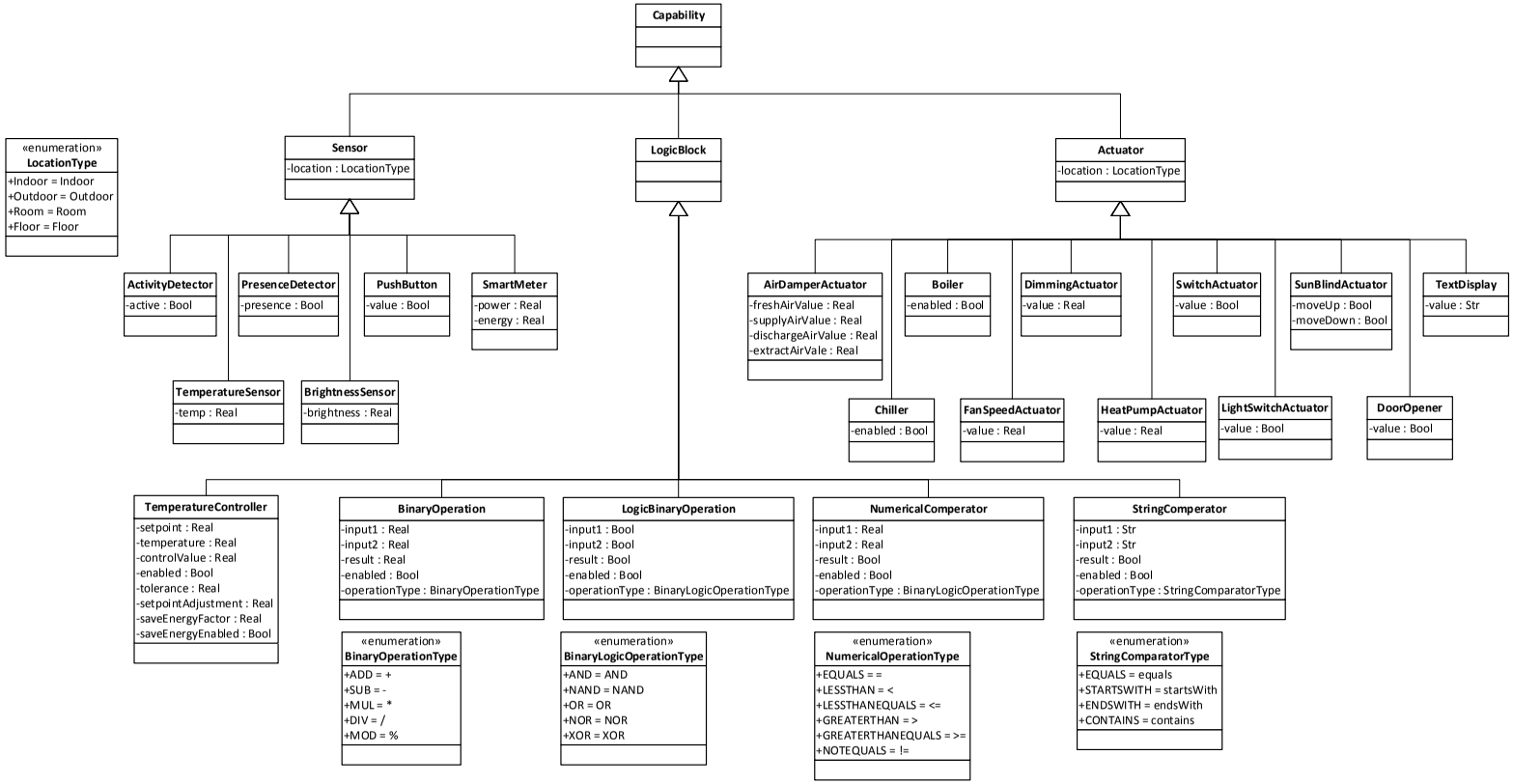


Figure 4.10: IoT capability hierarchy

The advantage of this approach is that semantic information about the type of devices and capabilities can be expressed for a communication client. However, the question is if this information is required or useful. If the client's features only rely on the simple OBIX object model or the basic datapoint type abstraction, then the further semantic information provides no use. Further, the problem is that a common standardization of the capabilities needs to be defined, which might be hard to identify within a heterogeneous vendor environment and different types of devices.

Below is a selection of some OBIX contracts for this capability hierarchy. The device abstraction ranges from a set of simple devices with only a single datapoint up to more complex mappings in which devices provide further datapoints and operations.

Listing 4.2: OBIX example contracts

```
<obj href="iot:TemperatureSensor">
  <real name="value" href="value" val="0.0" unit="obix:units/celsius"/>
</obj>
<obj is="iot:LightSwitchActuator">
  <bool name="value" href="value" val="false" writable="true"/>
</obj>
<obj is="iot:PushButton">
  <bool name="value" href="value" val="false"/>
</obj>
<obj is="iot:Cooler">
  <bool name="enabled" href="enabled" val="false" writable="true"/>
</obj>
<obj is="iot:Boiler">
  <bool name="enabled" href="enabled" val="false" writable="true"/>
</obj>
<obj is="iot:Pump">
  <int name="value" href="value" val="0" writable="true" min="0" max="100"/>
</obj>
<obj is="iot:FanSpeedActuator">
  <int name="fanSpeedSetpointValue" href="fanSpeedSetpoint" val="0" writable="true
      " unit="obix:units/percent"/>
  <bool name="enabled" href="enabled" val="false" writable="true"/>
</obj>
<obj is="iot:HumiditySensor">
  <real name="value" href="value" val="50.0" unit="obix:units/percent"/>
</obj>
<obj is="iot:LightIntensitySensor">
  <real name="value" href="value" val="1000.0" unit="obix:units/lumen"/>
</obj>
<obj href="iot:SunblindActuator">
      <bool name="moveDownValue" href="sunblindMiddleA/moveDownValue"
            val="false" writable="true" />
      <bool name="moveUpValue" href="sunblindMiddleA/moveUpValue" val="false"
            writable="true" />
</obj>
```

Datapoint object

A datapoint object is the most simple data object within the IoT stack. The purpose of a datapoint object is to host an atomic piece of information representing either an I/O-signal or a variable of the software process. OBIX foresees for this case a contract named *obix:Point* to reflect this datapoint semantic. According to the KNX specification [17], a datapoint object

type needs to define the data type consisting format and encoding and dimension which holds a unit and a range (Figure 4.11). The format could for example define whether the datapoint is represented through a boolean or floating point field. For floating point information, multiple encodings exist. Depending on the required granularity, more or less bits are used for representing the field. The dimension is a very important aspect of a datapoint. For example, taking a temperature sensor it is important to know which unit is used and if the value is represented in Fahrenheit, Kelvin or Celsius. The range is similar important for datapoints representing actuators in order to provide the information for clients which values can be taken by the actuator.

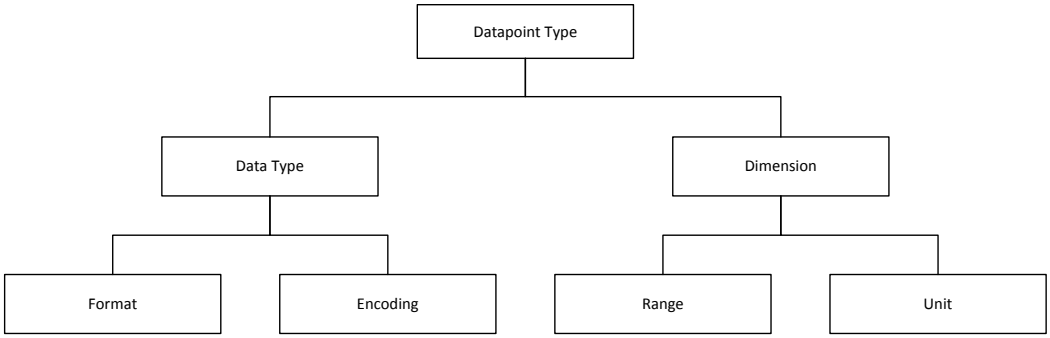


Figure 4.11: Datapoint object type [17]

There are several ways to realize datapoint objects in OBIX. Firstly, the information can be encoded in basic value objects like *bool*, *real*, *int* or *str*. This approach is shown in Listing 4.3. While for the *real* object the format, dimension and range information can be provided it is not possible for the *bool* object. There is no information whether the field represents the *on/off* or *open/close* state for an actuator. Such information would be valuable for a client interacting with a device.

Listing 4.3: Datapoint Objects

```
<bool is="obix:Point" name="channelA" href="channelA" writable="true" val="false"
    />

<real is="obix:Point" name="setpoint" href="setpoint" unit="/units/celsius" min="
    -273" max="670760" writable="true" val="false"/>
```

A more sophisticated way to represent datapoint objects in OBIX would be to use a complex object together with a type hierarchy. For a boolean datapoint, the plain information is stored within a *bool* object hosted by a complex object representing the datapoint. Within a subtype further semantic information about the encoding can be provided by using a *enum* field that holds encoding information for the whole value domain of the datapoint type.

Listing 4.4: Datapoint Objects

```
<obj href="iot:Bool" is="iot:Datapoint">
  <bool name="value" href="value" writable="true"/>
```

```
</obj>

<obj href="iot:BoolOnOff" is="iot:BoolOnOff iot:Datapoint">
  <enum name="encoding" href="encoding" range="/encodings/onoff" writable="true"/>
</obj>

<list href="encodings/onoff/" of="obix:bool" is="obix:Range">
<bool name="on" href="onoff/on" val="true" displayName="On"/>
<bool name="off" href="onoff/off" val="false" displayName="Off"/>
</list>
```

However, the more complex datapoint type representation provides only additional benefits for client communication partners and is especially useful for user interfaces. For interoperability and interworking between different devices, solely a common datapoint format is required.

Interaction pattern

The data access services define how smart objects based on the IoT stack can interact with each other and how external clients can access smart objects in order to modify control states or to collect sensor readings. For accessing data within the IoT stack, multiple interaction patterns are possible.

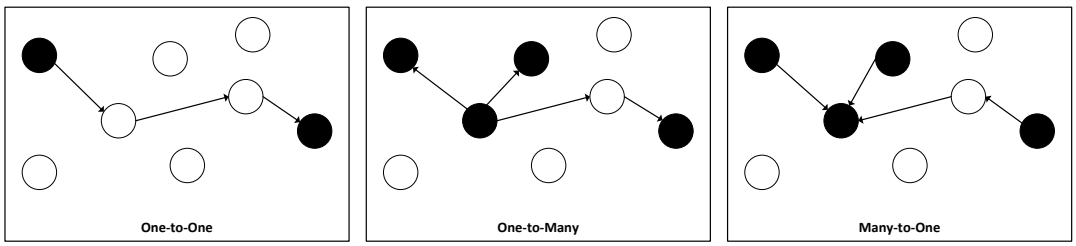


Figure 4.12: Communication interaction patterns [22]

In general, the communication mechanisms can be divided into one-to-one, one-to-many and many-to-one communication [22] as illustrated in Figure 4.12. For one-to-one communication, smart objects or nodes communicate directly with each other, having other smart objects only involved for forwarding messages or responses. Another interaction pattern is one-to-many communication in which a message from one smart object needs to be transmitted to multiple receivers or even all smart objects within a network. There are several ways to realize one-to-many communication within a WSAN. Whether reliability is an issue or not, simple flooding can be used in which each node simple retransmits the message in order to have the message eventually relayed to all nodes. Many-to-one communication is usually used if so-called *sink* nodes collect sensor data from multiple smart objects.

Furthermore, the interaction can be divided into request/response-based interaction, asynchronous communication and publish/subscribe-based communication. Furthermore, end-to-end connectivity should be provided for communication partners.

The HTTP binding provides only a request/response-based interaction. For updating an actuator or reading a sensor value, it is always necessary for a client to send a request in order to initiate a response. There is no unsolicited message sent from a server. This interaction pattern is state of the art for Web services but has a huge drawback within a sensor network, where several sensors need to be monitored. The client needs to poll the smart object to receive the latest sensor value which is inefficient and leads to a lot of traffic in the network and in consequence to congestion and high energy consumption. More efficient is asynchronous communication in which a smart object sends updated values in an unsolicited response to a client. Comparable is the publish/subscribe interaction pattern which is used within message-oriented middlewares such as MQTT but slightly different. Here, a communication partner can subscribe to a topic of interest at a message broker and smart objects can publish latest sensor readings at the topic. Asynchronous communication is supported in this interaction mechanism, however a message broker is required which relays the communication.

This leads to the next communication principle that should be applied by a communication stack which is end-to-end communication. Within end-to-end communication only the communicating partners should deal with the application layer part of a message and no other nodes in between should work on the application layer. Some protocols define intermediate nodes or proxies which act differently depending on the application payload. For the proposed IoT stack in this thesis, both principles can be found. For the CoAP binding, end-to-end connectivity is provided but for certain communication use cases involving commissioning or external clients based on HTTP or SOAP, a gateway component can translate between the different communication protocols.

Within the presented IoT stack, a one-to-one communication is supported through the HTTP and CoAP binding with a smart object, preferable using the CoAP binding in order to provide an efficient communication. The HTTP binding follows a request/response-based interaction, which is also possible with the CoAP binding. The CoAP binding offers further interaction mechanisms such as asynchronous communication and group communication in order to realize a one-to-many communication. A many-to-one communication could be realized also using the group communication mechanism. For this interaction style, a one-to-one communication with an asynchronous subscription to sensor readings is more meaningful.

Object identification based on URIs

Since OBIX follows a RESTful system architecture, the addressing of entities rests upon URIs. A URI is based on a hierarchical sequence consisting of a scheme, authority, path, query and fragment as illustrated in the following listing.

Listing 4.5: URI scheme

```
URI = <scheme name> : <authority> / [path] [? <query> ] [ # <fragment> ]
```

The scheme type defines the URI-type and is used to interpret the following content of the URI. If the scheme defines a protocol like HTTP or CoAP then the URI can be taken as a URL that can be used to access the resource by the specified protocol. For the authority, either a DNS name or an IPv6 address can be used as proposed in [52].

The URI scheme for a device depends on the taken modeling approach. For a generic device model, the scheme looks as follows:

Listing 4.6: Generic device addressing

```
Generic device URI scheme = <scheme name> : <authority> "/" path
path = <device-property> | "datapoints/" <datapoint-id> "/" <datapoint-object>
```

Consider for example a device that offers a temperature and humidity sensor. For the generic device modeling approach, two datapoints would be represented in the device datapoint list. A concrete example for CoAP addressing these two datapoints is given in Listing 4.9. This scheme does not distinguish between sensors and actuators. This differentiation can be done within the datapoint object representing the I/O signals marked either as writeable or only readable. The datapoint object is either a simple OBIX value object or a complex datapoint object.

Listing 4.7: Addressing scheme example

```
URI: coap://[2008:db8::1]/datapoints/1/temperature
Identified object:
<real is="obix:Point" name="setpoint" href="setpoint" unit="/units/celsius" min="
    -273" max="670760" writable="true" val="false"/>

URI: coap://[2008:db8::1]/datapoints/2/light
Identified object:
<obj href="iot:BoolOnOff" is="iot:BoolOnOff iot:Datapoint">
  <bool name="value" href="value" writable="true"/>
  <enum name="encoding" href="encoding" range="/encodings/onoff" writable="true"/>
</obj>
```

Capability-based device modeling leads to the scheme in Listing 4.8. In this case, a functional IoT contract specifies the capability of a device, for example, the capability of being a switching actuator. This approach induces a flat addressing scheme, however problems might arise through naming conflicts of capabilities having similar datapoint object names and URI identifiers.

Listing 4.8: Capability-based addressing

```
Capability URI scheme = <scheme name> : <authority> "/" path
path = <device-property> | <capability-name><capability-id> / capability-subpath
capability-subpath = <capability-property> | <datapoint-object>
```

Listing 4.9: Addressing scheme example for capability-based modeling

```
URI: coap://[2008:db8::1]/light1/
Identified object:
<obj is="iot:LightSwitchActuator">
  <bool name="value" href="value" val="false" writable="true"/>
</obj>

URI: coap://[2008:db8::1]/temperature1/
Identified object:
<obj href="iot:TemperatureSensor">
  <real name="value" href="value" val="27.0" unit="obix:units/celsius"/>
</obj>
```

IPv6 addressing proxy: Depending on the used protocol and media the URI takes a large part of the protocol header since the authority- or path-part can be represented using a text-based encoding. Therefore, it is desirable to reduce the protocol header and to use pure IPv6 addressing only. This can be achieved by using an IPv6 addressing proxy which allows to assign an IPv6 address not only on the device level but also at the resource level. In this way, not only devices can be identified through an IPv6 address but also capabilities or datapoint objects. IPv6 provides the required address space for these application scenarios. The proxy mechanism can either be based on IPv6 unicast addresses or IPv6 multicast addresses, which will be used for the group communication mechanism later on. Through the proxy mechanism, the message size can be decreased since an arbitrary long text-based URI-path can be avoided. Figure 4.13 shows the impact of URIs within CoAP based message exchange for IEEE 802.15.4 wireless networks.

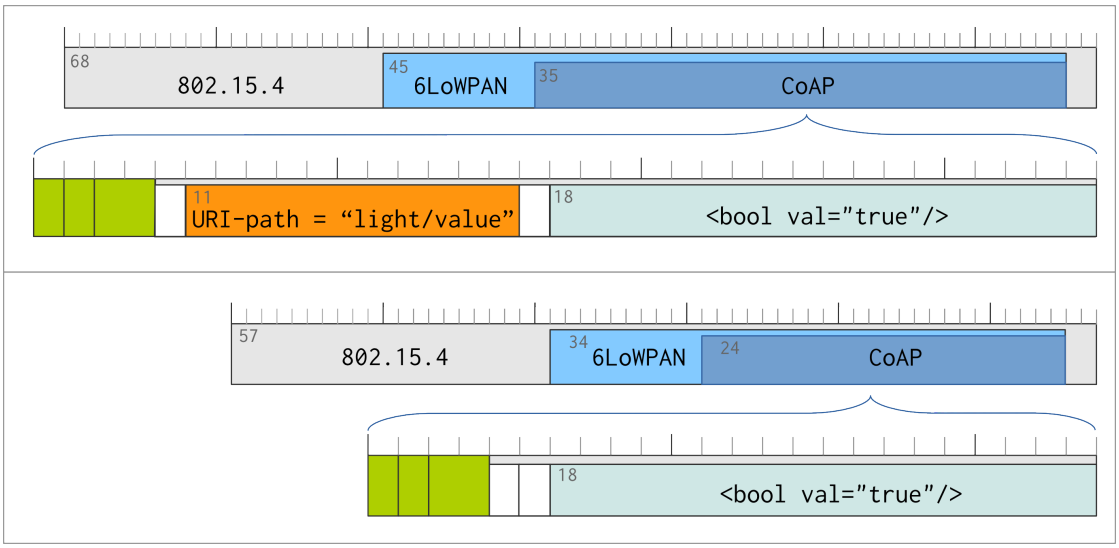


Figure 4.13: Impact of URI on IEEE 802.15.4 frame size [51]

Further, the frame size can be significantly reduced if only IPv6 addresses are used for addressing datapoint objects.

Data access

For data access, OBIX foresees to use the operations *read*, *write*, and *invoke* which can be mapped to the various protocol verbs according to Table 4.7. For updating object states, the *put* method of the protocols can be used. For the transmission, a one-to-one communication or one-to-many communication is possible. The one-to-many communication is only possible with the CoAP protocol binding, which does not provide a reliable means of communication and rests upon a best effort delivery of messages.

OBIX Operation	HTTP	CoAP	Target
Read	GET	GET	Any accessible object
Write	PUT	PUT	Any writeable object
Invoke	POST	POST	Any operation (OBIX type *op*)
Delete	DELETE	DELETE	Any writeable object

Table 4.7: Data access methods

Histories

Within monitoring and analysis of buildings, *histories* of datapoints play an essential role. OBIX standardizes the representation of time series and their resource representation. An OBIX history object collects datapoint values and provides a standardized interface to query, filter and aggregate them. The contract for the object is given in Listing 4.10.

Listing 4.10: OBIX history contract

```
<obj href="obix:History">
  <int name="count" min="0" val="0"/>
  <abstime name="start" null="true"/>
  <abstime name="end" null="true"/>
  <str name="tz" null="true"/>
  <list name="formats" of="obix:str" null="true"/>
  <op name="query" in="obix:HistoryFilter" out="obix:HistoryQueryOut"/>
  <feed name="feed" in="obix:HistoryFilter" of="obix:HistoryRecord"/>
  <op name="rollup" in="obix:HistoryRollupIn" out="obix:HistoryRollupOut"/>
  <op name="append" in="obix:HistoryAppendIn" out="obix:HistoryAppendOut"/>
</obj>
```

For numeric datapoint time series, an aggregation can be performed calculating simple statistic values. This can be useful if a series of values needs to be calculated for sensor values of interest. Consider a smart meter collecting the historic meter readings. The *rollup* operation can be used to receive an aggregated result.

Histories further provide means to query event feeds and to retrieve latest events. Data can also be actively pushed to history objects by clients by using the *append* operation.

Watches

Watches introduce an important OBIX feature for HTTP-based clients. Since an HTTP client needs to continuously poll for updates on new information such as the latest sensor reading, a lot of traffic is generated which puts load on the network and the server. For this problem, OBIX foresees the use of per-client state objects or so-called *watches* which keep track of changes for a dedicated client. A client can create an arbitrary number of watches and register URIs of objects that should be tracked. The main contracts are listed in Listing 4.11. The watch object takes a list of object URIs passed within a *WatchIn* object through the *add* operation. With the *pollChanges* operation the latest changes of the observed objects can be retrieved.

Listing 4.11: OBIX watch

```
<obj href="obix:Watch">
```

```
  <reltime name="lease" min="PT0S" writable="true"/>
  <op name="add" in="obix:WatchIn" out="obix:WatchOut"/>
  <op name="remove" in="obix:WatchIn"/>
  <op name="pollChanges" out="obix:WatchOut"/>
  <op name="pollRefresh" out="obix:WatchOut"/>
  <op name="delete"/>
</obj>

<obj href="obix:WatchIn">
  <list name="hrefs" of="obix:WatchInItem"/>
</obj>

<uri href="obix:WatchInItem">
  <obj name="in"/>
</uri>

<obj href="obix:WatchOut">
  <list name="values" of="obix:obj"/>
</obj>
```

Alarming

The observation of datapoints and monitoring for certain alarm conditions is an important feature for safety and security use cases in the IoT. OBIX standardizes the representation of alarming by providing alarming contracts and also specifying a mechanism for stateful alarm representation which can persist the information when the alarm state has entered and when it has been reset to normal. Further, alarms can support an acknowledgement mechanism which can be used by a client to confirm the reception of the alarm.

Service discovery

Service discovery within the IoT stack can occur at multiple layers and for different interaction scenarios. Firstly, there is a *global discovery* mechanism and *service repository* as described within Chapter 3. Secondly, there are local discovery mechanisms that can be used within a local wired or wireless network to discover devices and their capabilities. Thirdly, a device needs to express its capabilities and service interfaces to a client that wants to interact.

For RESTful Web service DNS-SD [124] can be used to realize service repositories and mDNS [125] for discovery within a local network. DNS is a hierarchical distributed naming system for computers, services or resources connected to the Internet. With DNS, names can be assigned to these entities and clients do not need to use IP addresses. A DNS name consists of a set of labels separated by a label, where each label can have a maximum length of 63 characters. The length of the complete name is limited to 255 characters. The name space is organized as a rooted tree. The labels of a path name identify the nodes within the name space tree, a subtree identifies a domain and the root node of a sub tree represents the domain name. Resource records represent the contents of a domain and convey information such as the IP address of the host representing this node (A record), mail server for the node (MX record) and relevant for DNS-SD records for specific services provided (SRV record), the canonical name of a host (PTR record) and arbitrary information (TXT record). Within DNS-SD, it is standardized how DNS and DNS resources can be used to facilitate service discovery. Given a certain service

type and domain at which this type of service is provided, a client looking for the service can lookup instances hosting this service. Domain names used for this mechanism take the form given in Listing 4.12.

Listing 4.12: DNS-SD domain names

```
<sn>._tcp . <servicedomain> . <parentdomain>.
<Instance> . <sn>._tcp . <servicedomain> . <parentdomain>.
<sub>._sub . <sn>._tcp . <servicedomain> . <parentdomain>.
```

The service name (<sn>) consists of a pair of DNS labels. The first label starts with an underscore following with a service name according to [126]. The service name should identify what the service does and which application protocol is used. An example given in the standard for such a DNS-SD name is "_http._tcp.example.com.", which allows clients to discover all Web servers at *example.com* domain. The _tcp and _udp have been identified in retrospect as not good design choices and a generic _srv label might have been a better choice. If an implementation wants to fully stick with the standard one of these labels has to be used. Although the authors of the standard argue against the use of custom labels for other application protocols such as SOAP it makes sense in the context of the IoT stack. The subtype (<sub>) can be used to identify a subset of service instances fulfilling certain properties.

For the IoT stack, the DNS-SD service domain could be either _http, _coap, or _soap and the service type would be a standardized name representing the stack such as _obix or _iot. Further, the sub name could be linked to the capability hierarchy. In this way, multiple service discovery scenarios can be realized as illustrated with the examples in Listing 4.13.

Listing 4.13: DNS-SD service discovery examples

```
_obix._http.auto.tuwien.ac.at // All service instances and devices providing an
    OBIX service based on the HTTP binding in the domain of the Automation
    Systems Group
_obix._coap.floor3.auto.tuwien.ac.at // All service instances and devices at
    floor 3 providing an OBIX service based on the CoAP binding
_lightswitchactuator._obix._coap.floor3.auto.tuwien.ac.at // All devices
    providing a light switch actuator capability
```

The service discovery can be managed based on DNS-SD in a hierarchical manner. As can be seen in the example it can be used to advertise the services at a certain site or building represented through a domain which is further subdivided into zones.

If devices need to discover each other in a peer-to-peer interaction style without the presence of any domain name server mDNS can be used. In this case, ".local." is used as top-level domain and a DNS resolver issues a multicast request based on IPv4 or IPv6 in order to query the network for service providers. In [127], an optimized lightweight version of mDNS and DNS-SD is presented for discovery of IPv6 based resources in the WoT.

Once a device is discovered, the OBIX *lobby* can be used to discover the resources hosted by the device. The lobby is provided through a standardized URI and provides a list of references to available objects and their contracts. An example of such a listing is given in Listing 4.14.

Listing 4.14: OBIX lobby

```
<obj href="obix/">
  <ref name="about" href="obix/about"/>
```

```
  <ref href="watchService" is="obix:WatchService"/>
  <ref href="alarms" is="obix:AlarmSubject"/>
  <ref href="IoTDevices/IndoorBrightnessSensor" is="iot:IndoorBrightnessSensor"/>
  <ref href="IoTDevices/OutsideTemperatureSensor" is="iot:OutsideTemperatureSensor
      "/>
  <ref href="IoTDevices/Presence" is="iot:PresenceDetectorSensor"/>
  <ref href="IoTDevices/PushButton" is="iot:PushButton"/>
  <ref href="IoTDevices/RoomRelativeHumiditySensor" is="iot:
      RoomRelativeHumiditySensor"/>
  <ref href="IoTDevices/RoomTemperatureSensor" is="iot:RoomTemperatureSensor"/>
  <ref href="IoTDevices/TemperatureSensor" is="iot:TemperatureSensor"/>
  <ref href="IoTDevices/SunIntensitySensor" is="iot:SunIntensitySensor"/>
  <ref href="IoTDevices/SwitchingSensor" is="iot:SwitchingSensor"/>
  <ref href="IoTDevices/SmartMeter" is="iot:SmartMeter"/>
  <ref href="IoTDevices/SimpleHVACvalveActuator" is="iot:SimpleHVACvalveActuator"/
      >
  <ref href="IoTDevices/ComplexSunBlind" is="iot:ComplexSunblindActuator"/>
  <ref href="IoTDevices/SunblindMiddleA" is="iot:SunblindActuator"/>
  <ref href="IoTDevices/SunblindMiddleB" is="iot:SunblindActuator"/>
  <ref href="IoTDevices/FanSpeed" is="iot:FanSpeedActuator"/>
  <ref href="IoTDevices/BrightnessActuator" is="iot:BrightnessActuator"/>
  <ref href="IoTDevices/HVACvalveActuatorImpl" is="HVACvalveActuator"/>
  <ref href="IoTDevices/AirDamperActuatorImpl" is="iot:AirDamperActuator"/>
  <ref href="IoTDevices/LightSwitchActuator" is="iot:LightSwitchActuator"/>
  <ref href="IoTDevices/PumpActuatorImpl" is="iot:Pump"/>
  <ref href="IoTDevices/DimmingActuatorImpl" is="iot:DimmingActuator"/>
  <ref href="IoTDevices/CoolerActuatorImpl" is="iot:Cooler"/>
  <ref href="IoTDevices/BoilerActuatorImpl" is="iot:Boiler"/>
</obj>
```

Security

For security, confidentiality, non-repudiation, integrity and authenticity between communication partners need to be provided as already stated in Chapter 3. Similar security mechanisms can be applied for securing the communication of the IoT stack. However, the memory and computational constraints of embedded devices impose limits on the applicability of several security methodologies. For example, key distribution and the maintenance of a public-key infrastructure might be too expensive for constrained devices such as micro-controllers. Further, the available memory might impose limits on the used key size, also the implementation size of protocols such as key-exchange protocols. If no special hardware such as a cryptographic co-processor is available, asymmetric encryption mechanisms might be not practical to be used. Therefore, most common are symmetric encryption mechanisms such as AES for smart objects due to the low computational effort. The security can be realized at multiple layers, starting from the link layer, over the network layer to the transport layer or finally the application layer. Link layer security can be provided by IEEE 802.15.4 through AES symmetric encryption. The problem with link layer security is that security can only be provided over a single hop but security needs to be ensured preferable in an end-to-end way between two communication partners including all hops and routers in-between. End-to-end security can be handled by IPSec or TLS. IPSec can be used within IPv4 or IPv6. The IPSec architecture consists of two protocols, the Authentication Header (AH) and the Encapsulating Security Payload (ESP). AH provides integrity for IP packets and ESP supports integrity and confidentiality. IPSec only takes care on security on a per-packet level. For securing the full stream of a communication,

Transport Layer Security (TLS) can be used. The main advantage of application layer security is that it is agnostic of the underlying transport protocol, however it needs to be specifically taken into account at the protocol design time and supported by communication partners. For SOAP-based message exchange, the full standardized features of WS-Security can be used. For XML-based messages, XML Encryption or XML Signature can be used to secure the exchange of payloads, but if other message encodings such as JSON are in use these standardized capabilities are not available. In this case, either custom security facilities need to be defined or the application layer has to rely on transport layer security facilities.

Authorization

For authorization, the XACML access control architecture can be used taking into account the object model and the capabilities offered by devices and smart objects. An XACML PEP component might be deployed on a 6LoWPAN border router or gateway component evaluating the access on devices and their resources. The XACML PDP component might be operated on a gateway or on central infrastructure components.

4.9 IoT peer-to-peer communication and Web-based commissioning

This section presents how a novel group communication protocol binding for OBIX can be used to realize distributed control logic. It is inspired by the group communication facilities found in nowadays home and building automation technologies, such as KNX or LonWorks, which rely on this design paradigm to efficiently realize the process data exchange between different devices of heterogeneous vendors. The information model of these systems is modeled around the concept of datapoints that represent the basic data structures of I/O signals or software variables of a device.

A similar group communication concept is presented for CoAP and IPv6 based communication. For establishing this group communication relationship within the IoT system architecture, a local Web-based commissioning tool can be used.

Group communication in CoRE

The IETF working group on constrained RESTful environments aims at providing efficient Web service based communication for low-power and lossy wireless networks (LLNs) with limited bandwidth and constraints regarding energy consumption. CoAP [9] is a key application protocol that provides an optimized UDP based alternative to HTTP for realizing RESTful Web service endpoints on constrained nodes. By being based on UDP it opens the opportunity to use IPv6 multicast packets for message exchange, which can be efficiently used for group communication. An IETF draft on group communication with CoAP [83] aims at exploiting this interaction pattern for CoAP based devices. However, the proposed approach assigns IPv6 multicast addresses on the device level and further uses a group URI identifier that finally ad-

dresses the data item that participates in the communication group. Several weaknesses can be identified in this approach.

Firstly, due to the use of IPv6 multicast addresses on the device level it is required to use further URI identifiers to identify a group communication endpoint. This is rather verbose since the URI identifier is text-based and verbose URIs have a severe impact on the CoAP header size, since no compression can be provided for the URI identifier. Secondly, the approach requires all devices to configure a new resource and URI identifier at runtime, which is not supported by most of the existing CoAP implementations out of the box. Further, all nodes have to share the same URI path identifier, which is error-prone and may lead to conflicts within the addressing scheme.

Finally, the standard leaves open the exchanged data format, so the current CoRE standardization falls short on providing true interoperability by leaving this important aspect of the application layer open.

IPv6 multicasting and CoAP based communication

For efficient group communication, devices need to be represented as objects following a datapoint-centric information model in contrast to a command-oriented design. A datapoint can be a boolean field that represents the state of a push button or switching actuator or a numeric field that represents a temperature or a setpoint of a room-thermostat. The RESTful Web service interaction model fits very well to this design approach, by offering simple service to read and write on resources, like datapoints.

To realize the concept of a shared network variable based on IPv6 and CoAP, a datapoint may now participate in one or more groups by assigning an IPv6 multicast addresses to it. An IPv6 multicast address is therefore not used to identify a common service or protocol at the device, instead it is used to identify a communication relationship between multiple devices. The device may offer a list of datapoints. A datapoint maintains a list of group addresses that can be changed dynamically at runtime. For the assignment of group addresses, a standardized interface needs to be provided.

So the paradigm should not follow to assign IPv6 multicast address on every device, instead IPv6 multicast addresses on every datapoint should be assigned. By doing so, the identification of group endpoints is simplified to an IPv6 multicast address instead of having an IPv6 multicast address combined with a URI based group identifier. Since 6LoWPAN also provides a compression scheme for multicast addresses the advantage regarding the protocol header size is significant.

The state of the data structure is synchronized between the different devices by making a CoAP PUT request directly on the transient link-local IPv6 multicast address. If a 6LoWPAN network is combined with other technologies the use of the site-local scope *FF15::/16* could allow to exchange the group messages amongst heterogeneous links.

Shared network variable

For the definition of an efficient group communication based on IPv6 multicasting and CoAP, the paradigm of shared network variables needs to be used. A shared network variable is a

Object type	CoAP payload ex.	XML bytes	EXI bytes
Bool	<bool val='false'/>	20	3
Int	<int val='58'/>	15	4
Real	<real val='58.12'/>	19	6

Table 4.8: Group communication payload

design paradigm for inter-process communication. In contrast to explicit remote procedure calls or Web service calls, the inter-process communication is managed by synchronizing simple data structures amongst distributed processes that need to interact. The data structures usually have a simple data type representing either a boolean, integer, numeric or string value and are typically encoded with a few bytes. If such a data structure is updated in one process the new value is transmitted through an underlying framework to other processes that have an association to the shared variable. By doing so the interaction between different devices can be reduced to this simple synchronization of basic data structures. For ensuring interoperability between different vendors, only a common encoding of the data structures and the interfaces to synchronize the values are required. The standardized interface includes the message exchange protocol and the application layer services required for the synchronization.

Standard object model and efficient message encoding

To have a standardized datapoint-centric object representation, the object model of OBIX is used. The object model provides a standardized XML-based representation of basic datapoints. Table 4.8 provides some examples and also lists the according encoding size for XML and schema-informed EXI.

Furthermore, OBIX provides a way to define operations that allow a datapoint to join or leave a group. Therefore, contracts are used to standardize the interface across heterogeneous vendors.

Listing 4.15: Group communication contract

```
<obj name="groupComm" href="iot:GroupComm">
      <list name="groups" href="groups" of="obix:str"/>
      <op name="joinGroup" href="joinGroup" in="obix:str" out="obix:list"/>
      <op name="leaveGroup" href="leaveGroup" in="obix:str" out="obix:list"/>
</obj>
```

Logic objects

Virtual logic objects provide functionality that is used to create complex control scenarios. This can range from simple numerical functions, boolean operators to more sophisticated functions like PID controllers.

A set of logic elements is illustrated in Table 4.9. The *binary numerical operation* with two input data types and a binary numerical operation can be selected out of a defined enumeration. A result datapoint provides the numerical outcome of the operation. The *binary*

numerical comparator can be used to compare two input datapoints. Depending on the numerical comparator type, the output is represented through a boolean output datapoint. For logical comparisons, the *binary logical operation* can be used. For simple HVAC control, a *temperature controller* is provided based on a simple two-point control logic. It provides an output between +100% and -100% acting as the control value for heating or cooling units. It further comes with an eco-mode functionality that is triggered by a boolean flag and reduces the control output by a given value. Furthermore, it provides an input field for setpoint adjustment that can be triggered through a room-control panel. All logic objects host a boolean flag that allows to enable or disable the control logic.

Containers and device classes

The overall concept and components of a shared network variable concept are illustrated in Figure 4.14. Within the concept different device classes can be identified. *Base class* devices only host the data access services of the IoT stack in order to interact with the device and its capabilities. *Extended class* devices may further host service discovery capabilities, more enhanced security components and authorization components. *Full feature class* devices offer all capabilities of the IoT stack including different protocol bindings, security components, authorization policy stores, a persistence layer based on a database, history and watch services, a proxy module for other devices and a user interface component. Such devices are typically too heavy-weight for a deployment on micro-controller based devices. Therefore, an embedded PC platform or standard PC platform is most suitable. In this case, these devices may also act as a 6LoWPAN border router to connect the WSAN to an existing network infrastructure.

The design of these devices as illustrated in Figure 4.14 consists of base class devices of the typical components of a smart object, such as a micro-controller, radio, sensor/actuator I/Os, CoAP REST and multicast engine, an OBIX device object with several child objects having the group communication capability. Full feature class devices additional offer the REST and SOAP engine and implement all message encodings. Further, proxy modules allow to integrate devices with less features in order to mimic the capabilities for them. For example, a virtual IoT device representation could be used to offer a SOAP endpoint for a device which is not directly capable of exchanging SOAP messages. Based on the HTTP and JSON protocol bindings, a Web based control and monitoring interface can be provided that offers a user the capability to interact with the IoT devices and to create control logic by grouping devices together.

4.10 Implementation

For realizing the IoT stack on constrained devices, a 6LoWPAN based platform is used which comes with a micro-controller and an IEEE 802.15.4 transceiver. To realize the firmware, the Contiki operating system is used which provides an optimized implementation for IPv6 and further comes with a CoAP reference implementation. Based on this implementation, an IoT application has been developed that provides an OBIX based object representation of sensors and actuators, attached to device and further implements the IPv6 based group communication mechanism.

