Alessandro Bassi Rob van Kranenburg

Martin Bauer Sebastian Lange

Martin Fiedler Stefan Meissner

Thorsten Kramp *Editors*

Enabling Things to Talk

Designing IoT solutions with the
IoT Architectural Reference Model

IoT-A
Internet of Things - Architecture

Springer Open

Enabling Things to Talk

Alessandro Bassi • Martin Bauer •
Martin Fiedler • Thorsten Kramp •
Rob van Kranenburg • Sebastian Lange •
Stefan Meissner
Editors

Enabling Things to Talk

Designing IoT solutions with the
IoT Architectural Reference Model

Springer

Editors

Alessandro Bassi
Alessandro Bassi Consulting
Nice, France

Martin Bauer
NEC Europe Ltd.
Heidelberg, Germany

Martin Fiedler
Fraunhofer Institute for Material Flow
    and Logistics IML
Dortmund, Germany

Thorsten Kramp
IBM Research
Rüschlikon, Switzerland

Rob van Kranenburg
Internet of People Ltd.
London, United Kingdom

Sebastian Lange
Deloitte & Touche GmbH
Berlin, Germany

Stefan Meissner
University of Surrey
Centre for Communication Systems Research
Guildford, United Kingdom

Foreword

Dear Reader,

The book that you are holding now in your hands is the result of a European success story and we want to share with you our feeling of being proud of it. Since 2007, the concept "Internet of Things" has clearly been driven by research and policy initiatives in Europe.

The Internet of Things (IoT) is an emerging network superstructure that connects physical resources and people together with software. It will enable an ecosystem of smart applications and services that will improve and simplify the life of the citizen and will contribute to sustainable growth, provided it combines and guarantees trust and security for people and businesses. At the same time, the IoT will bring hyper-connectivity to our society, using augmented and rich interfaces and characterised by higher semi-autonomous system behaviour than today.

Following a workshop in February 2008, co-organised with the European Technology Platform on Smart Systems (EPoSS),[1] a corresponding research call was developed where in particular Internet of Things architectural questions were pivotal. IoT-A has been the project in the past years in giving an answer by elaborating the Architectural Reference Model (ARM).

Whereas in the first run the IoT referred to the advent of barcodes and Radio Frequency Identification (RFID), helping to automate inventory, tracking and basic identification, the second, current wave of IoT is characterised by a strong verve for connecting sensors, objects, devices, data and applications. The next wave could be called a "cognitive IoT", facilitating object and data re-use across application domains, leveraging on hyper-connectivity, interoperability solutions and semantic enriched information distribution. We consider it being very important for Europe to be able to leverage each wave and to turn the research results into relevant innovation and products.

The Architectural Reference Model provided aims to connect vertically closed systems, architectures and application areas for creating open systems and integrated

---

[1] www.smart-systems-integration.org

environments and platforms. It constitutes the platform from which Europe can capitalise on the benefits of developing consumer-oriented platforms that closely involve the telecom, hardware, software and service industries. Innovative Internet ecosystems going beyond the smart phone must be created, and new multiple application sectors including potential new players and service providers need to collaborate in order to take advantage together of the technological progress.

By just accomplishing the goals of the Architectural Reference Model, a success would be achieved that would far overshadow everything previously created for the individual application areas. And it really can no longer be doubted that this would be achievable in the near future with a determined improvement of available engineering capabilities and with motivated pan-European, multidisciplinary teams ready to put thorough and serious scientific and technological effort to tackle the practical treatment of the IoT challenges, although we must not underestimate the extent of the difficulties that still have to be overcome. This will be the goal of the Internet of Things Objective in Horizon 2020 to build upon success stories like the Architecture Reference Model in order to rise up to today's and tomorrow's societal challenges.

European Commission Vice-President Neelie Kroes is committed to embody and promote a strong leadership presence in IoT technologies and applications in Europe, given the great opportunities they offer to both EU businesses and citizens in areas of general interest like the prediction, monitoring and alerting of natural hazards, the automation of processes in healthcare, utilisation of home metering solutions to assist in independent living, and support of the disabled persons. The Commission will continue to support research and innovation in this domain in the context of "Horizon 2020", the forthcoming EU research and innovation framework programme starting in 2014.[2]

If you start entering complex subjects you need both a framework and an explanation on how to advance and gain rapidly benefits. This "cookbook" provides you with all what you need for starting your IoT endeavour or refocus your current IoT activities. You will find the IoT Architectural Reference Model and compelling use cases – it is now in your hand to use this book and to expand the knowledge of the worldwide IoT community.

We enthusiastically invite you to read this book and opt-in to the Internet of Things! With your engagement, motivation and interactions, the future of the Internet of Things in Europe will be bright and successful.

<div align="right">

Gérald Santucci
Head of Unit "Knowledge Sharing",
Directorate General CONNECT, European Commission
Peter Friess
Scientific Officer Internet of Things,
Directorate General CONNECT, European Commission

</div>

---

[2] http://ec.europa.eu/research/horizon2020

Acknowledgements

Contents

Chapter 1
Introduction to the Internet of Things

Thorsten Kramp, Rob van Kranenburg, and Sebastian Lange

The expression "Internet of Things" (IoT), coined back in 1999 by Kevin Ashton, the British technology pioneer who cofounded the Auto-ID Center at the Massachusetts Institute of Technology, is becoming more and more mainstream. In opening the IoT Week 2013[1] with a pre-recorded video message,[2] Ashton insisted on the realization that IoT is here *now*; it is not the *future* but the *present*. While Gartner identifies IoT as one of the top ten strategic technology trends,[3] Cisco forecasts 50 billion devices connected by 2020,[4] a potential market in excess of $14 trillion,[5] and also claims that IoT is actually already here.[6] Similarly, it is not only companies with a technological focus, such as Ericsson, Bosch or Siemens that use IoT to advertise their cutting edge technologies – media companies such as the BBC are conducting research activities and have plans for IoT deployment. In short, we are currently on the verge of witnessing the emergence of a "mega-market", where markets such as home and building automation, electricity generation and

---

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

[2] http://kevinjashton.com/2013/06/17/pre-recorded-opening-talk-for-internet-of-things-week-helsinki-june-17-2013/

[3] http://www.gartner.com/newsroom/id/2209615

[4] http://share.cisco.com/internet-of-things.html

[5] http://iotevent.eu/cisco-sees-14-trillion-opportunity-in-iot/

[6] http://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1158640

T. Kramp (✉)
IBM Research, Säumerstraße 4, 8803 Rüschlikon, Switzerland
e-mail: thk@zurich.ibm.com; www.zurich.ibm.com

R. van Kranenburg
Internet of People Ltd., Suite 3, 32-38 Scrutton Street, London EC2A 4RQ, UK
e-mail: kranenbu@xs4all.nl; www.theinternetofpeople.eu

S. Lange
Deloitte & Touche GmbH, Kurfürstendamm 23, 10719 Berlin, Germany
e-mail: sebastian.lange@web.de

distribution, logistics, automotive, as well as telecommunications and information technology will steadily converge. As yet, we do not know the consequences of connecting all of these smart objects (smart meter, e-vehicle, cargo container, fridge etc.) to the Internet.[7]

At the same time, the Internet of Things (IoT) is not something you will experience as such itself. What you will see is that more and more objects become connected. If you are selling products, you will be negotiating with providers of connectivity. If you are building, selling or inventing models or tools for providing services or applications, you will notice that the convergence of IoT, big data and energy efficiency, combined with cheap hardware, software, data storage and analytics, favours open standards, innovation and interoperability. Daily activities that were distinct become interwoven in new formats and business models.

Thus, in effect, the Internet of Things is a combination of a technological *push* and a human *pull* for more and ever-increasing connectivity with anything happening in the immediate and wider environment – a logical extension of the computing power in a single machine to the environment: *the environment as an interface*. This push-pull combination makes it very strong, unstoppable, fast and extremely disruptive.

Mireille Hildebrandt, a Dutch professor working on the implications of emerging technologies and the rule of law, states that "we may need to develop an *Ambient Law* that is embodied in the algorithms and human machine interfaces that support Ambient Intelligence and for this we will have to break through our paralysis, ready to become literate in terms of a new script."[8] In a speech to the Pittsburgh Technology Council in 2009, Eric Schmidt, an American software engineer and executive chairman of Google, focused on the negative effects of (what he called) institutional fragmentation on innovation and integration. He wondered whether governments – and the very process of policy and policymaking itself – could benefit from the iterative cycles of measuring success and failure that characterize the engineering and design prototyping cycles. With this amount of real-time tracking and aggregated data and information rather than heuristics, the act of governing itself could benefit. Specific laws could take effect for 3 months and be evaluated and adjusted and then, based on real data rather than estimates, be adjusted again. It is this process that can lead to combinatorial and system innovation.

Two dominant characteristics unite these different perspectives: firstly, a sense that Internet connectivity is becoming increasingly ubiquitous and pervasive; and secondly, the idea that eventually everything – including mundane physical artefacts – will be connected.

---

[7] On the LinkedIn Group "Internet of Things" strueker@iig.uni-freiburg.de.

[8] Hildebrandt, Mireille and Koops, Bert-Jan, The Challenges of Ambient Law and Legal Protection in the Profiling Era, Modern Law Review, Vol. 73, Issue 3, pp. 428-460, May 2010.

IoT Application Example 1: Transport/Logistics
In transport logistics, IoT improves not only material flow systems but also the global positioning and automatic identification of freight. It also increases energy efficiency and thus decreases energy consumption.

In conclusion, IoT is expected to bring profound changes to the global supply chain via intelligent cargo movement. This will be achieved by means of continuous synchronisation of supply chain information and seamless real-time tracking and tracing of objects. It will make the supply chain transparent, visible and controllable, enabling intelligent communication between people and cargo/goods.

IoT Application Example 2: The Smart Home
Future smart homes will be conscious about what happens inside a building, mainly impacting three aspects: resource usage (water conservation and energy consumption), security and comfort. The goal is to achieve better levels of comfort while cutting overall expenditure.

Moreover, smart homes also address security issues by means of complex security systems for detecting theft, fire or unauthorized entry. The stakeholders involved in this scenario constitute a very heterogeneous group.

Different actors will cooperate in the user's home, such as Internet companies, device manufacturers, telecommunications operators, media service providers, security companies, electricity utility companies, etc.

IoT Application Example 3: Smart Cities
While the term smart city is still a fuzzy concept, there is general agreement that it is an urban area which creates sustainable development and high quality of life. Giffinger et al.'s model elucidates the characteristics of a smart city, encompassing economy, people, governance, mobility, environment and living.[9] Outperforming in these key areas can be achieved through strong human or social capital and/or ICT infrastructure. For the latter, an initial business analysis concludes that several sectors/industries will benefit from more digitalised and intelligent cities (examples for a city of one million people):[10]

(continued)

---

[9] http://www.smart-cities.eu/download/smart_cities_final_report.pdf
[10] http://de.slideshare.net/rlnicholson2/smart-cities-proving-ground-for-the-intelligent-economy

(continued)

(a) Smart metering, 600,000 m, $120 million opportunity
(b) Infrastructure for charging electric vehicles, 45,000 electric vehicles, $225 million opportunity
(c) Remote patient monitoring (diabetes), 70,000 people, $14 million opportunity
(d) Smart retail, 4,000 stores, $200 million opportunity
(e) Smart bank branches, 3,200 PTMs, $160 million opportunity

IoT Application Example 4: Smart Factory

In a global supply chain, companies will be able to track all of their products by means of radio frequency identification (RFID) tags. As a consequence, companies will reduce their operating expenses (OPEX) and improve their productivity due to tighter integration with enterprise resource planning (ERP) and other systems. Also, maintenance of machinery will be facilitated by connected sensors, allowing for real-time monitoring of the health and performance of the factory equipment.

Generally, IoT will provide automatic procedures that imply a drastic reduction in the number of employees needed. Workers will be replaced by bar code scanners, readers, sensors and actuators, and in the end by complex robots as efficient as a human being.

Without any doubt, these technologies will bring opportunities for white-collar workers and a large number of technicians will be required to program and repair these machines. This is synonymous to a transfer to maintenance jobs, but it also constitutes a new challenge for providing all blue-collar workers with an opportunity to move toward these types of jobs and to avoid unemployment.

As the developments got closer to the market and the everyday lives of citizens, the need for non-technical research in the area of machine to machine (M2M) communication and the Internet of Things was acknowledged in the 1996 EU Call for Proposals of i[3]: Intelligent Information Interfaces, an Esprit Long-Term Research initiative. The aim of i[3] (pronounced "eye-cubed") was to develop new, human-centred interfaces for interacting with information, aimed at the future broad population. This approach was also the starting point and rationale for the EU-funded proactive initiative "*The Disappearing Computer*", a cluster of 17 projects conducted by interdisciplinary research groups. Its mission was "to see how information technology can be diffused into everyday objects and settings, and to see how this can lead to new ways of supporting and enhancing people's lives

that go above and beyond what is possible with the computer today."[11] The third research iteration of this approach was Convivio (2003–2005), a thematic network of researchers and practitioners developing a broad discipline of human-centred design of digital systems for everyday life. The coordinator of Convivio stated that human-centred design "still has little influence either on governmental and supernational policies or on industrial strategies. As a result, it also has little impact on the quality of ICT in public and private life."[12]

However, in 2013, some 50 % of respondents[13] to a European Commission Public Consultation fell into the "interested citizen" category rather than belonging to a particular industrial, academic or other sector.[14]

Andreas Kirsch in the IoT Expert Group commented that the main point that emerged from the work of the subgroup on privacy was that everyone will be affected by IoT but many people will not realise it. It is vital that this realization is handled well. By default, the Internet of Things may involve function creep or have unintended consequences: "It was noted that most people use the same concepts when discussing IoT as when discussing the Internet in general. There is a significant difference, however. IoT involves objects talking to each other without user consent, with possibly un-envisaged functionalities. Cameras, for example, might take on functions that are different from their overt primary functions. These possibilities, once perceived, may cause user anxieties to rise. Moreover, what is the role of user consent if objects may be able to talk to each other spontaneously? It will be very difficult to backtrack after the deployment of million of chips employing a passive approach to connectivity."

Privacy, security and ideas in society about data storage and tracking could stall adoption when, for example, by combining the analysis of supply and demand, energy enterprises are able to supply more efficient demand shaping. They will not just give incentives to consumers; they will actually turn off devices that are not needed (e.g. turn off the freezer for 20 min). Furthermore, these actions must take place automatically. In IoT we always face a heterogeneous scenario involving diverse stakeholders. The main actors are of course energy utility companies, but public entities will also be important players. These services need to be coupled with educational programs that explain what is happening in reality.

---

[11] The Disappearing Computer II (DC) Proactive Initiative http://cordis.europa.eu/ist/fet/dc2-in.htm

[12] Letter to the Convivio community, Giorgio De Michelis, Convivio network coordinator, http://daisy.cti.gr/webzine/Issues/Issue%201/Letters/index.html

[13] Additional responses have been received since the last report, with the total number rising from 500 to more than 600. These additional responses did not affect the statistics for the exercise as a whole.

[14] Tenth Meeting of the Internet of Things Expert Group, Brussels, 14 November 2012. Tom Wachtel, rapporteur.

IoT Application Example 5: Retail

IoT realises both customer needs and business needs: price comparison of a product; looking for other products of the same quality at lower prices; with shop promotions, giving information not only to customers but also to shops and businesses. Having this information in real time helps enterprises to improve their business and to satisfy customer needs.

Obviously, big retail chains will take advantage of their dominant position to enforce the future IoT retail market, as was the case with RFID adoption, which was enforced by Walmart in 2004 (Wu et al. 2006). In particular, companies with controlling positions, such as Carrefour, Metro, Migros, Walmart, etc. will be able to push the adoption of IoT technology due to their sizable market shares.

IoT Application Example 6: E-Health

Control and prevention are two of the main goals of future health care. Already today, people have the option of being tracked and monitored by specialists even if the patient and specialist are not in the same place. Tracing peoples' health history is another aspect that makes IoT-assisted e-health very versatile. Business applications could offer the possibility of medical services not only to patients but also to specialists, who need information to proceed in their medical evaluation. In this domain, IoT makes human interaction much more efficient because it permits not only localization, but also tracking and monitoring of patients. Providing information about the state of a patient makes the whole process more efficient, and also makes people much more satisfied.

The most important stakeholders in this scenario will be public and private hospitals and institutes such as the Institute of Applied eHealth at Edinburgh Napier University, which participated in the first stakeholder session of IoT-A. It is worth mentioning that telecommunications operators are quite active in e-health (for instance, O2 UK).

The IoT Expert Group claims that, "As IoT will introduce new difficulties for contextual integrity, the principle whereby information supplied for use in one context (e.g. a meeting with one's doctor) is not expected by the owner of the data to be used in a different context (e.g. the doctor applying for a mortgage). There will be a social contract between people and objects, and the ethical ramifications of a contract of this kind must be considered".[15]

---

[15] Internet of Things Expert Group (E02514), Commission Decision of 10 August 2010 setting up the Expert Group on the Internet of Things. OJ C 217, 11.8.2010, p. 10–11, http://ec.europa.eu/transparency/regexpert/index.cfm?do=groupDetail.groupDetail&groupID=2514

All current computing and IoT paradigms position connectivity and content-centric networking centrally as an ecology of devices, protocols, services and networks, such as RFID, active sensors, biometrically-related smart camera data, 2D and 3D bar codes and 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks) or ZigBee. At the core of this ecology there is a seamless flow between:

- The **BAN** (body area network): e.g. the ambient hearing aide, the smart T-shirts
- The **LAN** (local area network): e.g. the smart meter as a home interface
- The **WAN** (wide area network): the bike, car, train, bus
- The **VWAN** (very wide area network): the smart city as e-government services everywhere; no longer tied to physical locations

Traceability, sustainability and security linking the gateways of these different area networks cannot be ensured without interoperability at architectural, domain-specific and application level. (see also the box on page 9 – The hierarchy of networks)

It is also highly likely that monitoring mechanisms will be built into devices themselves: for example, "if a guest is charging their electric car at a friend's house, we should consider applications that will understand that the charge should appear on the guest's electric bill and not that of the friend."[16] But there is a clear deadlock: clients do not know what they can expect, nor do they know what they could ask.

M2M vendors cannot interface their sensor capabilities beyond an optimizing function. No one is *asking* for an Internet of Things. People have no idea about what they can expect and why they should hand over their washing machines to a local grid to ensure energy efficiency, for example. Is a positive outcome feasible?

A successful IoT means the best possible feedback on our physical and mental health, the best possible deals based on real-time monitoring for resource allocation, the best possible decision-making based on real-time data and information from open sources, and the best possible alignments of our local providers with the global potential of wider communities.

Now that we have introduced the basic ideas of the Internet of Things concept and pointed out some aspects of the current discussion taking place in the Internet of Things community, you can see that the whole field is very much "in motion". New ideas, concepts and new technologies are appearing constantly, whereas others are disappearing, being ruled out as incompatible or not feasible. In the IoT concept, which is itself disruptive, other potential "disruptive" technologies (e.g. Google Glass etc.) strongly influence the direction of technological development as well as the related societal and political discussion.

Despite the high-level discourse that is necessary to assess the socio-economic impact of IoT in general, in this book we will focus on the underlying technological concepts, network architecture approaches and connectivity and interoperability requirements that are required to provide and realise the fundamental connectivity that will ultimately allow for the emergence of the Internet of Things to the benefit of mankind in general.

---

[16] http://tools.ietf.org/id/draft-roychowdhury-6lowappsip-00.txt

With a strong focus on network architectures, architecture models and guidelines for building a truly interoperable Internet of Things, this book summarises the results of the IoT-A[17] project, funded by the European Union and conducted between 2010 and 2013. More than 50 scientists and researchers contributed to the development of an "Architectural Reference Model" (ARM) for the Internet of Things.

This book is in two parts (I and II). Part I (Chaps. 2, 3, 4) introduces, on a more general level, the concepts developed over the course of the IoT-A project. It is targeted at a general audience including end users who want to employ IoT technologies, managers interested in understanding the opportunities generated by the new technologies, and system architects who are interested in an overview of the models developed. In Chap. 2 we explain the history behind and origin of the IoT-A project. In Chap. 3 we introduce the ARM as enabler, its terminology and methods for employing it. Chapter 4 then highlights use cases that exemplify how the ARM has been used in real life scenarios.

Part II (Chaps. 5, 6, 7, 8, 9, 10, 11, 12) contains Chap. 5, which provides an overview on guidance to the ARM, followed by Chap. 6 with very detailed and elaborate description of a process to generate concrete architectures. In Chap. 7 the IoT Reference Model is aiming at establishing a common grounding. Based on this, in Chap. 8 the IoT Reference Architecture is presented. Chapter 9 provides reference manuals with guidelines how to use the various Models and Perspectives presented in creating a concrete architecture. In Chap. 10 an interaction analysis on some selected scenarios is given to provide a general understanding on the interactions to be considered. The best practices and guidelines relating to how system engineers or other end users can use the ARM to develop specific IoT architectures for dedicated IoT solutions and how users can apply the concepts presented to develop a dedicated IoT architecture for a specific application case are illustrated in Chap. 11 and exemplified in reverse mapping exercises of existing standards and platforms to the IoT ARM up to a business case evaluation in Chap. 12.

In contrast to Part I, Part II addresses the topic on a very scientific and technical level and is targeted at the knowledgeable scientific or technical reader.

> **IoT Application Example 7: Smart Energy/Smart Grid**
> This field has many overlaps with other scenarios, such as smart home and smart city. The key issue in these scenarios is to detect ways to save energy. We are basically referring to what is known as a smart grid. In this application area, initiatives that imply a more distributed energy production must be highlighted, as many houses today have a solar panel, for example. As a vital constituent, smart metering is considered a prerequisite for enabling intelligent monitoring, control and communication in grid applications. The use of IoT platforms in smart metering will provide the following benefits:

(continued)

---

[17] www.iot-a.eu

(continued)

a. An efficient network of smart meters allows for faster outage detection and restoration of service. Such capabilities abound to the benefit of customers.
b. Customers will have greater control over their energy or water consumption, providing them with more choices for managing their bills.
c. IoT deployment of smart meters is expected to reduce the need for building power plants. Building power plants that are necessary only for occasional peak demand is very expensive: a more economical approach is to enable customers to reduce their demand through time-based rates or other incentive programs, or to use automatic recording of consumption to temporarily turn off devices which are not in use.

Finally, by combining the analysis of supply and demand, energy enterprises will able to supply more efficient demand shaping. They will not just give incentives to consumers, but will actually turn off devices that are not needed (e.g. turn off the freezer for 20 min). Furthermore, these actions must take place automatically. Here, we again face a heterogeneous scenario involving diverse stakeholders. The main actors are of course energy utility companies, but public entities will also be important players.