Logic element	Operations	Parameters	In/Out	Type	Encoding examples
Binary numerical operation	$+, *, :, \%$	input 1	In	Numerical	*<real val='5.0'/>*
		input 2	In	Numerical	*<int val='5'/>*
		enabled	In	Boolean	*<bool val='true'/>*
		operationType	In	Enumeration	*<enum val='+'/>*
		operationType	Out	Numerical	*<real val='10.0'/>*
Binary numerical comparator	$=, <, >, \leq, \geq$	input 1	In	Numerical	*<real val='5.0'/>*
		input 2	In	Numerical	*<int val='5'/>*
		enabled	In	Boolean	*<bool val='true'/>*
		operationType	In	Enumeration	*<enum val='+'/>*
		result	Out	Boolean	*<real val='10.0'/>*
Binary logical operation	$\wedge, \vee, \oplus$, NOR, NAND	input 1	In	Boolean	*<real val='5.0'/>*
		input 2	In	Boolean	*<bool val='false'/>*
		enabled	In	Boolean	*<bool val='true'/>*
		operationType	In	Enumeration	*<enum val='or'/>*
		result	Out	operation result	*<bool val='true'/>*
Temperature controller	Two-point controller	setpoint	In	Numerical	*<real val='25.0'/>*
		setpointAdjustment	In	Numerical	*<real val='2.0'/>*
		tolerance	In	Numerical	*<real val='2.0'/>*
		temperature	In	Numerical	*<real val='20.0'/>*
		enabled	In	Boolean	*<bool val='true'/>*
		safeEnergyEnabled	In	Boolean	*<bool val='true'/>*
		safeEnergyFactor	In	Numerical	*<real val='20.0'/>*
		controlValue	Out	Numerical	*<real val='100.0'/>*

Table 4.9: Logic objects

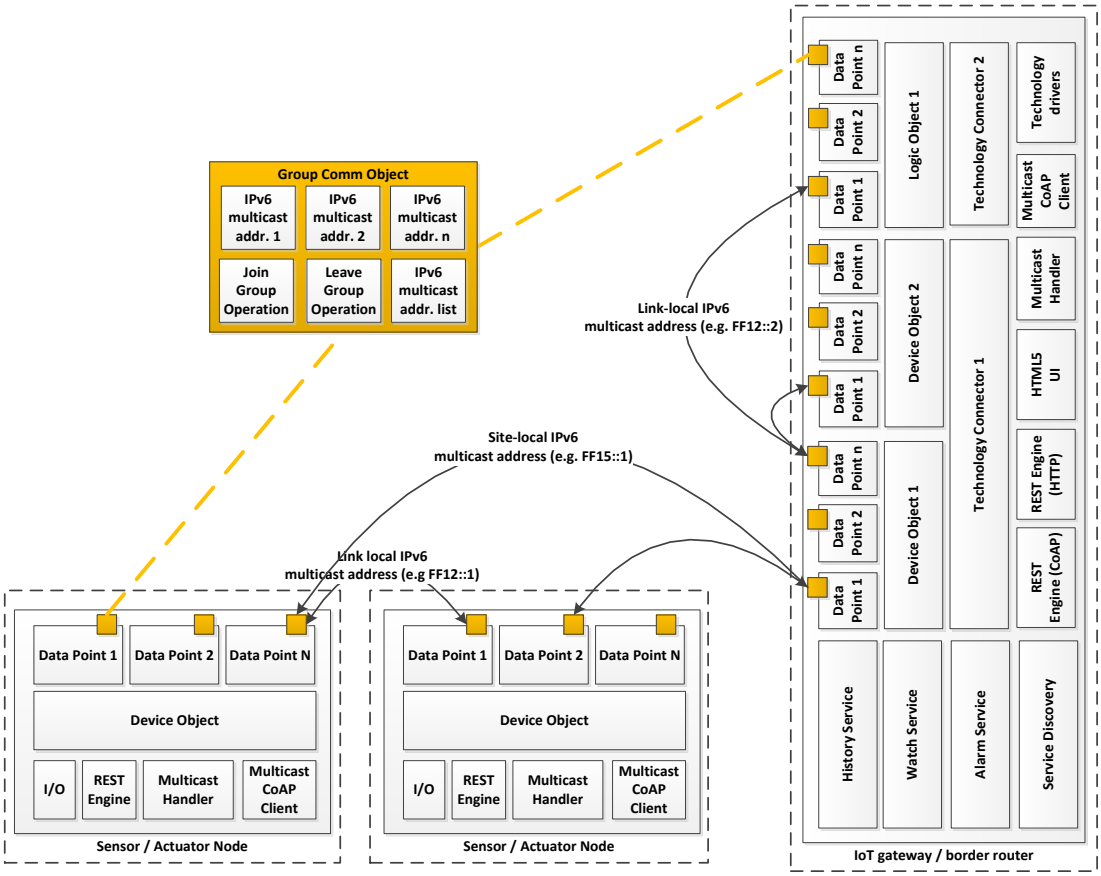


Figure 4.14: Shared network variable components [51]

Since Contiki 3.0, IPv6 multicasting based on different multicast engines (MPL, SMRF) is available. Further, a modified version of a flooding-based implementation is used that relies on the *LOWPAN_BC0* header to avoid duplicate retransmission of packets and allows to customize the multicast forwarding based on the proposed concept relying on transient IPv6 site-local multicast addresses. The Erbium framework has been extended to support multicast-based communication and to allow an application framework to register a handler routine for group communication messages that are received via IPv6 multicast packets.

For the implementation of the shared network variable concept, the proposed OBIX contract for the group communication object is optimized due to the memory constraints and only the *post* based operation requests for joining and leaving groups on a datapoint are implemented. An internal data structure keeps track of the mapping between the multicast addresses and the datapoint resource representation. As soon as a datapoint joins an IPv6 multicast address, the node is lined to the communication group. If requests arrive the basic OBIX value object is parsed and the internal value is updated. Similarly, if a change on a sensor input value is detected and the resource is linked to a multicast group, a CoAP non-confirmable *put* request

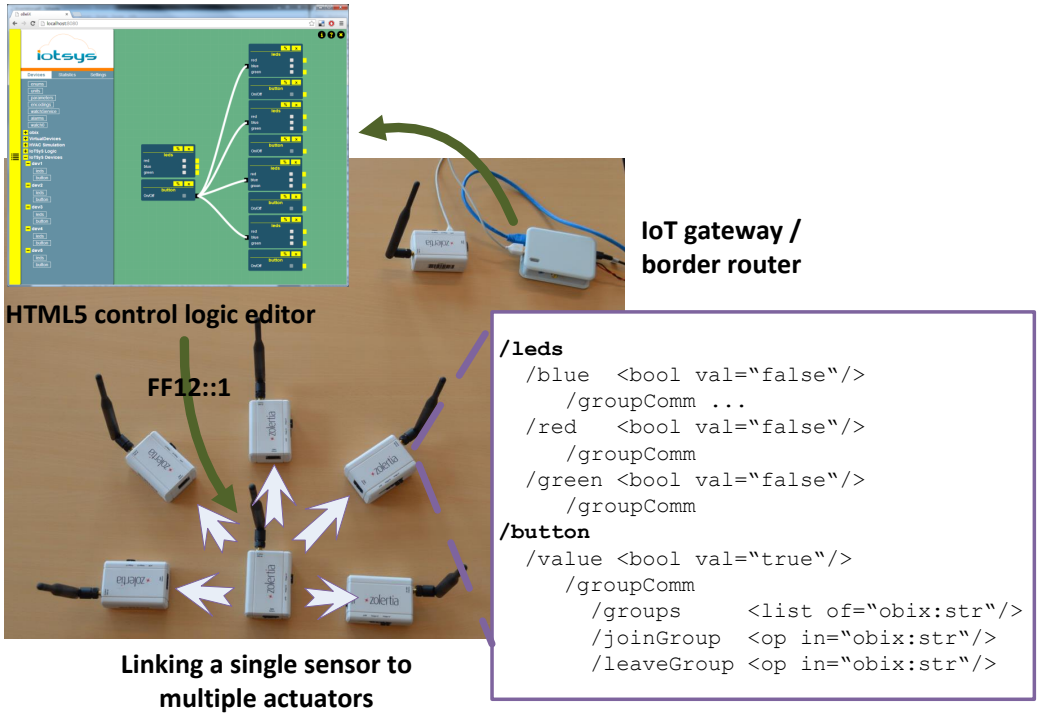


Figure 4.15: Test environment

is sent.

To forward site-local transient IPv6 multicast packets, the border router is modified to forward these requests to the tunnel interface of the host platform. The further routing resides in the responsibility of the host operating system. Furthermore, a Java-based gateway resides at the border router and provides additional protocol bindings for the sensors and actuators, and hosts services for service discovery, histories, watches and alarms. It provides also a persistence layer and hosts the Web-based commissioning tool. The gateway might also act as integration middleware for other technologies as described within Chapter 5.

Testbed

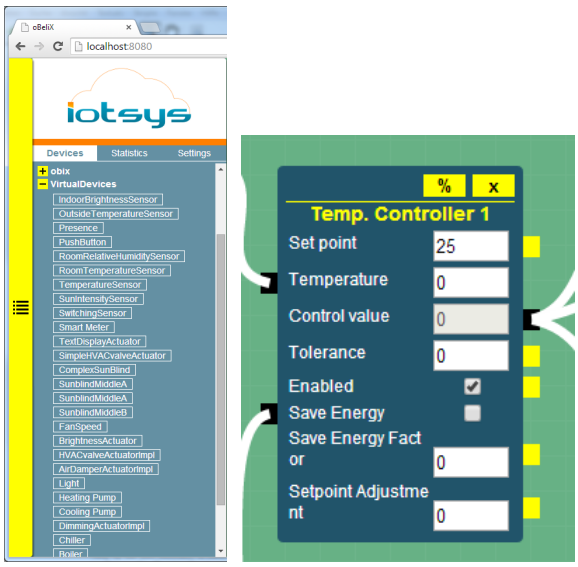
A testbed based on the Zolertia Z1 motes is used for an experimental evaluation. A mote offers a temperature sensor, accelerometer and button functionality on board. The proposed concept allows to wire multiple Z1 motes together, by linking for example the button of one mote to the LEDs of multiple other motes. For a direct user interaction, the user interface at the gateway can be used for wiring the button to the LEDs. The graphical control logic wire tool is used to download the addressing assignments. Figure 4.15 provides an overview of the used testbed.

Web-based commissioning and control logic editor

For commissioning of control logic, a Web-based control logic editor is implemented named Obelix. Obelix is an HTML5 control engineering interface based on JavaScript and CSS. It is a generic OBIX client that directly operates on the Web service interfaces provided by the gateway component. It is deployed as standalone file served by the gateway component and uses the HTTP and JSON protocol binding for OBIX. As the complete user interface is executed within the client browser there is no performance impact on the gateway.

The HTML5 control logic editor allows direct user control of devices and provides further means to configure communication relationships based on the IPv6 multicasting mechanism. The user interface consists of the following components illustrated in Figure 4.16.

Object browser: The object browser is the entry point for a user. It lists the available objects based on a query using the OBIX lobby. Since everything in OBIX is an object which may have an arbitrary number of subobjects the structure is recursively analyzed and an entry is displayed in the object browser based on the name. The user can drag an element out of the browser into the object canvas for display and to update values.



(a) Obelix obj. browser (b) Obelix object component

Figure 4.17: Obelix oject browser and object component

Object component: The object component as illustrated in Figure 4.17b can be used to display an OBIX object. Following the OBIX object model, an object consists of subobjects which are either base value types like, for example, *bool*, *int*, *real*, *str*, or complex objects. An object is rendered as component that provides HTML5 input elements for all base value types. For rendering the object, a simple *get* request is performed on the object URI and the object structure

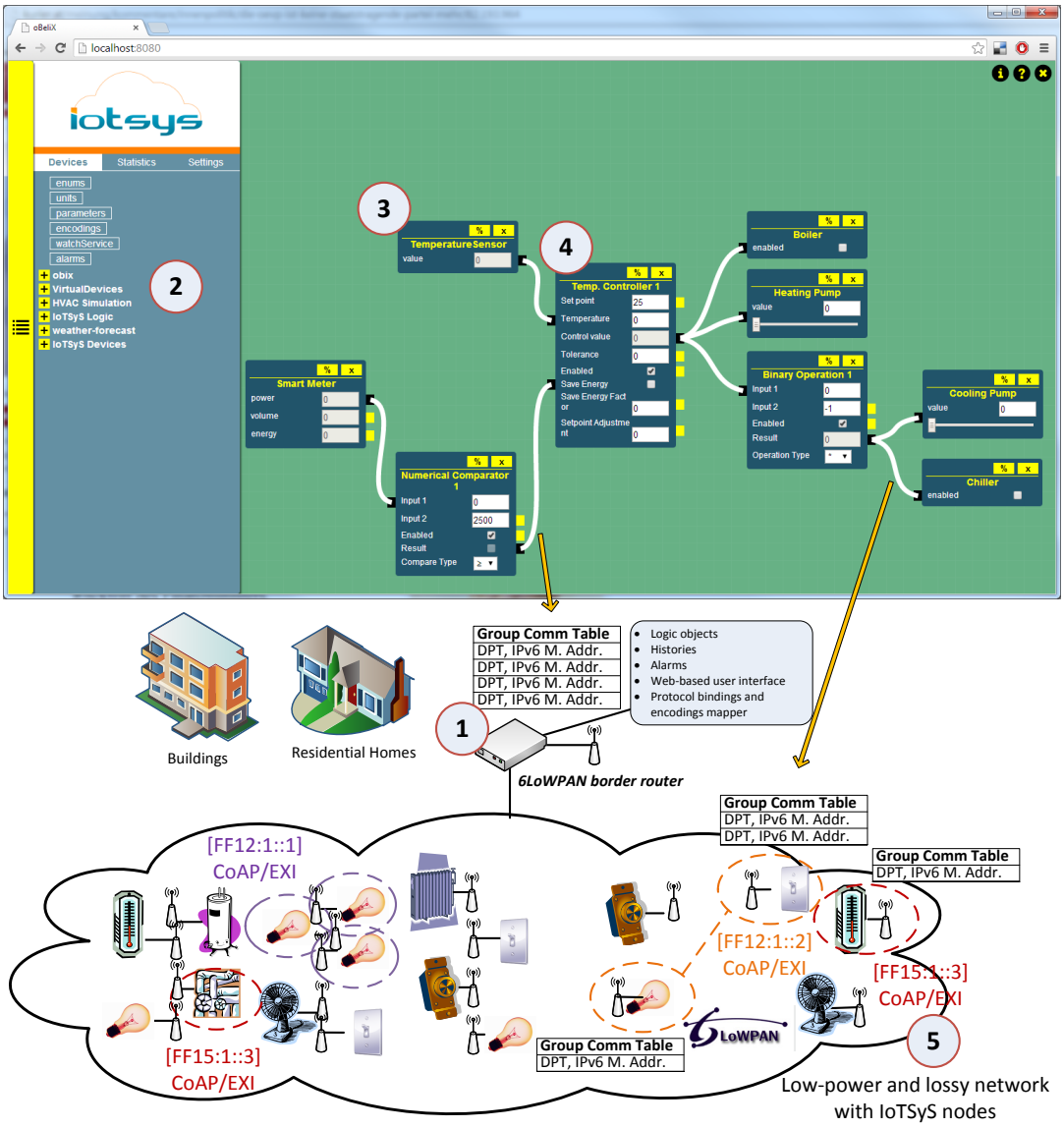
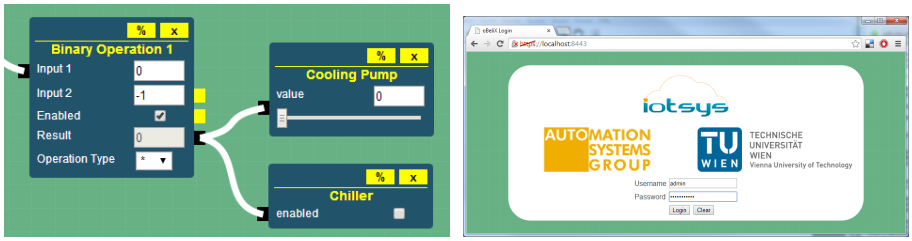


Figure 4.16: Obelix Overview

is parsed dynamically. On a change of a base value property, an according *put* request is performed and the object is updated at the server side. All objects represented in the object canvas are added to a watch that is used to monitor the current state of an object. This mechanism is required since the HTTP binding is used which requires to poll the gateway for updated states. The user interface does not depend on any domain specific object contracts and is therefore completely generic and reusable for any IoT application domain.



(a) Wire mechanism to create control logic (b) Obelix user authentication

Figure 4.18: Obelix wire mechanism and user authentication

Wire mechanism to create control logic: The object component allows a simple interaction with devices and virtual objects represented through OBIX objects. For engineering the group communication relationships, a graphical wire tool allows to group datapoints of different objects together. Whether a datapoint can participate in group communication or not is determined through a group communication object that is attached as child object to the basic value object. This is determined dynamically by the user interface. If such an object is present, connectors are displayed that can be used to graphically wire objects using a drag and drop mechanism as illustrated in Figure 4.18a. Once a connection is established, a dedicated IPv6 multicast address is added to the according group communication objects. If a connection is removed the multicast addresses are also released. For IoT "native" devices, the multicast address is added directly on the device using the HTTP/CoAP proxy mechanism. Once a device detects a change on the datapoint an according IPv6 multicast message using a CoAP *put* is sent to the multicast address and all receivers with a datapoint in the same address update their internal state.

Security: The Web commissioning tool is secured through HTTPSs and requires a user authentication through user-name and password (cf. Figure 4.18b). For external access, further authentication and security mechanism are provided as described within Chapter 3.

Obelix implementation: Obelix is implemented on top of the AngularJS framework[2] with heavy use of the two-way data binding capabilities provided by the framework which follows the Model-View-Controller (MVC) design pattern.

The execution on the client starts with an almost empty Document Object Model (DOM). Then, as JSON responses from the server are parsed into JavaScript objects representing the

[2]http://angularjs.org

model layer, AngularJS evaluates our declarative template and builds up the DOM (the view layer) to graphically represent the models. The template also establishes the binding between events on the DOM elements (such as a user clicking or dragging a device) and the controller, which handles these actions.

Several other frameworks are used to abstract lower level DOM manipulation and deal with cross browser compatibility, such as jQuery UI[3] which is used for drag and drop support and jsPlump[4] for visualizing and manipulating connections between DOM elements.

The resulting application consists of three single files: an Hypertext Markup Language (HTML) document containing the AngularJS template, the JavaScript file containing application source concatenated with the framework sources, and the Cascading Style Sheets (CSS) file. These static files are served by the built-in Web server with minimal performance overhead. The whole implementation is provided as open-source within the IoTSyS project[5].

4.11 Evaluation

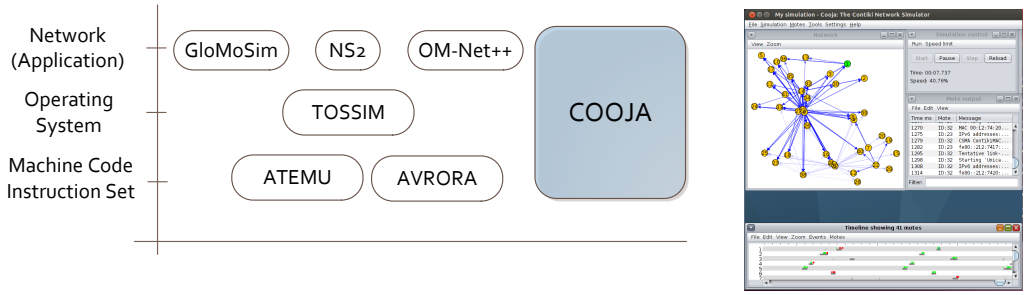
A scalability analysis of the IoT stack within a WSAN is conducted within this section. Therefore, a simulation based on the COOJA simulation environment of the Contiki operating system is used. This methodology is used since the COOJA simulation environment uses for the emulation of nodes the same firmware binaries that can be used on real hardware. In this way, compared to an analytic approach, higher effort has to be put on a proof of concept implementation but with the benefit of having more accurate and realistic results. The goals of the simulation are, firstly to analyze the impact of different protocol bindings and encodings, secondly to discuss the scalability limits, and thirdly to evaluate the impact of the group communication mechanism based on IPv6 multicasting. The metrics of interest are the latency, average response time, message failure rate and the energy consumption. The latency is important if information needs to be disseminated quickly amongst a group of appliances. The average response time needs to be kept under a certain time in order to reach quality-of-service goals if a client/server communication takes place.

COOJA wireless sensor network simulator

The COOJA network simulator [46] is a cross-level simulation environment that provides the capability of a holistic simulation at different levels within a wireless sensor network. COOJA is a Java-based simulator designed to simulate a network based on sensor nodes running the Contiki operating system [40]. As illustrated in Figure 4.19a, there are several alternatives that focus on different levels of simulation. NS2 or OM-Net++ are popular network level simulation frameworks. TOSSIM can be used to simulate TinyOS based nodes [128]. For instruction level simulation, ATEMU [129] or Avrora [130] provide a similar functionality. However, a simulation on all these different levels is only possible with COOJA. Therefore, COOJA is used to analyze the IoT stack regarding performance and scalability implications.

[3] http://jqueryui.com
[4] http://jsplumbtoolkit.com/home/jquery.html
[5] http://www.iotsys.org

COOJA can simulate different types of nodes. Contiki programs can be executed either as native code of the COOJA host platform or an instruction-level TI MSP430 emulator can be used. Further, nodes can be implemented completely in Java if the focus is more on the network communication protocol and the evaluation of distributed algorithms. Nodes can either be simulated at the network level, operating system level or machine code instruction set level but mixed in a certain simulation scenario. The network level is of interest if for example routing protocols or application specific protocols need to be evaluated. In this case, the specific hardware is not in focus of the simulation and other factors like the utilization of the radio medium, radio devices and duty cycles of sensor nodes might be in the scope of the evaluation. The operating system level is relevant if features of the operating system are modified and the proper function needs to be evaluated without the need of using real hardware. Finally, the machine code instruction level allows to evaluate compiled firmware for real hardware in a preliminary study. In this way, the development and test cycle can be significantly reduced and the effort of evaluating the firmware in a large scale deployment can be reduced. In certain cases, the required hardware for large-scale deployment tests might not be available so COOJA provides a convenient way to analyze the scalability of communication protocols.



(a) COOJA cross level simulation

(b) Simulation user interface

Figure 4.19: COOJA simulation

Within the COOJA environment, different arbitrary network topologies can be instantiated with multiple node instances of a similar node type. Figure 4.19b provides a screen shot of the graphical display. The environment allows to log certain simulation events and further to collect the log output of the nodes. The simulation is based on a discrete event time that is incremented where the simulation speed can be adjusted to follow a real-world speed. This is especially of use if the network traffic is visualized and a human can observe the process of network establishing and communication.

Simulation use cases

The following use cases inspire the simulation scenarios:

- **Query or update of sensor/actuator:** This use case represents the common situation that an external client queries a sensor. For example, the collection of temperature sen-

sor values from a set of sensors can be such a use case. In the case of a cloud based monitoring system, the client could reside externally somewhere in a datacenter. Similarly, an external client might activate an actuator. For example, the control of a set of light switch actuators could represent such an interaction. For this client/server-based interaction it is of interest which average response time can be achieved, depending on the protocol binding. It is of interest how CoAP performs in comparison to HTTP and how the different message encodings impact the overall communication performance.

- **Group communication between sensors and actuators:** This simulation scenario is focusing on local peer-to-peer communication between sensors and actuators. In contrast to the client/server communication paradigm, here the producer/consumer paradigm is taking place. A concrete use case could be a push button transmitting an updated value to a set of actuators. Latency between the first transmission of an updated information until the successful receive at all consumers is a relevant metric. Energy efficiency is essential and optimal routing of the updated information between involved communication partners, since battery-operated devices might be in use. For 6LoWPAN, the performance of different multicast routing protocols is of interest regarding latency and energy efficiency.

For the simulations, the following notation is used when describing the network topology of different communication scenarios. Figure 4.20 shows an example topology that consists of the following four node types. The communication range of the nodes is illustrated through the circled area around each node.

- **Server/Consumer**: A communication node providing access to its resources through an IoT stack communication interface or receiving message from a producer.
- **Client/Producer**: A communication client which sends requests to a server or acts as producer of messages in a group communication scenario.
- **Router**: A border router for a 6LoWPAN network, acting as root of the RPL DODAG.
- **Hop**: An intermediate 6LoWPAN node that can be an arbitrary node but beside the forwarding of messages without any specific responsibility within the communication for the simulation scenario.

Next to the used topology for a simulation scenario, the used communication configuration parameters are relevant for the interpretation of the simulation results. Especially for WSANs based on 6LoWPAN and IEEE 802.15.4, the radio duty cycling (RDC) mechanism has a strong influence on the performance and energy consumption. For all simulations, the ContikiMAC radio duty cycling mechanism is used.

There are several metrics of interest that can be studied within the WSAN simulation based on COOJA.

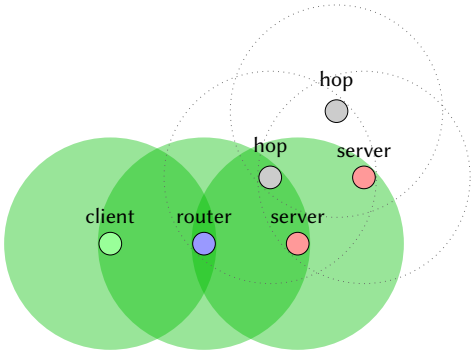


Figure 4.20: Example topology for simulations [131]

HTTP vs. CoAP

This first section compares Web service based communication within WSANs based on HTTP or CoAP. Different simulations are executed to compare metrics like average response time, message failure and energy efficiency. The results will show why CoAP based Web services are necessary within WSANs and also quantify the difference regarding the defined metrics.

Latency and average response time

The *latency* of a request represents the time between the first fragment of the message was transmitted and the last fragment of the message is finally delivered at the receiver's side. For a client/server-based interaction, the response time represents the time between the transmit of the request and the time when the fully response is delivered at the sender's side.

Message failure rate

The *message drop* metric represents the number of messages that are either not successfully received by the receiver or if a client receives no response within a certain time range.

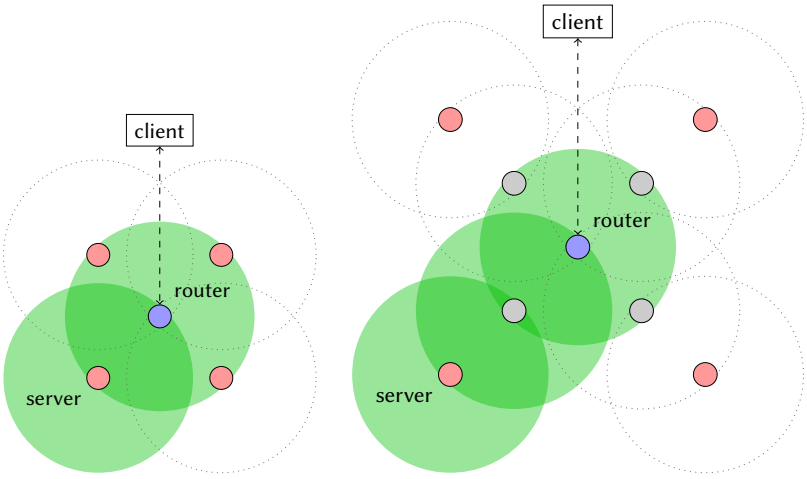
Energy consumption

The *energy consumption* is important for the operational costs of a WSAN. Most of the nodes are considered to be operated using batteries. Having nodes draining the available energy too fast leads to high maintenance costs. Within COOJA emulated nodes, the used energy cannot be measured directly. Instead the required CPU cycles provide a way to compare different communication mechanisms regarding their energy efficiency. The CPU cycles are used to measure three energy consumption relevant metrics:

- **CPU time:** represents the time while the CPU is active.
- **Radio TX time:** measures the time while the radio is active for sending.
- **Radio RX:** measures the time while the radio is active for receiving.

Average response times

A first study is performed on the latency and power usage implications introduced by the HTTP and CoAP protocol binding. Two topologies are used for the evaluation and illustrated in Figure 4.21. Within the networks one or multiple servers are queried with a *get* request of the according protocol. A resource payload with a variable payload is returned by the servers. A sequence of 10 requests is issued per simulation and every simulation is repeated 5 times. The average response time is measured at the border router.



(a) 1 hop (b) 2 hops

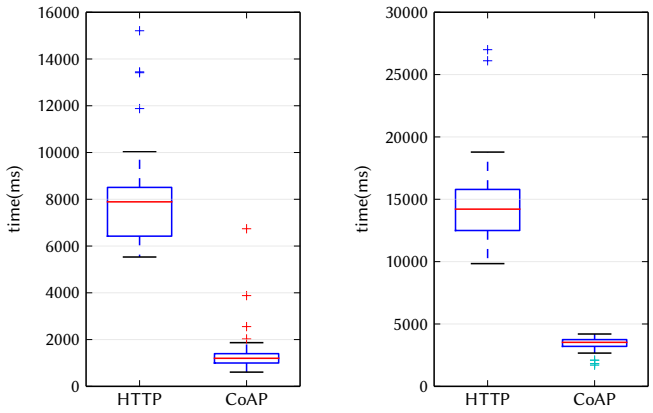
Figure 4.21: Simulation topologies CoAP vs. HTTP - latency [131]

Figure 4.22 contains the resulting average response time if either two or four parallel requests are issued in the two hop scenario. For this simulation, a 128 bytes payload is used. The average response time is significantly different. For the scenario where 2 requests are performed in parallel in multiple threads, the CoAP based communication outperforms the HTTP based communication by a factor of 8, but decreases to a factor of 3.5 to 4 for four parallel requests. A more detailed evaluation is given in Figure 4.23. In contrast, Figure 4.24 shows the impact of adding additional hops for a given number of parallel requests.

Both figures show that the scalability of HTTP is strongly limited within a WSAN and significant performance degradation can be observed already for a low number of parallel requests or a low number of hops within the network topology.

Message failure rate

Within the simulation, the response timeout limit needs to be relaxed in order to get comparable results. Considering a response time exceeding 10 seconds for message failure delivery, the failure rates illustrated in Table 4.10 can be observed. The results show that HTTP quickly suffers under the conditions found within WSANs. The very poor performance of HTTP can be explained due to the fact that with higher number of concurrent requests the underlying



(a) 2 requests (b) 4 requests

Figure 4.22: Distribution for latency for HTTP and CoAP, 2 hops, 128B [131]

TCP performs poorly and leads to a congestion of the network. The impact of the payload size is neglectable. For CoAP, every response arrived within the 10 seconds threshold.

	2 requests	**4 requests**
64B	0%	9.2%
128B	0%	7.6%

(a) 1 hop

	2 requests	**4 requests**
64B	0.4%	49.6%
128B	2.8%	56.4%

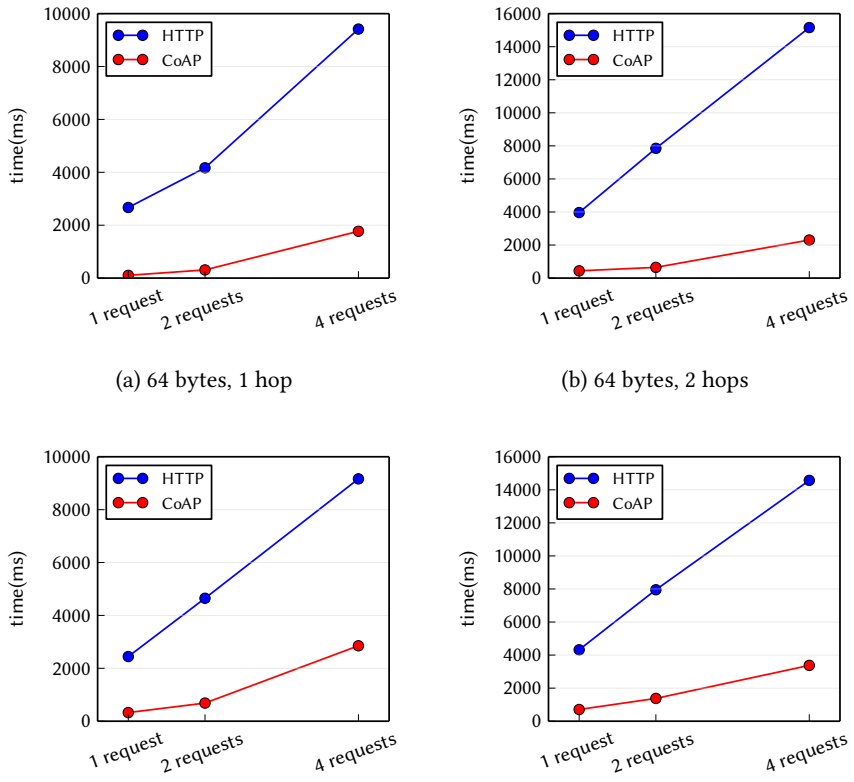
(b) 2 hops

Table 4.10: Request failure rate for HTTP [131]

Energy consumption

Furthermore, the energy consumption is a metric of high interest if the communication stack is deployed in a WSAN. For the evaluation, *get* requests of the HTTP and CoAP protocol are issued at a request rate of one request per 10 seconds, to a server node offering varying payload sizes between 2,4, 64 and 128 bytes. Figure 4.25 outlines the topology used for evaluating the energy consumption.

The power consumption was measured at the server node. Figure 4.26 presents the results of the simulation and shows that the CoAP-based message exchange has significant energy savings compared to the HTTP based interaction. This can be explained due to the expensive establishment of TCP-based communication for HTTP. The message payload size has not much effect on the HTTP-based interaction, since the initial communication effort is already high. In contrast, the increased payload size can be seen in the results for the CoAP protocol. As soon as fragmentation takes place additional communication effort is also visible for the CoAP protocol.



(a) 64 bytes, 1 hop

(b) 64 bytes, 2 hops

(c) 128 bytes, 1 hop

(d) 128 bytes, 2 hops

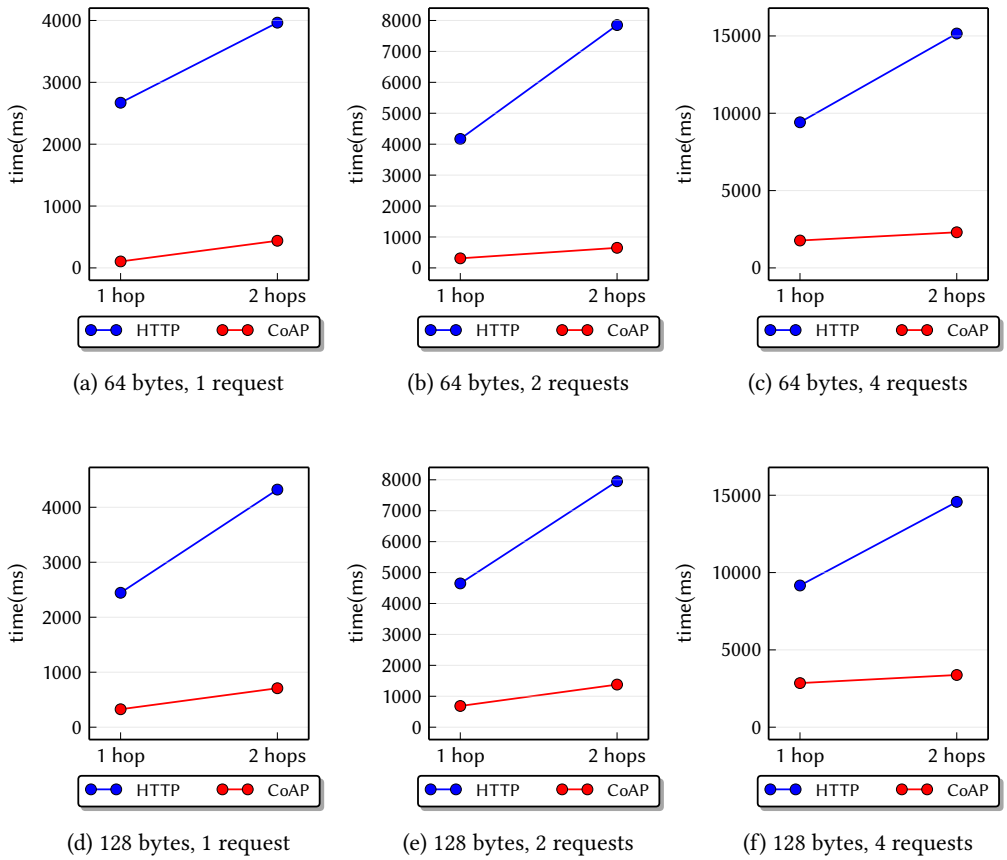
Figure 4.23: Average response time for HTTP and CoAP [131]

Conclusion

This section has compared Web-service-based communication based on HTTP and CoAP. The results show that HTTP cannot be considered for communication within WSANs since the scalability is not given due to the high average response time as soon the communication load is increased and further the energy consumption is too high.

CoAP scalability

The previous section shows that the scalability limits of HTTP are reached very early in a WSAN deployment and cannot be taken into consideration for Web service based communication. Therefore, this section investigates the scalability of the CoAP communication protocol. Especially the average response time depending on the number of hops within the network topology and the payload size are investigated. The scalability limits are analyzed regarding average response time limits and message failure rates. Finally, also an evaluation regarding the energy efficiency is conducted. The topology used for the simulation is illustrated in Fig-



(a) 64 bytes, 1 request (b) 64 bytes, 2 requests (c) 64 bytes, 4 requests

(d) 128 bytes, 1 request (e) 128 bytes, 2 requests (f) 128 bytes, 4 requests

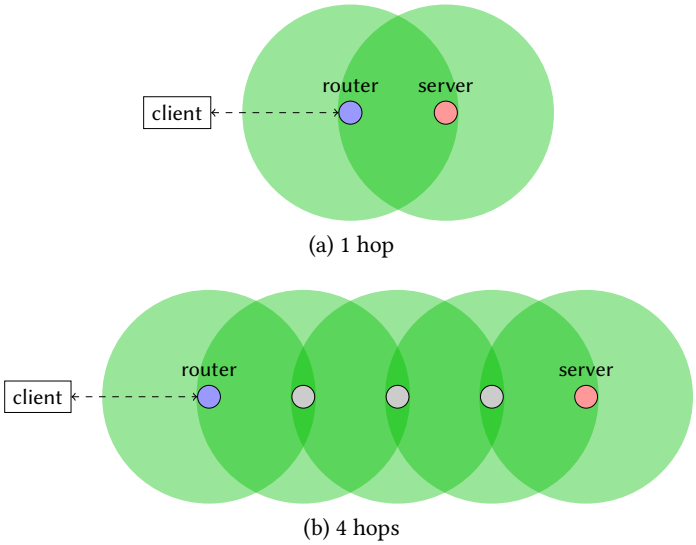
Figure 4.24: Average latencies for HTTP and CoAP, by hops [131]

ure 4.27.

Average response time

The effect of the number of hops and the payload on the average response time is presented in Figure 4.28. A major difference in the measured average response time can be observed if the payload is increased above 64 bytes which can be explained by the fragmentation of messages. The effect of increased hop counts shows a higher impact on the observed average response time. The effect is even increased if fragmentation takes place.

Message failure rate: The message failure rate starts to increase significantly after a server is requested that is more than 10 hops away from the client. After 14 hops it was nearly impossible to receive response messages from the server. This can be explained by the fixed response time threshold of 10 seconds. Responses that take more time than that time limit lead to an increase of message failures. The main reason for this is the use of radio-duty cycling which



(a) 1 hop

(b) 4 hops

Figure 4.25: Simulation topologies CoAP vs HTTP - power usage [131]

introduces latency between each hop. This leads to a severe delay, including retransmission of requests if confirmed CoAP communication is in use. This finally results in congestion within the network and response times that exceed the response time limit.

Energy consumption

The energy consumption depending on the number of hops and the payload size is given in Figure 4.30 on page 133. As soon as one more hop is involved, the energy consumption is increased but stays at a stable level. A more significant influence has the packet size. As soon as fragmentation and dropped messages need to be retransmitted an overall decrease in the energy efficiency can be seen.

Conclusion

The simulation results show that the performance of CoAP strongly depends on the number of hops between a client and server but much more on the payload size. Therefore, the payload size has to be kept as small as possible in order to keep the whole WSAN communication functional.