The Hierarchy of Networks: BAN (Body Area Network): The Ambient Hearing Aide, the Smart T-shirts
Control and prevention are two of the main goals of future health care. Already today, people have the option of being tracked and monitored by specialists even if patient and specialist are not in the same place. Tracing peoples' health history is another aspect that makes IoT-assisted e-health very versatile. Business applications could offer the possibility of medical services not only to patients but also to specialists, who need information to proceed in their medical evaluation. In this domain, IoT makes human interaction much more efficient because it permits not only localization, but also tracking and monitoring of patients. Providing information about the state of a patient makes the whole process more efficient, and also makes people much more satisfied. Trust is a key issue in this relationship. Patient to patient networks become more empowered as well.

LAN (local area network): the smart meter as a home interface
Future smart homes will be conscious about what happens inside a building, mainly impacting three aspects: resource usage (water conservation and energy consumption), security and comfort. The goal is to achieve better levels of comfort while cutting overall expenditure. Moreover, smart homes

(continued)

also address security issues by means of complex security systems to detect theft, fire or unauthorized entry. The stakeholders involved in this scenario constitute a very heterogeneous group. Different actors will cooperate in the user's home, such as Internet companies, device manufacturers, telecommunications operators, media service providers, security companies, electricity utility companies, etc.

WAN (wide area network): the bike, car, train, bus, ...

In transport logistics, IoT improves not only material flow systems but also global positioning and automatic identification of freight. It also increases energy efficiency and thus decreases energy consumption.

Prof. Dr. Michael ten Hompel, Managing Director at Fraunhofer-Institut for "Materialfluss und Logistik", describes the consequences for something as "solid" as logistics: "The logical consequence of the Internet of Things is not just a new philosophy of how we can control our production and logistics. It completely changes the paradigms of conventional supply chain management. Within the Internet of Things the supply chain will be created in real time: *Entities*, consisting of objects and a piece of (agent based) software, generates the resulting supply chain 'on the move.' Therefore the sequences of operations are not predicted. This leads to a new understanding of how to handle our logistic management which won't be a supply chain (!) anymore."

IoT is thus expected to bring profound changes to the global supply chain via intelligent cargo movement. This will be achieved by means of continuous synchronisation of supply chain information and seamless real-time tracking and tracing of objects. It will make the supply chain transparent, visible and controllable, enabling intelligent communication between people and cargo.

VWAN (very wide area network): the smart city as e-government services *everywhere;* no longer tied to physical locations

While the term smart city is still a fuzzy concept, there is general agreement that it is an urban area which creates sustainable development and high quality of life. Giffinger et al.'s model elucidates the characteristics of a smart city, encompassing economy, people, governance, mobility, environment and living (Giffinger 2007). Outperforming in these key areas can be achieved through strong human or social capital and/or ICT infrastructure. There are a number of critics who question whether the smart city as it is conceived now can be inclusive and educational.

Part I
General Concepts of the Architecture Reference Model (ARM)

Chapter 2
The Need for a Common Ground for the IoT: The History and Reasoning Behind the IoT-A Project

Alessandro Bassi and Sebastian Lange

The Internet of Things concept has evolved rapidly in recent years. It can be seen as an umbrella term for interconnected technologies, devices, objects and services. Nevertheless, after many years of heavy discussion, there is still no clear and common definition of the concept. And yet the application scenarios and market opportunities offered by objects communicating actively and autonomously extend far beyond the foreseeable horizon.

Looking at websites such as kickstarters.com and indiegogo.com, new applications and services envisaged by innovators and researchers are astonishing and clearly show the vast opportunities our future society will be confronted with.

The concept of IoT as introduced in Chap. 1 emerged primarily from the convergence of different technological developments and fields. In particular, it builds on the emergence of innovative enabling functionalities that stem from identification technologies such as RFID and bar codes, as well as from the development of networked sensors and actuators. In the early 2000s, RFID technology was developed and rolled out mainly across the logistics sector for tracking and tracing goods. At the same time, research was conducted on sensor networks and miniaturized smart systems. Sensors were becoming increasingly small and computing power dramatically increased. Nevertheless, innovative solutions were always developed for specific application cases, and there was no true interoperability and interconnectivity between different application areas.

For instance, in some fields such as manufacturing and logistics, communication and tagging solutions are well-established as they provide a clear business benefit in terms of asset tracking and supply chain management. However, the same solutions

---

A. Bassi (✉)
Alessandro Bassi Consulting, 3, Avenue de Cannes, 09160 Juan Les Pins, France
e-mail: alessandro@bassiconsulting.eu

S. Lange
Deloitte & Touche GmbH, Kurfürstendamm 23, 10719 Berlin, Germany
e-mail: sebastian.lange@web.de

do not apply for other fields such as domotics, where business synergies could provide services with clear added-value benefits.

As the IoT domain covers such a wide spectrum of application fields with very little in common, the development cycles and technologies used can be completely different. Often, early technological developments are driven by visionary small and medium-sized enterprises (SME) that are able to innovate faster and to catch emerging trends. However, the target is usually a product or service with a narrow scope, and as the focus and window of opportunity are slim, the solutions developed are usually non-interoperable, and while successful, they do not produce a common abstract infrastructure capable of marking significant progress in the whole field. The same holds true for large industry companies that often develop specialized solutions for dedicated business opportunities without implementing generally applicable concepts.

Therefore, current solutions can still be seen as island solutions, implementing some sort of "**INTRAnet of Things**" rather than an "**INTERnet of Things**".

While quite logical at this point, in the long term, this situation is unsustainable. Today, we can observe a similar situation to that in the networking field, where several solutions emerged at its infancy but were subsequently abandoned in favour of a unified communication infrastructure, the **TCP/IP protocol suite**.

The emergence of a common "lingua franca" for the IoT domain, representing the narrow central point in the Internet protocol suite, is a prerequisite for quick and pervasive development of innovative solutions that can leverage different technologies developed for different targets in different application domains.

After much discussion about the core concepts of the IoT for several years, in 2009 a group of researchers from more than 20 large industrial companies and research institutions joined forces to lay the foundation for the much needed common ground or a common "architecture" for the Internet of Things: the **IoT-Architecture project** (**IoT-A**) was born. IoT-A has become the European Commission's flagship project in the European Union's Seventh Framework Program for Research and Development with respect to establishing an architecture for the Internet of Things.

Leaving aside business considerations, and considering only the technical point of view, it was clear for the project partners that the existing solutions did not address the scalability requirements of a future IoT, both in terms of communication between smart devices and the orchestration and management of complex services. Furthermore, the IoT domain comprises several different governance models, which are often incompatible. This leads to a situation where privacy and security are treated on a per case and per legislation basis, retrofitting solutions to existing designs – this severely hampers portability, interoperability and deployment.

Of course, the spread of the IoT domain is so huge that it would be naive to consider a "one-size-fits-all" protocol, such as IP, or even a single layer where interoperability between all sorts of smart device communication can take place. However, it soon became clear that within this area, there was a need for a common ground in a more abstract layer.

We are convinced that different classes of devices will always co-exist. Taxonomies can be created according to different principles, such as critical or non-critical, or distributed or centralised. These classes can foster different profiles according to the specific needs and requirements of applications and domains.

As it is impossible to specify one single design pattern that can satisfy all application domains, the common ground has to be found at a more abstract level. We believe that the identification of a **reference model for the entire IoT domain** will provide the common ground. By **reference model** we mean an abstract framework that comprises a minimal set of unifying concepts, axioms and relationships for understanding significant relationships between the entities of an environment. This framework should enable the development of specific architectures which may have different levels of abstraction. At this level of abstraction we are independent of specific standards, technologies, implementations, or other concrete details.

This high-level work then drives the realisation of a framework for identifying specific **reference architectures** that subsequently describe both essential building blocks as well as design choices for dealing with conflicting requirements regarding functionality, performance, deployment and security. Interfaces need to be standardised, and best practices need to be provided in terms of functionality and information usage.

The central decision of the IoT-A project was to base its work on the current state of the art, rather than applying a clean slate approach. As a result, common traits have been derived to form the baseline of the **IoT Architectural Reference Model** (**ARM**). This has the major advantage of ensuring that the model is backward-compatible, as well as the adoption of established, working solutions for various aspects of the IoT.

It is no longer possible to build architectures in the lab or without real world input. IoT-A acknowledged this new reality, where the lines between R&D, innovation and emergent technologies are blurred, at a very early stage. With the help of end users, organised into the IoT-A stakeholders group, new requirements for IoT have been collected and introduced in the main model-building process. This stakeholder group was one of the most important sources for obtaining external input as well as feedback on the current status of project work. Thus far, the stakeholder contributions have been a main feature of the project, as the stakeholder requirements collected in an initial workshop formed the basis for the initial draft of the ARM, particularly the domain model and the functional decomposition. Each building block of the ARM was then developed to meet all requirements and enable the IoT-A holistic approach. Further stakeholder workshops and questionnaires were employed to review the progress of the ARM development and to fine tune the concepts and models.

Currently, the prevailing practice domain for stakeholder engagement is largely characterized by complex and dynamic environments that cover a wide range of stakeholders, from hostile to conciliatory, from obstructive to collaborative.

This is an apt characterisation of the Internet of Things: complex and dynamic environments containing a wide range of stakeholders. As such, it is an open and

ongoing ecology of environments, characterized by change and real-time combina-
torial innovation.

For all their different backgrounds – automotive, health, logistics, retail. . . – the
stakeholders were surprised to see that their requirements were often very similar.
In their real world cases, the same principles and same abstract level required that
"in this IoT world things become active participants"; the goal is a seamless chain
of real-time tracking and tracing, in which the elite of expensive high-level item
tracking and the multitude of low-level items should be balanced for cost efficiency.
Interoperability was validated by the stakeholders in the independently generated
use cases as the number one requirement.

The next chapter introduces the ARM in detail, its language and terminology, as
well as its beneficial role in the IoT application development process.

Chapter 3
The IoT Architectural Reference Model as Enabler

Martin Bauer and Joachim W. Walewski

As identified in the previous chapter, IoT-A has created an "Architectural Reference Model" (IoT ARM) as the common ground for the Internet of Things. The core idea is that the IoT ARM provides a common structure and guidelines for dealing with core aspects of developing, using and analysing IoT systems. The first part of this chapter provides a non-exclusive list of the beneficial uses of the IoT ARM. In the second part we focus on the role of the IoT ARM in the architecture development process.

3.1 Using the IoT ARM

In the following we present a non-exclusive list of the beneficial uses of the IoT ARM. The order in which they are discussed does not imply any ranking – we list them according to their degree of abstraction and remoteness from the product: i.e. the first usage type is concerned more with generic enabling (abstract and remote), while the last usage type concerns how the IoT ARM can be used for procuring system solutions (concrete, close to business). The usage type that is more important to any specific use of the IoT ARM depends on the perspective of the actors involved. A manager of an IoT development process, for instance, is more likely to favour the enabling aspects of the IoT ARM, while a procurement department is more likely to favour concrete advantages that are closer to the business process itself.

M. Bauer (✉)
NEC Laboratories Europe, Software & Services Research Division, NEC Europe Ltd.,
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
e-mail: Martin.Bauer@neclab.eu; www.nw.neclab.eu

J.W. Walewski
Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
e-mail: joachim.walewski@siemens.com; www.siemens.com

3.1.1 Cognitive Aid

When it comes to product development and other activities, an architectural reference model is of fourfold use.

Firstly, it helps to guide discussions, since it provides a language everyone involved can use, and which is intimately linked to the architecture, the system, the usage domain, etc.

Secondly, the high-level view provided in such a model is of high educational value, since it provides an abstract but also rich view of the domain. Such a view can help people new to the field to "find their way" and to understand the special features and intricacies of IoT.

Thirdly, the IoT ARM can assist IoT project leaders in planning the work at hand and the teams needed. For instance, the Functionality Groups identified in the IoT Functional View of the IoT system can also be understood as a list of independent teams working on an IoT system implementation. The Process Chapter (Chap. 6) provides more insight on how the IoT ARM can support the architecture generation process and also about how to separate it into different activity "islands". This type of approach is particularly interesting for enterprise architecture frameworks that incorporate system-architecting processes. Typically, these enterprise frameworks provide institutional rules and prescriptions for how the system-architecting process is to be conducted. The IoT ARM can inform such institutional rules and prescriptions. An example of the latter is the Zachman framework (Zachman 1987).

Fourthly, the IoT ARM helps to identify independent building blocks for IoT systems. This constitutes very valuable information when dealing with questions such as system modularity, processor architectures, third-vendor options, re-use of components already developed, etc.

3.1.2 Reference Model as a Common Ground

Establishing a common ground for a field is not an easy task. In order to be effective, it has to capture as many pertinent vantage points as possible. Establishing the common ground for the IoT encompasses defining IoT entities and describing their basic interactions and relationships with each other. The IoT ARM provides exactly such a common ground for the IoT field.

3.1.3 Generating Architectures

One of the main benefits is the use of the IoT ARM for generating compliant architectures for specific systems. This is done by providing best practices and

guidance for translating the IoT ARM into concrete architectures. For an overview on this, see Chap. 5. The benefit of this type of generation scheme for IoT architectures is not only a certain degree of automation in this process, and thus lower R&D efforts, but also that the decisions made follow a clear, documented pattern as described in Chap. 6.

3.1.4 Identifying Differences in Derived Architectures

When using the aforementioned IoT ARM-based architecture process, any differences in the derived architectures can be attributed to the special features of the use case in question and the design choices related to this case (Shames and Yamada 2004). When applying the IoT ARM, a list of system function blocks, data models, etc., together with predictions of system complexity, etc. can be derived for the architecture generated. Furthermore, the IoT ARM defines a set of tactics and design choices for meeting qualitative system requirements (for more details, see Chap. 6, Design choices). All of these facts can be used to predict whether two derived architectures will differ and where they will do so.

The IoT ARM can also be used for reverse mapping. System architectures can be cast in the "IoT ARM" language and the resulting "translation" of the system architectures is then stripped of incompatible language and system partitions and mappings. The differences that remain are then true differences in architecture.

3.1.5 Achieving Interoperability

As we explain later on in this book (see Chap. 6 on design choices), fulfilling qualitative requirements through the architecting process inevitably leads to design challenges. Since there is usually more than one solution to each of the design challenges (we refer to these solutions as design choices), the IoT ARM cannot guarantee interoperability between any two concrete architectures, even if they have been derived from the same requirement set. Nevertheless, it is an important tool in helping to achieve interoperability between IoT systems. This is facilitated by the design-choice process itself. During this process, one identifies and tallies the design choices made. By comparing the design choices made when deriving two architectures, one can readily identify where in the architecture measures are necessary to achieve interoperability. Interoperability may be achieved a posteriori by integrating one IoT system as subsystem in another system, or by building a bridge through which key functionalities of the respective other IoT system can be used. Notice though that these workarounds often fall short of achieving full interoperability. Nevertheless, building bridges between such systems is typically much more straightforward than completely re-designing either system and usually fair interoperability can be achieved.

3.1.6   System Roadmaps and Product Life Cycles

Above we discussed how the design choices made in order to derive a particular architecture, and also the features selected, are instrumental in describing the difference between two architectures. As well as identifying the differences between two "foreign" architectures, this approach can also be used to map the evolution of architectures. For instance, design choices are tied to qualitative requirements. Let us assume that during the requirements process (see Chap. 6, Sect. 6.4), two disjoint "design choice" islands are identified, i.e. groups of design choices that lead to non-interdependent functionalities, data models, etc. In this case, it is possible to embody only one "design choice" island in the systems produced and to embody the full set of design choices in the next product generation. Thus, the IoT ARM can be used to devise system roadmaps that lead to minimum changes between two product generations while still guaranteeing a noticeable enhancement in system capability and features. This approach also helps the designer to formulate clear and standardised, requirements-based rationales for the system roadmap chosen and the product life cycles that result from the system roadmap.

3.1.7   Benchmarking

Another important use of the IoT ARM is benchmarking. For example, NASA used a reference architecture that described its envisaged exploration vehicle in order to receive better benchmarking tenders during a public bidding process for the said exploration vehicle (Tamblyn et al. 2007). While the reference model prescribed the language to be used in the systems/architectures to be assessed, the reference architecture stated the minimum (functional) requirements for the systems/architectures. By standardising the description and also the ordering and delineation of system components and aspects, this approach also provided the benchmarking process with a high level of transparency and inherent comparability. Using this approach, besides just "ticking" off the minimum features each tender has to fulfil, even more insight can be gained into the proposed system. For instance, the number and "richness" of functional components belonging to the system and their interaction patterns allow an appreciation of the system complexity both in terms of composition and structure but also in terms of interaction. This information can be gleaned from the IoT Functional View (functional decomposition, interactions), the IoT Information View (data flow, data complexity) and the IoT Deployment View. It makes judging the overall system complexity easier during the tender review phase.

3.2   Architecture Development Process Based on the IoT ARM

Following the overview of the different cases of usage in which the IoT ARM plays a beneficial role, we will now focus on how the IoT ARM can be used during the process of generating concrete IoT architectures suitable for specific applications and use cases. We will first discuss the idea behind reference models and reference architectures and the underlying methodology.

The process of developing an architecture is about finding a solution to a pre-defined goal. In turn, the development and description of architectures is a modelling exercise. It is important to point out that the modelling itself does not take place in a vacuum but is based on a thorough understanding of the domain to be modelled. In other words, any architecture development is contingent on the understanding of the domain in question. The same is true for a generalisation of this process, i.e. the derivation of reference architectures. Thus, reference architectures, such as the one presented in this book, also have to be based on a detailed understanding of the domain in question. This understanding is commonly provided in the form of a reference model.

3.2.1   Reference Model and Reference Architecture

Reference models and reference architectures provide a description that is more abstract than what is inherent to actual systems and applications. They are more abstract than concrete architectures that have been designed for a particular application with particular constraints and choices. From literature, we can extrapolate the dependencies between a reference architecture, architectures and actual systems (see Fig. 3.1) (Muller 2008). Architectures do help in designing, engineering, building and testing actual systems. At the same time, a better understanding of system constraints can provide input for the architecture design, and this allows future opportunities to be identified. The structure of the architecture can be made explicit through an architecture description, or it is implicit through the system itself. Extracting essential components of existing architectures, such as mechanisms or the use of standards, allows the definition of a reference architecture.

Guidelines can be linked to a reference architecture in order to derive concrete architectures from the reference architecture (Fig. 3.2, left). These general architecture dependencies apply to the modelling of the IoT domain as well.

The transformation step from an application-independent model to a platform-independent model is informed by guidelines. The step from platform-independent model to platform-specific model is discussed later in this chapter.

While the model presented in Fig. 3.1 stops at the reference architecture, the IoT-A Architectural Reference Model goes one step beyond this and also defines a
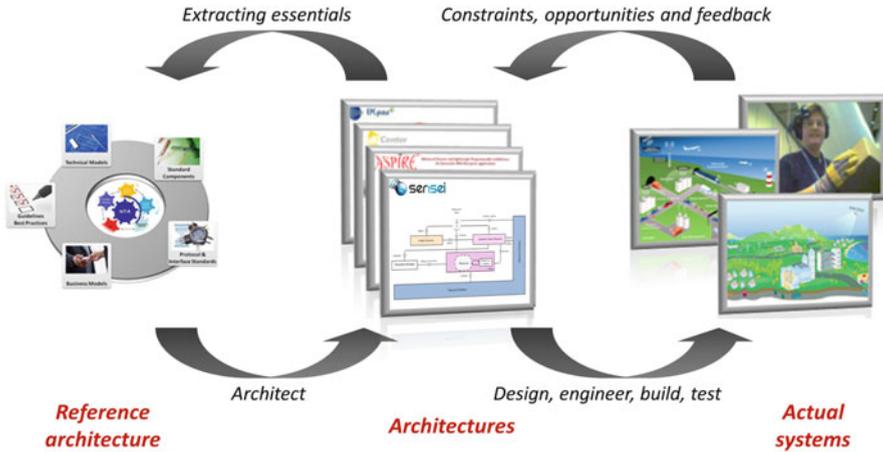
Fig. 3.1 Relationship between a reference architecture, architectures and actual systems (Adapted from Muller (2008))
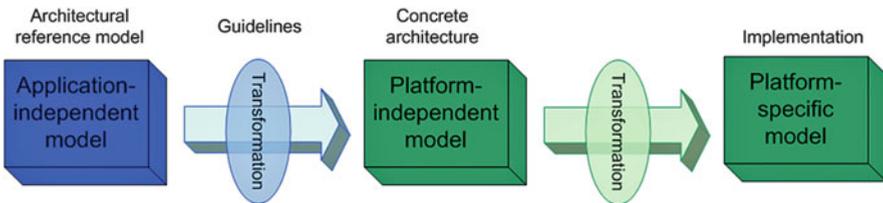
Fig. 3.2 Derivation of implementations (platform-specific models) from an architectural reference model (application-independent model) via the intermediate step of a concrete architecture (platform-independent model)

reference model. As already discussed, a reference model provides the ground for a common understanding of the IoT domain by modelling its concepts and their relationships. A detailed description of the IoT Reference Model can be found in Chap. 7.

3.2.2 Generating Architectures

Now that we have a general understanding about reference models, reference architectures and their relationships, the important question is how to derive the appropriate concrete architecture from the reference architecture. We dedicate an entire chapter to this issue, namely the Process Chap. 6, where we describe all aspects in great detail. However, the reader needs at least some appreciation of the

role of the IoT ARM here in order to take full advantage of the IoT Reference Model and the IoT Reference Architecture in the chapters in-between.

When applying the IoT ARM in designing systems, it is likely that in each individual case, different architectures will result. Thus, while Fig. 3.2 gives the impression that the process of translating the reference architecture into a concrete architecture is independent of the use case itself, this is, in reality, not so – the guidelines and the engineering practices chosen rely on a use case description and the requirements. This fact is reflected in Fig. 3.3. The role of the IoT ARM is to provide transformation rules for translating the rather abstract models into a concrete architecture. This step is strongly influenced by the use case and the related requirements. One entry point for this information is during the process of design choices, i.e. when the architect favours one avenue for realising the functionality or quality needed over another. The IoT ARM also recommends design patterns for such choices. The IoT ARM does not operate in a design vacuum but should be applied together with proven design process practices, which in themselves are contingent upon the guidelines provided and upon the use case and the requirements.

In Chaps. 7 and 8 we describe how both the IoT Reference Model and the IoT Reference Architecture can be used in this design process. Even though we describe the design process in a linear fashion, remember that in practice this will not always be the case. Depending on the engineering strategies used, some of the steps can be done in parallel or may even have to be reiterated due to additional understanding gained during the process or due to changes in the requirements.