IoT group communication based on IPv6 multicasting

A central focus of the scalability evaluation is put on the IoT group communication mechanism. Therefore, different group communication scenarios are simulated. For the simulation, a group of 5 devices interacting with each other is considered. Figure 4.31 on page 134 provides an overview of the used simulation scenarios. One client sends a *put* request to 4 servers. In the first scenario, one communication group is considered where the servers are out of range

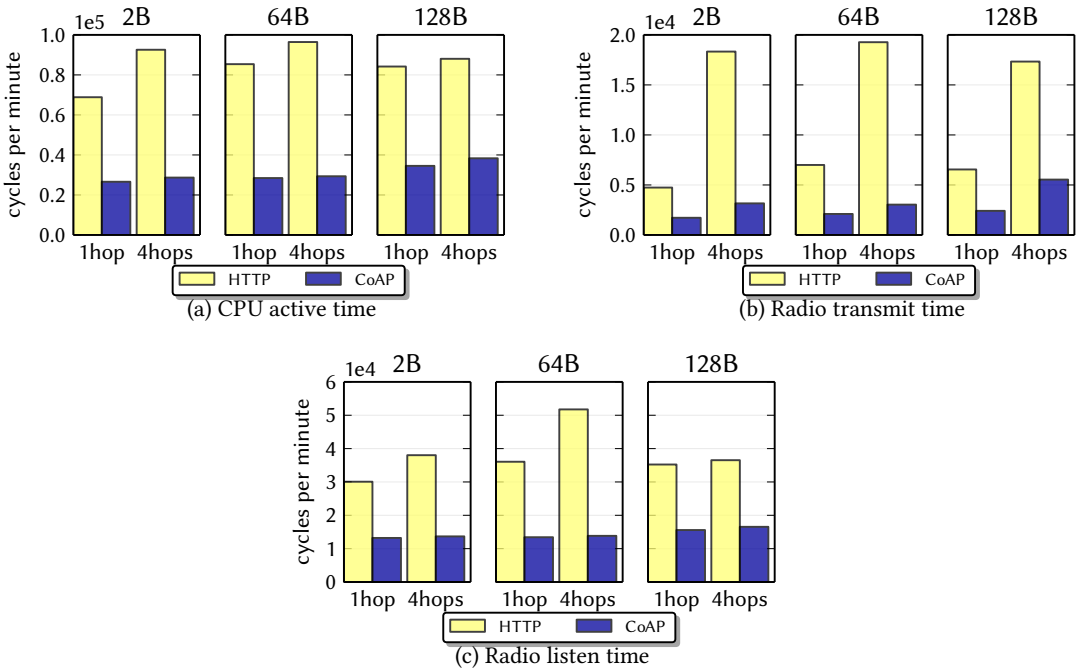


Figure 4.26: Power consumption HTTP vs. CoAP - RDC enabled [131]

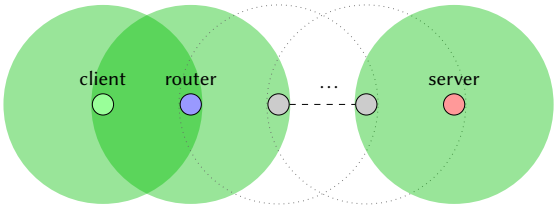


Figure 4.27: Simulation topology CoAP [51]

to each other. In the second scenario, the servers are in range of each other and also receive the message of neighboring servers. In the third scenario, three communication groups are considered. For the evaluation, unicast-based communication on dedicated client/server requests are compared to multicast-based group communication. The unicast-based communication represents the state of the art communication for CoAP-based communication and requires a blocking client request to each of the servers in the communication group. The multicast-based mechanism relies on the multicast mechanism using transient IPv6 site-local multicast addresses which only requires a single request to synchronize the state amongst all involved devices. The difference between the various multicasting routing algorithms is also taken into consideration.

A client is grouped with multiple servers into a communication group with servers having one intermediary node between. Within the WSAN, multiple communication groups co-exist. The client sends periodic updates to a dedicated group communication address used within

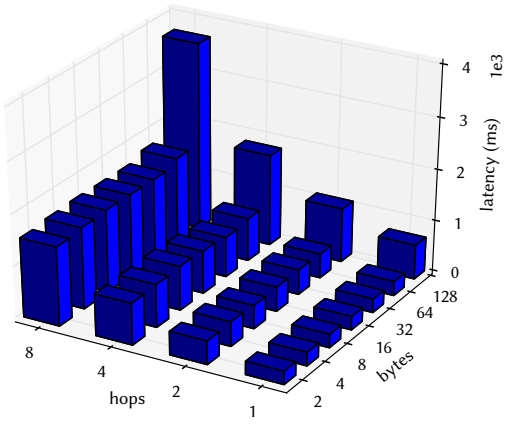


Figure 4.28: CoAP average response time depending on hops and payload size [51]

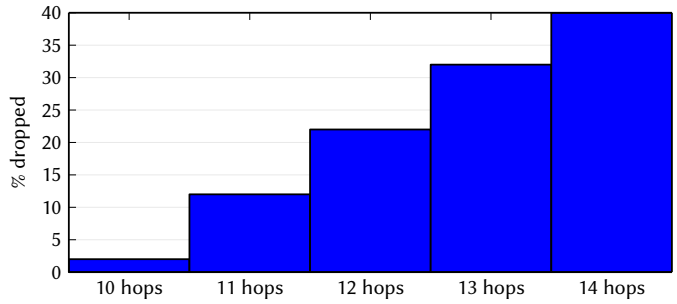
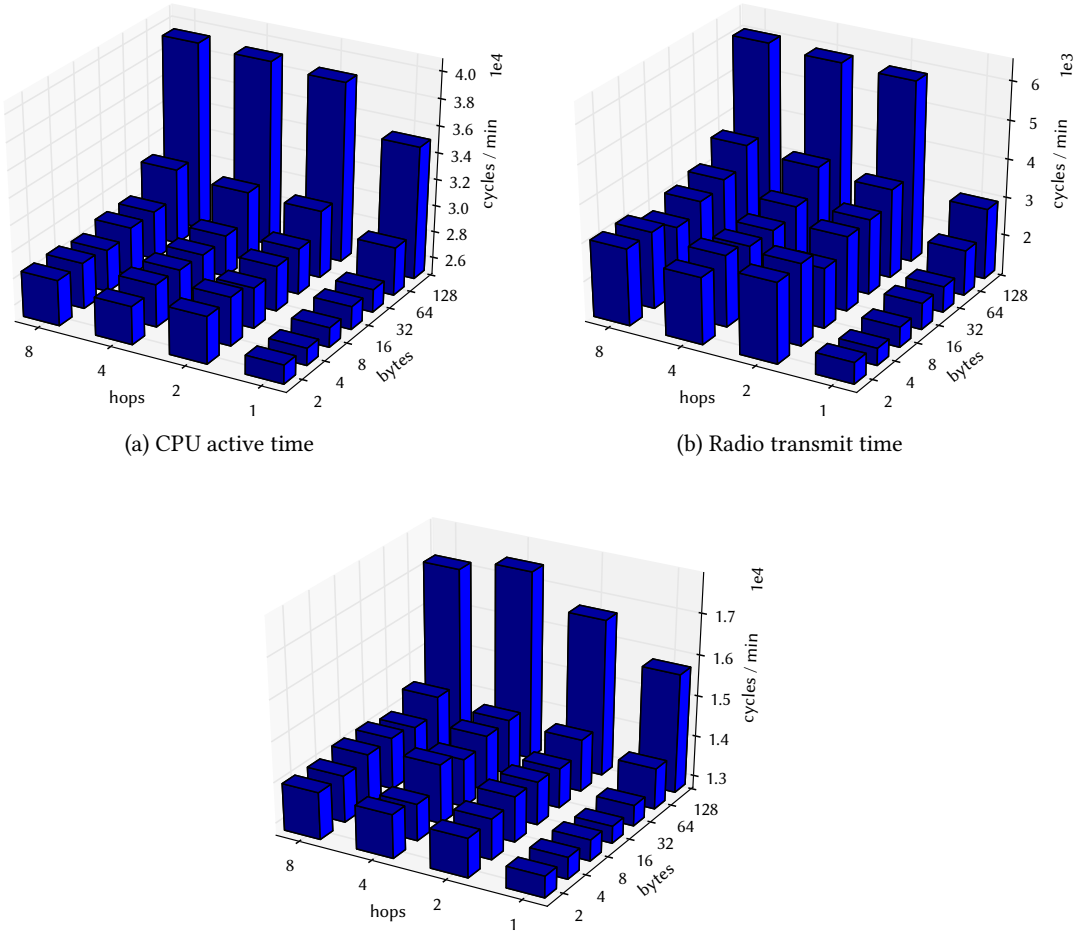


Figure 4.29: Dropped messages for CoAP - RDC enabled, 128B payload, 10s timeout [51]

the communication group. A concrete example can be a push button that sends a message to a number of switching actuators. For the evaluation, this scenario is realized first with unicast messaging based on the state of the art mechanism provided by the current implementation of Erbium. The main disadvantage is that the client has to send a unicast message to all receivers. In contrast, using the multicast-based interaction, only a single datagram needs to be transmitted by the client.

A 16 bytes payload is sent within a 10 seconds interval and the results are averaged over 5 repeated simulation runs. The latency represents the time when all servers have received the message and updated their internal resource state. MPL, Stateless Multicast Forwarding with RPL (SMRF) and LOWPAN_BC0 compared to unicast-based message exchange have been evaluated. As illustrated in Figure 4.32, the custom multicast-based interaction on LOWPAN_BC0 mechanism clearly outperforms the state of the art approach, which is based on unicast messages and also the other multicast routing algorithms. This can be explained by the fact that MPL is designed to distribute state information amongst a set of nodes in a periodic way. An example use case is the distribution of application firmwares within a WSAN. The performance is considerably bad due to the fact that every multicast message has to pass the MPL seed in



(a) CPU active time

(b) Radio transmit time

(c) Radio listen time

Figure 4.30: Energy consumption of CoAP based communication [51]

order to be distributed amongst all participating nodes. Further, the trickle timer has a strong impact on the perceived performance. For MPL, the I_{min} value is set to 125 ms. The same value is used for SMRF which shows a good performance in the taken simulation scenario. However, although SMRF works for this simulation scenario, a use case that involves the upward routing in a 6LoWPAN DODAG would not be possible. The best performance is shown by the flooding-based LOWPAN_BC0 mechanism, due to the direct forwarding to neighboring nodes, without any protocol dependent delay.

Further, the impact on the energy consumption is compared regarding the additional CPU utilization and the time the radio of a node is in transmit mode, both measured in CPU cycles. Figure 4.33a shows the differences regarding energy consumption between the unicast-based approach and the group communication using different multicast engines. According to the evaluation, SMRF followed by LoWPAN_BC0 behaves most energy efficient.

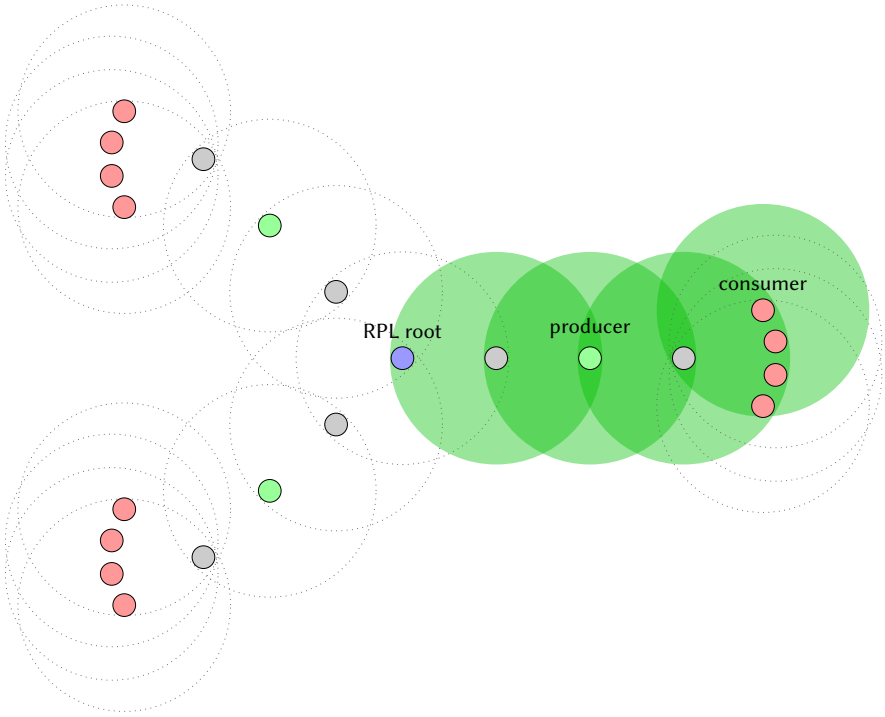


Figure 4.31: Simulated network topology [51]

Memory requirements

The impact of the proposed group communication mechanism using different multicast engines regarding ROM and RAM requirements of servers can be seen in Figure 4.33b. The implementation of the group communication based on LOWPAN_BC0 and SMRF shows only a minimal impact on the ROM and RAM requirements. MPL can be considered as the most heavy-weight solution.

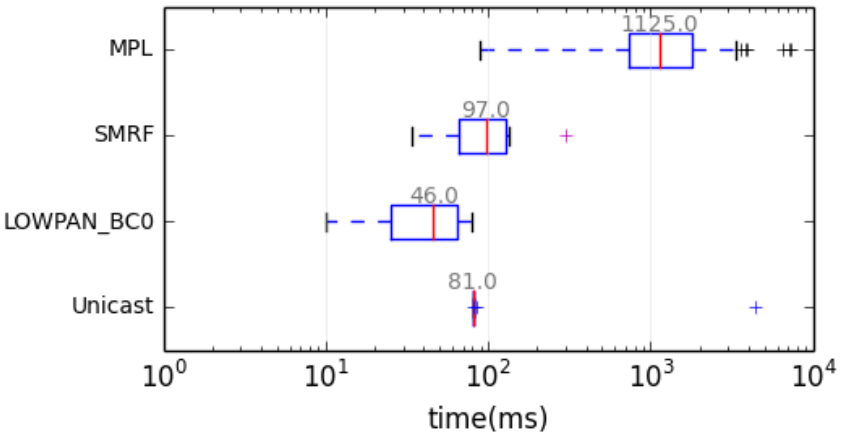
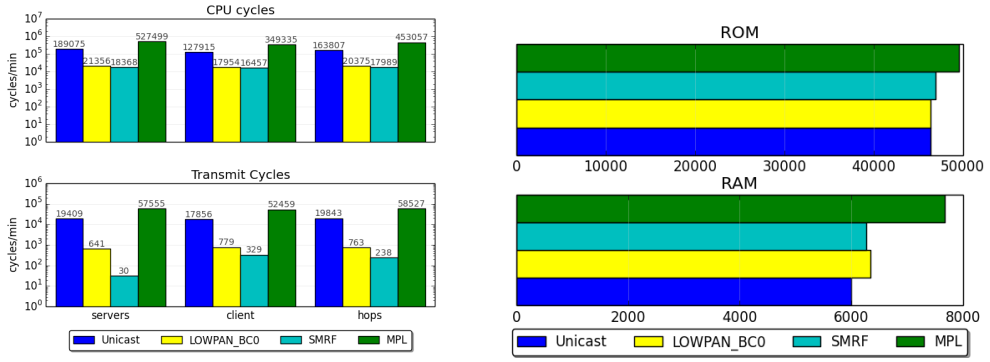


Figure 4.32: Comparing latency of unicast vs. multicast-based message exchange [51]



(a) Comparison of energy consumption

(b) Memory constraints

Figure 4.33: Multicast evaluation regarding energy and memory consumption [51]

4.12 Conclusion

Within this chapter a communication stack for the Internet of Things has been presented. Several requirements on the stack are identified such as interoperability, scalability, versatile interaction styles, energy efficiency, security and ease of use. A stack fulfilling these requirements is described following a bottom-up approach. The essential findings are that for media and data link wireless technologies IEEE 802.15.4 together with 6LoWPAN act as an enabler for a wireless Internet of Things. IPv6 offers global end-to-end connectivity combined with powerful multicasting capabilities that can be exploited by the application layer for peer-to-peer interaction styles. For message exchange and the application layer, CoAP together with OBIX provide an interoperable communication stack supporting the required application layer ser-

vices. A concept for a Web-based control logic engineering tool is presented, which shows how usability and an easy-to-use interface for interacting with IoT appliances and commissioning control logic can be provided. A proof of concept implementation together with a simulation-based evaluation illustrate that Web services using HTTP are too heavy weight for embedded appliances and constrained communication networks. Further, the scalability limits of CoAP based communication are identified and the performance of the presented group communication mechanism is evaluated. Here, also the impact of different multicast routing algorithm is investigated, where the different algorithms have their advantages and disadvantages regarding achieved minimal latency at the cost of energy efficiency.

CHAPTER 5

An IoT integration middleware

The IoT stack presented in the previous chapter can be used for smart objects that are directly integrated into the IoT. Within certain application domains such as building automation, the lifetime of appliances and devices is in the magnitude of decades. Therefore, a turnover to new IoT technologies will take time. Even in new buildings, due to the lack of available and mature IoT smart objects, it is quite likely that non-IP technologies will be in use for the next years. Also, a hybrid deployment scenario can be expected which will be in place until a full transition to an IoT-only scenario is achieved. In order to integrate non-IoT technologies, gateway devices will be required to translate between different technologies and protocols.

Although gateways might be an alternative to deploying IP at all, several reasons can be identified against their usage. The main problems or gateways are the inherent complexity and the lack of flexibility and scalability [22]. The complexity arises through the different protocol stacks coming with custom mechanisms for message exchange, routing, quality of service, transport, management and security. The flexibility and scalability problem occurs due to gateways becoming the bottleneck of the overall system. Nevertheless, gateways are for the first era of the IoT inevitable. Furthermore, as mentioned in the design of the IoT stack, some features can only be provided by devices with more computational power. Therefore, some type of gateway device will always be part of the architecture although their complex protocol translation functionality will become less relevant in the future.

This chapter will present an integration concept that provides a transparent integration of most relevant technologies and data sources such as wireless and wired home and building automation technologies, device identification technologies (e.g., RFID), smart meter protocols and other information sources such as weather data [4, 52, 132] into the IoT.

5.1 Requirements

The following subsections state the requirements of an IoT integration middleware.

Integration style

Typical BAS define several layers of the ISO/OSI reference model and contain a specific application model. The application models usually follow a *datapoint*-centric approach, meaning that every device is expressed as a collection of input and output datapoints of well defined data types. Datapoints are often grouped into functional blocks which define desired capabilities (e.g., light switch actuator with multiple channels). The meta data and semantics of functional blocks are provided in a human readable way. Furthermore, the application layer protocols are strongly aligned to the custom network and data link layers of the underlying BAS technology, which typically requires a variety of interaction styles like client/server, producer/consumer, or publish/subscribe communication. The network layer of a BAS is usually kept quite simple. However, reliable and non-reliable data transfer, point-to-point, multicast- or broadcast-based communication should be supported.

To integrate BAS into the IoT, it is not possible to simply put application layers of building automation technologies on top of a common network layer. These technologies usually define custom protocols for the transport and network layer. Instead, gateway devices are necessary. As shown in Fig. 5.1, these gateways may operate either following a N-to-N protocol mapping approach or may map all technologies to one protocol (R1) following a N-to-1 approach.

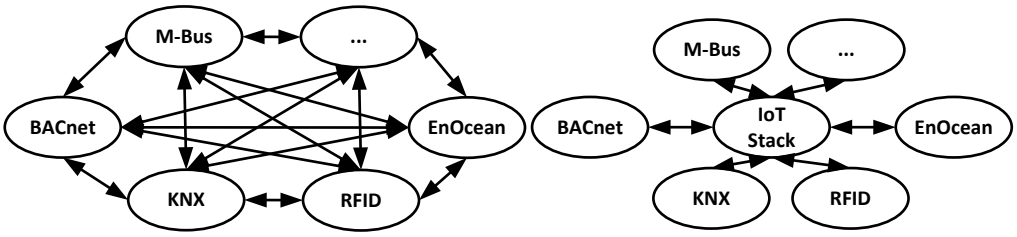


Figure 5.1: N-to-N or N-to-1 BAS integration

For N-to-N integration, two BAS are made compatible with each other through gateways featuring different network options. In the best case, the gateway is not visible for the other BAS system. If two BAS technologies rest upon the same network layer, multi-protocol devices may come into play as presented in [133]. However, for this kind of integration, $\frac{N*(N-1)}{2}$ mappings are required in the worst case.

The *N-to-1* integration approach refers to the integration of different building automation systems to a common target system. The mapping effort is reduced to N mappings, since all technologies have to be integrated only into one new technology. The problem remains to identify the commonly accepted technology stack.

Mapping communication principles

Figure 5.2 shows a *N-to-1* mapping of two BAS representatives (e.g., BACnet and KNX) to the IoT, where IPv6 acts as a common IoT network layer. An integration middleware needs to

map the various functionalities found on the different layers of the technologies to the features provided by a common stack. Some features might be lost or not easily representable.

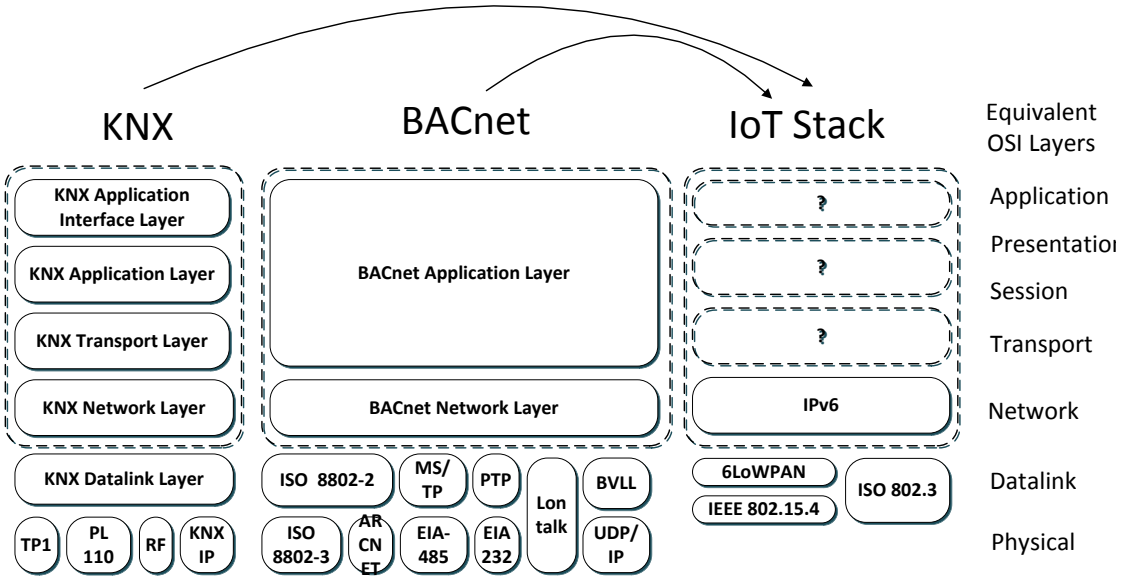


Figure 5.2: Integration challenges

Application layer service and information model

The various application layer services and information models need to be mapped by an integration middleware. Common application layer features such as datapoint-centric information representation (R2), support for histories (R3), alarming (R4) need to be provided.

A datapoint-centric information model is required with a type system for representing common signal types and information of automation systems. The information model capabilities need to offer a generic meta-model (R5) that can be used for representing all different technologies. An extension mechanism (R6) to specify domain and technology specific information models is required.

Naming and addressing

Naming is required to provide unique symbolic identifiers for entities and resources at the application layer. Depending on the BAS, a more or less fine-grained namespace is used. Some use hierarchical alphanumeric namespaces, whereas other technologies purely rely on numeric identifiers. Addressing deals with the identification of devices and communication partners at the network level within the BAS. Here, a sufficiently large address space (R7) needs to be used for an integration middleware.

Communication interaction patterns

The communication interaction patterns refer to the messaging facilities that are available in the various BAS. Typically, these systems have their own network and transport layer facilities supporting different interaction patterns. Most relevant communication interaction patterns include reliable point-to-point communication which is used for client/server communication (R8). Further, point-to-multipoint communication is required for group communication including a producer/consumer interaction pattern (R9).

Service discovery and metadata exchange

An integration middleware must use service discovery mechanisms of the integrated BAS in order to provide automatic resource provisioning (R10). For some technologies, run-time discovery services can be used, whereas for other technologies static project configuration files may be used and finally, only manual configuration might be applicable in some use cases.

Integration of Internet information sources

Several Internet information sources can be considered as relevant for the IoT. Weather data and forecasts (R11) are relevant for uses cases found in home and building automation and the Smart Grid. For example the expected solar radiation of the next hours or the expected wind allow forecasting the expected energy production through renewable energy data sources. If a bad weather front is expected, open windows or doors may cause a safety issue and might require the attention of a human person. The integration of RFID-based identification and according information systems (e.g., EPCIS) are highly relevant (R12) if logistics and building automation need to be combined. Further, the auditing and building maintenance process can be simplified through an integration.

Quality of service

An integration middleware must provide a certain quality of service. Here, especially the average response time of requests issued by external clients needs to stay within a certain limit (R13).

5.2 Integration middleware architecture

The integration middleware architecture addresses the heterogeneity of existing technologies in the IoT environment. For providing interoperability amongst heterogeneous technologies and also for new IoT devices directly communicating using IPv6, the IoT stack provides a common message format, message exchange protocol and vocabulary to ensure interoperability. Within the architecture several components are responsible to ensure multi-protocol interoperability. The gateway component, which can be deployed on an embedded PC or within a local control and monitoring system, provides an integration middleware for existing relevant technologies. It implements a mapping of the IoT stack to each technology that shall be integrated

and offers a communication interface according to the IoT stack for devices of these technologies. By adhering to this integration approach, a unified communication interface for the other components, such as cloud computing or mobile computing, can be provided. Figure 5.3 illustrates the overall architecture of the multi-protocol integration approach. Within the scope of the integration middleware are building automation technologies (KNX [134], BACnet [19], EnOCean [18]), Smart Grid technologies (W-MBus [135]), identification technologies (RFID, EPCIS) and Internet information sources (weather data).

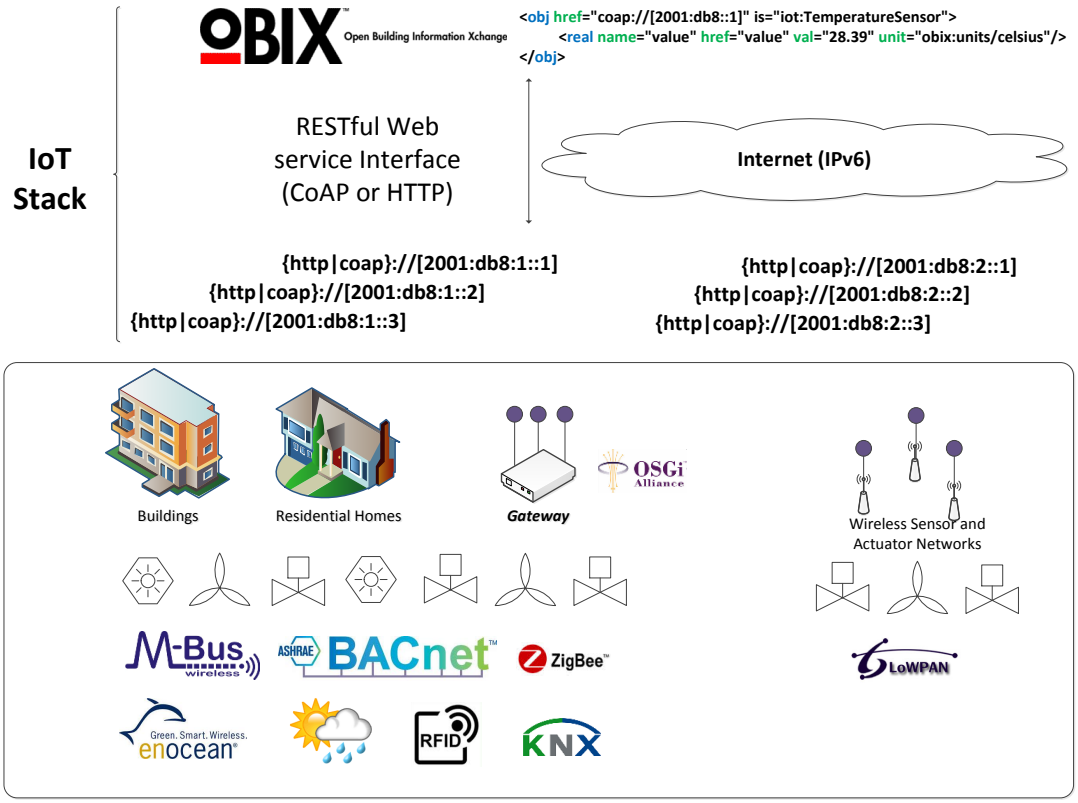


Figure 5.3: Multi-protocol integration architecture

Technologies considered within the integration architecture have been presented in Chapter 2. This section will focus on how the various communication principles and protocols can be mapped to the IoT stack. Further, it will be investigated how an automated mapping process can take place. Such a process allows an automated configured access to the various technologies through the IoT stack without the need for human supervision. Table 5.1 summarizes how the identified requirements are addressed by the integration middleware concept.

The following subsection contains details on the mappings of home and building automation technologies (KNX, BACnet, EnOCean), Smart Grid technologies (W-MBus), identification technologies (RFID, EPCIS) and Internet information sources (weather data) to the IoT stack.

Integration middleware requirement	Design decision
R1 N-to-1 integration	Integration mapping towards IoT stack
R2 datapoint-centric information model R3 support for histories R4 support for alarming R5 Generic meta-model	OASIS OBIX for standardized integration
R6 Extension mechanism for domain and technology information models	OBIX contracts
R7 Sufficient large address space	IPv6 addressing at the network layer URIs at the application layer
R8 Reliable point-to-point client/server communication	RESTful HTTP Web service communication
R9 Group communication following producer/consumer interaction pattern	RESTful CoAP Web services with IPv6 multicast
R10 Automatic resource provisioning	Automatic run-time service discovery Static service discovery Manual configuration depending on technology capabilities
R11, R12 Integration of Internet information sources (weather data, EPCIS)	IoT-stack mapping for weather data API and lightweight EPCIS interface
R13 Quality-of-service assurance	Analytic model to predict computational requirements

Table 5.1: IoT stack design decision fulfilling requirements

5.3 Integration of KNX

The interworking-model of KNX relies on the concept of group communication based on group communication objects (GO). These group communication objects are used by the end devices to exchange process data within the KNX network. In this way, communicating devices are linked together without the need of knowing a concrete communication endpoint. Figure 5.4 illustrates the KNX group communication concept.

If the so-called user application (firmware at device) recognizes a change on an input state the group object is updated. The network stack then checks all assigned group addresses to this group object and transmits a *group value write application service* request to the associated addresses using an unreliable network broadcast. The addressing scheme of KNX uses a 16 bit identifier, which is further sub-structured for organizational structuring of the address space. Other devices receive these updates and the network stack checks if there are associated group objects. If these associated group addresses are found, the state of the group object is updated and the user application performs the change on the output state. An important advantage of this communication mechanism is that multiple devices can interact with each other without

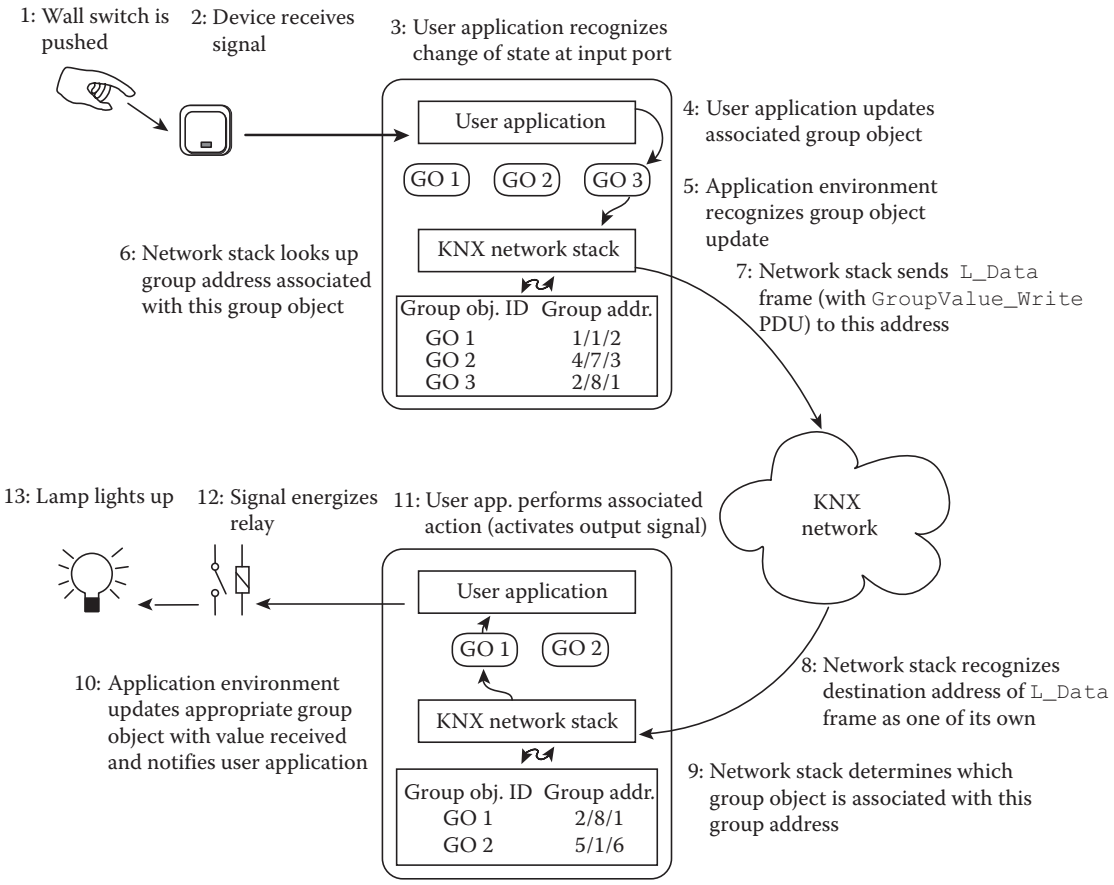


Figure 5.4: KNX interworking model [136]

the need of a centralized controller. Further, all the process related device interactions are independent from each other. So if one device fails, the communication of other related devices is not affected. However, this type of communication makes a mapping to a RESTful client/server interaction problematic. If a device-centric view on resources should be provided it might not be possible without side-effects, since there might be more than one device associated with the group address.

To realize a standardized set of device features, KNX uses the concept of function blocks. A function block defines a set of input and output datapoints that a device needs to implement in order to fulfil the contract defined by the function block. Typically, such input and output datapoints are related to the physical I/O signals of the device, but also so-called soft datapoints can be provided. For example, a setpoint value of a temperature controller is realized as such a datapoint that provides access to a variable in memory. The function block of a device provides a good starting point for a resource-centric representation of a KNX device. In this case, the OBIX contract mechanism can be used to define similar contracts like KNX function blocks as illustrated in Figure 5.5. The OBIX object provides a stateful representation of the associated KNX group objects. Therefore, it is required to keep the stateful representation synchronized

with the bus events that occur.

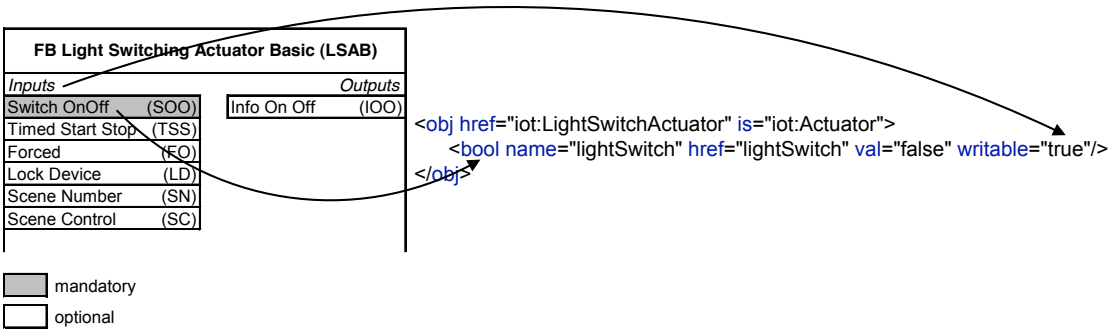


Figure 5.5: Mapping KNX function blocks to OBIX contracts

Besides the group communication interaction pattern, it is also possible to have reliable point-to-point communication. However, this type of communication is not for runtime process data exchange and only used for firmware download and configuration and commissioning purpose.

Mapping KNX datapoint types

To map the KNX bus system, the functional blocks provide a device centric view. For the associated datapoints, a mapping of the KNX datapoint types needs to be defined.

Datapoint Types B_1		
Format:	1 bit: B_1	
Range:	b = {0,1}	
Unit:	None	
Datapoint Types		
ID:	**Name:**	**Encoding: b**
1.001	DPT_Switch	0 = Off 1 = On
1.002	DPT_Bool	0 = False 1 = True
1.003	DPT_Enable	0 = Disable 1 = Enable
1.004	DPT_Ramp	0 = No ramp 1 = Ramp

Figure 5.6: KNX datapoint type main type 1 [17]

A KNX data type includes a format and encoding information, and a dimension with a range and unit. Figure 5.6 provides an example of a KNX data type that can be used to represent a simple *boolean* datapoint. For the main datapoint type, the information is encoded using a 1 bit representation with the given format. Further, the semantics of a “1” or “0” state are fully specified through the subdatapoint types. For example, a logical “1” or “0” can either have the

meaning associated of “on” or “off”, or “enabled” or “disabled”, “open” or “closed”. The concrete semantics is specified through the KNX standard.

The KNX main types and formats can be directly mapped to the OBIX value datapoint types (cf. Table 5.2). Range and unit information can be provided using a standard unit attribute provided by OBIX and the use of “enum” objects to represent the range and also the semantic information about the encoding.

KNX datapoint type	OBIX base type
Boolean Value (DPT_B)	bool
Unsigned Value (DPT_U)	int
Float Value (DPT_F)	real
Character (DPT_A)	string
String (DPT_A)	string
Time	abstime
Date	abstime

Table 5.2: Mapping KNX datapoint types to OBIX value object types [137].

A more sophisticated mapping is shown in Listing 5.1 which illustrates how the KNX main and subtypes can be represented using OBIX contracts and further semantics can be preserved by representing the encoding.

Listing 5.1: KNX datapoint mapping with encoding information

```
<obj name="P-0341-0_DI-3_M-0001_A-9803-03-3F77_O-3_R-4" href="switch_channel_a/"
      is="knx:DPST-1-1 knx:DPT-1 knx:Datapoint" display="On / Off"
      displayName="Switch, Channel A">
      <bool name="value" href="switch_channel_a/value" val="false"
            displayName="On / Off" null="true" writable="true" />
      <enum name="encoding" href="switch_channel_a/encoding" val="off"
            null="true" writable="true" range="/encodings/onoff" />
</obj>

<list href="onoff/" of="obix:bool" is="obix:Range">
      <bool name="on" href="onoff/on" val="true" displayName="On" />
      <bool name="off" href="onoff/off" val="false" displayName="Off" />
</list>
```

Automatic run-time discovery

KNX does not provide any discovery services that allow an automatic discovery of KNX devices at the bus during runtime. Without a priori knowledge of the available devices, their installed application firmware and group address relationships, it is usually not possible to derive the semantics of messages that are exchanged on the bus. Only the length of the binary exchanged application payload allows anticipating the type of communication. Due to the ambiguity of the semantics only the knowledge of a human observer could provide the required meta-information about the configuration.

Discovery based on KNX project configuration

For KNX, the engineering tool ETS offers the means to configure devices, to download the application firmware on devices and to configure communication endpoints. Therefore, all required information to create an automated mapping to the IoT stack is available within the ETS project file. The ETS allows exporting an XML-based description of the KNX bus system that can be used for automated configuration of the gateway.

5.4 BACnet

BACnet is a prominent representative of a building automation technology for commercial buildings. By having a collapsed protocol architecture of 4 layers (as shown in Figure 5.7) it is tailored and optimized to the specific needs of a building automation control network.