3.2.3 Choice of Design and Development Methodology

The choice of a design and development methodology can be understood in two ways: firstly, a methodology for the IoT ARM development and secondly, a methodology for the generation of specific concrete architectures. We have so far only provided high-level views of either case. In reality, more guidance is required, i.e. a recipe for how to derive all aspects of the IoT ARM model as well as how to derive guidelines for the application of the IoT ARM for the generation of architectures.

In the case of the IoT ARM there are, to our knowledge, no standardised approaches for developing such a model. Furthermore, compared to typical reference architecture domains, the IoT usage domain is extremely wide and varied, and common denominators are thus rather few and abstract. For examples of reference architectures and models, the reader is directed to the following literature: (Consultative Committee 2006; MacKenzie et al. 2006; Muller 2008; Open GeoSpatial Consortium 2002; Shames and Yamada 2004; Tamblyn et al. 2007; Usländer 2007). This high level of abstraction in terms of the domain to be modelled stands in contrast to the input needed for established and standardised methodologies such as Aspect-Oriented Programming (AOP), Model-Driven Engineering (MDE), Pattern-Based
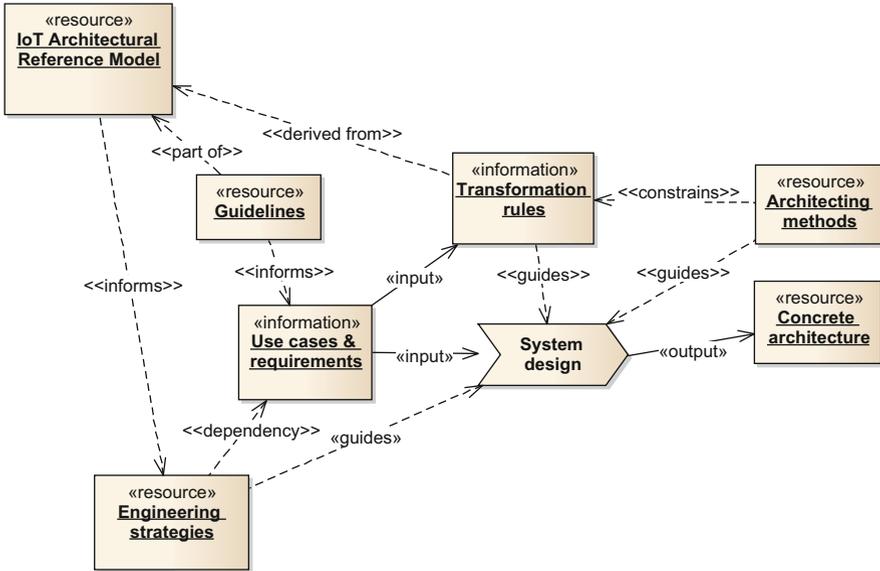
Fig. 3.3 Process for generating concrete architectures

Design and SysML. All of these methodologies were designed for very concrete use cases and application scenarios. Unfortunately, this high degree of specificity defines even their inner workings. In other words, if they are applied to generalised use cases, the result is not generalised models on the abstract level of an IoT ARM – in fact, the result is nothing. We illustrate this using the example of MDE.

MDE for the generation of Model-Driven Architectures (MDA) is standardised by the *Object Management Group* (OMG) (Miller and Mukerji 2003). The main application area of this methodology is the development of software systems. MDE prescribes four steps for a development process:

1. Specify a system independently from the platform;
2. Specify platforms;
3. Choose a particular platform for the system;
4. Transform the system specification into that of the particular platform.

The goals behind this approach are portability, interoperability and reusability through the architectural separation of concerns (Vicente-Chicote et al. 2007). Thus, on the face of it, this all sounds very similar to the goals of our IoT ARM development process.

Figure 3.4 summarises the main idea of MDA. A platform-independent model, i.e. an architecture, is to be transformed into a platform-specific model, i.e. an implementation. An example for the former is a GUI user interface described in

Fig. 3.4 Generalised architecture approach according to the Model-Driven Architecture methodology, otherwise known as Model-Driven Engineering (Miller and Mukerji 2003)
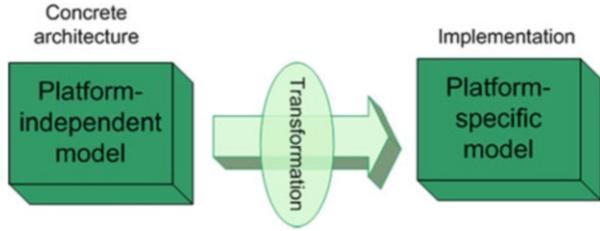
Table 3.1 Use of standardised architecture methodologies for the development of the IoT ARM

Methodology	Aspect adopted in our work
Aspect-oriented programming	Delineation of functionalities by aspects. This is embodied in the concept of functionality groups (see IoT Functional view in Chap. 8)
Model-driven engineering	General concept of transformation from a generic to a more specific model. We use this concept for describing and developing our guidelines
Views and perspectives	We adopt the concept of views and perspectives to derive the IoT Reference Architecture, i.e. we arrange all aspects of our reference architecture according to views and perspectives (see IoT Reference architecture in Chap. 8)

UML, and the latter is an implementation of this interface in a mobile phone model featuring a particular operating system.

This sounds very much like the transformation introduced in Fig. 3.3, but it actually takes place at a lower abstraction level, as becomes apparent from Fig. 3.2. The IoT ARM and the MDE approach are thus linked to each other through platform-independent models (architectures). While the general idea of a model transformation, as promoted by MDE, resonates with our IoT ARM approach, the methodology developed for deriving transformations between platform-independent and platform-specific models can, alas, not be transferred and adapted for deriving best practice transformations.

Table 3.1 summarises how we use ideas borrowed from standardised architecture methodologies for our work on the higher abstract level of our IoT ARM.

Chapter 4
IoT in Practice: Examples: IoT in Logistics and Health

Martin Fiedler and Stefan Meissner

The previous chapters gave a first impression of the ARM as common ground for the Internet of Things. In the following sections we will introduce use cases and sample scenarios (scenes) that have been used as a practical evaluation of the ARM in specific applications. Using the ARM in a top-down process, starting from an application description, most of the scenes introduced were realized as demonstrators within a specific work-package of IoT-A. The second, bottom-up approach of reverse mapping an existing application to the ARM is shown with a scene brought in by the stakeholder group of IoT-A.

The use cases described focus on the domains of retail/logistics and healthcare. This is due to the importance and relevance of these domains, but as IoT-A aims to provide an ARM for the Internet of Things, the ARM should also be applicable to other major domains such as manufacturing or entertainment. The denominator that the two domains considered have in common is that they affect many people – both now and in the future. There is a connection between nutrition and health, with many people opting for healthy food to prevent diseases; others act in accordance with a health plan prepared by their doctor after a diagnosis. Technology can be used to support both cases and makes it easier to eat and stay healthy.

The first part of this chapter covers the retail and logistics use case. It focuses more on enterprise-related processes. Here we also introduce a "red thread" example, which is used within the technical part of the book (see Chaps. 5, 6, 7, 8, 9, and 12). The second part introduces the health use case and an existing application which will be used to reverse map to the ARM.

M. Fiedler (✉)
Fraunhofer Institute for Material Flow and Logistics IML, Joseph-von-Fraunhofer Str. 2-4, 44227 Dortmund, Germany
e-mail: Martin.Fiedler@iml.fraunhofer.de; www.iml.fraunhofer.de

S. Meissner
University of Surrey, Stag Hill, Guildford GU2 7XH, UK
e-mail: s.meissner@surrey.ac.uk; www.surrey.ac.uk

In general, we defined day-in-a-life scenes with specific characters living through them in the storyline.

4.1 Storyline of the IoT-A Use Case "IoT in Retail and Logistics"

Nearly every single one of us has to do some shopping and in doing so, we gather experience with the retail industry; therefore it is part of our daily life. As technological innovations permeate other parts of our lives, retail is also being increasingly penetrated by different technologies that support and help us in many different situations: for example, smart mobile phones equipped with shopping applications that manage our shopping lists or our dietary information. This presents retailers with the new challenge of integrating their business into the consumer's world and vice versa.

On one hand, information which is generated by customers (e.g. product ratings) might be of high interest to retailers, especially product and category managers. On the other hand, information which is owned by a retailer is not always just of interest for retailers themselves: for example, looking at the traceability of individual goods, real-time queries on a mobile device for the customer or the availability of products in a certain store.

To realize traceability of individual goods during the whole product life cycle and to create transparency all along the supply chain, the first step is to serialise each individual item. This takes place at some point during the manufacturing process. Adding additional sensors to the items to collect various environmental information as well as counting the carbon emitted by the products itself helps to increase transparency (Fig. 4.1).

Privacy concerns that arise from tracking the items of customers outside the store using tags can be addressed by solutions that help both the supply chain and the customers.

Other examples of serialized objects include an NFC tag on a laptop to track the ownership and add built-in accelerometers to it to record physical transport damage. Many kinds of different data are recorded on the way from the point of origin to the point of destination and are transmitted during the transport or at handover and could be made visible by the system at any time.

Using smart mobile phones in combination with RFID- or NFC-tagged products provides advantages not only for manufacturers, retailers and customers, but also for delivery and anyone involved in logistical processes for these products.

With this scenario in mind, the future Internet of Things applied to the retail domain could unfold as follows from a user's perspective:

> Ted, the delivery man for a gardener, uses his IoT phone to manage transport orders, scan tagged items or load carriers and receive status messages from sensors added to the items he is currently transporting (in our case, sensitive orchids). This way, he knows the

©Acik / fotolia.com   ©Mikael_Damkier / fotolia.com

Fig. 4.1 The retail and logistics domain

circumstances of the products without having to do a visual inspection (i.e. stop his truck, which would mean a delay).

When he arrives at the local supermarket, Ted lets the load carriers he delivered pass through an RFID gate which recognizes them automatically. After briefly talking to John, the store manager, Ted sends the sensor record history saved on his IoT phone to John's IoT phone via NFC. The manager can now see that on the way to the store, there was a critical rise in temperature at one point, which causes him to visually inspect the orchids and decide whether he still wants to accept the delivery. Since John identifies the orchids as fine, he sends a message of approval to Ted's IoT phone.

To look at the customer's perspective of our scenario, we switch to Salomée, a young woman representing a customer:

This Saturday, Salomée decides to try out the new supermarket (where John is the store manager) that opened recently. She is a young lawyer and a single mother. Salomée recently took up her first position at a big law firm, and therefore has to put in long hours, leaving her son in day care. Balancing work and family time is difficult. Therefore, she usually does all of her shopping on Saturday morning, even though she hates the long queues that usually form then. As a single parent she is very price-conscious, but she still wants her child to get healthy nutrition and she also cares about the environment, preferring local products with a small carbon footprint.

As she enters the supermarket, she is positively surprised by its spaciousness and its calm atmosphere. Salomée has a shopping application installed on her IoT phone – it allows her to receive information about products when she scans them or when the store's backend system recognizes certain behaviour or circumstances. The software also keeps track of Salomée's shopping behaviour in order to provide more personalized and thus more efficient suggestions.

Today, Salomée is looking for cheese, so she enters the refrigerated section. Once she finds a packet of cheese that catches her interest, she reads its NFC tag to get more information about it from her virtual Shopping Assistant and to compare it to other kinds of cheese she has bought before. Thanks to the application, she quickly finds the cheese she wants to buy.

Now that Salomée has found a cheese she likes, she wants to buy her favourite wine. The Shopping Assistant on her IoT phone can now tell her about the prices of wines she has bought before and if one of these wines is out of stock, shows a recommendation for a similar wine.

Today, Salomée has to acknowledge that there are no affordable wines available that she likes so she starts to leave. This and the fact that she has got cheese in her shopping cart

causes a big TFT display to show an announcement for a 30 % discount on wine for anyone buying cheese. Salomée is happy to return and buy a bottle of wine she had considered too expensive earlier.

In parallel, John and his crew have to struggle with the always busy Saturdays. They have to replenish the empty shelves and need to know what the customers need next. Cameras on the ceiling and other ways of understanding the customers support them, helping them to be more efficient and provide the best services to the customers.

Some automatic processes simplify the staff's tasks. The orchids (delivered by Ted) have sensors attached to them that monitor environmental features critical to the quality of the flowers. The sensors send this information to the price tags to enable automatic price adjustment according to product quality. Since the air conditioning in the store is currently not set up correctly, the orchids' price is lowered by 10 % due to a rise in ambient temperature. Continuing her shopping, Salomée passes the orchids, sees how beautiful they are, and to her surprise, realises there is a discount on them as displayed on the electronic shelf labels. She immediately takes one as a present for her neighbour who loves flowers. As the supermarket is crowded today, she uses her IoT phone to participate in a virtual queuing system at the checkout, meaning that she can browse the shelves while already being in the queue for checkout.

After Salomée has finished grocery shopping and is about to return home, she receives a notification on her IoT phone telling her that Robert, her father, has used his last ampoule of insulin. The notification recommends that she stops at a pharmacy to buy new medication. Salomée is glad to see that her IoT phone can show her the location of the closest pharmacy in the area.

She enters the pharmacy and picks up a package of insulin ampoules. The clerk scans the medication and is asked by the local pharmacy software to ask for Salomée's health ID to verify that she is allowed to buy this kind of drug. Salomée hands over her health ID and when the clerk scans it, the software confirms that Salomée is permitted to buy the medication.

In summary, this storyline gives us an impression of how IoT-A components can help consumers and retailers to handle or manage daily challenges. It shows that IoT affects the whole supply chain: starting with the production site, through transport and retail, up to the customer, IoT can facilitate the whole process and improve the service.

4.2 Introducing the ARM with a Recurring Example (Logistics)

The ARM itself – and therefore this description as well – has a certain complexity. In order to ease the process of understanding the overall concepts and the different components that make up the ARM, we will exemplify concepts of the ARM with a "recurring use case" scene (also known as a "red thread") throughout the book. This allows us to complement the sometimes abstract and top-down discussions of ARM concepts with a real, tangible use case.

We have selected a modification of the specific IoT-A use case scene of "Transport monitoring with smart load carriers" that can be found in (Fiedler et al. 2012), because the issue of transport monitoring in logistics is familiar to

many readers and offers aspects that are relevant for basically all components in the ARM, while at the same time not being too complex itself.

The "Transport monitoring with smart load carriers" use case scene shows how live sensor monitoring of smart load carriers can prevent the goods transported from being damaged due to environmental influences. The load carrier is equipped with sensors and can communicate with other devices via wireless radio technology. With this hardware, every load carrier continuously measures its environmental parameters and sends all measurements via the embedded event service to the mobile phone of the truck driver who has subscribed to this service.

The business value of the scene is clear: as around 20 % of perishable goods never reach the consumer, but are disposed of beforehand, either in the store or in the supply chain, the utilization of IoT sensors is an interesting concept for implementing quality control for perishable goods and thus reducing waste and increasing margin gains at the same time. In transportation, there is a huge potential for innovative logistics models, such as rescheduling at distribution centres based on the estimated quality of the goods in order to reduce waste and finally get the products to the consumers in good shape (see Fig. 4.2).

To make the use case description more concrete and easy to grasp, we present the use case from a user's perspective. This description supplements the specific IoT-A scene with security features:

> Ted is a truck driver transporting highly sensitive orchids (can be substituted with any perishable goods) to a retail store. After loading the orchids on his truck, he attaches an array of sensors to the load carriers in order to measure the temperature. While he is driving, Ted gets hungry and decides to stop and have lunch. He parks the truck at a resting spot, turns off the engine and goes into a nearby restaurant. Unfortunately, Ted forgot that by turning off the engine, the air conditioning for the transported goods (the highly sensitive orchids) shuts off too, and since it is a very hot day, the temperature inside the truck starts to rise. When the temperature reaches a predefined critical level inside one of the load carriers, one of its sensors notices this and sends an emergency signal to Ted's IoT phone. Due to its delicate nature, this signal cannot be received by the phones of other drivers.
>
> On the IoT phone's display, Ted can now see that the orchids in load carrier number 6 are in danger due to a high temperature. He therefore rushes back to the vehicle and turns the air conditioning back on. The IoT phone also keeps track of any alert messages it receives from the load carriers and saves this message history for future inspection in a way that cannot be altered. When the truck reaches the retail store for delivery, the sensor history is transferred to the store's enterprise system and the sensors authenticate themselves as not having been tampered with.

4.3 Use of the ARM in the Scene "Sensor-Based Quality Control" (Retail)

Another short example shows how sensors monitor perishable goods in a store. The sensor infrastructure measurements are used to estimate the quality of a rare and expensive form of Chinese orchid. Depending on the luminance, humidity, and temperature of the environment, the estimated future quality of the orchids is

Fig. 4.2 The recurring example ("*red thread*")

determined and prices are reduced, even before a perceivable degradation of quality occurs. By applying this sensor-based quality control and combining it with dynamic pricing, the store can ensure that the goods are sold before quality degradation is likely to occur.

From a user's perspective this scene is as follows:

> This Saturday, Salomée decides to try out the new supermarket that opened recently. As she enters, she is positively surprised by its spaciousness and its calm atmosphere. Her mobile shopping application points her to a special offer of non-food items, namely rare and fragile orchids from China.
>
> She immediately thinks of her neighbour, Heinrich, who loves flowers and would appreciate them as a gift from her. Just as she approaches the shelf with the orchids, she sees their price going down by 10%. Happy about the price reduction, she immediately picks an orchid and continues shopping.

From a business and industry perspective, the scene demonstrates two important retail-related concepts: dynamic pricing and quality control of perishable goods. Dynamic pricing as a real-time tool for price optimization strategies has always been crucial for profit maximization. In contrast to the state of the art, dynamic pricing in the use case featured is not performed based on static information such as best before end dates in the transaction data of the backend Enterprise-Resource-Planning (ERP) system, but is based on real-time IoT data gathered from a sensor infrastructure.

4.4 Storyline of the IoT-A Use Case "IoT in Health and Home"

Leaving behind the retail and logistics domain, we now introduce the second use case domain, health and home. This domain focuses more on human-related processes in the IoT. The patient's related perspective of the complete day-in-a-life scenario is as follows:

> After having enjoyed a nice dinner with his daughter Salomée the night before, 55 year old Robert wakes up in the morning. Robert is proud of his daughter, with whom he has good contact. It's always nice meeting up with her, as Salomée has a lot to talk about, which reminds him a little of himself when he was younger and more healthy. Robert suffers from high blood pressure and has type II diabetes, and since he has already suffered from one heart attack, he is considered a high-risk patient. He is participating in a program organized by his health insurance company – the program monitors his health continuously and remotely. As his wife died a while ago, Salomée is registered at the health insurance company as a family member who supports Robert with his housekeeping and simple medical care. This relationship is stored in Robert's electronic health record (EHR).
>
> This morning, Robert is still thinking about the things Salomée was talking about the night before, and he leaves his IoT phone behind in his bedroom. The IoT phone is Robert's new IoT-capable smart phone which Salomée has told him a lot about. Normally, Robert carries his IoT phone everywhere he goes. A backend system reminds him to take medical measurements in a daily routine, usually three times a day. Now, Robert cannot hear the reminder alarm. Since the alarm is not acknowledged, the system looks for nearby IoT devices such as lights or buzzers in the vicinity of Robert and uses those devices to attract his attention. Robert sees the lights flashing in his living room and instantly remembers what this means, as it has happened before. He goes to his bedroom and picks up the IoT phone to acknowledge the alarm.
>
> He is guided through the measurements he has to take by the application on his smart tablet. He has to measure his blood pressure, heart beat, blood glucose level, current weight and give an indication of the activity he was performing immediately before taking the measurements. All measurements are stored in the system and are analysed automatically, with a notification sent to his doctor if any values are outside the normal range. The system calculates the amount of insulin he must inject. As Robert has to take more insulin than usual he takes his last NFC-tagged ampoule of insulin out of his medicine cupboard and this action is recognised immediately. The insulin stock level in Robert's medicine cupboard is tracked, and as soon as it reaches a predefined refill level, an alarm is raised. Salomée's IoT phone is notified to advise Salomée to buy insulin on behalf of Robert at a nearby pharmacy as she is registered as a supporting family member. After Robert takes his insulin dose, his electronic health record is updated accordingly.
>
> Later on, Robert suddenly feels lightheaded and he presses a panic button he is wearing as part of a bracelet. The system detects that his mother, Jane, who he lives with in the same flat, is nearby and notifies her of the situation. Jane finds Robert and sees that there is no need for further action since he has already eaten a candy bar he always carries with him.
>
> In the afternoon, Robert leaves his flat. He is driving to visit his daughter when he is involved in a car accident. The other driver must have overlooked him in the bad weather conditions. Luckily, the acceleration sensor of Robert's IoT phone instantly recognizes that something dangerous may have happened and queries his condition from his body sensors. The devices agree that Robert is in danger and, after a short time during which Robert can confirm he is safe, an emergency message is released, sending the location data of where the dangerous condition arose as well as his personal ID to the emergency centre. Using location-based lookup, the nearest emergency centre is alerted and asked to send an

ambulance to Robert immediately. The ambulance arrives at the car crash within seven minutes and picks up Robert and the other driver to take them to a hospital.

Arriving at the hospital, the check-in is quick, even though Robert is unconscious. Fortunately, the check-in procedures can be performed directly through interactions between Robert's identity card or health ID card and the hospital admission desk. The clerk first looks for Robert's health ID card but cannot find it. He finds only the identity card, which he can also use to check Robert in. Using an IoT-enabled mouse, Robert can be identified by the local hospital software via his national identity number. The software can be used to grant the hospital access to data from the national identity database that is required for check-in and also allows the receptionist to look up all necessary medical insurance data as well as his entire medical file – making it easy to prepare all helpful information for the doctor beforehand with no time-consuming effort. And again, it pays off to have signed up for the program as Robert gets precedence over a young man who seems to have broken his arm but still has to spend time filling in all the paperwork. After the quick check-in, Robert is looked after by the medical personnel who treat his wounds.

The hospital Robert is staying at is equipped with the Hospital Information System (HIS). This system continuously monitors the environmental conditions (temperature, humidity) in the rooms and prevents incorrect medicine being administered to the patients. Robert had to stay at the hospital overnight as the doctors had to monitor his reaction to the medical treatment. The next day during the morning routine, the temperature readings for Robert and a fellow patient in the same room are too high, indicating a small fever. However, an analysis shows that since the patients shall only be exposed to constant environment conditions the related room temperature in turn was too low due to a failure of the heating system. A facility manager is automatically called by the HIS to repair the defect.

For further medical treatment and to monitor his condition, a nurse visits Robert twice a day. The nurse administers medication to Robert as he needs this for the pain caused by his wounds. In the evening, the nurse scans Robert's identification tag followed by the tag for the box of medicine (ampoules) and suddenly an alarm is triggered. The medicine the nurse is about to administer is the correct one, but the dose in the ampoule is too high due to an error in the hospital pharmacy. The problem occurred could be resolved by the nurse administering the correct dose manually and documenting the mismatch in her tablet PC.