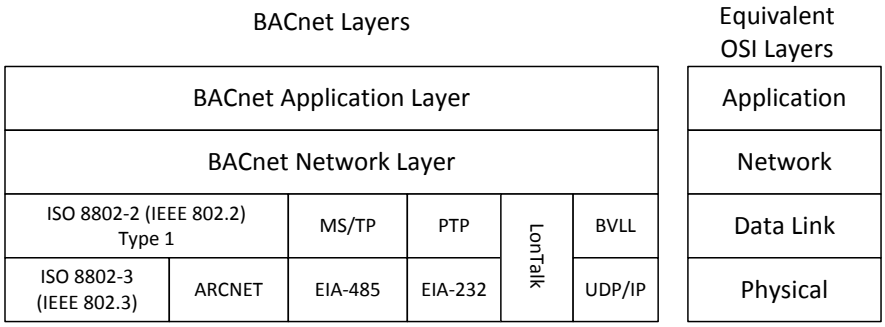


Figure 5.7: BACnet collapsed architecture

The BACnet application layer uses an object-oriented information model to offer access to the control network. The application layer services provide all required functionalities for inter-process and end-to-end communication between BACnet enabled devices. Every information asset or interface is modeled in an object-oriented way. Each BACnet device holds one or more BACnet objects. Every object can contain a list of standardized properties. A set of standardized property and object types is available. Further, a property might be optional, mandatory but read-only, or mandatory and read as well as writeable. A minimum set of required properties are the *Object_Identifier*, the *Object_Name* and the *Object_Type* which must be provided by each BACnet object. The BACnet standard defines an enumeration of the available property identifiers.

BACnet follows an object-oriented approach in modeling its devices and datapoints. A BACnet device can be seen as a collection of objects. Every device must have the special "device"-object that provides additional information about itself, including its device identifier, its name, and a list of all objects available on the device. The BACnet standard defines a number of object types. These objects can represent physical points or describe processes or internal operations. The *Program* object type, for example, represents a process running within a BACnet device. Physical points can be modeled using generic object types for binary and analog values.

There are also separate object types for input, output and value objects, resulting in a total of 6 object types: *AnalogInput*, *AnalogOutput*, *AnalogValue*, *BinaryInput*, *BinaryOutput* and *BinaryValue*. Input objects receive their value from an external source (e.g., sensors) and therefore are not writeable. Output objects are writeable and represent control outputs. Value and output objects can be used as setpoints e.g. the target temperature in a heating system. Every object has a collection of properties. Among these properties are name and type of the object, as well as a description. For analog objects, a property provides a way to specify the units of the value. There are a number of other properties, and each object type has its own set of properties that are useful for that type.

Present value property: An important property is the *Present_Value*. If this property is writeable, then the properties *Priority_Array* and *Relinquish_Default* are also present on the object. These properties are part of the prioritization mechanism. The priority array is a read-only array property of 16 values which correspond to 16 available levels of priority. The elements are in order of decreasing priority, so the first element (priority 1) has the highest priority. An entry in the priority array can be either a value or *NULL*. The non-*NULL* value with highest priority gets mapped to the *Present_Value* property. If there is no non-NULL value, then the value of the *Relinquish_Default* property is used. To set entries in the priority array, a *WriteProperty* request including the priority to override is issued on the *Present_Value* property. To clear entries, the value of the *WriteProperty* request shall be *NULL*. If no priority is specified on the request, a default priority of 16 (the lowest priority) is assumed.

Object identifier: Objects within a device are identified through their object identifier. This identifier consists of two parts: The object type and an instance number. In order to uniquely identify a property of an object inside a BACnet network three values are required: the device identifier of the device, the object it is residing on, the object identifier (object type and instance number) and the property identifier.

Mapping BACnet object types

A way to represent a BACnet network and its devices and objects inside an OBIX server is desired in order to be able to access and manipulate them through an OBIX representation. To achieve this, BACnet objects can be mapped to OBIX objects. Within this thesis, a closer look on the the mapping of the BACnet object types *AnalogInput*, *AnalogOutput*, *AnalogValue*, *BinaryInput*, *BinaryOutput* and *BinaryValue* to OBIX objects is done, as they are sufficient to model a wide variety of devices and applications, including sensor, actuator values and setpoints. The object hierarchy is illustrated in Figure 5.8.

Data types: First, the mapping of data types is examined. The group of analog objects in BACnet is of the data type real, which maps directly to the *real* OBIX object type. BACnet binary objects are of the data type *BACnetBinaryPV*, which can take the values active and inactive. These values map to true and false of the OBIX object type *bool*. To create a functional mapping, the present value is the only mandatory property. Therefore, it is possible to define

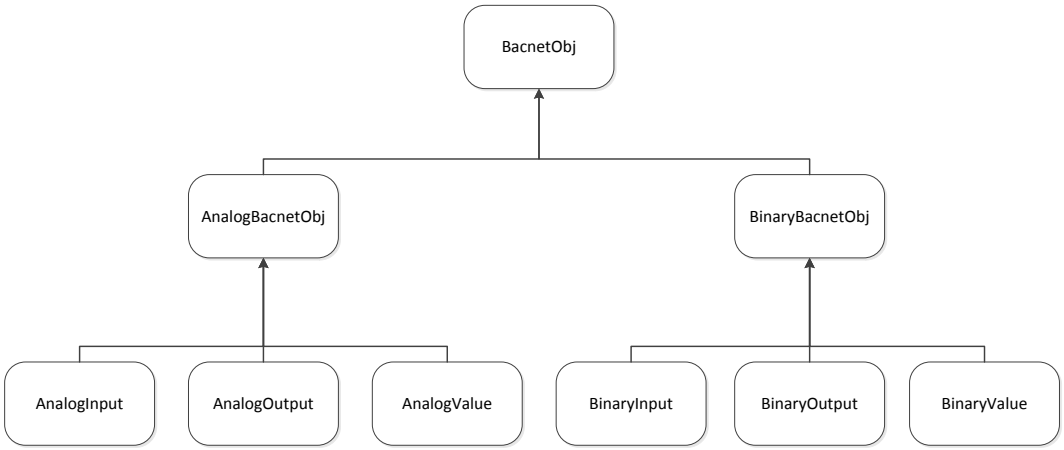


Figure 5.8: OBIX BACnet object hierarchy

basic OBIX contracts corresponding to the BACnet objects. These contracts are shown in Listing 5.2. BACnet input objects are by default not writable, while output objects are writable by default. This is reflected in the contract definition.

Listing 5.2: Basic OBIX contracts corresponding to BACnet object types

```
<obj href="iot:AnalogInput">
      <real name="value" val="0" writable="false" />
</obj>
<obj href="iot:AnalogOutput">
      <real name="value" val="0" writable="true" />
</obj>
<obj href="iot:AnalogValue">
      <real name="value" val="0" />
</obj>
<obj href="iot:BinaryInput">
      <bool name="value" val="false" writable="false" />
</obj>
<obj href="iot:BinaryOutput">
      <bool name="value" val="false" writable="true" />
</obj>
<obj href="iot:BinaryValue">
      <bool name="value" val="false" />
</obj>
```

Name: Every object has an *Object_Name* property. A name is unique within the BACnet device that contains the object. In OBIX, the direct children of an object must have unique names, too. Names in BACnet are restricted to printable characters only, and a minimum length of one character. OBIX imposes stricter constraints on names and only allows ASCII letters, digits, underbars, and dollar signs. Additionally, a digit must not be used as first character. Invalid characters have to be stripped from the *Object_Name* before it can be used as name

for the OBIX object. The problem of duplicate names from different BACnet objects is largely avoided by structuring the OBIX representation as discussed in the next chapter.

Description: BACnet also provides a description property. This property can be useful to understand the purpose of the object and can be included in the constructed OBIX object. To this end, a new child object of type *str* is added to the object. The name of the object is "description" and its value is the value obtained from the description property by a *ReadProperty* service request. No conversion of the obtained string is required, as it is only restricted to printable characters.

Units: A unit is required to interpret analog values correctly. Both BACnet and OBIX provide means to specify the units for a value. In BACnet, the *Units* property of the data type *BACnetEngineeringUnits* represents the units of the *Present_Value* property. *BACnetEngineering-Units* is an enumeration of many units from different domains, such as acceleration, area, currency, electrical, energy, enthalpy, entropy, force, frequency, humidity, length, light, mass, mass flow, power, pressure, temperature, time, torque, velocity, volume, volumetric flow and others. *Units* with an enumerated value in the range 0-255 have been reserved for definition by ASHRAE, values in the range 256-65535 may be used freely. OBIX features a flexible way to define units mathematically. Dimensions are specified using the *obix:Dimension* contract using the seven fundamental SI units and their exponent (Listing 5.3).

Listing 5.3: OBIX dimension contract

```
<obj href="obix:Dimension">
      <int name="kg" val="0" />
      <int name="m" val="0" />
      <int name="sec" val="0" />
      <int name="K" val="0" />
      <int name="A" val="0" />
      <int name="mol" val="0" />
      <int name="cd" val="0" />
</obj>
```

An actual unit is represented with the *obix:Unit* contract (Listing 5.4). It contains a dimension that can be scaled and offset (unit = dimension * scale + offset) and the unit symbol. Listing 5.5 shows how a kilowatt can be expressed using this system.

Listing 5.4: OBIX unit contract

```
<obj href="obix:Unit">
      <str name="symbol" />
      <obj name="dimension" is="obix:Dimension" />
      <real name="scale" val="1" />
      <real name="offset" val="0" />
</obj>
```

Listing 5.5: KiloWatt as OBIX unit

```
<obj href="obix:units/kilowatt" display="kilowatt">
      <str name="symbol" val="kW" />
      <obj name="dimension">
            <int name="m" val="2" />
```

```
            <int name="kg" val="1" />
            <int name="sec" val="-3" />
      </obj>
      <real name="scale" val="1000" />
</obj>
```

Some units do not fit into this model, like logarithmic units or units dealing with angles. Such units should use a dimension where every exponent is set to zero. OBIX provides a database of predefined units. If possible, *BACnetEngineeringUnits* should be mapped to the corresponding unit in this database. New units can be defined. For example, BACnet revolutions-per-minute (with an enumerated value of 104) maps to *obix:units/revolutions_per_minute*. BACnet has a special enumerated value representing the absence of a unit called no-units. This value can be mapped by simply omitting the unit attribute of the OBIX object.

Complete OBIX object representation: By mapping these additional properties, a much more meaningful object is obtained as illustrated in Listing 5.6.

Listing 5.6: Complete BACnet object contract

```
<obj name="lfan" href="/BACnet/10003/AnalogOutput1" is="iot:AnalogOutput">
      <real name="value" href="value" val="4200"
            units="obix:units/revolutions_per_minute" writable="true" />
      <str name="description" href="description" val="left fan speed setpoint" />
</obj>
```

Mapping of read and write requests

Assuming the address of a BACnet object is known, the appropriate OBIX contract can be chosen for an OBIX object since the object type is part of the address. To get the current value of the BACnet object when reading a mapped OBIX object, the OBIX read request has to be mapped to the BACnet *ReadProperty* service. The *ReadProperty* service takes three arguments: object identifier of the object to read from, the property identifier of the property to be read, and optionally a property array index. The object identifier is already known. The property identifier to choose is the identifier of the *Present_Value* property. The property array index is only needed if the property being read is an array. As the present value is a single value this argument is omitted. The response of the service contains the read property value. Its data type is either *REAL* or *BACnetBinaryPV*, depending on the object type. The value can be interpreted as previously discussed. For write requests, a similar mapping has to be defined. The write request maps to the *WriteProperty* service. This service takes five arguments: the object identifier, property identifier, property array index, property value and priority. For the identifier, property identifier and property array index, the same considerations described for the *ReadProperty* service apply. The property value argument contains the value we want to write to the object. As priority an integer in the range 1-16 can be chosen. If priority is omitted, a priority of 16 (lowest priority) is implied. The highest priorities should be reserved for life safety emergency overrides, so a mid-range priority of around 10 is suggested. The priority mechanism allows to override a value with a lower priority, later revoke the new high priority value and return to the original value. To revoke a value with a certain priority, its entry in the

priority array has to be reset to *NULL*. OBIX does not have a null object, instead it uses a facet or attribute to indicate a null value. Therefore, writing to an OBIX object with a null value can be used to reset its value. In this case, the BACnet *NULL* data type is the property value to be used in the *WriteProperty* service request.

The contract definitions already set the default of the writeable facet for each object type. However, under certain circumstances the present values of *Input* and *Value* objects can become writeable. In order to correctly reflect this in the OBIX object, these conditions have to be checked. Output objects are always writeable. Input objects are writeable, if the property *Out_Of_Service* is set to *TRUE* or if the *Present_Value* property is commandable, otherwise they are read only. The boolean property *Out_Of_Service* indicates whether physical input to the object is currently in service. If the *Present_Value* is commandable, then the properties *Priority_Array* and *Relinquish_Default* are both present on the object, too. To check if these properties are available, a *ReadProperty* service request can be attempted. If they are not available, the request will result in an *UNKNOWN_PROPERTY* error, indicating that the *Present_Value* property is not commandable.

These conditions have to be repeatedly checked every time the OBIX object is read, as they can change over time. Using the techniques described so far, a BACnet object can be mapped to a functional OBIX object that can be read and written. An example of an object using this mapping is shown in Listing 5.7.

Listing 5.7: Example OBIX object representing a BACnet Analog Output object

```
<obj href="/BACnet/10003/AnalogOutput1" is="iot:AnalogOutput">
      <real name="value" href="value" val="100" writable="true" />
</obj>
```

Automated mapping of BACnet devices

BACnet remote device management services: These services provide a number of functions, including operator control and auto-configuration functions. Among them two can be used to discover devices, the *Who-Is* and *Who-Has* services. They eliminate the requirement to program the network addresses of other devices into each device. The service messages are broadcast in the BACnet network to every device, and the receiving devices may respond with an acknowledgement message containing their address.

Who-Is and I-Am: The *Who-Is* service is used to determine the device identifier, the network address, or both of other BACnet devices that are on the same network as the device issuing the service request. It is an unconfirmed service, meaning that it does not require a response. There are multiple ways of using the service. It can be used to determine the device object identifier and network addresses of all devices on the network. If a device object identifier is already known, then the service can be used to determine the address of the corresponding device. The *I-Am* service informs other devices about its sender. It broadcasts an unconfirmed request containing the device identifier among other information. The service may be used at any time. Usually, an *I-Am* request is sent after a device has initialized to inform other devices about its availability. When a device receives a *Who-Is* request whose parameters include the

device, then it answers with an *I-Am* service request. Another way to locate devices on the network is provided by the unconfirmed services *Who-Has* and *I-Have*. The *Who-Has* service is similar to the *Who-Is* service. It can be used to ask for the device identifier of devices that contain an object that has a given object name or object identifier. In response, devices that have the requested resource send an *I-Have* service request, containing the device identifier, as well as both the object name and object type of the requested object.

Device instance ranges: Optionally, a range of device instances can be specified as an argument for the *Who-Is* and *Who-Has* service requests. If the range is omitted, then every device that receives the request will process it. Otherwise, only devices whose device object's instance number are within the specified range will answer. In large networks, an unbound *Who-Is* request to all devices at once would result in a lot of answers at the same time and a sudden network load. As a consequence, packets containing the *I-Am* response are more likely to be dropped on their way to the requesting device, and since the *I-Am* service is unconfirmed, the packets will not be resent. This may lead to not all devices being discovered. The device instance ranges provide a solution to this problem. Instead of one *Who-Is* request to all devices, the request can be split into several smaller requests, thereby reducing the network load per request.

Device objects and object lists: Every BACnet device is required to have a device object. Its instance number is the same as the device identifier. The device object offers a property called *Object_List*. It is an *Array* type property that contains all objects available on the device. This makes it possible to query all the objects contained in a device which has previously been discovered by the *Who-Is* service.

5.5 EnOcean

EnOcean is a wireless automation technology that is gaining importance, due to the fact that it relies mainly on energy harvesting mechanisms to power its sensors. The energy can be harvested from mechanical processes, like pressing a button, or by retrieving the energy from solar panels. The EnOcean alliance stands behind the technology and it aims at establishing an international standard. Within ISO/IEC 14543-10, the physical, data link layer and network layer are standardized. The stack is illustrated within Table 5.3.

The EnOcean specification covers multiple parts as outlined in Figure 5.9. There is the EnOcean Radio Protocol (ERP), which specifies the frame format for radio transmissions. Further, there is a specification for the communication between EnOcean RF modules and hosts based on the EnOcean Serial Protocol (ESP).

Most relevant for interoperability amongst different devices and vendors is the EnOcean Equipment Profile (EEP), which defines the structure of exchanged application data. The characteristics of a device are specified through the ERP radio telegram type (*RORG*), the basic functionality (*FUNC*) of the data content and the type of the device and its individual characteristics (*TYPE*). The EEP is identified through a 21 bit number structured in the way as illustrated in Figure 5.10 and expressed using hexadecimal numbers.

Wireless short-packet protocol (WSP) stack			
Standard	**Layer**	**Services**	**Data units**
Not defined in this standard	Application		
	Presentation		
	Session		
	Transport		
ISO/IEC 14543-3-10	Network	Destination addressed telegrams (Encapsulation/Decapsulation) Switch telegram conversion (RORG and STATUS processing) Repeating (STATUS processing)	TELEGRAM
	Data Link Layer	Sub telegram structure Hash algorithm Sub telegram timing Listen before talk	SUBTELEGRAM
	Physical	Encoding/Decoding (INV and SYNC) Wireless receiving/transmitting)	BITS/FRAME

Table 5.3: EnOcean specification

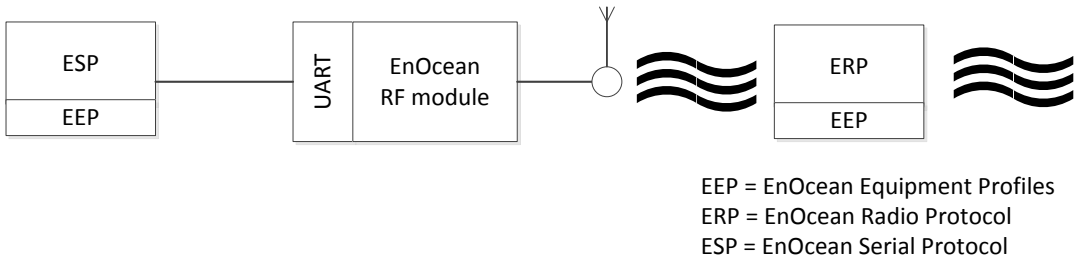


Figure 5.9: EnOcean specification overview

Due to the constraints imposed by the required energy efficiency the protocol is kept quite simple. It is the goal to keep these profiles as generic as possible in order to avoid vendors need to add additional profiles, which would decrease the overall interoperability.

The standard defines several telegram types as given in Table 5.4, where the telegram types for repeated switch communication (RPS), 1 byte communication (1BS), and 4 byte communication are the most relevant for the mapping to the IoT stack. For the RPS and 1BS telegram only 1 byte of user data is available. The rest of the telegram consists of a 1 byte telegram type identifier, a 4 byte sender identifier and a 1 byte status field.

For example, the EEP code F6-02 is used for rocker switches with 2 rockers. The type field is then used to further distinguish between different application styles, like whether a blind control is included or not. D5-00 is used for simple contacts and switches. Only one concrete EEP exists here, which is D5-00-01 (single input contact). For temperature sensors, the A5-02

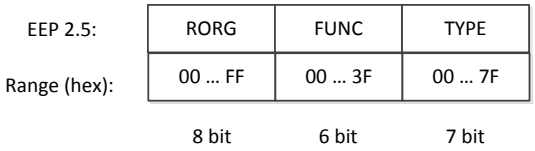


Figure 5.10: EnOcean Equipment Profile (EEP) structure

Telegram	**RORG**	
RPS	F6	Repeated Switch Communication
1BS	D5	1 Byte Communication
4BS	A5	4 Byte Communication
VLD	D2	4 Variable Length Data
MSC	D1	Manufacturer Specific Communication
ADT	A6	Addressing Destination Telegram
SM_LRN_REQ	C6	Smart Ack Learn Request
SM_LRN_ANS	C7	Smart Ack Learn Answer
SM_REC	A7	Smart Ack Reclaim
SYS_EX	C5	Remote Management
SEC	30	Secure telegram
SEC_ENCAPS	31	Secure telegram with R-ORG encapsulation

Table 5.4: EnOcean telegram types

group of EEPs, is another example. Here, the specific types categorize the temperature range. A5-02-01 is for temperature sensors with a range between -40°C and 0°C, whereas A5-02-01 identifies a temperature sensor with a range between -30°C and +10°C.

Mapping EnOcean equipment profiles

EnOcean does not come with a strong definition of an information model. Instead, every EEP has its own definition for interpreting the information conveyed in the telegram. The information is structured into a valid range, a scale and unit or into an enumeration of a fixed value definition for the binary code. Table 5.5 shows an example definition of an EEP for a temperature sensor. Here, the first identifier A5 refers to a 4 byte telegram, and the subidentifier 02 groups all temperature sensors together. The subidentifier 01 refers to the type, which is in this case specifying the range of the temperature sensor. The other types within the temperature sensor subgroup just specify other scales for the 1 byte data field representing the temperature. For this specific EEP, a lot of bits are unused, but other types add further information, such as for example a humidity field in the case the sensor support temperature and humidity as well.

Table 5.6 gives an example for another EEP, which is a contact sensor. The telegram type foresees 1 byte for user data. For representing whether the contact is opened or closed, a one bit field is used. The exact semantics of the 1 bit field is represented in the enumeration, which is custom defined within every EEP.

For the mapping of the EnOcean information model to the IoT stack, either the IoT con-

Size	Bit range	Data	ShortCut	Description	Valid Range	Scale	Unit
16	DB3.7 ... DB2.0	Not used (=0)					
8	DB1.7 ... DB1.0	Temperature	TMP	Temperature (linear)	255...0	-40...0	°C
4	DB0.7 ... DB0.4	Not used (=0)					
1	DB0.3	LRN Bit	LRNB	LRN Bit	Enum: 0: Teach-in telegram 1: Data telegram		

Table 5.5: A5-02-01 Temperature Sensor Range -40°C to 0°C

Size	Bit range	Data	ShortCut	Description	Valid Range	Scale	Unit
1	DB0.3	Learn Button	LRN	...	Enum: 0: pressed 1: not pressed		
1	DB0.0	Contact	CO	...	Enum: 0: open 1: closed		

Table 5.6: D5-00-01 Contact Sensor

tracts or an EnOcean specific mapping can be used. Due to the lack of a generic standardized information model it is only possible to map every EEP type to a custom OBIX contract, representing this EEP. Listing 5.8 shows an OBIX representation of the D5-00-01 EEP.

Listing 5.8: EnOcean

```
<obj name="EasyclickWindowContact" href="EasyclickWindowContact/" is="enocean:
    Entity" display="null" displayName="Window contact" writable="true">
      <str name="manufacturer" href="EasyclickWindowContact/manufacturer" val="
          PEHA"/>
      <list name="datapoints" href="EasyclickWindowContact/datapoints" of="obix:
          ref enocean:Datapoint">
        <obj name="SingleInputContact" href="EasyclickWindowContact/datapoints
            /single_input_contact" is="enocean:DPTBoolOpenClosed enocean:
            DPTBool enocean:Datapoint" display="Open/Closed" displayName="
            Single Input Contact">
          <bool name="value" href="EasyclickWindowContact/datapoints/
              single_input_contact/value" val="true" null="true"/>
          <enum name="encoding" href="EasyclickWindowContact/datapoints/
              single_input_contact/encoding" val="closed" null="true"
              range="/encodings/openclosed"/>
        </obj>
        <obj name="TeachIn" href="EasyclickWindowContact/datapoints/
            teachin_mode" is="enocean:DPTBoolOnOff enocean:DPTBool enocean:
            Datapoint" display="On/Off" displayName="TeachIn mode">
          <bool name="value" href="EasyclickWindowContact/datapoints/
              teachin_mode/value" val="false" null="true"/>
          <enum name="encoding" href="EasyclickWindowContact/datapoints/
              teachin_mode/encoding" val="off" null="true" range="/
              encodings/onoff"/>
```

```
            </obj>
        </list>
</obj>
```

The EnOcean telegram type can be seen as equivalent to the KNX main type, since both represent the number of transmitted bits. However, for KNX the main type is usually associated with a single datapoint. For EnOcean multiple datapoints might be encoded within the telegram, which makes a more generic mapping infeasible. Here, the mapping has to be done separately for each EEP.

Auto discovery

For creating the mapping of an EnOcean installation, an automatic discovery of devices is not possible. Due to the energy constraints, a sender might be only identified at runtime with its sender ID. Further, the telegram type can be automatically inferred but the concrete semantics require the knowledge of the EEP which is not contained in the frame.

5.6 Wired and wireless M-Bus

Smart grids are an important application scenario for the IoT. A popular communication protocol for automated meter read-out is the M-Bus protocol suite, which can be used for wired and wireless communication. The M-Bus protocol stack only uses three layers and builds upon the standard defined by IEC EN 61334-4-1. Wireless-M-Bus systems enable the communication between measurement entities and non-stationary units (for example, master devices such as a laptop acting as a data collector). To achieve the communication, several operation types are specified, such as stationary operation method (S), frequent send operation method (T) and time frequent receiver operation method (R), which align the protocol towards certain interaction scenarios.

				Body					
Header				Body Header				CRC	Body Payload
L	C	M	A	CI	Acc	S	Sig	CRC	Body Payload
3E	44	2C 4C	74 44 00 15 1E 02	7A	07	00	30 85	87 01	B1 B2 D2 97 F3 ....

Figure 5.11: Wireless M-Bus frame [138]

The Wireless M-Bus telegram consists of a header and body part. Within the header part, a length field, control field, manufacturer field and address field are provided. Within the body, the control information field identifies the application payload and contained information. For parsing the telegram, the Advanced Encryption Standard (AES) decryption needs to be used.

For M-Bus, there are two alternatives for the application layer protocol. To represent smart meter information either the M-Bus specific application layer format or the Device Language Specification (DLMS)/Companion Specification for Energy Metering (COSEM) can be used. COSEM uses the concept of interface classes, following an object-oriented approach. An interface class specifies methods and attributes an object instantiation has to provide in order to fulfil the class. Figure 5.12 gives an example for a generic *Register* interface class and two example objects instantiating this class and providing access to the current energy and to the current flow temperature value. The *logical_name* attribute uses the Object Identification System (OBIS) identifier scheme. The *value* attribute contains the current value, with a format specific to the actual object instance. OBIS is a hierarchical structured identification code. The identifier is built based on six value groups (A to F), describing in a hierarchical way the detailed meaning of a value.

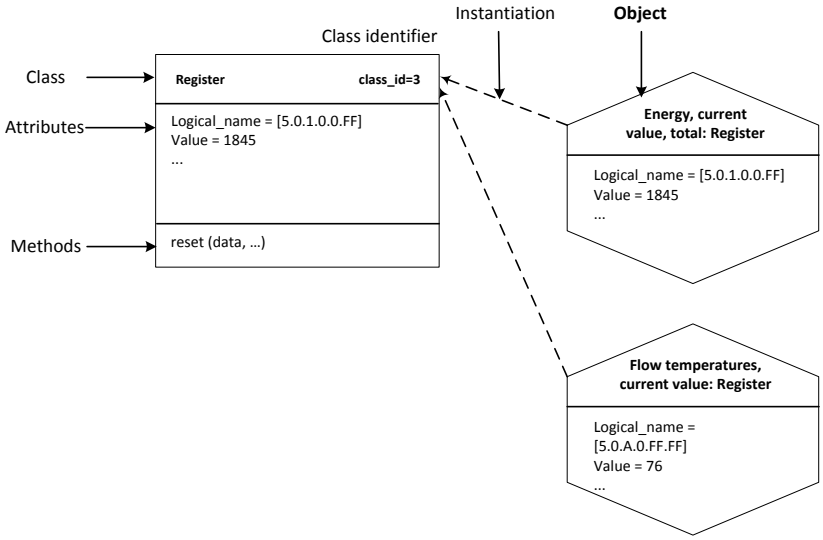


Figure 5.12: COSEM interface class and instance example [138]

Another, information representation is standardized by the dedicated M-Bus application layer. Here, the specification defines different application payload types. For variable payloads, the structure illustrated in Table 5.7 is used. The application payload consists of a data record header with semantic information about the application data. The information consists of a data information block and a value information block. Both blocks consist of an information field (DIF, VIF) and an extension field (DIFE, VIFE). The data information block specifies the format of the payload including the number of transmitted bits and data representation format. The value information block further completes the semantic information, such as range, scale and unit of the data.

DIF	DIFE	VIF	VIFE	Data
1 byte	0 to 10 (each 1 byte)	1 byte	0 to 10 (each 1 byte)	0 to N byte
Data information block (DIB)		Value information block (VIB)		
Data record header				

Table 5.7: M-Bus dedicated application layer format [138]

Application layer mapping

To integrate DLMS/COSEM into the IoT, a mapping based on OBIX specific contracts can be used to represent the COSEM interface classes. The OBIS identification code can be used for the URI to address objects.

The dedicated M-bus application layer consisting of a raw data field information is mapped to generic IoT smart meter contracts as given in Listing 5.9, with a flat mapping of the typical registers offered by a smart meter. The OBIX base value object types can be used to represent the different register values. Through the unit attribute further semantics about the field value is offered to a client.

Listing 5.9: Smart meter

```
<obj href="smartmeter/" is="iot:SmartMeter">
      <real name="power" href="smartmeter/power" val="0.0" unit="obix:units/watt"/
          >
      <real name="volume" href="smartmeter/volume" val="0.0" unit="obix:units/
          liter"/>
      <real name="energy" href="smartmeter/energy" val="0.0" unit="obix:units/
          kilowatthours"/>
</obj>
```

5.7 EPCIS and RFID

Low-cost passive RFID tags are replacing traditional barcodes for physical object identification and are heavily used within logistics and trade. They are an essential aspect of identifying physical objects, therefore an integration into the IoT is required. EPCIS refers to the information system that tracks objects and provides standardized interfaces, for RFID readers and external application clients that want to access the information. This section explains how the current EPCIS can be integrated into the IoT, extending it to a so called Smart Things Information System (STIS) [139]. The integration follows a two-fold approach. On the one side, the information of the EPCIS is made available for IoT devices and services, by providing a lightweight query interface. On the other side, the information of non-RFID related devices, such as sensors and actuators of home and building automation systems are made available to EPCIS. Figure 5.13 shows the architecture of the STIS that includes the original EPCIS. There are three layers in the STIS, the device layer at the bottom where smart physical things reside, a filter and collection middleware in the middle, and an application layer, where application logic

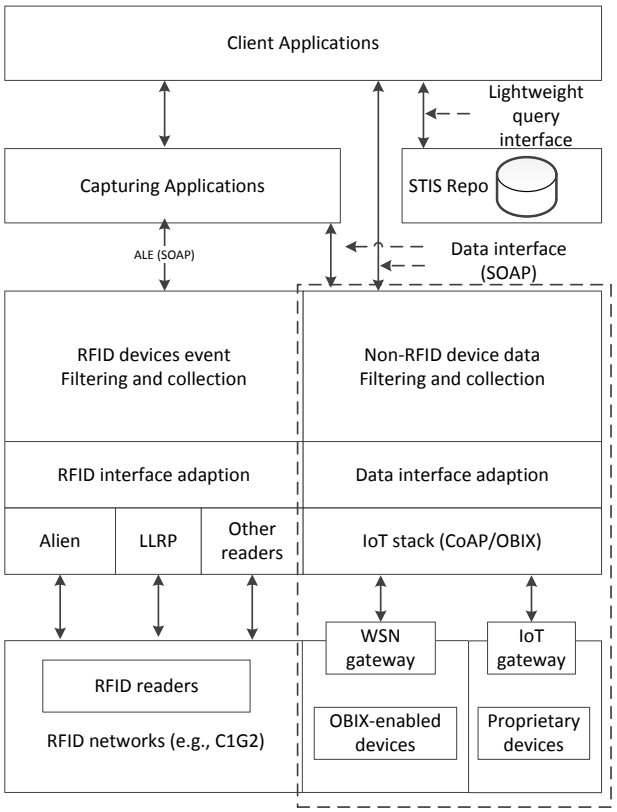


Figure 5.13: STIS components [139]

is hosted. The extension includes additional communication modules to acquire information related to things. These modules use a communication stack according to the IoT stack.

EPC scheme

The EPC is designed as universal identifier for physical objects. It supports up to 7 different formats for identification keys. An EPC can be represented in a canonical form through a URI. For example, the Global Returnable Asset Identifier (GRAI) provides a unique identification for individual objects, whereas the Global Trade Item Number (GTIN) only identifies the product type or the stock-keeping unit. In order to identify an individual instance of a product, the Serialized Global Trade Item Number (SGTIN) adds a further serial number to the GTIN. Listing 5.10 and Listing 5.11 illustrate the syntax and provide an example EPC for these coding schemes.

Listing 5.10: Global Returnable Asset Identifier

```
urn:epc:id:grai:CompanyPrefix.AssetType.SerialNumber
```

```
<ns2:ECReports specName="reportSpecName" date="2000-11-10T00:00:
 <reports>
  <report reportName="reportName">
   <group>
    <groupList>
     <member>
      <epc>urn:epc:id:sgtin:886348213.7508.136164026610</epc>
      <rawHex>urn:epc:raw:96.x304f4d499b57551fb40228f2</rawHex>
      <extension>
       <fieldList>
        <field>
         <name>temperature</name>
         <value>24</value>
         <fieldspec>
          <fieldname>unit</fieldname>
          <datatype>float</datatype>
          <format>degree C</format>
         </fieldspec>
        </field>
       </fieldList>
      </extension>
```

Figure 5.14: Sample ALE Event: Using extension in EPC standard event [139]

```
urn:epc:id:grai:0614141.12345.400
```

Listing 5.11: Serialized Global Trade Item Number

```
urn:epc:id:sgtin:CompanyPrefix.ItemRefAndIndicator.SerialNumber
urn:epc:id:sgtin:0614141.112345.400
```

For linking building automation systems and the EPCIS, OBIX can be used to represent devices and appliances and an EPC code could be linked to the OBIX object as further metadata allowing a unique identification within the EPCIS.

IoT to EPCIS integration

To integrate IoT-related data into the EPCIS, either an IoT-gateway or standalone IoT-stack-enabled objects can be used. A plugin is required at the STIS, which can interface with the presented stack. The plug-in retrieves the data represented as OBIX objects and the Filter and Collection (FnC) component generates an Application Level Event (ALE) that will be forwarded to capturing applications. The ALE event follows the EPCIS specification using the EPCIS extension for storing product data. Figure 5.14 shows how the extension is used to incorporate product data into normal ALE events. The sample ALE event incorporates an extension field called “temperature” with a value of “24” and some other information such as data type and format. This ALE event is captured by the STIS repository. Finally, the data is made available to end users or physical devices.

For the integration with the STIS the datapoint centric representation and the history contract are most relevant. One option is to let a sensor or gateway device actively push sensor information to the STIS. Another option is to register datapoints and their URIs at the STIS which will then continuously poll for new sensor data.

For the push based interaction, the STIS can provide an OBIX history object linked to a certain sensor metric (e.g., temperature, location) of an object identified through an EPC code.

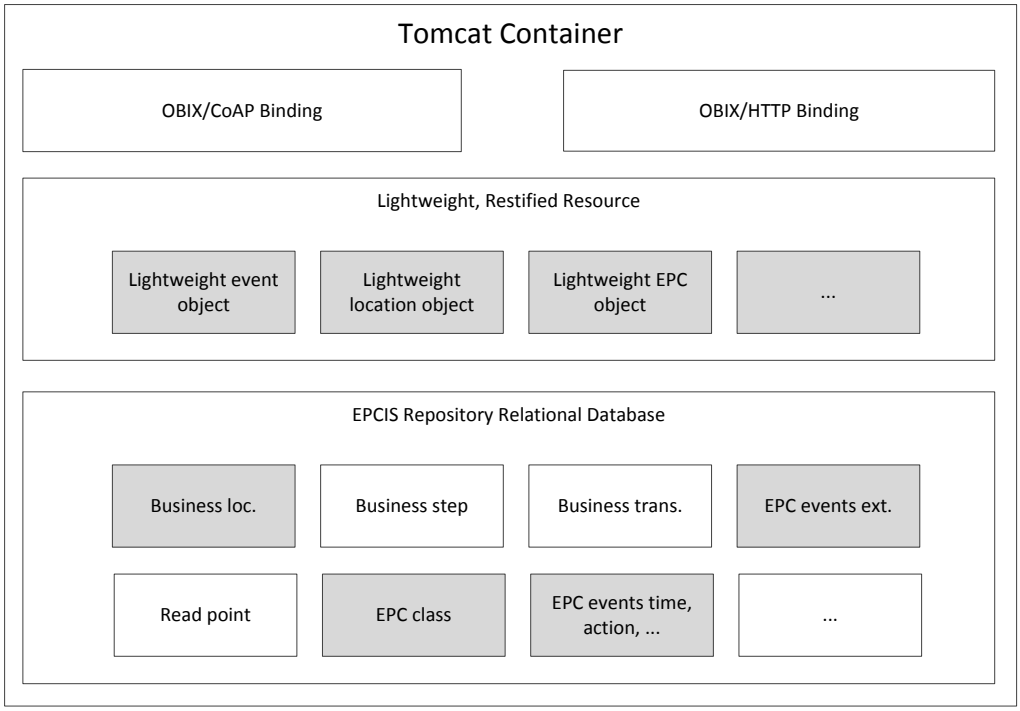


Figure 5.15: STIS lightweight query interface [139]

In this case, the sensor or the gateway pushes the new sensor reading to the EPCIS server. The address of the server object has to be provided within the sensor. To realize this interaction, the lightweight-query interface of the STIS makes the interaction feasible since only a CoAP client library is required at the sensor level.

EPCIS to IoT integration

The EPCIS interfaces are designed as heavy-weight enterprise-level Web service interfaces typically using SOAP over HTTP. For IoT smart objects, it is desirable to make the EPCIS accessible for constrained devices. For example, physical objects can subscribe to data streams of other objects in order react to certain conditions or to fulfil a certain control purpose. Therefore, a lightweight interface adhering to the IoT stack can be used. Figure 5.15 shows the structure of the lightweight interface. Only a part of the most relevant data is made accessible through a resource URI. In most cases the clients are mainly interested in the contextual data of the products. The accessible part of the EPCIS repository is grouped into lightweight resource objects such as a lightweight EPC event object, a lightweight location object, and a lightweight EPC object. These lightweight resource objects are designed to give client applications a minimum

set of most necessary information. For example, the lightweight EPC event object has only three properties: event time, event action and event data extension, which are useful for tracking the current contextual status of the object without knowing other information such as the read point or business location of the event. The lightweight location object holds information about the location of the reader where the event occurred.

5.8 Weather data

For the weather data, several free services offer an XML-based representation of weather forecasts for the most important areas in the world. Relevant information is, for example, the perception of rainfall and the expected wind speed. This information is important in the smart home context and allows to warn the inhabitants, if windows or doors are open. Temperature information is relevant to plan the required energy consumption that can be expected to heat or cool a residential home or building towards a desired setpoint. Information such as cloudiness, fog and wind speed help again to estimate the local energy production of renewable energy sources.

For the integration of the weather data, a free weather data Application Programming Interface (API) of the Norwegian Meteorological Institute[1] is used. It provides access to weather data in a REST-style. An XML schema document is provided together with the structure of the weather data. The weather data service provides a weather data element that holds a time series of weather forecasts for a certain location.

To represent weather data for the IoT, an OBIX-based representation following the structure given in Listing 5.12 is used. Figure 5.16 illustrates the mapping as class diagram. To make the weather data available to IoT smart objects and applications, a weather forecast object consists of a location object and an upcoming weather forecast object. The upcoming weather forecast offers a time series of future weather forecasts as OBIX history. A weather forecast consists of a predicted temperature, wind direction, wind speed, humidity, pressure, cloudiness, fog, cloudiness and dew point temperature. As can be seen, the simple elements of the weather forecast are mapped to OBIX basic value objects where the unit attribute provides further information of the semantics.

Listing 5.12: Weather data representation in OBIX

```
<obj name="yr.no" href="vienna/" is="iot:WeatherForecast">
      <obj name="location" href="vienna/location" is="iot:WeatherForecastLocation"
          >
            <str name="description" href="vienna/location/description" val="Vienna
                " />
            <real name="latitude" href="vienna/location/latitude" val="48.21"
                  unit="obix:units/degree" />
            <real name="longitude" href="vienna/location/longitude" val="16.37"
                  unit="obix:units/degree" />
            <int name="height" href="vienna/location/height" val="171"
                  unit="obix:units/meter" />
      </obj>
      <obj name="upcoming" href="vienna/upcoming" is="iot:UpcomingWeather">
            <abstime name="timestamp" val="2014-03-25T15:00:00.000+01:00"
```

[1] http://www.yr.no

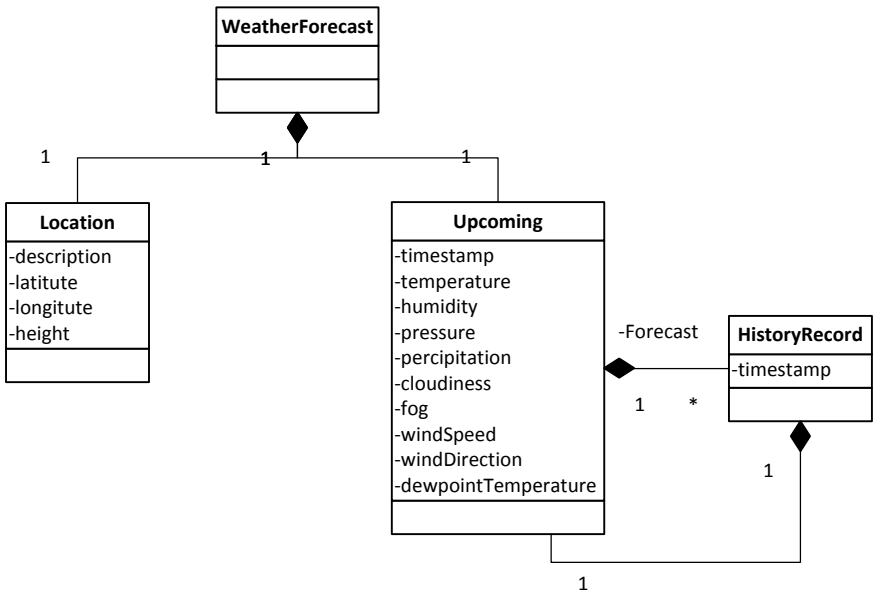


Figure 5.16: Mapping of weather data

```
                tz="Europe/Berlin" />
            <real name="temperature" href="vienna/upcoming/temperature" val="9.7"
                unit="obix:units/celsius" />
            <real name="humidity" href="vienna/upcoming/humidity" val="40.0"
                unit="obix:units/percent" />
            <real name="pressure" href="vienna/upcoming/pressure" val="1009.0"
                unit="obix:units/hectopascal" />
            <real name="precipitation" href="vienna/upcoming/precipitation"
                val="0.0" unit="obix:units/millimeter" />
            <real name="cloudiness" href="vienna/upcoming/cloudiness" val="96.1"
                unit="obix:units/percent" />
            <real name="fog" href="vienna/upcoming/fog" val="0.0"
                unit="obix:units/percent" />
            <int name="windSpeed" href="vienna/upcoming/windspeed" val="1"
                unit="obix:units/beaufort" />
            <str name="windDirection" val="W" />
            <real name="dewpointTemperature" href="vienna/upcoming/
                dewpointTemperature"
                val="-3.5" unit="obix:units/celsius" />
        </obj>
        <ref name="upcoming forecast" href="vienna/upcoming/forecast" is="obix:
            History" />
</obj>
```

5.9 Implementation

Figure 5.17 illustrates a closer view on the required components of the integration middleware gateway. The gateway provides interfaces according to the IoT stack for several existing state of the art technologies, which are non-IPv6 based. It has a common interface for direct

browser-based interaction with these devices, for control and monitoring systems, for Cloud-based software-as-a-service interaction, for global discovery services and interfacing through mobile computing. Multiple HTTP and CoAP handlers are offered and bound to virtual and physical network interfaces. An HTTP and a CoAP handler can be used to offer a centralized interface conforming to the traditional OBIX approach to interact with devices. In case of HTTP, this is fully compliant to the OBIX standard, at the same time per-device interfaces can be offered with HTTP and CoAP. The HTTP interfaces remain OBIX compliant and the CoAP interfaces are compliant to the CoRE standards. The message encoding and decoding are accomplished transparently between the HTTP and CoAP handlers and the OBIX server. As encoding XML, JSON, EXI or OBIX binary encoding can be used to encode the transferred OBIX objects. The OBIX server takes care of read, write and invoke requests to the OBIX objects that are internally controlled by an object broker. The OBIX server is independent of the protocol that is used to interact with the objects. The object broker takes care about OBIX watches, histories and alarms. The OBIX watches can be used to monitor changes of objects and follows the observer pattern to realize this functionality. In case of a CoAP get request with an observe option, it also takes care of sending asynchronous responses whenever changes occur. For clients, the traditional OBIX polling, the CoAP observe approach or both ways can be used to observe a resource. Furthermore, the OBIX broker publishes the represented devices through the traditional OBIX lobby which can be easily mapped to the CoAP /.well-known listing of available resources according to the CoRE link format [140]. Beside this centralized device access approach, the OBIX handler also publishes each device using a separate HTTP and CoAP handler on a per-device IPv6 address. It is even possible to offer centralized and per-device access in parallel, since requests to all endpoints will be handled by the same OBIX objects. Further, the IPv6 address can be assigned on a more fine-grained level by using assigning it to any kind of OBIX object. IPv6 multicast addresses can be assigned to basic OBIX value objects.

For realizing the IoT stack group communication mechanism, a group communication service intercepts the requests that arrive at the CoAP handler and directly interacts with the object broker to update the internal states of the objects. Further, it maintains a group communication address table and monitors the OBIX objects which are linked to an IPv6 multicast address. On a state change, the according update is sent to the network using a non-confirmable CoAP *put* request. Beside this, the object broker uses a discovery service client to register the available devices and resources at the IoT global discovery service. The protocol adapters (e.g., KNX Adapter) are key components of the gateway architecture. They provide the interface to the BAS specific application layer protocol. Depending on the underlying technologies the connections to different physical and data link layers need to be provided. Furthermore, the mapping of BAS specific concepts to the generic objects adhering to the IoT-OBIX contracts happens here. These contracts allow to map various technologies into a common object-oriented representation. Next to the BAS-specific adapters, also other data sources are integrated based on this mechanism. For example, the weather data service or IoT *logic blocks*.

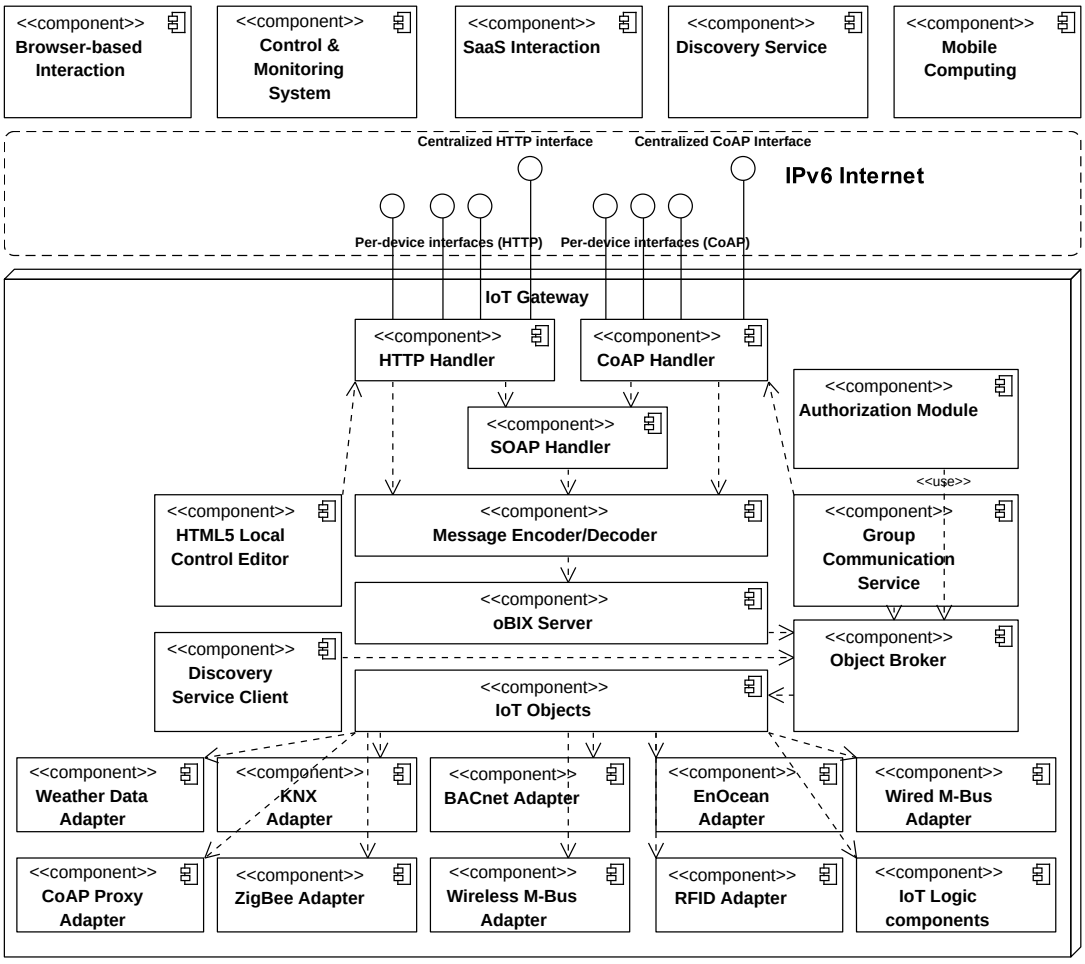


Figure 5.17: Gateway architecture

Gateway implementation

The implementation of the gateway is based on Java 7, which acts as a reference implementation of the developed IoT integration middleware. The OBIX implementation is based on the OBIX toolkit. Tomcat is used as HTTP server. The open source implementation Californium acts as CoAP server. For the implementation, the Open Service Gateway Initiative (OSGi) service framework requires a component-oriented development and a modularization of the different connectors to the integrated technologies. The communication drivers and connectors as well as technology-specific mappings of OBIX objects based on the generic abstraction are handled via so called protocol bundles. A protocol bundle is required to offer its devices through a mapping to OBIX objects. The protocol bundle needs to implement a *connector* interface, although the gateway is agnostic of available protocols. The gateway provides an object broker that can be used by protocol bundles to register its OBIX objects. In this way, the dependency

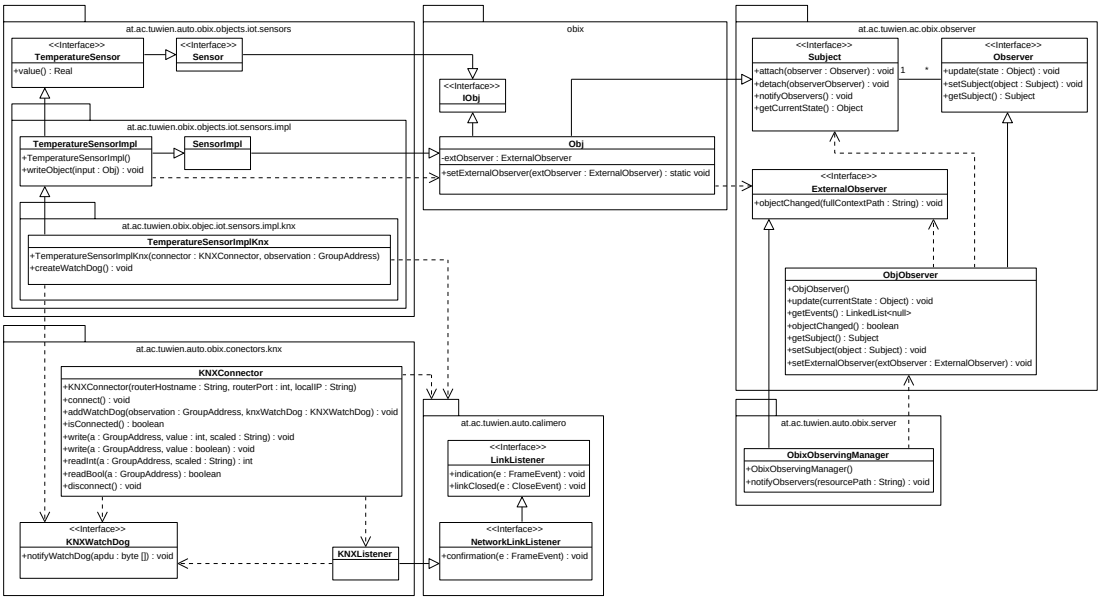


Figure 5.18: KNX protocol bundle internals [141]

is shifted to the protocol bundle provider and all the gateway services. Group communication for example can be kept generic for all technologies. The device representation of an integrated technology needs to follow the internal designs. The object refresh and notification mechanism are central mechanisms within the gateway. If the technology comes with a request/response communication mechanism the object refresh can lead to a direct read request, otherwise if the technology is event-based then an object notification mechanism needs to be used. This is important for gateway services, such as the watch service in order to provide near real-time updates to subscribed clients. To illustrate the details of a protocol bundle and the internal gateway mechanisms, the KNX protocol bundle is described in more details in the following subsection.

KNX protocol bundle

The KNX protocol bundle reuses the open source KNX Java library Calimero[2]. The class diagram in Figure 5.18 shows the concept of how the KNX bus is monitored and the mapping to the OBIX objects is performed. Since the bus communication of KNX relies on transient event-based communication that might not provide a state based read access, it is required to provide an according OBIX object that retains the state of the current bus communication.

Due to the RESTful nature of the IoT stack it is required to provide a resource that keeps the current state of the KNX device. To reach this goal, group communication on the KNX bus is monitored and the OBIX objects mapped to the group communication address are updated. To

[2]https://github.com/calimero-project

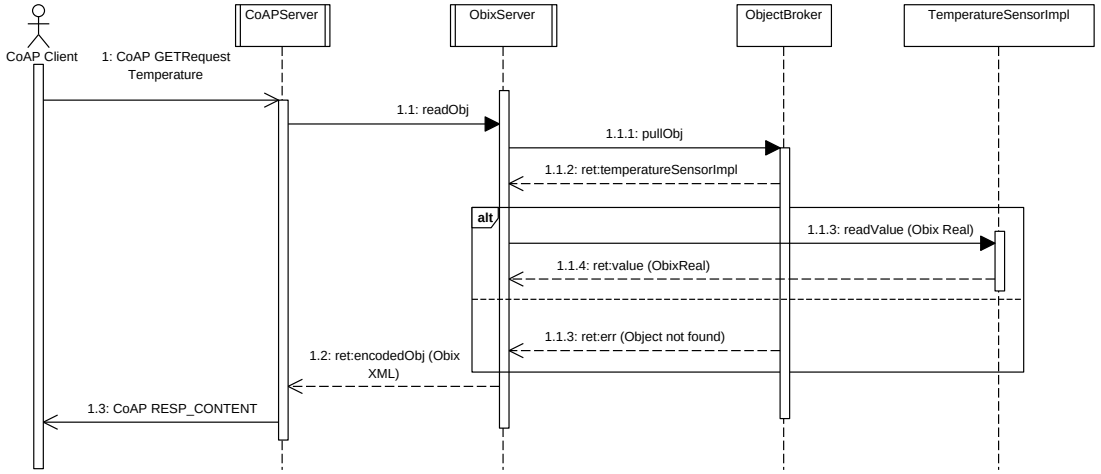


Figure 5.19: Get temperature

provide an example for this process, Figure 5.19 illustrates the concept for a KNX temperature sensor. The KNX connector of the KNX protocol bundle provides a Java API to interface the bus. Part of the connector is a watchdog mechanism that follows the observer design pattern and allows an interested client to be notified if an update on a KNX group address occurs. A client is in this case another Java object that will get notified. The base OBIX object contract is defined through the Java interface *TemperatureSensor*, which provides a simple datapoint value with the current temperature and directly inherits from the base OBIX *IObj* interface. This representation is technology-agnostic and a generic implementation is provided through *TemperatureSensorImpl*. This allows to instantiate the object within the gateway and to perform OBIX related requests on the object representation. A technology-specific implementation, as it is done for KNX, has to subclass the implementation and to provide the related KNX connector API calls, in the case a read or write on the object occurs. The *TemperaturSensorImplKnx* provides this implementation and further relies on the watchdog mechanism to get notified if an update on the bus occurs.

The UML sequence diagram (cf. Figure 5.19) illustrates the read of a KNX temperature sensor by issuing a CoAP get request. The request is first handled by the CoAP server which relies on the message decoder/encoder to parse the message. The OBIX read or write message is then passed to the *ObixServer* which retrieves the according OBIX object through the *ObjectBrowser*.

Further, another application of the observer design pattern is done within the gateway. Other OBIX related services like the watch service of OBIX or the CoAP get with observe option want to be notified if an object state has changed. For that purpose, a client object can subscribe to the changes on any subclass of OBIX *Obj* types. Based on this mechanism the OBIX watch service and the history service are realized. They automatically keep track of the changes and the history of objects. Also the CoAP server implementation maintains a session

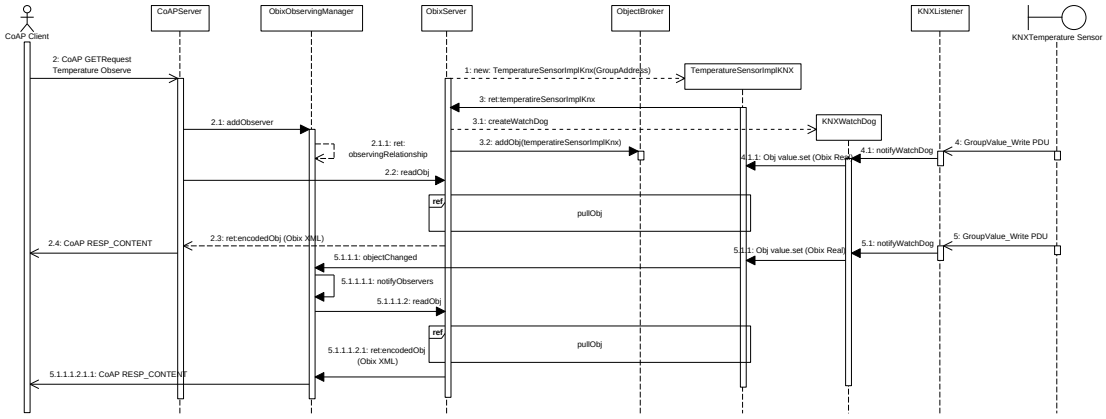


Figure 5.20: Temperature observe

for the CoAP clients that observe resources. The following sequence diagram illustrates the full interaction and the different observe relationships between the IoT gateway and the KNX bus and between the CoAP client and the IoT gateway.

Gateway configuration and device discovery

The gateway can be configured dynamically regarding the active protocol bundles. In an OSGi container the bundles can be added or removed at runtime. For discovery of devices, a static XML configuration can be provided, which contains the information about the bus-specific communication addresses of the various technologies to the IoT gateway. Beside this static information source, for some technologies an automatic discovery of devices can be realized at run-time. The so-called *devices.xml* defines the information on how technology specific address information is mapped to the OBIX based implementation. If the gateway is started in a standalone mode without an OSGi container a device loader section in the XML file allows to configure which technologies should be used. Listing 5.13 gives an example of how the configuration can be done for the KNX bus system. Either a static mapping of the KNX group addresses or an automated mapping based on the KNX project file export can be used.

Listing 5.13: devices.xml example

```
<devices>
      <!-- Configure which technologies should be loaded -->
      <deviceloaders>
            <device-loader>at.ac.tuwien.auto.iotsys.gateway.connectors.knx.
                KNXDeviceLoaderImpl
            </device-loader>
            ...
      </deviceloaders>

      <!-- static mapping between OBIX objects and KNX bus system -->
      <knx>
            <connector>
                  <name>KNX</name>
```

```
                <enabled>false</enabled>
                <router>
                      <ip>192.168.1.14</ip>
                      <port>3671</port>
                </router>
                <localIP>auto</localIP>

                <device>
                      <type>at.ac.tuwien.auto.iotsys.gateway.obix.objects.iot.
                          actuators.impl.knx.TextDisplayActuatorImplKnx
                      </type>
                      <address>null, 0/0/3</address>
                      <href>textDisplay</href>
                      <historyEnabled>true</historyEnabled>
                      <historyCount>200</historyCount>
                </device>
          </connector>
    </knx>

    <!-- automated configuration based on ETS project file -->
    <knx-ets>
          <connector>
                <name>KNX</name>
                <enabled>false</enabled>
                <knx-proj>knx-config/projects/KNX.knxproj</knx-proj>
                <forceRefresh>false</forceRefresh>
                <router>
                      <ip>192.168.1.100</ip>
                      <port>3671</port>
                </router>
                <localIP>auto</localIP>
          </connector>
          ...
    </knx-ets>
</devices>
```

Security and privacy

For ensuring security and privacy, the gateway rests upon state of the art encryption and authentication methods. The Tomcat server container can be configured to protect the HTTP-based traffic. Therefore, the cryptography provider of the the underlying Java platform is used [142]. To authenticate clients either user-name and password or a client certificate can be taken. Various asymmetric algorithms such as RSA, DSA and ECC are available, where especially ECC is attractive due to the reduced key size, bandwidth and power requirements [143]. Once authenticated, the exchanged payload is encrypted using a transport-layer security approach. Here, various symmetric algorithms and cypher suites are available. The selection of the cypher suite mainly depends on the interoperability requirements with respective clients. A reasonable secure and efficient algorithm for symmetric encryption is for example provided by the AES algorithm. For SOAP-based access, the security proxy presented in Chapter 3 can be used. It intercepts incoming calls and authorizes the request either using a local or remote policy decision point. The gateway itself includes an XACML policy decision point.

5.10 Integration middleware case study

To evaluate the multi-protocol integration, a case study is conducted based on a real-world testbed. Therefore, the Java-based gateway is operated using an OSGi framework on an embedded PC platform. The case study, is performed with a standard Raspberry PI platform and a custom board developed within the IoT6 research project[3].

Evaluation use cases

In order to test the multi-protocol interaction, a set of representative interaction use cases found within a typical building automation system is used (cf. Figure 5.21). To illustrate and to show the feasibility of a common multi-protocol integration, heterogeneous technologies have been taken to realize the defined use cases. The test environment involves state of the art non-IPv6 based technologies combined with new IoT enabled sensors and actuators.

For testing the multi-protocol interaction as methodology, an experimental evaluation based on a real testlab is done. The aim of these scenarios is to show the benefits of the taken integration approach based on the IoT gateway using the IoT stack. The use cases are described in more details in the appendix in Table A.1 to Table A.4 based on the casual use case description template of Cockburn [144].

Test environment

For the multi-protocol interaction tests, a lab environment is used. The lab consists of several home and building automation testbeds including a room model for KNX, a BACnet HVAC-model connected to a W-MBus smart meter. The room itself is automated using KNX including a presence sensor, window contact sensor, a fan coil unit, a user control panel, light switch actuators and a door access control. An RFID tag reader mounted next to the lab door provides access to lab room with further system equipment. In the following, the specific testbeds are shortly described.

KNX equipment: The KNX lab environment provides typical room automation devices, including a presence sensor, room lights, push buttons, a room thermostat, an air conditioning unit, a sun blind actuator, a window contact sensor and a room lighting model.

(a) KNX light and presence sensor (b) KNX room control unit (c) KNX room HVAC (d) KNX window contact sensor

Figure 5.22: Smart object hardware architecture and example platform

[3] www.iot6.eu

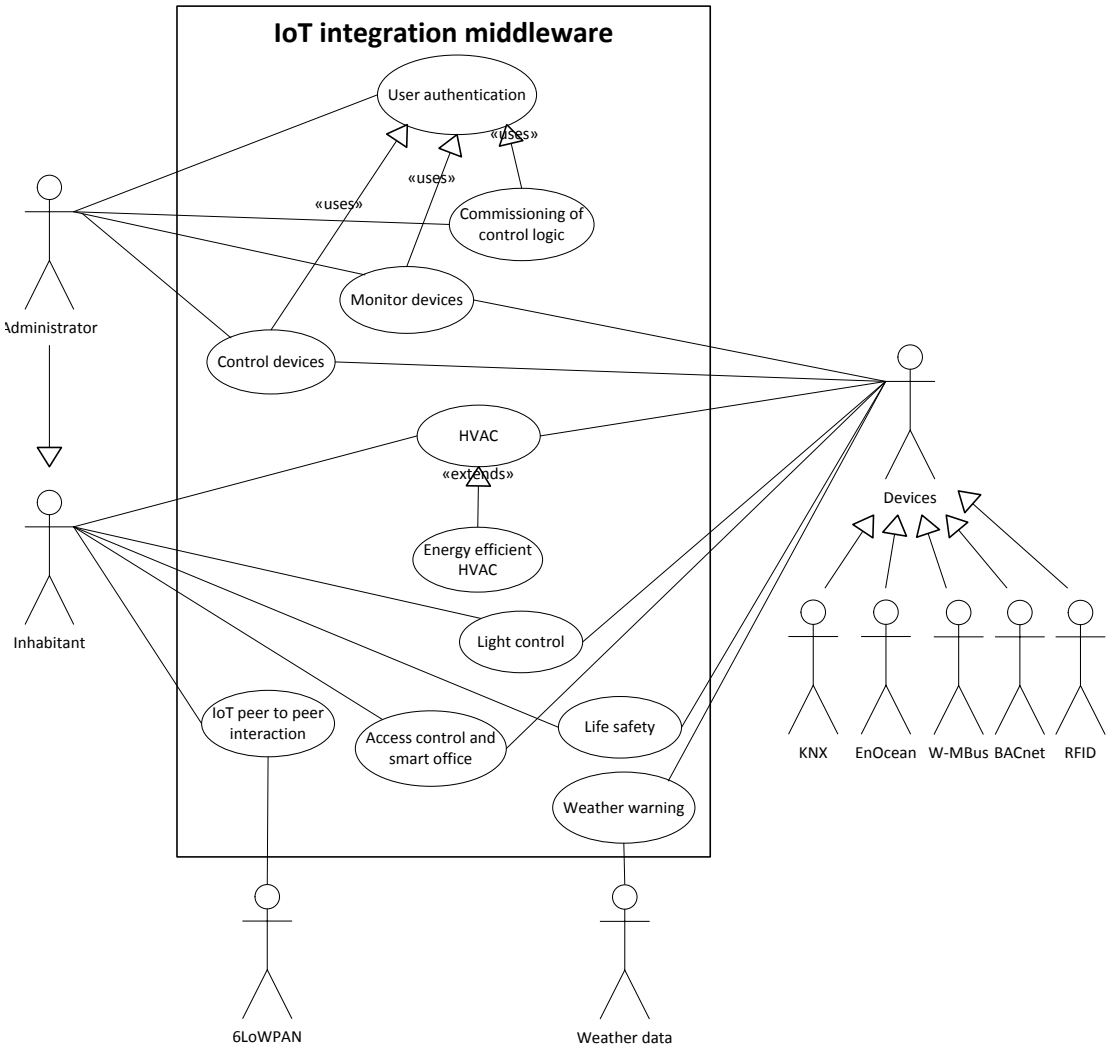


Figure 5.21: Use cases of the case study

BACnet environment: A BACnet-based HVAC model as illustrated in Figure 5.23 can be used to test the HVAC use case scenario. The model consists of a boiler and chiller device to heat or cool dedicated water cycles, which are pumped into heating or cooling registers. Fans and valves control the air circulation over registers, in order to heat or cool a miniaturized room environment.

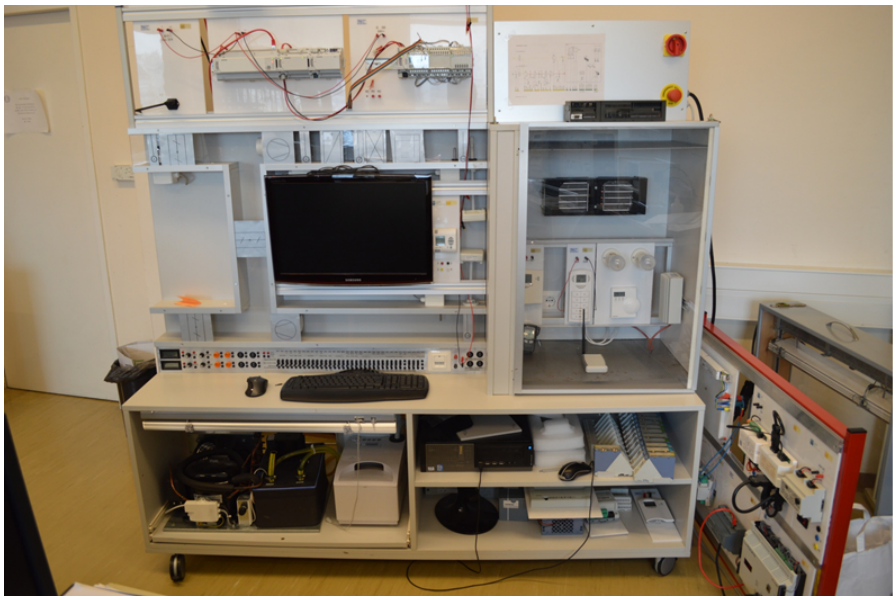
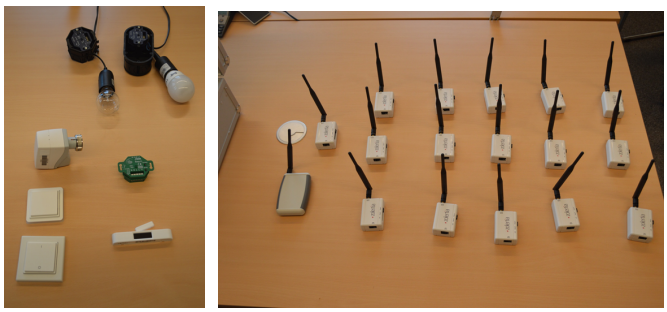


Figure 5.23: BACnet HVAC model

The room model is further enhanced with a 6LoWPAN temperature sensor that measures the current room temperature and an electricity smart meter. A wireless M-Bus meter measures the current power load in real-time.

EnOcean wireless room automation: Several actuators and sensors based on EnOcean (cf. Figure 5.24a) are available to test the room automation integration.



(a) EnOcean testbed (b) 6LoWPAN testbed

Figure 5.24: EnOcean and 6LoWPAN testbed

6LoWPAN testbed: A 6LoWPAN testbed (cf. Figure 5.24b) is deployed within the research group office area.

Gateway platform Figure 5.25 provides an overview of the taken platforms for the experimental evaluation of the integration middleware. The use of Java as application platform comes at the cost of having no access to hardware specific APIs. This is a problem, if a direct interaction with device drivers for bus connectivity is required. However, the use of virtual serial interfaces provides a convenient way to avoid this problem and to use Java as gateway platform. For the connection to the heterogeneous technologies and bus systems, various USB-based connectors are available, which provide a virtual serial interface. By using the RXTX library support for Java, this universal serial interface can be used to exchange telegrams with the different technologies (e.g., KNX, EnOcean, W-MBus, RFID).

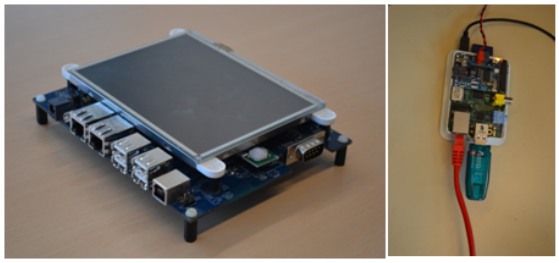


Figure 5.25: IoT gateway platforms (custom smart board and Raspberry Pi)

Interoperability tests

This sections describe how the different use cases are instantiated with the components created in the proof of concept implementation. Integration tests are performed in order to validate the interoperability amongst the heterogeneous technologies.

Light control: For the lighting control scenario, the group communication of the IoT stack is used. The boolean datapoint representation of the EnOcean push button, the KNX light switch actuator and the BACnet light switch actuator are assigned to the same IPv6 multicast address (e.g., FF12::1). As soon as the EnOcean push button is pressed, the IoT gateway receives the telegram and checks internally the group communication table. It transmits the new value by sending a CoAP PUT on the multicast address. It receives the multicast request itself and checks again the related OBIX objects that reside in the same group and updates the corresponding values. This leads to the transmission of a KNX telegram on the according KNX group address. Similarly, a write property request is performed for the related BACnet object.

HVAC control scenario: One of the more sophisticated interaction scenarios is the HVAC control scenario. It uses the BACnet-based HVAC process model that operates on the various devices to control the room temperature. The process model is enhanced by a 6LoWPAN

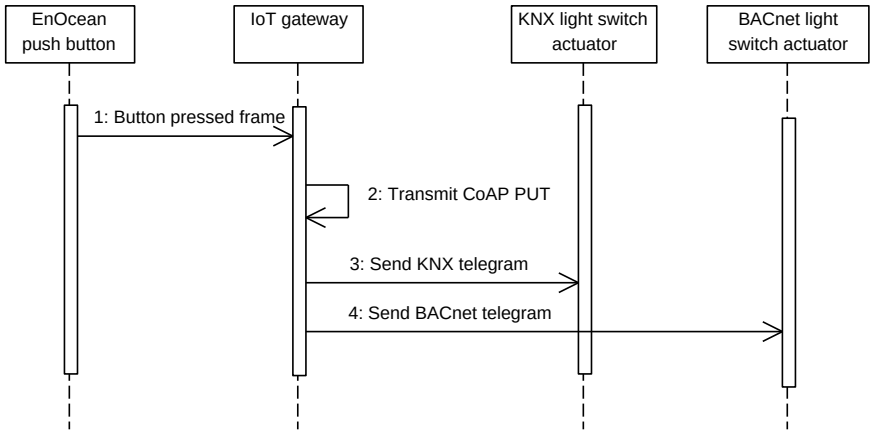


Figure 5.26: Light control scenario

temperature sensor that monitors the current room temperature. A logic object realizes a temperature controller inside the gateway and operates on the OBIX object representation of the various technologies. It observes the current room temperature, the current energy consumption with the W-MBus smart meter and the state of an EnOcean window contact sensor. A KNX room thermostat allows adjusting the desired setpoint with a relative value. It controls the BACnet control devices if the current room temperature is not close to the desired setpoint. If the window is opened the HVAC process is set to stand-by. This is visualized on the KNX room thermostat. Further, if a certain threshold of the current power consumption is reached the control value is reduced to a relative value.

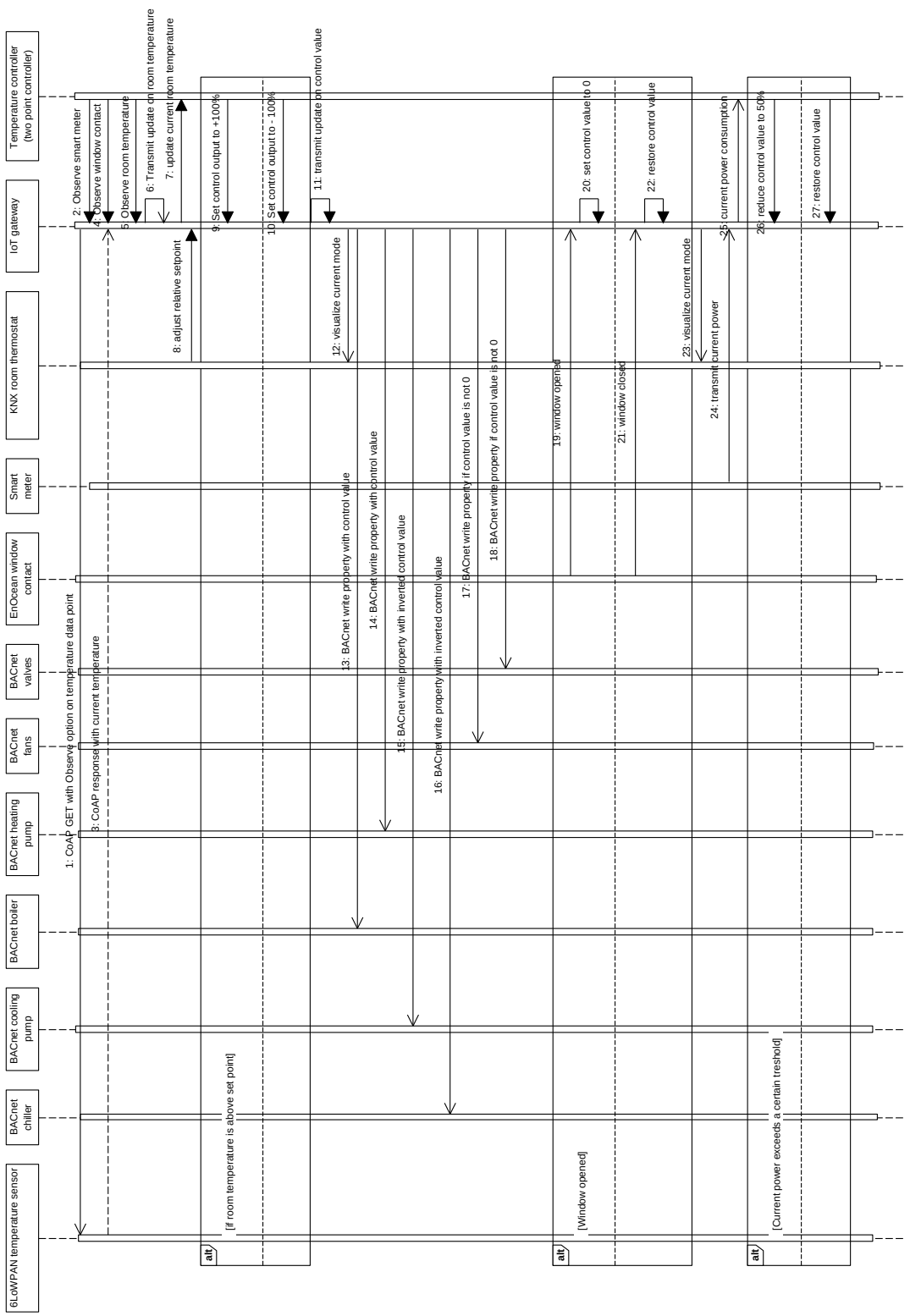


Figure 5.27: HVAC control scenario

Alarming scenario: For the alarming scenario, the weather data is required. In case of an approaching storm, the user receives a warning. This is realized through a KNX text display that provides a flashing light together with a simple text message. The user needs to manually acknowledge the warning. If a storm is approaching and a window is open, an acoustic alarm is added that requires the immediate attention of a resident.

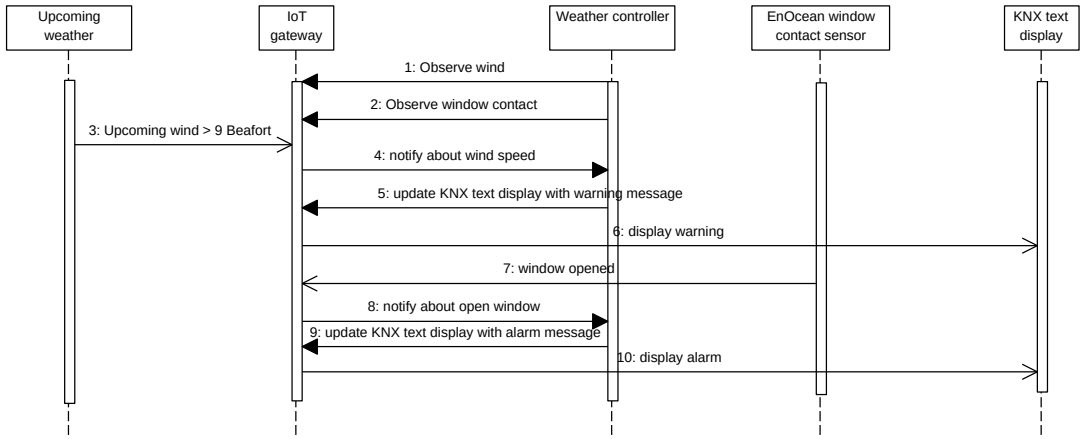


Figure 5.28: Alarming scenario

Access control and room environment settings scenario: In this scenario, an RFID reader is combined with a KNX-controlled door opener. Based on the employee card or the passport the access to the lab room is granted and further the environment is adjusted to the predefined settings for the person.