4.5   Use of the ARM in the Scene "Remote Patient Notification" (Homecare)

In the first scene of the health use case, the Remote Patient Care application notifies the patient of actions they have to perform. These actions can be related to administering medicine or to taking measurements at a regular interval.

Patients carry personal devices such as smart phones or tablets which can become IoT-enabled. Applications running on these devices can hence make use of all functions of the IoT-A compliant platform.

In this scene, the patient is notified by an alarm ringing on his IoT phone. This alarm is not acknowledged and therefore the application looks for nearby resources such as light switches or buzzers in the vicinity of the last known location of the patient and uses these devices to attract the patient's attention. The scene ends when the patient finally acknowledges the alarm.

Fig. 4.3 The MUNICH scene of tracking stomach towels during surgery © Technical University of Munich

4.6   Reverse Mapping of the ARM in the Scene "In-Surgery Tracking of RFID-Tagged Stomach Towels" (Hospital)

This scenario will be used to reverse map an existing real-world implementation to the ARM. Further details on reverse mapping of the reverse mapping of the MUNICH platform can be found in Chap. 12. This scenario was included by Prof. Christoph Thümmler, who is actively contributing in the e-health area. The specific application was implemented with help of the MUNICH platform by Celestor, Napier University Edinburgh, Technical University of Munich and Siemens.

This use case scene is about counting stomach towels used inside the abdomen during surgery on a patient. After the operation, no towels may be left in the abdominal cavity (the human body) and assurance is required that this is the case. Therefore, each towel is fitted with an RFID tag enabling it to be tracked during surgery. Figure 4.3 shows ongoing surgery with the blue stomach towels.

The RFID-tagged towels can be tracked by three antennas from different positions in the operating theatre:

- Mayo stand (instrument table): towel is unused.
- Operating table: towel is in use.
- Used towel container: towel is used.

Each towel is used in a specific order. Initially, a batch of "unused" stomach towels is located on the instrument table. Towels which are put into the patient's body are "in use". Finally, towels which are not needed any more after the surgery are put into the towel container and are attributed status "used".

Assurance is required that no towels are left inside the patient's abdomen when the operation has finished. In more technical terms, this means that after the operation has finished, all the towels that were "in use" must be in status "used", meaning in the waste bin.

From a business perspective, up to 100 stomach towels can be used within one single surgical procedure. Towels left in the patient's abdomen can cause severe and even fatal infections. As there are no official numbers, e.g. no central databases on towels left in a patient's body, the numbers differ: studies indicate 6,000–9,000 incidents per year. A business case evaluation example based on its use case can also be found in Chap. 12.

Part II
A Guidance to the Architecture Reference Model (ARM)

Chapter 5
Guidance to the ARM: Overview

Stefan Meissner and Joachim W. Walewski

A major goal of this Guidance Chapter is to provide guidance for system architects. In other words, we aim at explaining the usage of the IoT ARM. One of the major focus areas of this guidance is the derivation of domain-specific architectures from the ARM. For other potential usages of the IoT ARM see Chap. 3. The structure of the technical part B of this book is depicted in Fig. 5.1.

On about 250 pages, this book provides a technical description of the IoT ARM together with multifaceted guidance to the user of the IoT ARM. In the various chapters, we cover various interests of the user, such as generating architectures by aid of the IoT ARM (Chaps. 6, 11 and 12), and how to use the IoT Reference Models presented in Chap. 7. We also shed light on how other IoT architectures relate to the IoT ARM (Chap. 12 sections about reverse mapping ETSI M2M, EPC Global, Ucode, BUTLER), and we also illustrated, how already existing systems can be mapped onto the IoT ARM (Chap. 12, Sect. 12.6). Notice that by its very nature this part of the book is not an insulated part of the IoT ARM, but it provides many pointers back to the IoT Reference Model (Chap. 7) and the IoT Reference Architecture (Chap. 8 including Carrez et al. 2013). Also notice that while many of the Sections of the Chapter are oriented toward the generation of concrete architectures, they can also be consulted when using the IoT ARM along any of the other avenues listed in Chap. 3. One example is Chap. 6, Sect. 6.9, which covers the design-choice process for translating qualitative requirements into view requirements. This process is not only of importance for the generation of concrete architectures but comes also to pass when identifying the differences between architectures (Chap. 3, Sect. 3.5), outlining avenues toward interoperability (Chap. 3, Sect. 3.6), and generating system roadmaps (Chap. 3, Sect. 3.7).

S. Meissner (✉)
University of Surrey, Stag Hill, Guildford GU2 7XH, UK
e-mail: s.meissner@surrey.ac.uk; www.surrey.ac.uk

J.W. Walewski
Siemens AG, Otto-Hahn-Ring 6, Munich 81739, Germany
e-mail: joachim.walewski@siemens.com; www.siemens.com

Chapter 6	Chapter 7	Chapter 9
Process	IoT Reference Model	Reference Manual
• Process steps to generate IoT architectures • Compatibility with other architecting methods • IoT architecture generation and related activities • Requirements process and "other views" • Usage of Unified Requirements • Threat analysis • Design Choices	• IoT Domain Model • IoT Information Model • IoT Functional Model • IoT Communication Model • IoT Trust, Security, Privacy Model	• Usage of IoT Domain Model • Usage of IoT Information Model • Usage of IoT Communication Model • Usage of Perspectives

Chapter 11
Concrete Architecture
generate a concrete IoT architecture for a Pay-By-License-plate parking use case

Chapter 8	Chapter 10	Chapter 12
IoT Reference Architecture	Interactions	Testimonials

• Usage of Views and Perspectives in the IoT ARM
• IoT Functional View
• IoT Information View
• IoT Deployment & Operation View
• Perspectives
	• Management centric scenarios
	  • Adding a device
	  • Changing device configuration
	• Service-centric scenarios
	  • Discovering services
	  • Managing service choreography
		• Reverse Mapping ETSI M2M
		• Reverse Mapping EPC Global
		• Reverse Mapping Ucode
		• Reverse mapping MUNICH
		• Reverse mapping BUTLER
		• Business evaluation example healthcare use case

Fig. 5.1 Chapter structure for the guidance to the ARM

5.1 Chapter Structure

As you can see in Fig. 5.1, the technical part of this book consists of seven chapters in total. The starting point to guide and architect through the development of concrete architectures based on the ARM is Chap. 6, "Process". Chapters 7 and 8 describe the ARM specification by their IoT Reference Model and IoT Reference Architecture. Adjacent to Chap. 7, "IoT Reference Model" you will find the Chap. 9, "Reference Manual" in which the usage of the respective models is explained further. For guidance on the IoT Reference Architecture Chap. 9 provides material about the usage of perspectives. Typical management and service-centric scenarios are illustrated in Chap. 10, "Interactions". Chapter 11, "Concrete Architecture" illustrates the process of creating a domain specific concrete architecture along the process described in Chap. 6 for an example use case. The technical part of this book is concluded by Chap. 12, "Testimonials" that contains reverse mappings to IoT related standards and initiatives as well as an example for a business case evaluation of an IoT enabled use case.

5.1.1 Chapter 6 "A Process for Generating Concrete Architectures Process"

Provides the reader with detailed guidance on how to derive concrete architectures from the IoT ARM as briefly introduced in Chap. 3. It presents the process steps architects need to follow in order to generate an IoT architecture. That chapter also contains extensive treatises on how to use the IoT-A unified requirements (UNIs; see [Appendix, requirements]); on the common contents of an IoT threat analysis; and, last but not least, on how qualitative requirements are translated into design choices concerning their impact on designing the functional view, the information view, and the deployment view (see Chap. 8). Furthermore it is analysed in that chapter how compatible the presented IoT architecture generation process is with other well-known architecting methods.

5.1.2 Chapter 7 "IoT Reference Model"

Contains several sub-models the IoT Reference Model is made of, such as the IoT Domain Model, the IoT Functional Model, the IoT Information Model, the IoT Communication Model, as well as the IoT Trust, Security, Privacy Model. The IoT Reference Model provides the concepts and definitions on which the IoT Reference Architecture (see Chap. 8) can be built.

5.1.3   Chapter 8 "IoT Reference Architecture"

Provides architectural views and perspectives that are relevant for IoT systems. After a short introduction about views and perspectives the chapter presents the IoT Functional View, the IoT Information View and the Deployment & Operation View of the ARM. As explained in Sect. 8.3 of that chapter, the functional view of a concrete architecture typically consists of three viewpoints: functional decomposition (viz. the logical structure), interfaces, and behaviour. In Sect. 8.4.1 of Chap. 8, we provide an overview of the functional decomposition of an IoT system. More information on this logical viewpoint is provided in Carrez et al. (2013) of the ARM [Use cases, sequence charts and interfaces], and the interfaces of the FCs proposed in the functional decomposition are detailed in the same Appendix. That Appendix also contains a rudimentary interaction analysis, viz. illustrations of how the FCs can be interacted with and what the outcome of each interaction is. Chapter 8 also presents the architectural perspectives of the ARM which address quality aspects of the system to be designed, e.g. scalability and availability.

5.1.4   Chapter 9 "Reference Manual"

Contains reference manuals on the IoT Domain Model, the IoT Information Model, the IoT Communication Model, and the usage of Perspectives. While the Process Chapter outlines, how and when the modules of the IoT ARM (for instance the information model) can instruct the architecting process, the pertinent Section in the Reference Model, viz. Section 7.4, might not contain sufficient information on how to use the models of the IoT ARM. The respective reference manual section complements the guidance on the use of the model.

5.1.5   Chapter 10 "Concrete Architecture"

In order to further elucidate the guidance provided in the Process chapter, we discuss for a concrete example (pay-by-license-plate parking) how the IoT ARM can be utilised for the generation of a domain-specific architecture.

5.1.6   Chapter 11 "Interactions"

As can be appreciated by looking at already existing IoT systems, the operation of such systems generally involves sequences of FC interactions. Since the IoT ARM covers a huge range of usage domains and an even larger range of architectures that

can be derived from it, it is, unfortunately, beyond our capabilities to detail every possible FC interaction sequence for every possible architecture. However, in order to provide the reader at least with a general understanding of how such interactions can look like, we provide analyses for some few usage scenarios. These scenarios are divided into management-centric and service-centric scenarios.

5.1.7 Chapter 12 "Testimonials"

As discussed in Chap. 3, generating domain-specific architectures is not the only purpose of an ARM. Another important use is the identification of differences in architectures (see Chap. 3, Sect. 3.5). In Chap. 12 we provide examples for this usage. We looked at two IoT-related standards (ETSI M2M, EPC Global, uCode, and BUTLER), and we also showed how an already existing system, viz. the MUNICH platform, can be analysed with the concepts provided by the IoT ARM. The numerous examples provided in Chap. 12 are meant as an inspiration for how the reader can perform her own reverse mapping of existing architectures onto the IoT ARM. Furthermore, the high degree to which the mapping of our new framework onto already existing architectures and systems actually works is an initial indication for both the comprehensiveness and the utility of the IoT ARM. Additionally Chap. 12 presents a business evaluation example for an IoT enabled use case in the healthcare domain.