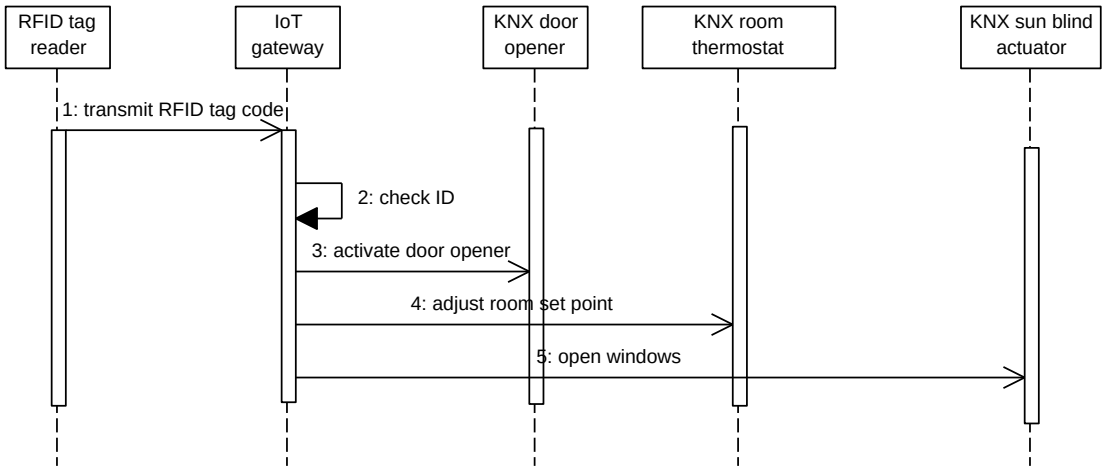


Figure 5.29: Access control scenario

Life safety: By linking a 6LoWPAN accelerometer sensor to a KNX alarm signal and text display, a life safety use case can be demonstrated. Therefore, an IPv6 site-local multicast address is assigned to the boolean datapoint representing the free fall and linked through the gateway to an according alarming datapoint at a KNX text display. As soon a free fall is detected, the alarm is raised and a rescue person has to manually acknowledge the alarm.

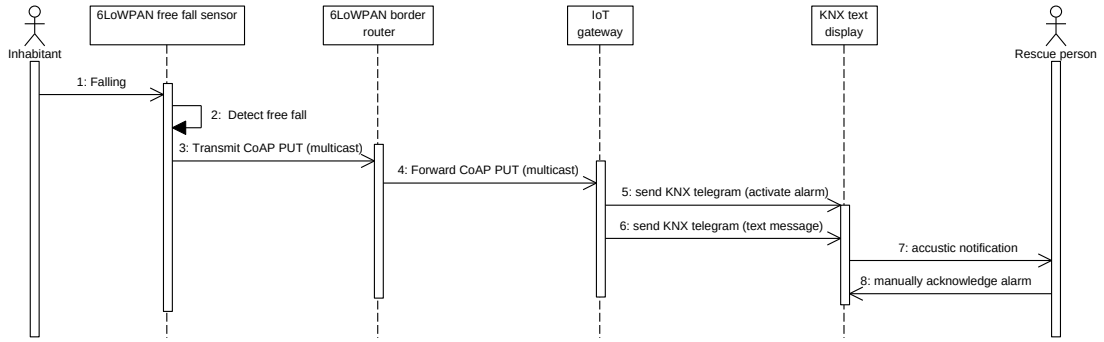


Figure 5.30: Life safety

IoT peer-to-peer interaction: In this use case scenario, multiple 6LoWPAN devices are linked together by grouping boolean datapoints representing a push button and LED datapoints. The gateway is used for engineering the group communication relationship. However, for run-time interaction no gateway is required. A human user can switch off the gateway and tap the 6LoWPAN push button. The changed state is transmitted using an IPv6 multicast, which is received by all nearby devices. The devices check the internal address assignment and update the according datapoints.

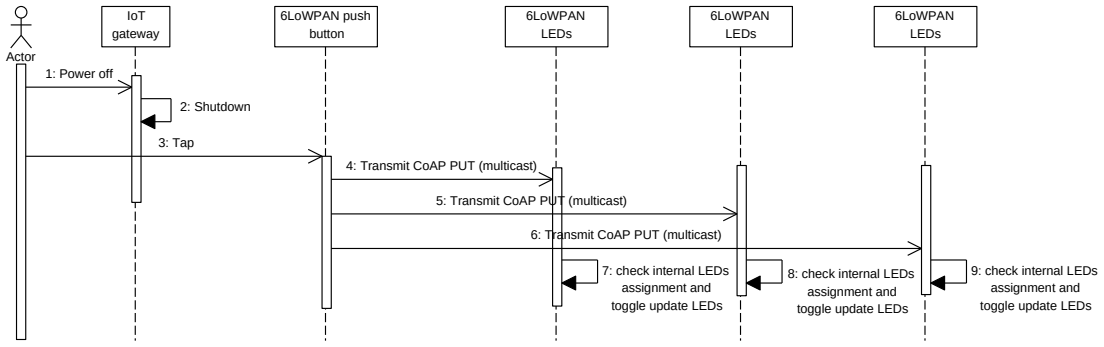


Figure 5.31: Peer-to-peer interaction

5.11 Scalability analysis

For the scalability analysis of the proposed integration middleware, a queuing network model of the gateway allows to predict the resource demand and the performance behaviour. Therefore, the different request types that impose load on the gateway are analyzed, measured and used to establish and quantify an analytic model that allows to estimate the scalability of the gateway in certain load scenarios.

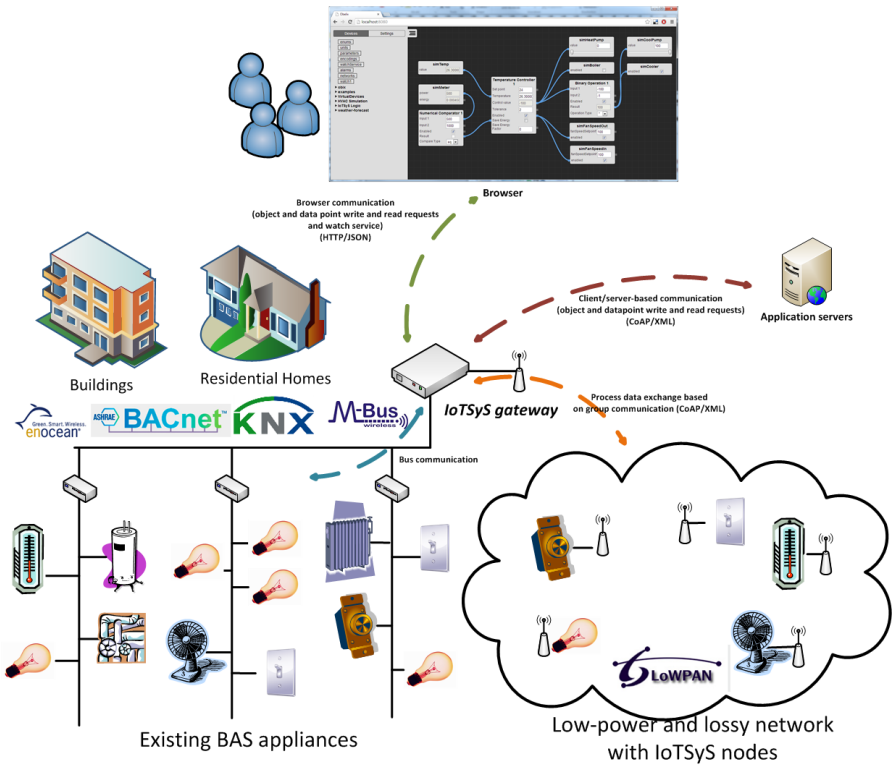


Figure 5.32: Gateway evaluation scenario

Notation

The following notation is used throughout this chapter:

- K: number of devices or service centers of the model
- R: number of classes of customers
- M_r: number of terminals of class r
- Z_r: think time of class r
- N_r: class r population

- λ_r: arrival rate of class r
- $S_{i,r}$: average service time of class r customers at device i
- $V_{i,r}$: average visit ratio of class r customers at device i
- $D_{i,r}$: average service demand of class r customers at device i; $D_{i,r} = V_{i,r} * S_{i,r}$
- $R'_{i,r}$: average residence time of class r customers at device i (i.e., the total time spent by class r customers at device i over all visits to the device); $R'_{i,r} = V_{i,r} * R_{i,r}$
- $\overline{n}_{i,r}$: average number of class r customers at device i
- $\overline{n}_i$: average number of customers at device i
- $X_{i,r}$: class r throughput at device i
- $X_{0,r}$: class r system throughput
- R_r: class r response time

Mixed queuing network model for gateway analysis

A mixed queuing network model is taken to analyze the scalability. This model approach and according algorithms to solve this type of model follow the methodology presented in [41]. A mixed model consists of set of R customers, that is mixture of open and closed class customers. C denotes the set of closed class customers and O denotes the set of open classes, where $R = C \cup O$. In this way, the mixed model can be seen as a combination of independent sub-models that share the same computing resources. For the specification of the load intensity, the closed and the open model need to be defined. The closed model is specified through a vector $\overrightarrow{C} = (N_1, N_2, ..., N_C)$ that defines the customer population, the open model is represented through a vector of arrival rates, $\overrightarrow{O} = (\lambda_1, \lambda_2, ..., \lambda_O)$. Therefore, the vector pair $(\overrightarrow{C}, \overrightarrow{O})$ specifies the load intensity of the overall mixed model.

For the analysis of the gateway, a model needs to be defined. The gateway has to deal with various request types that are represented through the different customer types and queuing network models. The gateway needs to deal with requests that represent the process data exchange. Group communication used for creating distributed control logic scenarios falls into this category. Also read and write requests on OBIX objects and datapoints for creating application logic can be categorized into this type of traffic. Furthermore, human users may interact with the gateway in order to directly interact with objects or to create further control logic applications.

A mixed queuing network model can be taken to represent these different types. A closed model reflects the interactive user requests that are placed on the gateway. An open model is used to represent related requests to the process data exchange. The resource of interest is the CPU of the gateway.

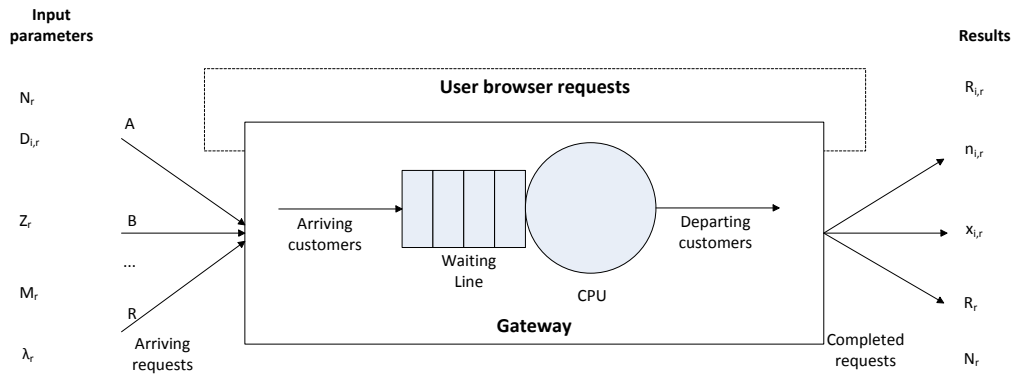


Figure 5.33: QN model of gateway

Closed model The closed model is represented through the load intensity vector $\overrightarrow{N}$ and the relevant class descriptor parameters of $(D_{i,r}, M_r, Z_r)$. R is defined with the set of possible request types $R = (WatchService, Read, Write, GroupManagement)$. The $WatchService$ customer class represents the recurring requests that occur through the OBIX watch service, issued by a client to update the user interface. In case of the HTTP binding of the gateway, this service can be polled for updates that happened since the last poll. The watch service comes with two different request types. The *pollChanges* operation provides a delta of changes since the last poll and the *pollRefresh* operation provides a complete list of the objects that are currently monitored by the watch. The computational effort strongly depends on the required number of OBIX objects that need to be encoded. The positive effect of the *pollChanges* further depends on the update ratio of the represented objects. If the update ratio is higher than the polling interval, no improvement of *pollChanges* compared to *pollRefresh* can be observed.

Table 5.8 provides the service demands measured on different hardware platforms operating the gateway. For the evaluation a x86 PC platform is compared to a custom developed embedded smart board and the popular Raspberry PI platform. The smart board was developed within the IoT6 research project. For acquiring the measurements the operational analysis methodology presented in [108] is used.

Class	Service Demand CPU (ms) Pi	Smart Board	x86 PC
WatchService	159.13	41	1
Write	95.45	32	2
Read	51.54	17.7	7
Group Management	57.71	31	2.5

Table 5.8: Service demands

For the analysis of the closed model, it is important to know the number of concurrent transactions of these request classes at the same time. A single human user might impose one

transaction at the same time for all these request types. So if one user opens the Web-based UI, the session is linked to a watch service, further the user might at the same time perform *read*, *write* and *group management* activities. The *think time* is another important parameter for interactive jobs within the closed model.

For solving a closed queuing network model, alternative ways for modelling and solving the model can be used. A classic approach would be to use a Markov model with a state space description. However, the problem with Markov models is that they are exposed to state space explosion. The main problem is here the set of linear equations, since each state is described through a set of linear equations. An alternative technique is the Mean Value Analysis (MVA) that allows to iteratively derive device performance metrics (e.g., response time). It is based on the fact that given the average number of customers at each device with $(n-1)$ customers in the system, the device residence times when there are n customers in the network can be derived. The MVA algorithm for a single class system is described by Algorithm 5.1 illustrates the iterative process of calculating the performance metrics.

Initialize the average number of customers at each device i: $\overline{n}_i = 0$.
for *each customer population* $n = 1, 2, ...N$ **do**
    calculate the average residence time for each device i:
        $R'_i(n) = V_i * S_i * [1 + \overline{n}_i(n-1)] = D_i * [1 + \overline{n}_i * (n-1)]$
    calculate the overall system response time:
        $R_0(n) = \sum_{i=1}^{K} [V_i \times R_i(n)] = \sum_{i=1}^{K} R'_i(n)$
    calculate the overall system throughput:
        $X_0(n) = \frac{n}{R_0(n)}$
    calculate the throughput for each device i:
        $X_i(n) = V_i \times X_0(n)$
    calculate the utilization for each device i:
        $U_i(n) = S_i \times X_i(n)$
    calculate the average number of customers at each device i:
        $\overline{n}_i(n) = X_0(n) \times R'_i(n)$
end

Algorithm 5.1: The MVA algorithm

In order to make the algorithm usable for a closed multi-class model, several enhancements need to be done. The following equations provide the base of the enhanced MVA algorithm for multi-class models.

$$X_{0,r}(\overrightarrow{N}) = \frac{N_r}{Z_r + \sum_{i=1}^{K} R'_{i,r}(\overrightarrow{N})} \tag{5.1}$$

Equation 5.1 is used to calculate the overall system throughput for every request class. It can be calculated by dividing the number of customers in this request class through the sum of all residence times at the different devices and the think time of the request class.

$$\begin{aligned}\overline{n}_{i,r}(\overrightarrow{N}) &= X_{i,r}(\overrightarrow{N}) * R_{i,r}(\overrightarrow{N}) \\ &= X_{0,r}(\overrightarrow{N}) * V_{i,r} * R_{i,r}(\overrightarrow{N}) \\ &= X_{0,r}(\overrightarrow{N}) * R'_{i,r}(\overrightarrow{N})\end{aligned} \tag{5.2}$$

The overall system throughput for a request class can be used as outlined in Equation 5.2 to calculate the average number of customers of a certain request class at a given device.

$$\begin{aligned}\overline{n}_i(\overrightarrow{N}) &= \sum_{r=1}^{R} \overline{n}_{i,r}(\overrightarrow{N}) \\ &= \sum_{r=1}^{R} X_{0,r}(\overrightarrow{N}) * R'_{i,r}(\overrightarrow{N})\end{aligned} \tag{5.3}$$

The average number of customers of a certain request class at a given device can be summed up to get the overall number of average customers at a given device (cf. Equation 5.3). The key equation of the MVA technique is based on the fact that the response time depends on the customers' service time and the time to complete the number of customers already queued upon arrival. Equation 5.4 shows how to calculate the response based on $\overline{n}^A_{i,r}$, which is the average queue length at device i seen by a customer of request class r.

$$\begin{aligned}R_{i,r}(\overrightarrow{N}) &= S_{i,r}[1 + \overline{n}^A_{i,r}(\overrightarrow{N})] \\ V_{i,r} * R_{i,r}(\overrightarrow{N}) &= V_{i,r} * S_{i,r}[1 + \overline{n}^A_{i,r}(\overrightarrow{N})] \\ R'_{i,r}(\overrightarrow{N}) &= D_{i,r}[1 + \overline{n}^A_{i,r}(\overrightarrow{N})]\end{aligned} \tag{5.4}$$

Using Equation 5.2 to Equation 5.4, the MVA algorithm for a single class of request types can be extended for multiple types as given in Listing 5.4. The key idea of the algorithm is similar to the idea of the single-class MVA algorithm. The expected response time is calculated incrementally using the customer population starting from zero, (cf. line 4 to 7). Beginning from zero the service demands are taken to calculate the residence time (cf. line 13), which is used to calculate the request class throughput (cf. line 19). Using the throughput for the customer class, the average number of customers at a certain device can be calculated (cf. line

22).

Input Parameters: $D_{i,r}$ and N_r
Initialization: For i = 1 to K to $\overline{n}(\overrightarrow{0}) = 0$
Iteration Loops:
for $j_1 = 0$ **to** N_1 **do**
  for $j_2 = 0$ **to** N_2 **do**
    ...
    for $j_R = 0$ **to** N_R **do**
      $\overline{N} = (j_1, j_2, ..., j_R)$
      if $\overrightarrow{N} \neq \overrightarrow{0}$ **then**
        for $r = 1$ **to** R **do**
          if $j_r > 0$ **then**
            for $i = 1$ **to** K **do**
              $$R'_{i,r}(\overrightarrow{N}) = \begin{cases} D_{i,r} & delay \\ D_{i,r}[1 + \overline{n_i}(\overrightarrow{N} - \overrightarrow{1_r})] & LI \end{cases}$$
            end
          end
          else
            $R'_{i,r}(\overrightarrow{N}) = 0$
          end
          $X_{0,r}(\overrightarrow{N}) = \frac{j_r}{Z_r + \sum_{i=1}^{K} * R'_{i,r}(\overrightarrow{N})}$
        end
        for $i = 1$ **to** K **do**
          $\overline{n}_i(\overrightarrow{N}) = \sum_{r=1}^{R} * X_{0,r}(\overrightarrow{N}) R'_{i,r}(\overrightarrow{N})$
        end
      end
    end
  end
end

Algorithm 5.2: Exact MVA algorithm for multiple classes

Open model For the requests related to process data exchange, an open model is used, since requests may originate within the local network by devices directly sending requests and also local or remote applications that perform control logic scenarios. The open model is represented through R customer classes with a load intensity vector $\overrightarrow{\lambda} = (\lambda_1, \lambda_2, ..., \lambda_R)$ on K devices, where λ_r indicates the arrival rate of class r customers.

Table 5.9 illustrates the service demands measured at the gateway for the open model.

Algorithm 5.3 provides the solution to solve models with multiple open classes. The utilization of a device i by a customer class r can be calculated using the arrival rate and the service demands. The sum of the device utilization per customer class leads to the overall de-

Class	Service Demand CPU (ms) Pi	Smart Board	x86 PC
Datapoint Write	298	52	12
Datapoint Read	259	45	1
Object Write	334	51	13
Object Read	265	44	1
Group Communication Write	72	41	16
Bus Event	3	6	1

Table 5.9: Service demands open model

vice utilization (cf. line 3-6). The average residence and response time are calculated using the service demands of a certain customer class on a device and the overall device utilization (cf. line 7-9). The utilization can also be used to calculate the average number of customers of a certain customer class at a device i (cf. line 10-12).

1 **Input Parameters:** $D_{i,r}$ and λ_r
2 **Steps:**
3 Calculate the utilization per device and per customer class for the given load vector:
4 $U_{i,r}(\overrightarrow{\lambda}) = \lambda_r V_{i,r} S_{i,r} = \lambda_r D_{i,r}$
5 Determine the overall device utilization:
6 $U_i(\overrightarrow{\lambda}) = \sum_{r=1}^{R} U_{i,r}(\overrightarrow{\lambda})$
7 Based on the overall device utilization derive the average residence time per device and customer class:
8 $R'_{i,r}(\overrightarrow{\lambda}) = \begin{cases} D_{i,r} & delay \\ \frac{D_{i,r}}{1-U_i(\overrightarrow{\lambda})} & LI \end{cases}$
9 $R_r(\overrightarrow{\lambda}) = \sum_{i=1}^{K} R'_{i,r}(\overrightarrow{\lambda})$
10 Calculate the average customer population per device and per customer class and for the overall device population:
11 $\overline{n}_{i,r}(\overrightarrow{\lambda}) = \frac{U_{i,r}(\overrightarrow{\lambda})}{1-U_i(\overrightarrow{\lambda})}$
12 $\overline{n}_i(\overrightarrow{\lambda}) = \sum_{r=1}^{R} \overline{n}_{i,r}(\overrightarrow{\lambda})$

Algorithm 5.3: Algorithm for solving a QN with multiple open classes

Mixed Model To solve the combined model of the multi-class open and multi-class closed model, the principle approach is to solve the independent models. In order to represent the influence of the different models on each other, first the utilization of the resources due to the open class model is calculated and depending on the utilization, the service demand of the closed model is elongated. The closed model with the modified service demand is then solved and the average customer queue length of the closed model customers is used for calculating

the response times of the open model.

Input Parameters: $D_{i,r}$, N_r and λ_r
Steps:
Solve the open submodel and obtain the device utilization per device and request class:
    $U_{i,r}(\overrightarrow{O}) = \lambda_r D_{i,r} \forall r \in \{1, 2, ..., O\}$
Determine the overall device utilization in the open model:
    $U_{i,open} = \sum_{r=1}^{O} U_{i,r}$
Elongate the service demands of the closed classes:
    $D_{i,r}^{e} = \frac{D_{i,r}}{1-U_{i,open}}, \forall r \in \{1, 2, ..., C\}$
Using the MVA algorithm, compute performance results for the closed model:
    $R'_{i,r}(\overrightarrow{O}) = \sum_{r=1}^{C} \overline{n}_{i,r}(\overrightarrow{C})$
Determine the average customer population per device in the closed model:
    $\overline{n}_{i,closed}(\overrightarrow{C}) = \sum_{r=1}^{C} \overline{n}_{i,r}(\overrightarrow{C})$.
Compute the average residence time for the open submodel:
    $R'_{i,r}(\overrightarrow{O}) = \frac{D_{i,r}[1+\overline{n}_{i,closed}(\overrightarrow{C})]}{1-U_{i,open}(\overrightarrow{O})}$
Determine the average customer population per device in the open model:
    $\overline{n}_{i,open}(\overrightarrow{O}) = \lambda_r R'_{i,r}(\overrightarrow{O})$

Algorithm 5.4: Algorithm for mixed multiclass models

The algorithm to solve the mixed class QN model is implemented in Java and calculates the expected gateway response time for certain load scenarios.

Evaluation results

For the evaluation, different scenarios are taken into the account. The main input parameters are the arrival rate of the open class request, the think time and the number of closed class customers. To keep the complexity of the model reasonable the same think time is used for all closed class requests. For the open class requests, the arrival rate is grouped into an arrival rate for process related communication that consists of group communication and bus communication, and further into a group that represents ad-hoc communication generated through external applications which operate through client/server based communication with datapoints and objects. The process communication takes place at run-time without the involvement of any system engineer. With the IoT stack, this might be traditional non-IP bus communication, as well as IP-based communication using CoAP. The following results show the average response time to be expected for a certain arrival rate and number of clients for the Raspberry Pi platform.

Scenario 1 - High ad-hoc communication and high user interaction: In this scenario, an equal distribution of process communication related requests and ad-hoc communication requests is considered. A think time of one second represents a high frequent interaction through the user interface and a fast update cycle of the user interface. As it can be seen in Figure 5.34 and Figure 5.35, the average response time in this load scenario is acceptable for up to 2 concurrent users if the overall request arrival rate for the open model requests is around 2 requests per second. The main bottleneck are the expensive ad-hoc communication requests based on CoAP. The impact on the process communication is severe after more than two concurrent users are accessing the gateway. Figure 5.34 illustrates the average response time for the closed class request types. Figure 5.35 shows the according results for the open class request types.

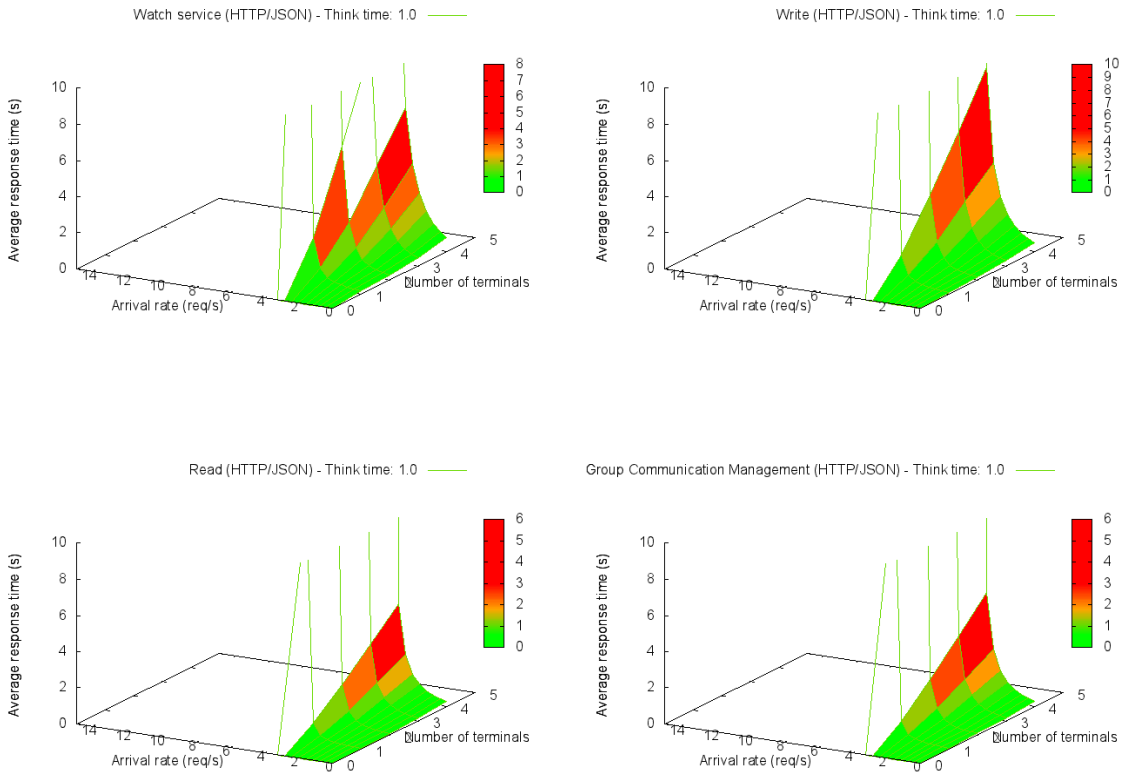


Figure 5.34: Scenario 1 - Closed class expected response time

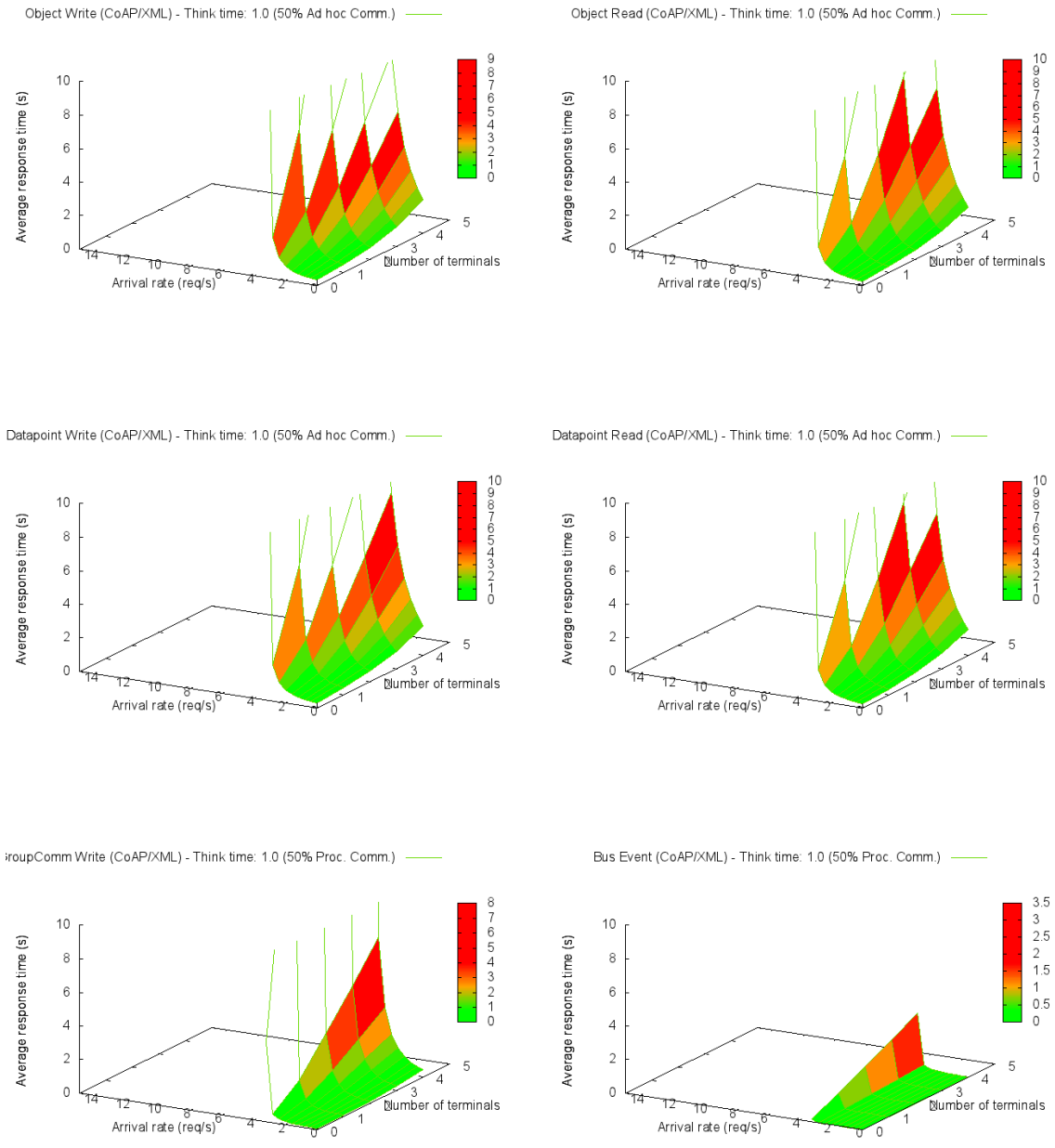


Figure 5.35: Scenario 1 - Open class expected response time

Scenario 2 - High ad-hoc communication and medium user interaction Within this scenario, the load imposed through the human interaction is lowered through doubling the user

think time to 2 seconds. The effect on the experienced average response time is slightly improved. Although, a significant performance improvement cannot be achieved (cf. Section B.1).

Scenario 3 - High ad-hoc communication and low user interaction In this scenario, the user interaction is limited through a think time of 3 seconds, which imposes a significant delay for refreshing user interface objects and mediocre user activity. Again, an improvement is measurable but no significant improvement can be achieved (cf. Section B.2).

Scenario 4 - High process communication and medium user interaction In this scenario, the process communication represents 80% of the incoming requests of the open customer class. By avoiding too much ad-hoc communication a significant improvement can be achieved (cf. Section B.3).

Scenario 5 - Highest process communication and medium user interaction The best performance can be achieved if ad-hoc communication can be reduced around 10%, leading to a significant performance improvement (cf. Section B.4).

Scenario comparison As the previous scenario shows, the achieved gateway performance strongly depends on the pattern and composition of the traffic and the user interaction. Taking the Raspberry Pi platform, the worst case performance can be observed if the process communication is disturbed by high load imposed through ad-hoc communication requests and high user interaction. Reducing the ad-hoc communication to 20% and decreasing the user delay provides a significant performance improvement, but the best performance can be achieved if the ad-hoc communication is limited to 10% and the user interaction is kept at a low interaction rate.

For the comparison of different traffic scenarios, the average response time of the process communication based on CoAP/XML is used.

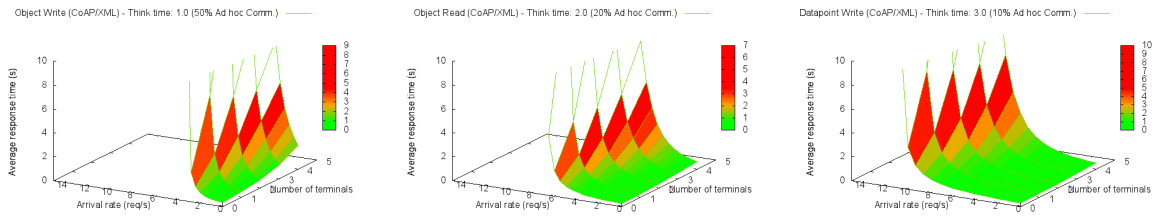


Figure 5.36: Scenario comparison

Platform comparison The previous results only focused on the Raspberry Pi platform. The results below show the difference between the other hardware platforms that have been considered. A custom developed smart board within the IoT6 research project shows a much better performance compared to the Raspberry Pi platform

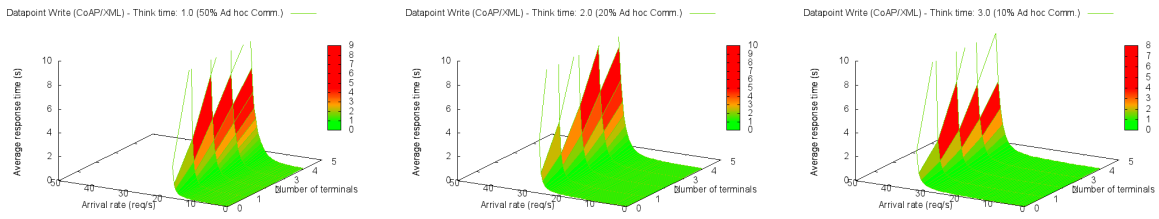


Figure 5.37: Scenario comparison - smart board

However, the performance of an x86 based PC platform is still a magnitude better. The cost difference for the various platforms is also significant.

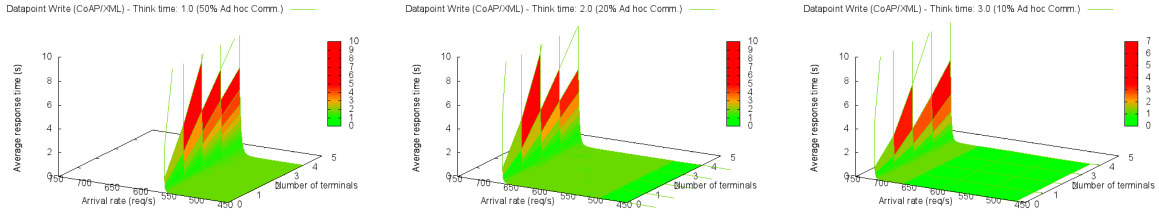


Figure 5.38: Scenario comparison - X86 platform

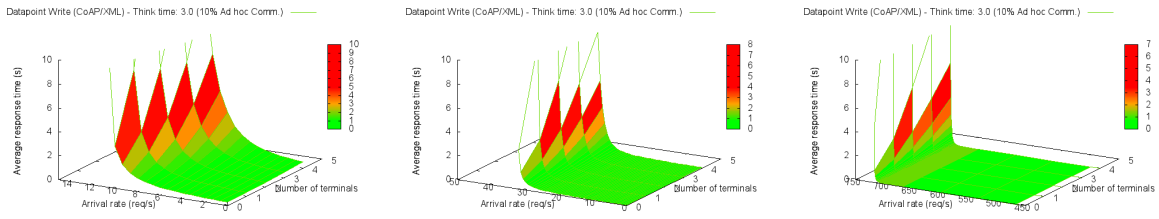


Figure 5.39: Scenario comparison - X86 platform

5.12 Conclusion

This chapter covered an IoT integration middleware, which can integrate state of the art home and building automation technologies, object identification technologies and Internet information sources into the IoT SoA. A gateway concept design is introduced, mapping the various

technologies to the IoT communication stack presented in Chapter 4. The component-oriented design of the gateway makes it easy to plug new technology connectors into the gateway and to add additional services based on the generic abstraction layer. The gateway is usable also in an IoT-only deployment, since it hosts services that demand more computational resources such as a database for persisting historic values, watch and alarming services, a group communication service, security and authorization components and a user interface to enable the engineering of constrained devices. The gateway concept is analyzed regarding its scalability through an empirical benchmark and a mixed-class analytic queuing network model. The results show that user-based interaction or external unicast-based communication degrades the performance and should be avoided in favor of process communication resting upon multicast-based group communication.

CHAPTER 6

Conclusion and further outlook

This chapters summarizes the main contributions of the thesis and provides an outlook on open issues and challenges within the context of the Internet of Things.

6.1 Contributions

This thesis presented an integration middleware consisting of an overall ICT infrastructure for the IoT, a communication stack for future IoT devices and an integration concept for state of the art technologies out of various domains relevant for the IoT. Instead of focusing on a very detailed problem, the holistic system architecture is taken into consideration in order to address the major problems of interoperability and scalability for IoT communication infrastructures.

- **IoT SoA:** Different requirements on an ICT infrastructure for the IoT are identified in the context of the Smart Grid application use case and architectural design decisions are evaluated [47,48,120]. A SoA based on Web services is identified as promising solution to reduce the integration effort within the IoT. Several application use cases can be realized on top of a such a SoA. The need for a secure and privacy aware communication infrastructure is addressed through a generic access control concept for IoT data sources [49].
- **IoT communication stack:** A holistic communication stack for the IoT is presented that demonstrates how interoperability can be achieved through providing a standardized information model and application layer services built around existing Internet and Web standards following the RESTful design paradigm. A key contribution is the concept of an efficient peer-to-peer interaction style between IoT devices through IPv6 multicasting [50, 51]. Latency and reliability can be significantly increased through this concept and the control logic can be stored in a decentralized way avoiding central controllers which might impose a single point of failure. A proof of concept implementation running on the open source operating system Contiki shows the feasibility of realizing the proposed communication stack on constrained devices. Further, the performance and

scalability of the communication stack, especially the group communication feature in constrained WSANs is evaluated using a discrete-event simulation environment.

- **Web-based commissioning:** A concept for a Web-based commissioning tool [53] shows how the communication stack can be used to provide a generic user interface for any IoT domain and how a scripting-free control logic editor and commissioning tool can support the linking of different technologies.

- **Integration middleware:** The integration concept for existing home and building automation, Smart Grid and RFID technologies beside other information sources enables a smooth path towards a large-scale deployment of IoT devices in different application domains [4, 52, 139]. With this concept, interoperability can be achieved amongst existing appliances and heterogeneous standards [132]. The integration middleware is implemented using an OSGi framework on the Java platform. Different technologies of a real-world test lab are used to execute case studies on several use cases that require the interaction of heterogeneous technologies in the context of an IoT. Furthermore, an analytic mixed-class queuing network model is used to analyze the scalability of the gateway component taking into account different hardware platforms.

- **IoTSyS:** The proof of concept implementations performed within this thesis have been released as open source project called "IoTSyS"[1] including the communication stack for constrained devices and the Java-based middleware. The project has been demonstrated within academic conferences [145, 146] as well as at industry fairs and IoT-related competitions[2].

- **Standardization:** The concept and results gathered within the thesis have been partly standardized as new protocol bindings for the OBIX standard [121, 122].

6.2 Discussion

The proposed IoT SoA can be applied to various application use cases within the Smart Grid, for example to provide access to home and building automation appliances to realize demand response interaction with third-party providers, or to offer energy feedback and being a base for energy consulting. The architecture identifies central components that are required to operate this infrastructure, but the question which stakeholder should fulfil this role and host the required components for mediating the various service consumers and service clients is still to be answered. A trustworthy entity needs to be found, especially due to the important role of hosting data access policies and protecting the privacy of consumers. The proposed access control mechanism is analyzed regarding its scalability using an analytic model and experimental evaluation based on proof of concept implementations. Here, it can be seen that a cloud-based deployment of a central policy decision point significantly improves the scalability

[1] www.iotsys.org

[2] http://www.ipso-alliance.org/challenge/ipso-challenge-2014-interviews/iotsys

of the overall systems. Nevertheless, a hierarchical system design would be beneficial, where certain policy sets are distributed to local components residing next to the data sources.

The defined policy structure is capable to specify access control policies for arbitrary SOAP-based Web services in a generic way. This general applicability is on the one side nice, since any data source can be protected, but on the other side no further domain specific semantics and information can be used for specifying the policy. For the end-user, it is hard to understand the technical terminology which is still present in formulating the policies. Although it is shown how for example OBIX-services can be controlled with the mechanism, it is still an integration effort to support other application layer protocols. Further, the access control mechanism heavily relies on the WS-* stack features, where a holistic concept also integrating RESTful Web services is required.

The presented IoT communication stack and system design are capable of replacing state of the art BAS with IoT devices. The requirements on the stack such as interoperability, scalability, versatile interaction styles, energy efficiency, security and ease of use can be fulfilled. The essential ingredients for this stack are wireless technologies. IEEE 802.15.4 together with 6LoWPAN may act as an enabler for a wireless Internet of Things. IPv6 offers global end-to-end connectivity combined with powerful multicasting capabilities that can be exploited by the application layer for peer-to-peer interaction styles. For message exchange and the application layer, CoAP together with OBIX provide an interoperable communication stack supporting the required application layer services. The presented stack has a strong focus on home and building automation scenarios, other domains such as wearable devices, consumer electronics, industrial components or devices from the traffic management domain are a bit neglected. Here, it is questionable if the OBIX meta-model is the right choice. Finally, although it is feasible to deploy such a Web service based and interoperable stack on constrained devices, it can be seen that the resource demand is significant and only due to heavy optimizations it is possible to make the complete firmware fitting on resource limited nodes. For the proof of concept and simulations, several features required for discovery and security have been omitted in order to fit the application program. It is essential not only to focus on the functional aspects of the application program, but also on the non-functional features required in a real-world deployment. To find the right balance between the services operated on a constrained device will be a challenging task.

The Web-based control logic engineering tool contributes a significant step towards an end-user ready IoT environment. The realized application use case scenarios show the feasibility of realizing complex control scenarios. Nevertheless, for operating a complete building further enhancement of the tool would be required. For example, it will be necessary to structure the control logic to multiple domains. Furthermore, currently there are no means to provide additional communication-related properties to datapoints whether they are only intended to transmit or receive signals. Here, there is also a lack for an algorithm that identifies control logic cycles that lead to an infinite control loop and make the complete system unstable.

The group communication mechanism can show the direction how a peer-to-peer interaction scheme for the IoT could look like in future. The datapoint centric communication approach makes significant improvements regarding message size efficiency or the amount of exchanged messages and also regarding memory requirements on constrained nodes. How-

ever, through the evaluation of different communication mechanisms the lack of an efficient IPv6 multicast mechanism within 6LoWPAN networks has been identified. The simple implementation of a flooding based mechanism is used for demonstration of the feasibility of the overall peer-to-peer interaction model but more research on efficient multicasting for this kind of interaction pattern needs to be found in order to increase the energy efficiency. The performed simulations make the performance improvement and the impact on energy consumption apparent.

The IoT integration middleware can integrate state of the art home and building automation technologies, object identification technologies and Internet information sources into the IoT SoA. The gateway concept design allows to easily plug new technology connectors into the gateway and to add additional services based on the generic abstraction layer. The gateway concept is analyzed regarding its scalability through an empirical benchmark and a mixed-class analytic queuing network model. The results show that user-based interaction or external unicast-based communication degrades the performance and should be avoided in favor of process communication resting upon multicast-based group communication. The proposed concept uses transient IPv6 multicast addresses in an innovative way in order to realize a smooth integration of native IoT devices and legacy devices. Nevertheless, the current concept lacks to incorporate a multi-gateway setup which would be required in the context of a large deployment within commercial buildings.

6.3 Future challenges

(Semantic) interoperability: This thesis presented a concept that achieves interoperability by using a standardized application layer information model with complementing application services. Therefore, the existing OBIX standard is used and brought down to the most constrained devices within WSANs further enhanced by a peer to peer interaction mechanism. In principle, this could be achieved with several other competing standards. However, no dominant technology has emerged, yet. Also it has been identified that the RESTful design paradigm suits well for the IoT and the integration of existing home and building automation systems. Furthermore, the achieved interoperability resides in the syntactical layer by specifying standardized exchange formats and also in the semantic layer through the use of annotated OBIX contracts to express the type of capabilities of a device. In this way, a semantic Web of Things needs to be established [147]. However, the current semantic interoperability is only accessible for human beings. As a next step, this semantics needs to be available in a machine-processable format. Expressing the information in a formalized way, opens the application to automatic control logic engineering, eases data mining and knowledge extraction in order to create, for example, behavioral profiles.

Efficient and reliable group communication WSANs: The results within this thesis show promising results for using group communication mechanisms within WSANs. It can be seen that there is still a huge standardization effort for optimal routing in WSANs. The support for multicasting is still in its infancy for networks based on 6LoWPAN. Also, the use of multicasting for communication between sensors and actuators is not well established [51, 87, 148]. The

performance results show that there is a need for new multicasting routing protocols that take into account the commissioning information which devices take part in the communication in order to optimize the communication effort. Further, the CoAP communication mechanism only provides best effort message delivery. It is still an open topic to address reliable group communication mechanisms with adjustable layers of quality-of-service depending on given constraints regarding the on energy efficiency.

Privacy-preservation and security: The proposed generic authorization mechanisms relies on SOAP-based message exchange and can be used to secure the inter-enterprise communication and the data exchange with third-parties. Here, also effective mechanisms have to be found that enable a fine-grained access control within constrained RESTful environments. This open research scope is also identified within [149]. There, the requirements for a policy language are stated, capable to define policies for different IoT domains and powerful to express different types of data assets and context of the data access. Within this thesis, a contribution towards such an IoT privacy language and privacy protection mechanism has been performed but there are still open research topics and challenges. One of these challenges are how to specify a language capable of expressing the different types of context in the environment, also to express the types of different data owners and a privacy language support for different types of physical entities. Further, the required openness to interconnect various systems in an IoT communication infrastructure leads to security problems [149], which have been partly addressed within this thesis, but leaving still significant space for future research.

Scalability: The used performance models provide an estimation of the scalability of an IoT gateway and communication stack but leave ample room for incorporating and modeling more system details. It is desirable to have further queuing network models that take beside the gateway also the connected WSANs into account and the control networks of integrated state of the art technologies, in order to make more accurate predictions on the system scalability. Such models could be especially useful for analyzing the impact of different routing mechanisms and different deployment scenarios such as a multi-gateway scenario. In general, there are no limits on the complexity of specifying such models, although a reasonable balance between simplicity and accuracy of the models have to be found in order to keep the modeling and solving effort reasonable. Nevertheless, the presented analysis in this thesis provides a methodology how to assess the scalability of an IoT system deployment and a first estimation of the system scalability bounds. The used performance models provide an estimation of the scalability of an IoT gateway and communication stack but leave place for incorporating and modeling more system details. It is desirable to have further queuing network models that take beside the gateway also the connected WSANs into account and the control networks of integrated state of the art technologies, in order to make more accurate predictions on the system scalability. Such models could be especially useful for analyzing the impact of different routing mechanisms and different deployment scenarios such as a multi-gateway scenario. In principle, there are no limits on the complexity of specifying such models, although a reasonable balance between simplicity and accuracy of the models has to be found in order to keep the modeling and solving efforts reasonable. Nevertheless, the presented analysis in this thesis provides a sound

methodology how to assess the scalability of an IoT system deployment and a first estimation of the system scalability bounds.

Glossary

6LoWPAN IPv6 over Low Power Wireless Personal Area Networks

AES Advanced Encryption Standard

AH Authentication Header

ALE Application Level Event

API Application Programming Interface

BluetoothLE Bluetooth Low Energy

BACnet Data Communication Protocol for Building Automation and Control Networks

BACnet/WS BACnet Web services

BAS Building Automation System

CAP Compact Application Protocol

CIM Common Information Model

CoAP Constrained Application Protocol

CoRE Constrained RESTful Environments

COM Component Object Model

CPU Central Processing Unit

COSEM Companion Specification for Energy Metering

CSS Cascading Style Sheets

DAO Destination Advertisement Object

DCOM Distributed Component Object Model

DLMS Device Language Specification

DM Device Management

DNS Domain Name System

DNS-SD DNS Service Discovery

DODAG Destination Oriented Directed Acyclic Graph

DOM Document Object Model

DPWS Device Profile for Web Services

DTLS Datagram Transport Layer Security

ebXML Electronic Business using XML

EEP EnOcean Equipment Profile

EPAL Enterprise Privacy Authorization Language

EPC Electronic Product Code

EPCIS EPC Information System

ERP EnOcean Radio Protocol

ESP Encapsulating Security Payload

ETSI M2M ETSI M2M

EXI Efficient XML Interchange

FnC Filter and Collection

FTP File Transfer Protocol

GAE Google App Engine

GRAI Global Returnable Asset Identifier

GTIN Global Trade Item Number

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS HTTP Secure

HVAC Heating, Ventilation and Air Conditioning

IaaS Infrastructure as a Service

ICT Information- and Communication-Technologies

IDP Identity Provider

IoT Internet of Things

IP Internet Protocol

IPSec IPSecurity

IPSO Internet Protocol for Smart Objects

IPv4 Internet Protocol Version 4

IPv6 Internet Protocol Version 6

JSON Javascript Object Notation

LDAP Lightweight Directory Access Protocol

LWM2M Lightweight M2M

mDNS Multicast DNS

MPL Multicast Protocol for Low power and Lossy Networks

MQTT Message Queueing Telemetry Transport

MQTT-S MQTT for Sensor Networks

MVA Mean Value Analysis

MVC Model-View-Controller

M2M Machine to Machine

NFC Near Field Communication

OASIS Organization for the Advancement of Structured Information Standards

OAuth OAuth 2.0 Authorization Framework

OBIS Object Identification System

OBIX Open Building Information eXchange

OMA Open Mobile Alliance

OPC OLE for process control

OPC UA OPC Unified Architecture

OSGi Open Service Gateway Initiative

PaaS Platform as a Service

PAP Policy Administration Point

PDP Policy Decision Point

PEP Policy Enforcement Point

PID Proportional-Integral-Derivative

PIP Policy Information Point

PKI Public Key Infrastructure

QN Queueing Network

QoS Quality of Service

RDC Radio Duty Cycling

RMI Remote Method Invocation

RPC Remote Procedure Call

REST Representational State Transfer

RFID Radio-Frequency Identification

RTP Real Time Protocol

RPL Routing Protocol for Low-Power and Lossy Networks

SaaS Software as a Service

SAML Security Assertion Markup Language

SC Service Consumer

SenML Sensor Markup Language

SGTIN Serialized Global Trade Item Number

SIP Session Initiation Protocol

SMRF Stateless Multicast Forwarding with RPL

SNMP Simple Network Management Protocol

SoA Service-oriented Architecture

SOAP Simple Object Access Protocol

SP Service Provider

SSO Single-Sign On

STIS Smart Things Information System

TCP Transmission Control Protocol

TLS Transport Layer Security

UDDI Universal Description, Discovery and Integration

UDP User Datagram Protocol

UPnP Universal Plug and Play

URI Uniform Resource Identifier

URL Uniform Resource Locator

WoT Web of Things

WSAN Wireless Sensor and Actuator Network

WADL Web Application Description Language

WSDL Web Service Description Language

WSN Wireless Sensor Network

XACML Extensible Access Control Markup Language

XML eXtensible Markup Language

XMPP Extensible Messaging and Presence Protocol

APPENDIX A

Use case description for integration middleware case study

This chapter contains a detailed description used for the integration middleware case study based on the casual use case description defined by Cockburn [144].

Title:	**User authentication**
Primary actor:	Administrator
Scope:	System
Level:	User goal
Story:	The administrator opens with the Web browser the HTML5 user interface. A username and a passphrase are provided. If the username and passphrase are correct, the user is authenticated as administrator. The user can end the session through a logout.

Table A.1: User authentication

Title:	**Commissioning of control logic**
Primary actor:	Administrator
Scope:	System
Level:	User goal
Story:	The administrator opens the object browser and drags device objects to the canvas. Further, logic blocks are added to the canvas. By graphically wiring datapoints of device objects and logic blocks together the control logic is created.

Table A.2: Commissioning of control logic

Title:	**Monitor devices**
Primary actor:	Administrator
Scope:	System
Level:	User goal
Story:	The authenticated administrator can browse the available device objects and drag them into the canvas area of the user interface. Datapoint values of sensors and actuators, are updated in real-time. Historic values are graphically illustrated.

Table A.3: Monitor devices

Title:	**Control devices**
Primary actor:	Administrator
Scope:	System
Level:	User goal
Story:	The authenticated administrator can browse the available device objects and drag them into the canvas area of the user interface. For datapoint values of sensors and actuators input widgets like check-boxes, text-boxes are displayed. The administrator can modify values and the involved devices change their state.

Table A.4: Monitor devices

Title:	**Light control**
Primary actor:	Light controls
Scope:	System
Level:	User goal
Story:	The most simple interaction scenario is the use of heterogeneous light switch actuators combined with push buttons of other technologies. For this scenario, KNX switching actuators are combined with an EnOcean push button and a 6LoWPAN sensor also acting as push button.

Table A.5: Light control

Title:	**Life safety**
Primary actor:	Free fall sensor, Alarm text display
Scope:	System
Level:	User goal
Story:	The life safety use case includes an accelerometer sensor that is linked to an alarm signal and a text display. If an accident of an inhabitant is detected, an alarm is raised and a text message is displayed. A human person needs to manually acknowledge the alarm.

Table A.6: Life safety

Title:	**HVAC**
Primary actor:	Inhabitant, Devices
Scope:	System
Level:	User goal
Story:	In this scenario, an existing HVAC control process based on BACnet is enhanced with KNX room automation devices and 6LoWPAN temperatures sensors. The BACnet controller is responsible for controlling a water boiler to create heating water and a chiller, which is used for the cooling process. The hot and warm water are circulated using pumps that are also controlled through BACnet. Heating and cooling registers are combined with fans and valves that allow the air flow and finally cool or heat a room to a desired setpoint. Through the IoT architecture, the closed BACnet HVAC process can be enhanced with KNX devices. A KNX room thermostat allows adjusting the desired room setpoint and can further be used to visualize the current mode (comfort or standby) of the HVAC process. An EnOcean window contact sensor can be used to activate the standby mode if a window is opened. Further, a smart meter can measure the current energy consumption in real-time and depending on a given policy the HVAC process can be adjusted to reduce the energy consumption. This can be done either in an indirect way by overwriting the room setpoint, or directly by adjusting the mode of the chiller, boiler, fans and pumps.

Table A.7: HVAC

Title:	**Energy efficient HVAC**
Primary actor:	Inhabitant, Devices
Scope:	System
Level:	User goal
Story:	This use case extends the HVAC use case through an energy safe mode that can be activated at the temperature controller. If the measured power consumption exceeds a certain threshold the safe energy mode is activated.

Table A.8: Energy efficient HVAC

Title:	**Access control and smart office**
Primary actor:	Light controls
Scope:	System
Level:	User goal
Story:	In this scenario, an RFID reader is combined with a KNX switching actuator that controls an automated door opener. An employee identity card or a passport can be used for identification and upon a successful identification the door is opened. Further, personalized settings of the card holder can be used to configure the room temperature setpoint and lighting scenario.

Table A.9: Access control and smart office

Title:	**Weather warning**
Primary actor:	Device, Weather data
Scope:	System
Level:	User goal
Story:	For the alarming scenario, the weather data can be used to inform the residents of an upcoming storm. If a storm is approaching and the windows are closed a simple visual notification and warning using a KNX text display can be done. If the EnOcean window contact sensor indicates an opened window at that time an acoustic signal is also provided to warn the inhabitants about the approaching storm.

Table A.10: Weather warning

Title:	**IoT peer to peer interaction**
Primary actor:	Inhabitant, 6LoWPAN devices
Scope:	System
Level:	User goal
Story:	The IoT peer to peer interaction demonstrates the group communication capabilities. A communication relationship between a 6LoWPAN push button and a 6LoWPAN LED actuator is established and the IoT gateway is switched off. The interaction between the devices keeps working.

Table A.11: IoT peer to peer interaction

APPENDIX B

Gateway scalability analysis results

This chapter contains all results related to the scalability analysis of the integration middleware gateway.

B.1 Scenario 2

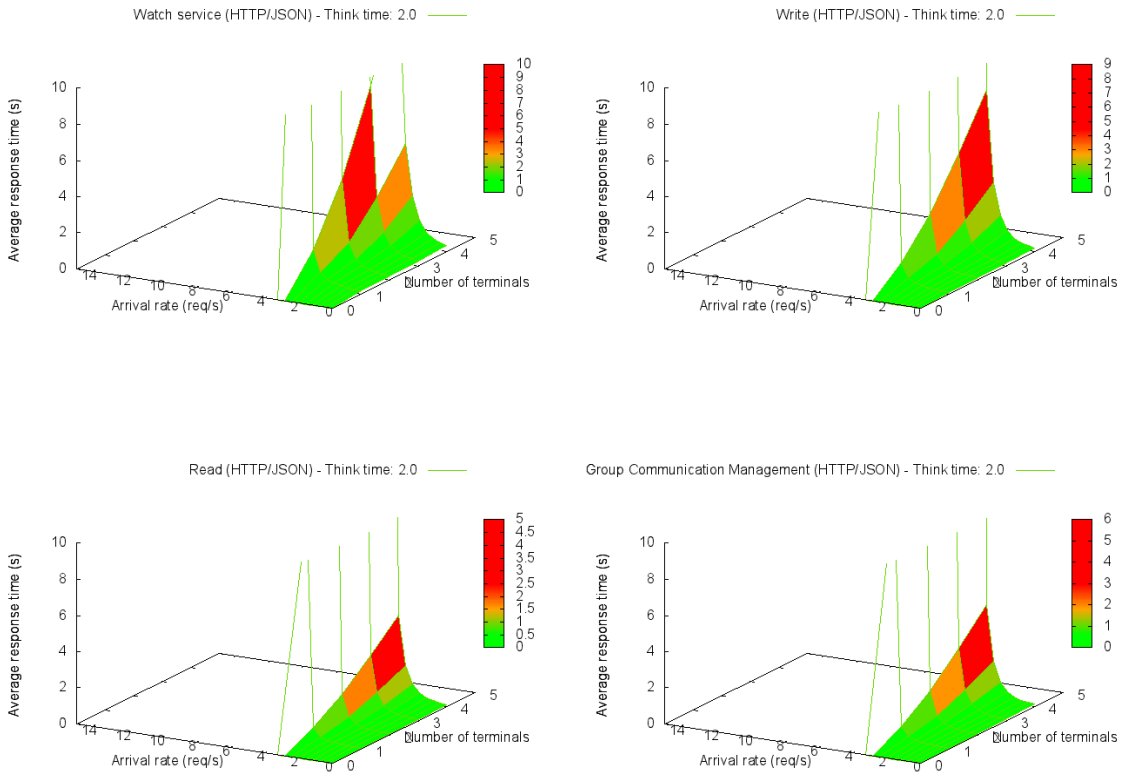


Figure B.1: Scenario 2 - Closed class expected response time

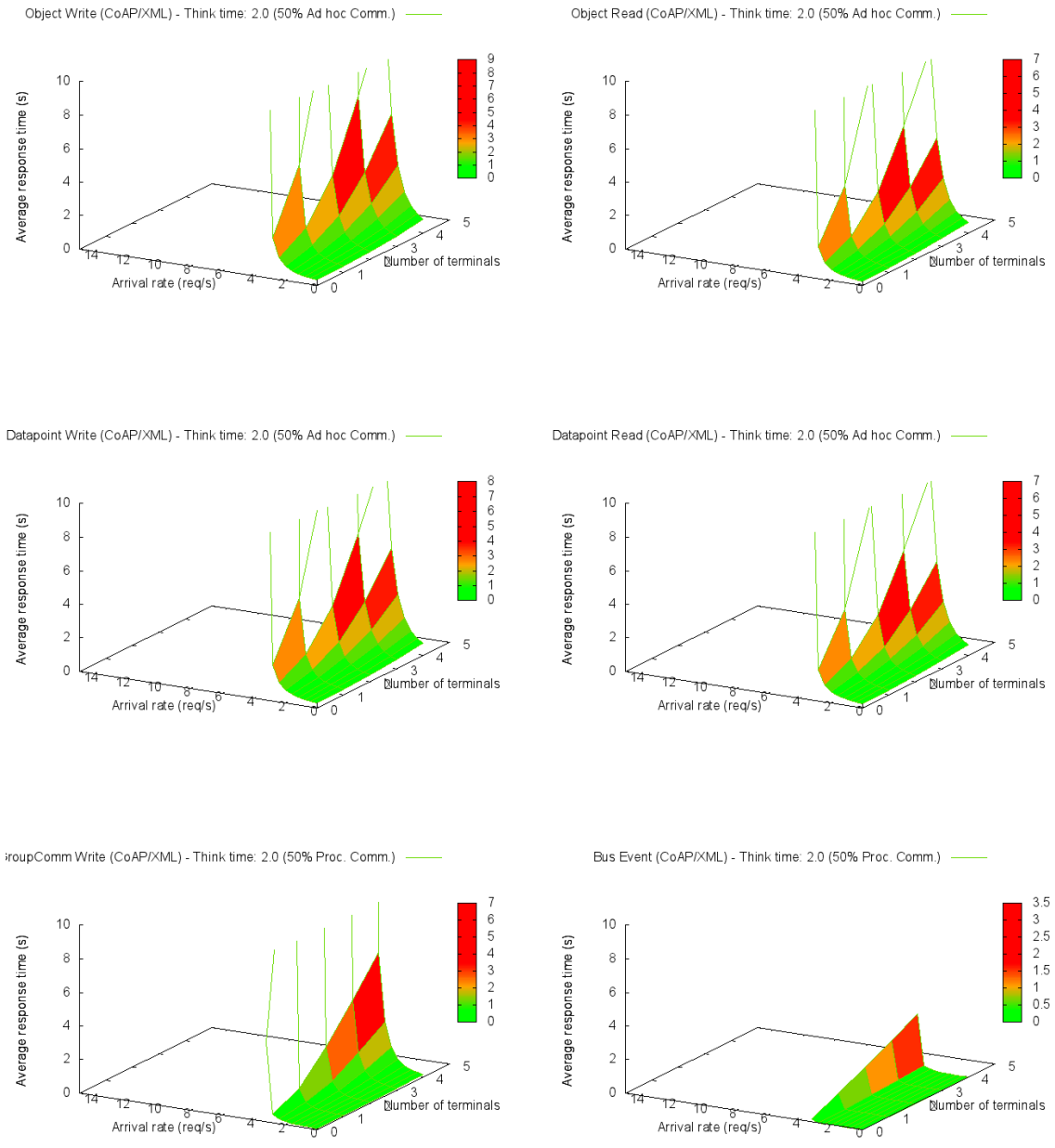


Figure B.2: Scenario 2 - Open class expected response time

B.2 Scenario 3

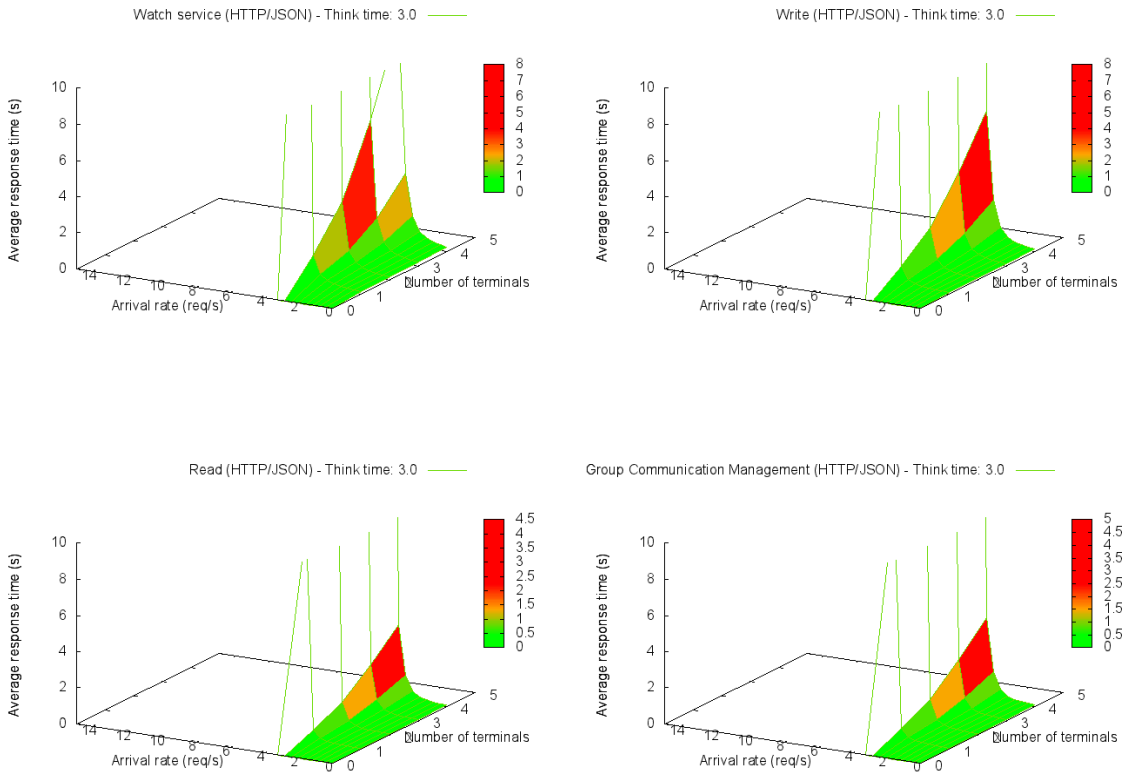


Figure B.3: Scenario 3 - Closed class expected response time

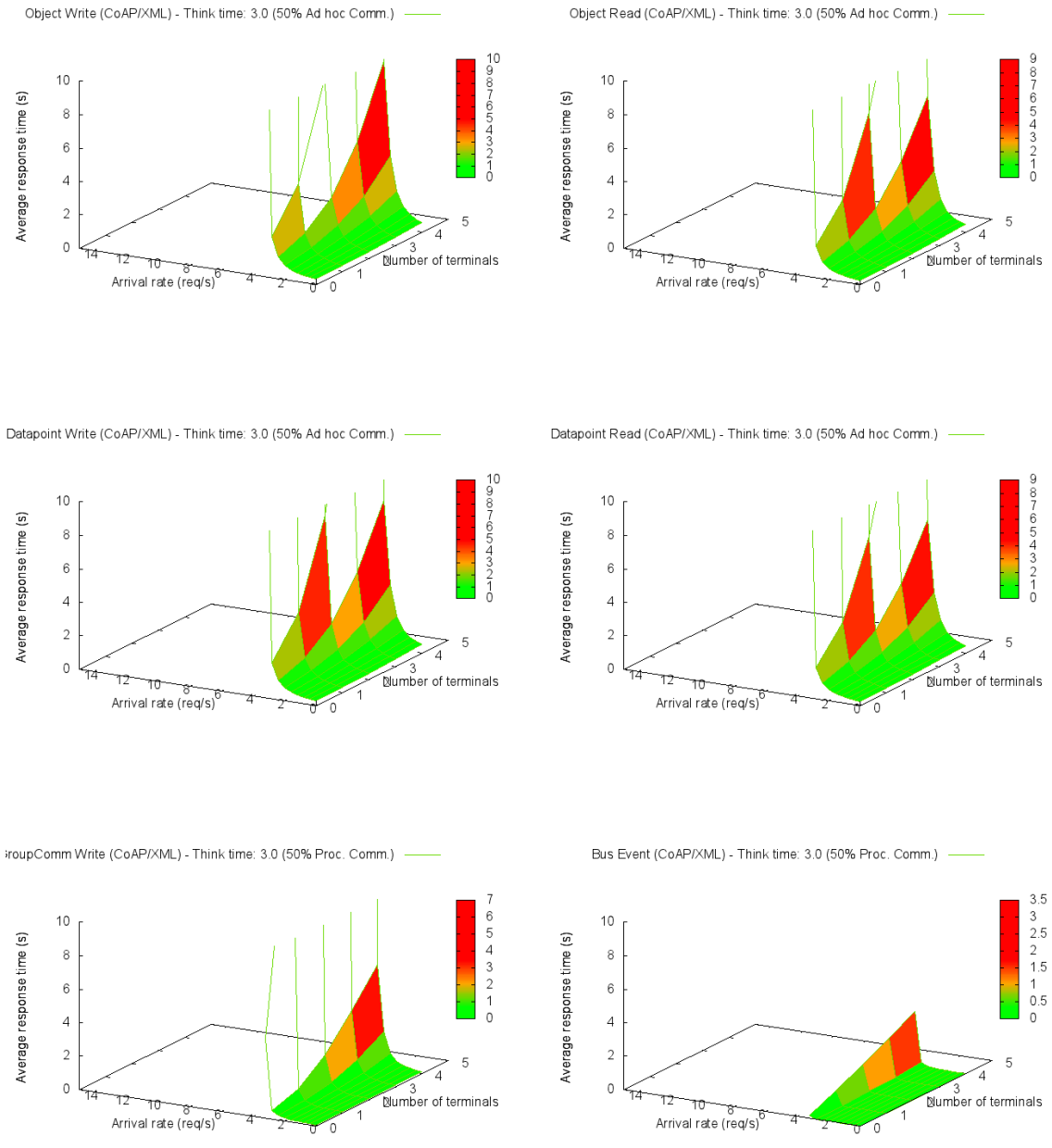


Figure B.4: Scenario 3 - Open class expected response time

B.3 Scenario 4

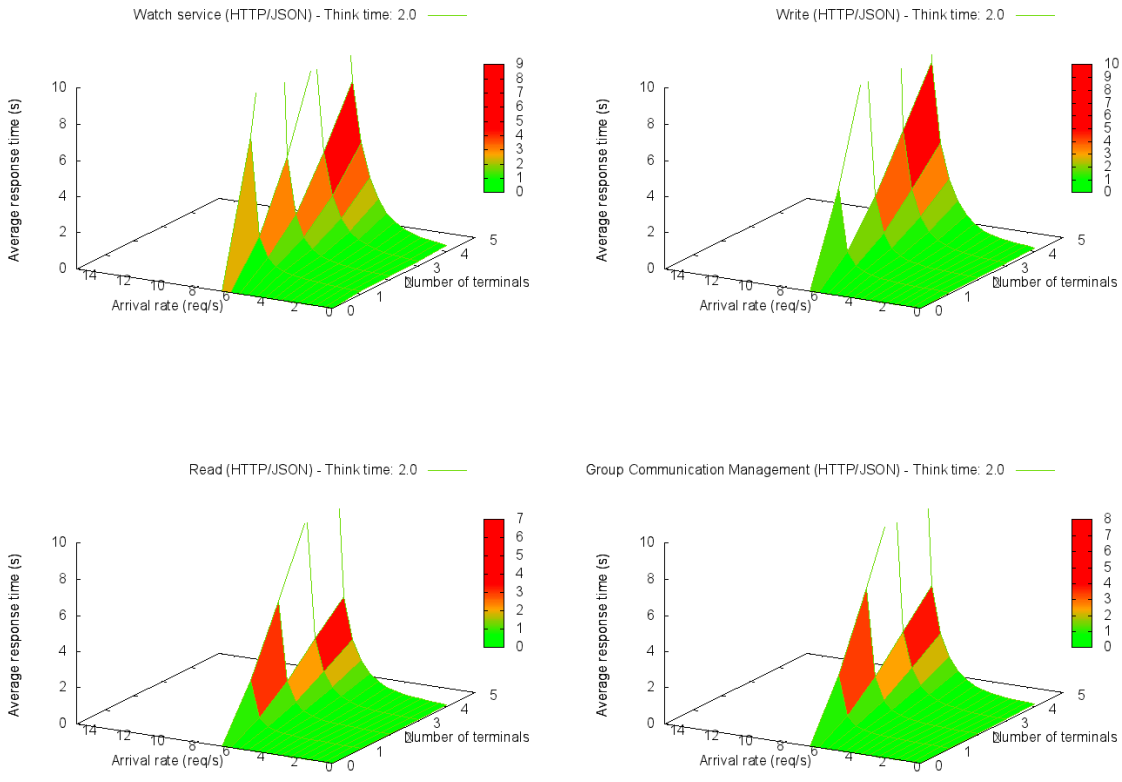


Figure B.5: Scenario 4 - Closed class expected response time

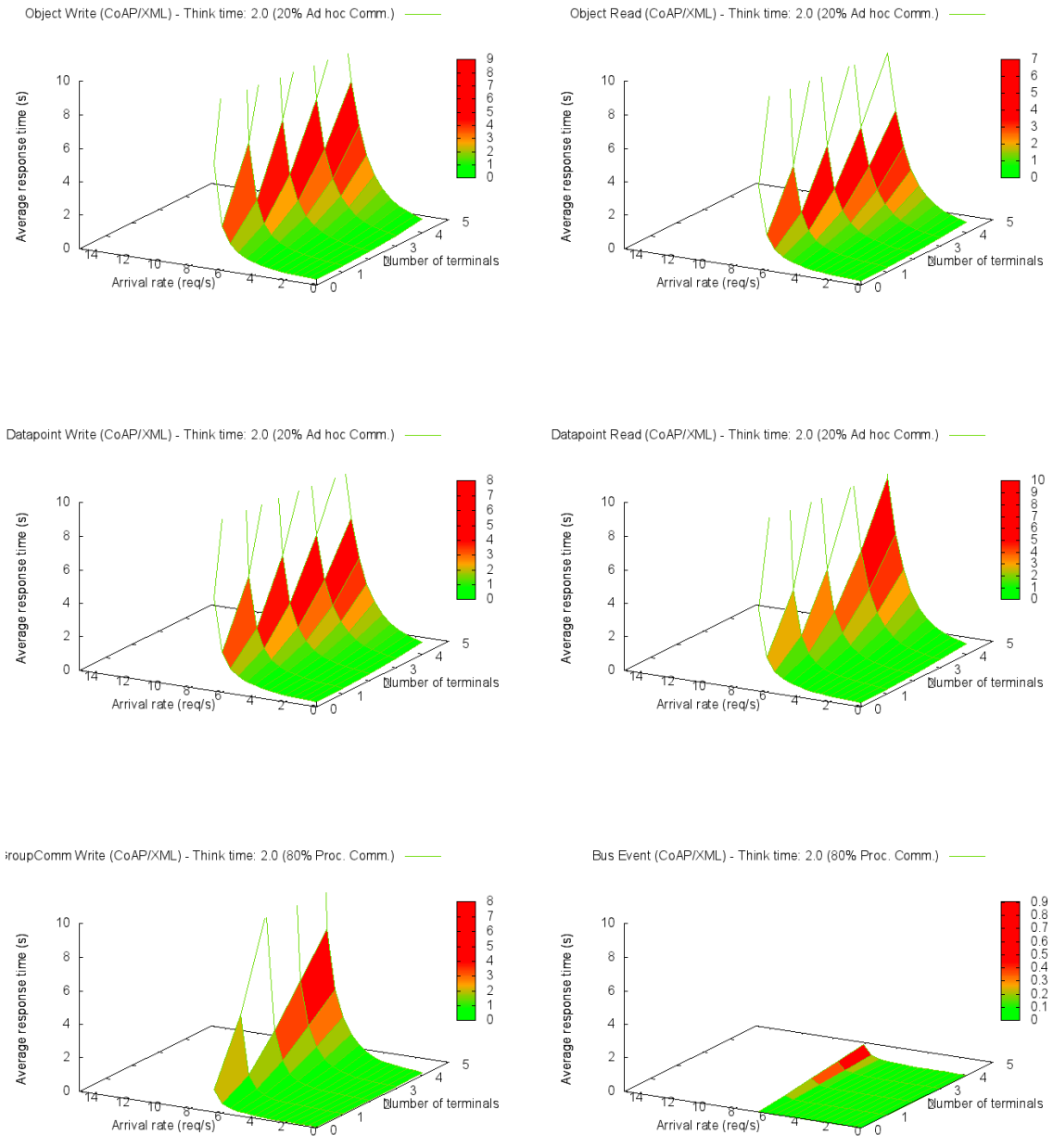


Figure B.6: Scenario 4 - Open class expected response time

B.4 Scenario 5

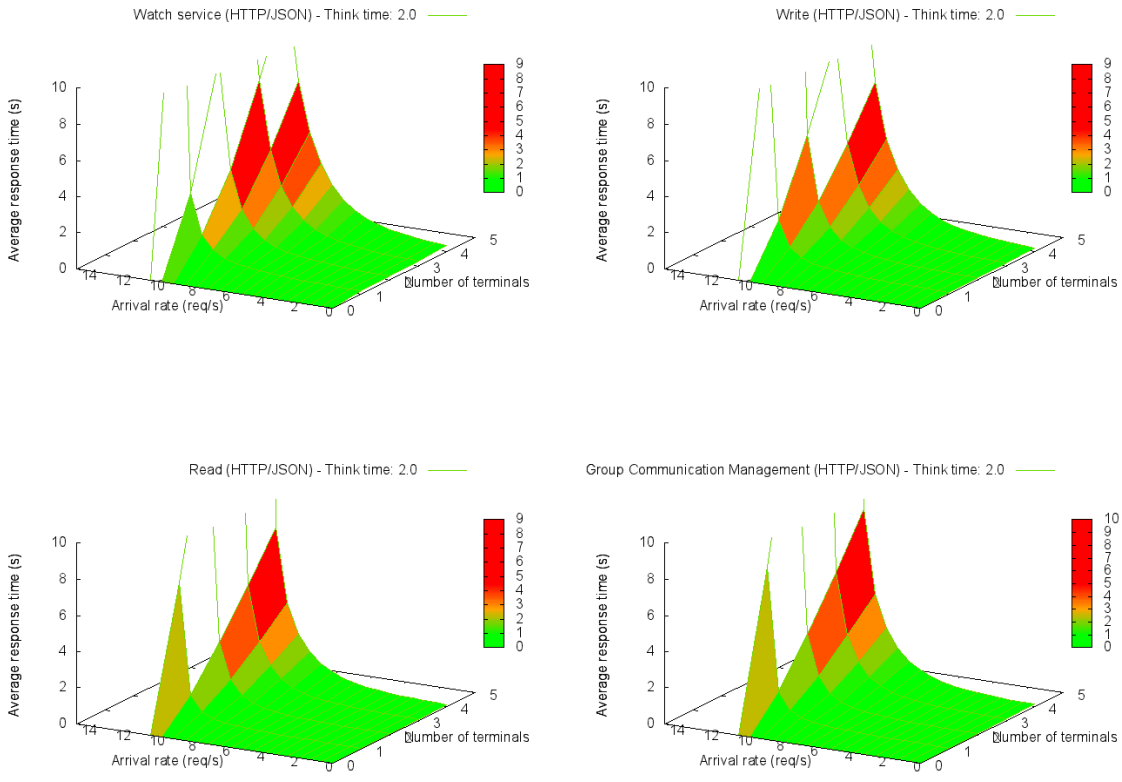


Figure B.7: Scenario 5 - Closed class expected response time

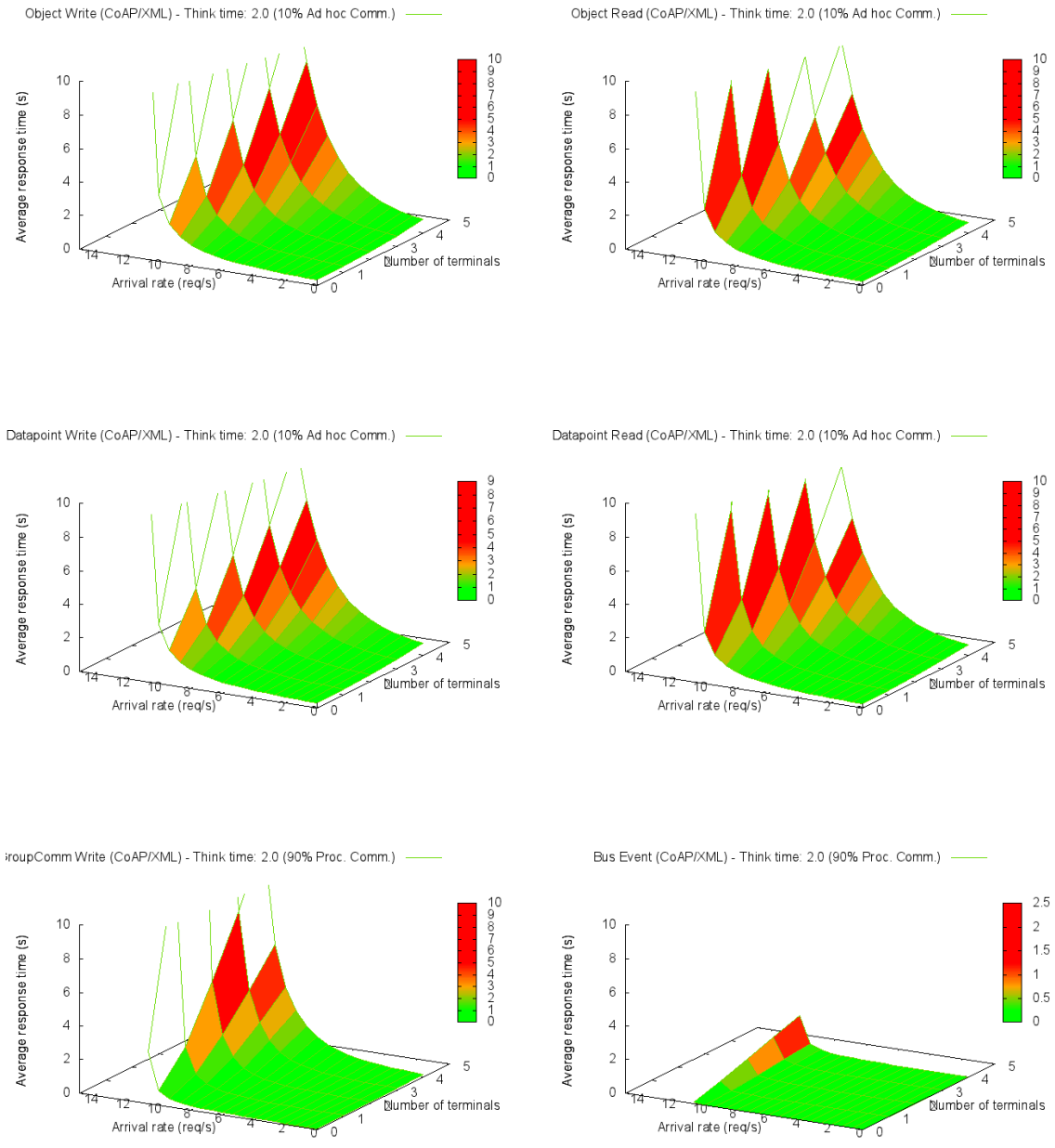


Figure B.8: Scenario 5 - Open class expected response time

Bibliography

[1] D. Brock, "The Electronic Product Code (EPC)," *A Naming Scheme for Physical Objects*, 2001.

[2] Z. Shelby and C. Bormann, *6LoWPAN: The wireless embedded Internet*. John Wiley & Sons, 2011, vol. 43.

[3] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[4] M. Jung, C. Reinisch, and W. Kastner, "Integrating building automation systems and IPv6 in the Internet of Things," in *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2012, pp. 683–688.

[5] A. Dunkels, "Full TCP/IP for 8-bit architectures," in *Proceedings of the International Conference on Mobile systems, Applications and Services*. ACM, 2003, pp. 85–98.

[6] Internet Engineering Task Force (IETF), "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals – RFC 4919," Aug. 2007.

[7] D. Guinard, "A Web of Things Application Architecture – Integrating the Real-World into the Web," Ph.D. thesis, ETH Zurich, 2011.

[8] R. Fielding, "Representational state transfer," *Architectural Styles and the Design of Network-based Software Architecture*, pp. 76–85, 2000.

[9] Internet Engineering Task Force (IETF), "The Constrained Application Protocol (CoAP) – RFC 7252," 2014.

[10] Z. Shelby, "Embedded Web Services," *Wireless Communications, IEEE*, vol. 17, no. 6, pp. 52–57, 2010.

[11] A. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, "Web Services for the Internet of Things through CoAP and EXI," in *Proceedings of the International Conference on Communications Workshops*. IEEE, 2011, pp. 1–6.

[12] D. Emmerich and E. Bloom, "Commercial Building Automation Systems - Security and Access, HVAC Controls, Fire and Life Safety, Building Management Systems and Lighting Controls: Global Market Analysis and Forecasts," 2012.

[13] C. Petersdorff, T. Boermans, and J. Harnisch, "Mitigation of CO2 emissions from the EU-15 building stock. Beyond the EU directive on the energy performance of buildings," *Environmental Science and Pollution Research*, vol. 13, no. 5, pp. 350–358, 2006.

[14] S. Darby *et al.*, "The effectiveness of feedback on energy consumption," *A Review for DEFRA of the Literature on Metering, Billing and direct Displays*, vol. 486, pp. 1–24, 2006.

[15] NIST, "Framework and Roadmap for Smart Grid Interoperability Standards, Release 2.0," *NIST special publication*, vol. 1108, 2012.

[16] W. Kastner, M. Jung, and L. Krammer, "Future Trends in Smart Homes and Buildings," in *Industrial Communication Technology Handbook, Second Edition*, R. Zurawski, Ed. CRC Press, Inc., 2014, ch. 59.

[17] "KNX Specifications, Version 2.0," Konnex Association, 2009.

[18] International Electrotechnical Commission, "Information technology – Home electronic system (HES) architecture – Part 3-10: Wireless short-packet (WSP) protocol optimised for energy harvesting – Architecture and lower layer protocols," IEC 14543-3-10, 2012.

[19] "BACnet – A Data Communication Protocol for Building Automation and Control Networks," ANSI/ASHRAE Std. 135, 1995–2010.

[20] "Building automation and control systems (BACS) – Part 5: Data communication protocol," ISO 16484-5, 2012.

[21] "ZigBee IP Specification," ZigBee Alliance, 2013.

[22] J.-P. Vasseur and A. Dunkels, *Interconnecting smart objects with IP: The next Internet*. Morgan Kaufmann, 2010.

[23] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: A tiny aggregation service for ad-hoc sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 131–146, 2002.

[24] P. J. Marrón, D. Minder, A. Lachenmann, and K. Rothermel, "TinyCubus: An Adaptive Cross-Layer Framework for Sensor Networks (TinyCubus: Ein Adaptives Cross-Layer Framework für Sensornetze)," *it-Information Technology*, vol. 47, no. 2, pp. 87–97, 2005.

[25] J. Heidemann, F. Silva, C. Intanagonwiwat, R. Govindan, D. Estrin, and D. Ganesan, "Building efficient wireless sensor networks with low-level naming," in *ACM SIGOPS Operating Systems Review*, vol. 35, no. 5. ACM, 2001, pp. 146–159.

[26] O. Gnawali, K.-Y. Jang, J. Paek, M. Vieira, R. Govindan, B. Greenstein, A. Joki, D. Estrin, and E. Kohler, "The tenet architecture for tiered sensor networks," in *Proceedings of the International Conference on Embedded Networked Sensor Systems*. ACM, 2006, pp. 153–166.

[27] A. Dunkels, F. Österlind, and Z. He, "An adaptive communication architecture for wireless sensor networks," in *Proceedings of the 5th International Conference on Embedded Networked Sensor Systems*. ACM, 2007, pp. 335–349.

[28] D. E. Culler, P. Dutta, C. T. Ee, R. Fonseca, J. Hui, P. Levis, J. Polastre, S. Shenker, I. Stoica, G. Tolle *et al.*, "Towards a sensor network architecture: Lowering the waistline," in *HotOS*, 2005.

[29] C. T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker, and I. Stoica, "A modular network layer for sensorsets," in *Proceedings of the Symposium on Operating Systems Design and Implementation*. USENIX Association, 2006, pp. 249–262.

[30] MarketsandMarkets, "Internet of Things (IoT) & Machine-to-Machine (M2M) communication market," 2014.

[31] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful Web services vs. big Web services: making the right architectural decision," in *Proceedings of the International Conference on World Wide Web*. ACM, 2008, pp. 805–814.

[32] N. Strother and B. Gohn, "Home Energy Management – In-Home Displays, Networked HEM Systems, Standalone HEM Systems, Web Portals, and Paper Bill HEM Reports: Market Analysis and Forecasts," 2012.

[33] C. Groba and S. Clarke, "Web services on embedded systems – a performance study," in *Proceedings of the International Conference on Pervasive Computing and Communications Workshops*. IEEE, 2010, pp. 726–731.

[34] D. Schall, M. Aiello, and S. Dustdar, "Web services on embedded devices," *International Journal of Web Information Systems*, vol. 2, no. 1, pp. 45–50, 2006.

[35] M. Berger, T. Hofer, F. Judex, M. Jung, G. Kienesberger, M. Meisel, M. Pichler, S. Prost, W. Prüggler, and K. Röderer, "SGMS – Smart Web Grid," http://www.smartgridssalzburg.at/forschungsfelder/ikt/smart-web-grid/, 2014.

[36] D. Pilone and N. Pitman, *UML 2.0 in a Nutshell*. O Reilly Media, Inc., 2005.

[37] J. Tyree and A. Akerman, "Architecture decisions: demystifying architecture," *IEEE software*, vol. 22, no. 2, pp. 19–27, 2005.

[38] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proceedings of the Conference on Software Architecture*. IEEE/IFIP, 2005, pp. 109–120.

[39] P. B. Kruchten, "The 4+1 view model of architecture," *Software, IEEE*, vol. 12, no. 6, pp. 42–50, 1995.

[40] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the International Conference on Local Computer Networks*. IEEE, 2004, pp. 455–462.

[41] D. A. Menasce, V. A. Almeida, L. W. Dowdy, and L. Dowdy, *Performance by design: computer capacity planning by example*. Prentice Hall Professional, 2004.

[42] D. A. Menascé and V. Almeida, *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall PTR, 2001.

[43] F. Baskett, M. Chandy, R. Muntz, and F. Palacios, "Open, closed, and mixed networks of queues with different classes of customers," *Journal of the ACM (JACM)*, vol. 22, no. 2, pp. 248–260, 1975.

[44] L. Kleinrock, *Queueing Systems, Volume I: Theory*. Wiley Interscience, 1975.

[45] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi, *Queueing Networks and Markov Chains*. John Wiley & Sons, Inc, 2000.

[46] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-Level Sensor Network Simulation with COOJA," in *Proceedings of the IEEE Conference on Local Computer Networks*. IEEE, 2006, pp. 641–648.

[47] A. J. Jara, A. C. Olivieri, Y. Bocchi, M. Jung, W. Kastner, and A. F. Skarmeta, "Semantic Web of Things: an analysis of the application semantics for the IoT moving towards the IoT convergence," *Journal of Web and Grid Services*, vol. 10, pp. 244 – 272, 2014.

[48] M. Jung, W. Kastner, G. Kienesberger, and M. Leithner, "A comparison of Web service technologies for smart meter data exchange," in *Proceedings of the IEEE PES Innovative Smart Grid Technologies Europe Conference*. IEEE, 2012, pp. 1–8.

[49] M. Jung, T. Hofer, W. Kastner, and S. Döbelt, "Protecting data assets in a Smart Grid SoA," *Journal of Internet Technology and Secured Transactions*, vol. 2, pp. 155 – 166, 2013.

[50] M. Jung and W. Kastner, "Efficient group communication based on Web services for reliable control in wireless automation," in *Proceedings of the Conference of the Industrial Electronics Society*. IEEE, 2013, pp. 5716–5722.

[51] M. Jung, P. Raich, and W. Kastner, "The relevance and impact of IPv6 multicasting for Wireless Sensor and Actuator Networks based on 6LoWPAN and Constrained RESTful Environments," in *Proceedings of the International Conference on the Internet of Things*, 2014.

[52] M. Jung, J. Weidinger, C. Reinisch, W. Kastner, C. Crettaz, A. Olivieri, and Y. Bocchi, "A transparent IPv6 multi-protocol gateway to integrate Building Automation Systems in the Internet of Things," in *Proceedings of the International Conference on Green Computing and Communications*. IEEE, 2012, pp. 225–233.

[53] M. Jung, E. Hajdarevic, W. Kastner, and A. J. Jara, "Short Paper: A Scripting-Free Control Logic Editor for the Internet of Things," in *Proceedings of the IEEE World Forum on Internet of Things*. IEEE, 2014, pp. 193–194.

[54] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer, "Simple object access protocol (SOAP) 1.1," *W3C Note*, 2000.

[55] "Web Services Description Language (WSDL) 1.1," W3C Note, 2001.

[56] "Extensible Markup Language (XML) 1.1," W3C Recommendation, 2006.

[57] "XML Schema Part 1: Structures," W3C Recommendation, 2004.

[58] "XML Schema Part 2: Datatypes," W3C Recommendation, 2004.

[59] "Efficient XML Interchange (EXI) Format 1.0," W3C Recommendation, 2004.

[60] International Electrotechnical Commission, "Common Information Model (CIM) / Energy Management," IEC 61970-301, 2003.

[61] M. Papazoglou and D. Georgakopoulos, "Service-oriented computing," *Communications of the ACM*, vol. 46, no. 10, pp. 25–28, 2003.

[62] R. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. thesis, University of California, 2000.

[63] "CoRE Resource Directory," IETF draft, 2013.

[64] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web services*. Springer, 2004.

[65] European Telecommunications Standards Institute, "Machine-to-Machine communications (M2M); mIa, dIa and mId interfaces - ETSI TS 102 921 V1.1.1," Feb. 2012.

[66] FI-WARE, "ETSI M2M mId Open RESTful API Specification," 2012.

[67] Interdigital, "White paper: Standardized Machine-to-Machine (M2M) Software Development Platform," 2012.

[68] W. Mahnke, S.-H. Leitner, and M. Damm, *OPC unified architecture*. Springer, 2009.

[69] D. Guinard, V. Trifa, F. Mattern, and E. Wilde, "From the Internet of Things to the Web of Things: resource-oriented architecture and best practices," in *Architecting the Internet of Things*. Springer, 2011, pp. 97–129.

[70] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the Web of Things," in *In Proccedings of the International Conference on Internet of Things*. IEEE, 2010, pp. 1–8.

[71] V. Trifa, D. Guinard, V. Davidovski, A. Kamilaris, and I. Delchev, "Web messaging for open and scalable distributed sensing applications," in *Proceedings of the International Conference on Web Engineering*, 2010, pp. 129–143.

[72] E. Wilde, "Putting things to rest," *School of Information*, 2007.

[73] D. Raggett, "The Web of Things: Extending the Web into the real world," in *SOFSEM 2010: Theory and Practice of Computer Science*. Springer, 2010, pp. 96–107.

[74] D. Guinard and V. Trifa, "Towards the Web of Things: Web mashups for embedded devices," in *Proceedings of the International World Wide Web Conferences*, 2009, pp. 1–8.

[75] V. Trifa, S. Wieland, D. Guinard, and T. Bohnert, "Design and implementation of a gateway for Web-based interaction and management of embedded devices," in *Proceedings of the International Workhop on Sensor Network Engineering*, 2009.

[76] G. Moritz, F. Golatowski, and D. Timmermann, "A Lightweight SOAP over CoAP Transport Binding for Resource Constraint Networks," in *Proceedings of the International Conference on Mobile Adhoc and Sensor Systems (MASS)*. IEEE, 2011, pp. 861–866.

[77] D. van der Linden, W. Granzer, and W. Kastner, "OPC Unified Architecture (OPC UA) new opportunities of system integration and information modeling in automation systems," in *Proceedings of the International Conference on Industrial Informatics*. IEEE, 2011.

[78] W. Kastner and S. Szucsich, "Accessing KNX networks via BACnet/WS," in *Proceedings of the International Symposium on Industrial Electronics*. IEEE, 2011, pp. 1315–1320.

[79] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner, "Web Services in Building Automation: Mapping KNX to oBIX," in *Proceedings of the International Conference on Industrial Informatics*. IEEE, 2007.

[80] C. Reinisch, W. Granzer, F. Praus, and W. Kastner, "Integration of Heterogeneous Building Automation Systems using Ontologies," in *Proceedings of the Conference Industrial Electronics Society*, IEEE, 2008.

[81] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, M. Pagel, M. Hauswirth, M. Karnstedt *et al.*, "SPITFIRE: toward a semantic Web of Things," *Communications Magazine*, vol. 49, no. 11, pp. 40–48, 2011.

[82] M. Kovatsch, M. Lanter, and S. Duquennoy, "Actinium: A RESTful Runtime Container for Scriptable Internet of Things Applications," in *Proceedings of the International Conference on the Internet of Things*, 2012, pp. 135–142.

[83] A. Rahman and E. Dijk, "Group Communication for CoAP," IETF Draft, 2014.

[84] I. Ishaq, J. Hoebeke, F. Van den Abeele, I. Moerman, and P. Demeester, "Group Communication in Constrained Environments Using CoAP-based Entities," in *Proceedings of the International Conference on Distributed Computing in Sensor Systems*, 2013, pp. 345–350.

[85] C. Reinisch, W. Kastner, and G. Neugschwandtner, "Multicast communication in wireless home and building automation: ZigBee and DCMP," in *Proceedings of the International Conference on Emerging Technologies and Factory Automation.* IEEE, 2007, pp. 1380–1383.

[86] O. Gaddour, A. Koubaa, O. Cheikhrouhou, and M. Abid, "Z-Cast: a multicast routing mechanism in ZigBee cluster-tree wireless sensor networks," in *Proceedings of the International Conference on Distributed Computing Systems Workshops.* IEEE, 2010, pp. 171–179.

[87] G. Oikonomou and I. Phillips, "Stateless multicast forwarding with RPL in 6LoWPAN sensor networks," in *Proceedings of the International Conference on Pervasive Computing and Communications Workshops.* IEEE, 2012, pp. 272–277.

[88] Internet Engineering Task Force (IETF), "The Trickle Algorithm – RFC 6206," Mar. 2011.

[89] M. Blackstock and R. Lea, "IoT mashups with the WoTKit," in *Proceedings of the International Conference on the Internet of Things*, 2012, pp. 159–166.

[90] D. C. Pintus, Antonio and A. Piras, "Paraimpu: a platform for a social Web of things," in *Proceedings of the International Conference Companion on World Wide Web.* ACM, 2012.

[91] International Electrotechnical Commission, "Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM)," IEC 61970-301, 2007.

[92] M. Postina, S. Rohjans, U. Steffens, and M. Uslar, "Views on service oriented architectures in the context of smart grids," in *Proceedings of the International Conference on Smart Grid Communications.* IEEE, 2010, pp. 25–30.

[93] S. Rohjans, M. Uslar, and H. Appelrath, "OPC UA and CIM: Semantics for the Smart Grid," in *Proceedings of the Transmission and Distribution Conference and Exposition.* IEEE, 2010, pp. 1–8.

[94] A. Claassen, S. Rohjans, and S. Lehnhoff, "Application of the OPC UA for the Smart Grid," in *Proceedings of the International Conference and Exhibition on Innovative Smart Grid Technologies.* IEEE, 2011, pp. 1–8.

[95] C. Ardagna, E. Damiani, S. De Capitani di Vimercati, and P. Samarati, "A Web service architecture for enforcing access control policies," *Electronic Notes in Theoretical Computer Science*, vol. 142, pp. 47–62, 2006.

[96] M. Menzel, C. Wolter, and C. Meinel, "Access control for cross-organisational web service composition," *Journal of Information Assurance and Security*, vol. 2, no. 3, pp. 155–160, 2007.

[97] S. Lakshminarayanan, "Authentication and authorization for Smart Grid application interfaces," in *Proceedings of the Power Systems Conference and Exposition*. IEEE, 2011, pp. 1–5.

[98] T. Baumeister, "Adapting PKI for the Smart Grid," in *Proceedings of the International Conference on Smart Grid Communications*, 2011, pp. 249 –254.

[99] S. Meyer and A. Rakotonirainy, "A survey of research on context-aware homes," in *Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers*. Australian Computer Society, Inc., 2003, pp. 159–168.

[100] F. Mäyrä, A. Soronen, I. Koskinen, K. Kuusela, J. Mikkonen, J. Vanhala, and M. Zakrzewski, "Probing a proactive home: Challenges in researching and designing everyday smart environments," *Human Technology: An Interdisciplinary Journal on Humans in ICT Environments*, vol. 2, no. 2, pp. 158–186, 2006.

[101] T. Flick and J. Morehouse, *Securing the Smart Grid: next generation power grid security*. Elsevier, 2010.

[102] Xively. Business solutions for the Internet of Things.

[103] Google Inc., "Google Protocol Buffers."

[104] P. Mell and T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, pp. 1–3, 2009.

[105] C. Steel, R. Nagappan, and R. Lai, *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Pearson Education, 2006.

[106] T. Moses, "eXtensible Access Control Markup Language (XACML) Version 2.0," 2005.

[107] D. Guinard, I. Ion, and S. Mayer, "In search of an Internet of Things service architecture: REST or WS-*? A developer's perspective," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 2012, pp. 326–337.

[108] P. J. Denning and J. P. Buzen, "The operational analysis of queuing network models," *ACM Computing Surveys*, vol. 10, no. 3, 1978.

[109] "Intelligente Messgeräte-AnforderungsVO (IMA-VO 2011)," Bundesgesetzblatt für die Republik Österreich, 2011.

[110] "Directive 2009/72/EC of the European parliament and of the council of 13th July 2009 concerning common rules for the internal market in electricity and repealing directive 2003/54/EC," Official Journal of the European Union, 2009.

[111] N. Chohan, C. Bunch, S. Pang, C. Krintz, N. Mostafa, S. Soman, and R. Wolski, "AppScale: Scalable and Open AppEngine Application Development and Deployment," in *Cloud Computing*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, vol. 34, ch. 4.

[112] T. Hofer, "Secure and privacy aware data exchange in a service-oriented architecture," Master's thesis, Vienna University of Technology, 2012.

[113] S. Procter, "A brief introduction to XACML," 2003.

[114] G. Ryba, M. Jung, and W. Kastner, "Authorization as a Service in Smart Grids: Evaluating the PaaS Paradigm for XACML Policy Decision Points," in *Proceedings of the International Conference on Emerging Technologies & Factory Automation*. IEEE, 2013.

[115] M. Buettner, G. V. Yee, E. Anderson, and R. Han, "X-MAC: a short preamble MAC protocol for duty-cycled wireless sensor networks," in *Proceedings of the International Conference on Embedded Networked Sensor Systems*. ACM, 2006, pp. 307–320.

[116] G. Mulligan, "The 6LoWPAN architecture," in *Proceedings of the Workshop on Embedded Networked Sensors*. ACM, 2007, pp. 78–82.

[117] Internet Engineering Task Force (IETF), "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks – RFC 6550," Mar. 2012.

[118] ——, "Design and Application Spaces for IPv6 over Low-Power Wireless Personal Area Networks – Informational RFC 6568," Apr. 2012.

[119] "Multicast Protocol for Low Power and Lossy Networks (MPL) (Internet-Draft, version. 04)," IETF draft, 2013.

[120] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, "Building automation and smart cities: An integration approach based on a service-oriented architecture," in *Proceedings of the International Conference on Advanced Information Networking and Applications Workshops*. IEEE, 2013, pp. 1361–1367.

[121] "Bindings for OBIX: REST Bindings Version 1.0," OASIS Committee Specification Draft, 2014.

[122] "Encodings for OBIX: Common Encodings Version 1.0," OASIS Committee Specification Draft, 2014.

[123] "OBIX Version 1.1," OASIS Committee Specification Draft 02, 2013.

[124] Internet Engineering Task Force (IETF), "DNS-Based Service Discovery – RFC 6763," 2013.

[125] ——, "Multicast DNS – RFC 6762," 2013.

[126] ——, "Internet assigned numbers authority (IANA) procedures for the management of the service name and transport protocol port number registry – RFC 6335," 2011.

[127] A. J. Jara, P. Martinez-Julia, and A. Skarmeta, "Lightweight multicast DNS and DNS-SD (lmDNS-SD): IPv6-based resource and service discovery for the Web of Things," in *Proceedings of the Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IEEE, 2012, pp. 731–738.

[128] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proceedings of the International Conference on Embedded Networked Sensor Systems*. ACM, 2003, pp. 126–137.

[129] J. Polley, D. Blazakis, J. McGee, D. Rusk, and J. S. Baras, "ATEMU: a fine-grained sensor network simulator," in *Proceedings of the Communications Society Conference on Sensor and Ad Hoc Communications and Networks*. IEEE, 2004, pp. 145–152.

[130] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Proceedings of the International Symposium on Information processing in Sensor Networks*. IEEE, 2005, pp. 1–6.

[131] P. Raich, "Scalability Analysis of a Web-based IoT Stack for Automation Systems," Master's thesis, Vienna University of Technology, 2015.

[132] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, "Heterogeneous device interaction using an IPv6 enabled service-oriented architecture for building automation systems," in *Proceedings of the Symposium on Applied Computing*. ACM, 2013, pp. 1939–1941.

[133] W. Granzer, W. Kastner, and C. Reinisch, "Gateway-free integration of BACnet and KNX using multi-protocol devices," in *Proceedings of the International Conference on Industrial Informatics*. IEEE, 2008, pp. 973–978.

[134] "Information technology – Home electronic system (HES) architecture," ISO/IEC 14543, 2006-2007.

[135] International Electrotechnical Commission, "Communication systems for meters and remote reading of meters," EN 13757, 2002.

[136] W. Kastner, F. Praus, G. Neugschwandtner, and W. Granzer, "KNX," in *The Industrial Electronics Handbook-Industrial Communications Systems*, 2nd ed., Bogdan M. Wilamowski, J. David Irwin, Ed. CRC Press, 2012, ch. 42.

[137] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner, "Web services in building automation: Mapping KNX to oBIX," in *Proceedings of the International Conference on Industrial Informatics*. IEEE, 2007, pp. 87–92.

[138] "Communication systems for and remote reading of meters – Part 3: Dedicated application layer," DIN EN 13757-3:2005-02, 2011.

[139] G. Nam, S. H. Kim, D. Kim, M. Jung, and W. Kastner, "Extending the EPCIS with Building Automation Systems: a New Information System For the Internet of Things," in *Proceedings of the International Workshop on Extending Seamlessly to the Internet of Things*, 2014, pp. 1–6.

[140] Internet Engineering Task Force (IETF), "Constrained RESTful Environments (CoRE) Link Format – RFC 6690," Aug. 2012.

[141] M. Jung, J. Weidinger, and W. Kastner, "A seamless integration of KNX into Constrained RESTful Environments," in *Proceedings of the KNX Scientific*, 2012, pp. 1–13.

[142] "Java Cryptography Architecture Oracle Providers Documentation for JDK 8," https://docs.oracle.com/javase/8/docs/technotes/guides/security/SunProviders.html, accessed: 2014-11-26.

[143] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, 2007.

[144] C. Alistair, *Writing effective use cases*. Addison-Wesley, 2001.

[145] M. Jung, J. Weidinger, D. Bunyai, C. Reinisch, W. Kastner, and A. Olivieri, "Demonstration of an IPv6 multi-protocol gateway for seamless integration of Building Automation Systems into Constrained RESTful Environments," in *Proceedings of the International Conference on Internet of Things*. IEEE, 2012, pp. 242–243.

[146] M. Jung, J. Chelakal, J. Schober, W. Kastner, L. Zhou, and K. N. Giang, "IoTSyS: an integration middleware for the Internet of Things," in *Proceedings of the International Conference on Internet of Things*. IEEE, 2014, pp. 1–2.

[147] M. Ruta, F. Scioscia, and E. Di Sciascio, "Enabling the Semantic Web of Things: Framework and Architecture," in *Proceedings of the International Conference on Semantic Computing*, 2012, pp. 345–347.

[148] T. Clausen and U. Herberg, "Comparative study of RPL-enabled optimized broadcast in wireless sensor networks," in *Proceedings of the International Conference on Intelligent Sensors, Sensor Networks and Information Processing*. IEEE, 2010, pp. 7–12.

[149] J. Stankovic, "Research Directions for the Internet of Things," *Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, 2014.

Markus Jung, MSc.
Research Assistant

Vienna University of Technology
Institute of Computer Aided Automation
mjung@auto.tuwien.ac.at
Tel. +43 1 58801-18322
Fax +43 1 58801-18391

Curriculum Vitae

Personal information

Address	Gattring 11, 3143 Pyhra
E-Mail	markus.jung85@gmail.com
Nationality	Austria
Compulsory Military Service	Served (2004 to 2005)
Date of birth	22.02.1985

Personal skills and competences

Mother tongue	**German**
Foreign languages	**English (Fluent), Korean (Basic)**
Social Actions	Since 07/2009 part of the TU Buddy network, NGO - supporting exchange students
Other Interests	Mountain biking, Running, Piano
Driving License	Yes

Awards, Certificates, Activities and International Experience

International Experience

- Visiting researcher at Real-Time and Embedded Systems Lab at Korean Advanced Institute of Science and Technology (KAIST), 07/2013 to 09/2013
- Visiting researcher at Institute of Information Systems, University of Applied Sciences Western Switzerland, 07/2012 to 09/2012

Awards

- IPSO Challenge 2013 – semi-finalist, IPSO Challenge 2014 – semi-finalist with IoTSyS Internet of Things integration middleware
- IEEE conference IECON 2013 – best paper in session award for paper "Efficient group communication based on Web services for reliable control in wireless automation"
- Winner Team (Austria) of the Accenture Campus Challenge 2007 (Development of a business case and prototype for wireless sensor networks)

Standardization and Open Source Activities

- OASIS Open Building Information Exchange – Editor for protocol bindings and message encodings specifications: SOAP binding, REST binding, Encodings
- Lead for open source project IoTSyS - Internet of Things integration middleware for home and building automation technologies (www.iotsys.org, blog)
- Google Summer of Code 2014 – Mentor for IoTSyS projects (A lightweight yet scalable persistent layer for IoTSyS gateway, IoTSyS security)
- Google Summer of Code 2013 – Mentor for IoTSyS projects (oBIX 1.1 Specification WD Implementation, a discovery service in OBIX standardization)
- Google Summer of Code 2011 – Student Implementation of a cross-site request forgery protection module for Apache Tapestry 5

Certificates

- Java Certified Programmer
- Java Certified Developer
- Stock trader diploma for the spot market of the Vienna Stock Exchange

Training

- Volksbanken Academy– Basic banking knowledge (Basic@I)
  GNC – Oracle SQL and PL/SQL Goodies – speed up your Application

Work experience

Vienna University of Technology

Date	Since May 2011
Position	Research Assistant
Institute	Computer Aided Automation
Research projects	IoT6, Smart Web Grid, Networked miniSPOT
Lectures	Home and Building Automation, Wireless in Automation

TeleTrader Software AG

Date	August 2010 - April 2011
Position	Software Architect & Product Manager
Main activities and responsibilities	Product management for Web services and database infrastructure
Type of Business	Software and data services provider in the financial sector

Santander Consumer Bank Austria (formerly GE Money Bank GmbH)

Date	August 2008 - June 2010
Position	Application developer
Main activities and responsibilities	Implementation of an online banking platform (https://service.santanderconsumer.at/eva/) Project Management, Software Architecture and Design, Security Audits
Technologies	Java EE (EJB 3.0, Hibernate, Web Services, …), Struts, Grails, Oracle, Ajax (Prototype, Yahoo UI)
Type of Business	Bank – IT

Volksbank AG

Date	September 2007 – December 2008
Position	Project Manager, Digital Signature Coordinator

Main activities and responsibilities	Project management, Responsibility for the online representation of the Volksbank AG (www.volksbank.at, www.volksbank.com)
Type of Business	Bank – Organisation Department/Project Management Group
	Xion IT Systems AG
Date	Mai 2005 – August 2007
Position	Software Developer
Main activities and responsibilities	Software projects, Java development in the area of online sports betting (www.wettpunkt.com), Testing and quality assurance (Federal Ministry of the Interior - central population register)
Technologies	Java EE (EJB 2.1, RMI), Flash, Linux
Type of Business	IT service provider

Education

	Vienna University of Technology – PhD candidate
Date	since 05/2011
Field of study and research	Computer Science, Smart Grids & Internet of Things
	Vienna University of Technology – Computer Science, Master - passed with distinction
GPA	4.0 of 4.0
Date	09/2008 - 09/2010
Field of study	Software Engineering & Internet Computing
	Vienna University of Technology – Computer Science, Bachelor - passed with distinction
Date	09/2005 – 07/2008
Field of study	Software & Information Engineering
	Federal Secondary College of Engineering St. Pölten, Department Electronic Data Processing and Economics
Date	September 1999 – July 2004
Principal subjects	Programming, Database development, Network administration, Project management, Accounting

Publications

Conference papers

Markus Jung, Jomy Chelakal, Jürgen Schober, Wolfgang Kastner, Luyu Zhou, and Giang Ky Nam. **IoTSyS: an integration middleware for the Internet of Things**. In Proceedings of the 4th International Conference on the Internet of Things (IoT 2014), Cambridge, MA, USA, October 2014

Markus Jung, Philipp Raich, Wolfgang Kastner. **The relevance and impact of IPv6 multicasting for Wireless Sensor and Actuator Networks based on 6LoWPAN and Constrained RESTful Environments**. In Proceedings of the 4th International Conference on the Internet of Things (IoT 2014), Cambridge, MA, USA, October 2014

Giang Nam, Seong Hoon Kim, Daeyoung Kim, Markus Jung, Wolfgang Kastner. **Extending the EPCIS with Building Automation Systems: a New Information System For the Internet of Things**. In International Workshop on Extending Seamlessly to the Internet of Things (esIoT), Birmingham, UK, July 2014

Markus Jung, Esad Hajdarevic, Wolfgang Kastner, Antonio J. Jara. Short Paper: **A Scripting-Free Control Logic Editor for the Internet of Things**. In Proceedings of the IEEE World Forum on Internet of Things, Seoul, Republic of Korea, March 2014

Markus Jung and Wolfgang Kastner. **Efficient group communication based on Web services for reliable control in wireless automation**. In Proceedings of the 39th Annual Conference of the IEEE Industrial Electronics Society, Vienna, Austria, November 2013 (Best paper in Session Award)

Gregor Ryba, Markus Jung, and Wolfgang Kastner. **Authorization as a Service in Smart Grids: Evaluating the PaaS Paradigm for XACML Policy Decision Points**. In Proceedings of the 18th IEEE International Conference on Emerging Technologies & Factory Automation, Cagliari, Italy, September 2013

Markus Jung, Jürgen Weidinger, Wolfgang Kastner, and Alex Olivieri. **Building automation and smart cities: An integration approach based on a service-oriented architecture**. In Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications, Barcelona, Spain, March 2013.

Markus Jung, Jürgen Weidinger, Wolfgang Kastner, and Alex Olivieri. **Heterogeneous device interaction using an IPv6 enabled service-oriented architecture for building automation systems**. In Proceedings of the 28th ACM Symposium on Applied Computing, Coimbra, Portugal, 2013

Markus Jung, Jürgen Weidinger, Christian Reinisch, Wolfgang Kastner, Cedric Crettaz, Alex Olivieri, and Yann Bocchi. **A transparent IPv6 multi-protocol gateway to integrate Building Automation Systems in the Internet of Things**. In Proceedings of the IEEE International Conference on Internet of Things (iThings 2012), Besancon, France, November 2012

Markus Jung, Thomas Hofer, Susen Döbelt, Georg Kienesberger, Florian Judex, and Wolfgang Kastner. **Access control for a Smart Grid SOA**. In Proceedings of the 7th IEEE Conference for Internet Technology and Secured Transactions, London, UK, December 2012

Markus Jung, Christian Reinisch, and Wolfgang Kastner. **Integrating Building Automation Systems and IPv6 in the Internet of Things**. In Proceedings of the 2012 IEEE International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS' 12), Palermo, Italy, July 2012

Markus Jung, Jürgen Weidinger, Dominik Bunyai, Christian Reinisch, Wolfgang Kastner, and Alex Olivieri. **Demonstration of an IPv6 multi-protocol gateway for seamless integration of Building Automation Systems into Constrained RESTful Environments**. In Proceedings of the IEEE International Conference on Internet of Things (IoT 2012), Wuxi, China, October 2012.

Markus Jung, Wolfgang Kastner, Georg Kienesberger, and Manuel Leithner. **A comparison of Web service technologies for smart meter data exchange**. In Proceedings of the 2012 IEEE PES Innovative Smart Grid Technologies (ISGT) Europe Conference, Berlin, Germany, October 2012

Georg Kienesberger, Thomas Hofer, Markus Jung, and Susen Döbelt. Smart Web Grid - **Eine serviceorientierte Informationsdrehscheibe für Smart Grids**. In Proceedings of the ComForEn 2012, Wels, Austria, September 2012

Georg Kienesberger, Markus Jung, Susen Döbelt, Florian Judex, and Thomas Hofer. **Smart Web Grid: Eine Informationsdrehscheibe für Smart Grids**. In Proceedings of the Smart Grids Week, Bregenz, Austria, May 2012

Markus Jung. **Multi-Stakeholder-Datenaustausch in zukünftigen Smart Grids: Smart Web Grid**. In Proceedings of the ComForEn 2011, FH Oberösterreich, September 2011

Georg Kienesberger, Markus Jung, Friederich Kupzog, and Wolfgang Kastner. **Smart Web Grid: Eine integrierte Informationsdrehscheibe für das Smart Grid**. In Smart Grids Week, Linz, Austria, May 2011.

Industry conference papers	Markus Jung, Jürgen Weidinger, and Wolfgang Kastner. ***A seamless integration of KNX into Constrained RESTful Environments***. In Proceedings of the KNX Scientific, Las Palmas, Spain, November 2012.
	Markus Jung ,Christian Mauser, Wolfgang Kanster, ***BACdroid - A versatile platform for building automation, Droidcon***, Germany, March 2012
Articles	Antonio J. Jara, Alex C. Olivieri, Yann Bocchi, Markus Jung, Wolfgang Kastner, Antonio F. Skarmeta, ***Semantic Web of Things: An analysis of the application semantics for the IoT. Moving towards the IoT convergence*** in International Journal of Web and Grid Services, 2014
	Markus Jung and Thomas Hofer and Wolfgang Kastner and Susen Döbelt. ***Protecting data assets in a Smart Grid SOA. Journal of Internet Technology and Secured Transactions (JITST)***, 2:155 - 166, 2013.
Books	Wolfgang Kastner, Markus Jung, and Lukas Krammer. **Future Trends in Smart Homes and Buildings**. In Richard Zurawski, editor, Industrial Communication Technology Handbook, Second Edition, chapter 59. CRC Press, Inc., 2014
	Markus Jung, Wolfgang Kastner, and Susen Döbelt. **The Smart Web Grid project**. In Smart City: Viennese expertise based on science and research, pages 174-180. Erich Schmidt Verlag, 2013
	Sebastien Ziegler, Cedric Crettaz, Latif Ladid, Srdjan Krco, Boris Pokric, Antonio F. Skarmeta, Antonio Jara, Wolfgang Kastner, and Markus Jung. IoT6 - ***Moving to an IPv6-Based Future IoT***, volume 7858 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013.