5.2 ARM History and Evolution

This third and final full version (v3) of the IoT Architectural Reference Model builds upon the intermediary version (v2) release end October 2012. Following its dissemination a third feedback process took place and eventually led to this version, where much technical improvements and new material can also be found. Therefore this version is not only a great improvement to the former full version 2 but also a consolidated version that takes into account many received comments (spread among three distinct feedback processes) from external stakeholders, from external technical experts, from internal partners involved into the other technical Work Packages of IoT-A and finally from the projects involved in the IERC AC1 activity chain on Architecture.[1]

Compared to IoT ARM v2, the technical improvements touch all aspects of the Models, Views and Perspectives – respectively found in Chaps. 7 and 8 – already introduced in former versions of the ARM. But it is also worth mentioning that Chaps. 6, 9, 10, 11, and 12 on Guidance has been drastically improved; for instance

---

[1] http://www.internet-of-things-research.eu/activity_chains.htm

it provides now also a very precise and comprehensive description of the whole process about deriving a concrete architecture out of the ARM (see Chap. 6). This chapter is a central part of this ARM master piece (300+ pages); it is fully dedicated to making this ARM useful to IoT system developers, by providing best-practice guidance and a large set of Design Choices that provide the system architects with concrete option when designing a concrete architecture out of the IoT ARM. This chapter also provided some elements of validation materialised through a "reverse mapping" exercise, applied to existing IoT Architectures.

As said above, this book is the final version of the ARM from the IoT-A "era". Still the project reckons that the ARM should live longer than just those 3 years project life-time; ensuring the sustenance of the ARM is therefore a major concern for IoT-A, and something that we definitely must organise and drive.

The IoT ARM is not a "Style exercise" aiming at staying on the corner of someone's desk. In order to fully reach its objective, which is wide-spread adoption by IoT system architects, the IoT ARM needs to be challenged even more and eventually improved. Only then it will reach its full maturity. From November 2013 onward, the ARM will be taken care of by the IoT Forum (which was officially founded in June 2013), within the "Technology" Working Group. Through this work, we will identify specific ARM "profiles" and make relevant design and technology choices needed to specify the profiles (e.g. "Semantic Interoperability" profile with a number of related technologies, functional components and interfaces, languages, semantic information model, etc.).

It is of the utmost important that industrial actors step into this activity and drive it, as they are the ones which will put the ARM into practice in the context of their own businesses. Sustaining the ARM and specifying profiles is a compulsory step on the path leading to standardisation.

Chapter 6
A Process for Generating Concrete Architectures

Mathieu Boussard, Stefan Meissner, Andreas Nettsträter, Alexis Olivereau, Alexander Salinas Segura, Matthias Thoma, and Joachim W. Walewski

This chapter addresses the question of how to generate concrete architectures with the IoT ARM, which is one of the many uses to which an architectural reference model can be put (see Chaps. 3 and 4). This topic was already touched upon in Section "Generation of Architectures" in Chap. 3, but it is covered in greater depth in this section.

Note that we do not prescribe any specific architecting methodology for generating concrete architectures. Instead, this section outlines how and where

---

M. Boussard (✉)
Alcatel-Lucent Bell Labs France, Route de Villejust, 91620 Nozay, France
e-mail: mathieu.boussard@alcatel-lucent.com; www.alcatel-lucent.com

S. Meissner
University of Surrey, Stag Hill, Guildford GU2 7XH, UK
e-mail: s.meissner@surrey.ac.uk; www.surrey.ac.uk

A. Nettsträter
Fraunhofer Institute for Material Flow and Logistics IML, Joseph-von-Fraunhofer Str. 2-4, 44227 Dortmund, Germany
e-mail: andreas.nettstraetter@iml.fraunhofer.de; www.iml.fraunhofer.de

A. Olivereau
Commissariat à l'Energie Atomique, Avenue des Martyrs 17, 38054 Grenoble, France
e-mail: alexis.olivereau@cea.fr; www.cea.fr

A. Salinas Segura
University of Würzburg, Josef-Stangl-Platz 2, 97070 Würzburg, Germany
e-mail: alexander.salinas@uni-wuerzburg.de; www.bwl.uni-wuerzburg.de

M. Thoma
SAP AG, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany
e-mail: Matthias.thoma@sap.com; www.sap.com

J.W. Walewski
Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
e-mail: joachim.walewski@siemens.com; www.siemens.com

during the architecting process the IoT ARM can provide help and input for the architect. We return to this topic of "methodology agnosticism" in Sect. 6.2.

As can be seen in Chap.3 Figure "Process for the generation of concrete architectures", the IoT ARM informs the engineering strategies for the design of a concrete IoT system, and the transformation rules are derived from the entirety of the IoT ARM. Also, the IoT ARM informs the requirement-generation process. In this section we are focusing in greater detail on the generation of requirements and on the transformation of these requirements into a concrete architecture. Notice that a concrete architecture implies that it meets a selected use case and application scenario.

6.1 Process Steps to Generate IoT Architectures

What are the main building blocks of a domain-specific architecture that adheres to the IoT ARM framework? The answer is: architectural views. As discussed at the beginning of Chap. 7 and Sect. 8.1 "Short Definition of Views and Perspectives" in Chap. 8, we chose to arrange a system architecture according to views, with the totality of all views constituting the architecture description. Figure 6.1 outlines how the views are related to each other and how they contribute to the system design. All views shaded yellow are covered in detail in the IoT Reference Architecture (see Chap. 8) or in this Section. These views are:

- Physical Entity view
- Deployment view
- Operational view
- IoT Context view

  - IoT Domain Model

- Functional view
- Information view

  In this figure:

- In dark red: views that are treated in Chap. 8 and in Carrez et al. (2013) or in this section;
- In orange: related models (see Chap. 7).

Note that since the IoT Domain Model also encompasses the role of users, it actually implicitly covers the enterprise view as advocated by RM-ODP (Raymond 1995) (see Chap. 3 for a discussion of the enterprise view and Sect. 7.3 for a discussion of roles in the IoT Domain Model). Note also that although the other views shown in Fig. 6.1 ("... view") are not covered in the IoT ARM, this does not imply that they are not important for generating concrete architectures. This
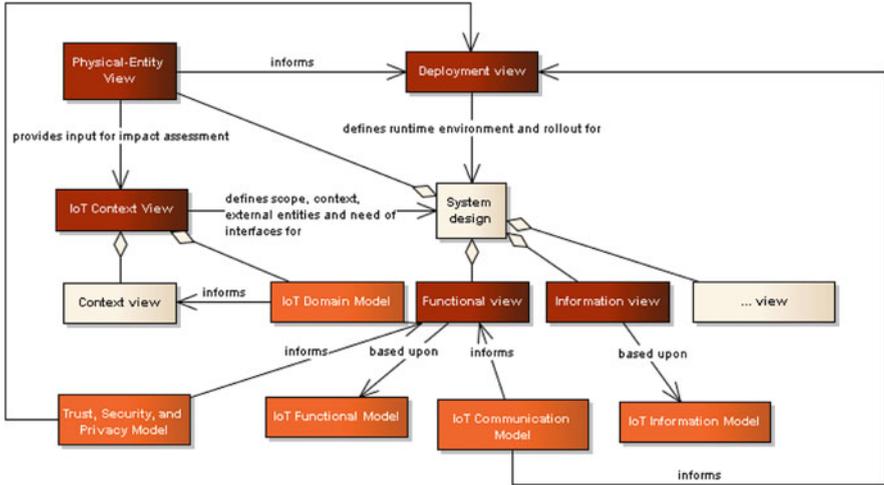
Fig. 6.1   Relationship of architectural views (based on Fig. 15-1 in Rozanski and Woods 2011)
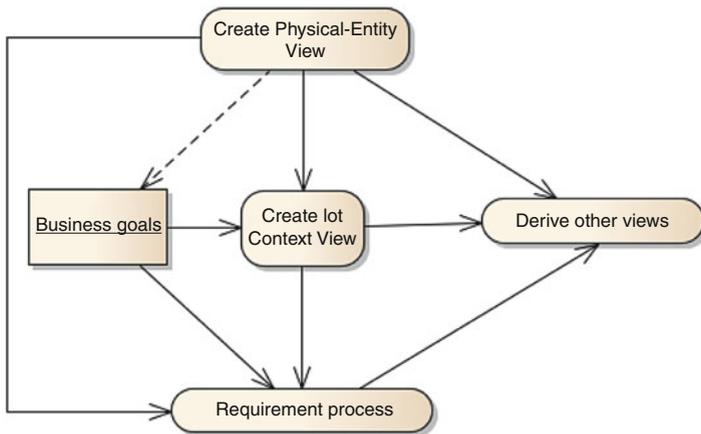
Fig. 6.2 UML activity diagram of the IoT architecture generation process (generation of requirements and transformation of requirements into a concrete architecture)

becomes clearer when we look at the architecture generation process in more detail. Figure 6.2 outlines the activities involved in generating an architecture. These are:

- Create Physical Entity view
- Create IoT Context view
- Requirements process
- Derive other views

In this figure, *dashed arrows* represent dependency, while *solid arrows* represent control flow (can be understood as either the next step or expressing a logical contingency of the target on the source).

As you can see, the creation of the Physical Entity View and IoT Context View (see Fig. 6.1) are explicit activities in the architecting process. All other views are comprised in the activity "derive other views". Before we look at each of these activities in more depth, let us return to the question of architecture methodologies and how the IoT ARM relates to them.

6.2 Compatibility with Other Architecting Methodologies

Figure 6.2 could give rise to the impression that we prescribe a sequential approach for generating architectures: (1) Define the scope, i.e. the business goals; (2) Create the Physical Entity View and the IoT Context View; (3) Define requirements; and (4) Generate the remaining views. This type of sequential approach to architecting lies, for instance, at the heart of the waterfall approach (Royce 1970). This interpretation of Fig. 6.2 is indeed true if all arrows in Fig. 6.2 are understood as arrows in time. However, they can also be understood as logical dependencies. For instance, in order to conduct the requirements process, we need a set of formulated business goals, an IoT Context View and a Physical Entity View. If we interpret the process described in this Section in the latter way, it can be mapped onto a plethora of popular architecting methodologies, such as Model-Driven Engineering (MDE) (Miller and Mukerji 2003), Pattern-Based Design (Gamma et al. 1994), and the Spiral Model (Boehm 1988).

The only limitation we see is in the choice of views. Some architectural methodologies prescribe different sets of views. Some of them, for instance the 4 +1 approach, lack some of the views we prescribe (mainly the information and context views) (Kruchten 1995). In this case we could choose to embed the 4+1 framework into the process described in this Section. On the other hand, other methodologies comprise views that are not part of the IoT ARM set. In this case, the option is to integrate the IoT ARM views (and the manner in which they are derived) into this other methodology.

6.3 IoT Architecture Generation and Related Activities

Since neither the IoT Context View nor the Physical Entity View are addressed in the IoT Reference Architecture (see Chap. 8), and since they are integral parts of the architecting process (Fig. 6.2), we need to look more closely at both of these views and understand how they inform the architecting process.

6.3.1   Physical Entity View

Before we describe the Physical Entity View we need to discuss what it is not: i.e. the "traditional" physical view in system architecting – a well-established view in software system architectures (see, for instance, Kruchten 1995). "It is concerned with the topology of software components on the physical layer, as well as the physical connections between these components. This view is also known as the deployment view." (Wikipedia 2013a; 4+1 view). As Fig. 6.1 implies, we are not using the term physical view for the deployment view in order to avoid semantic tension with the Physical Entity View.

   The Physical Entity View does of course refer to the Physical Entity in the IoT Domain Model (see Sects. 7.3.2 and 9.1). The Physical Entity is "any physical object that is relevant from a user or application perspective" (Appendix). For a concrete use case and application scenario, this is of course a well-defined set of physical objects. For instance, in the recurring example (see Sect. 4.2), the Physical Entities are the orchids that are transported in a truck and these orchids are subject to environmental monitoring.

   It is obvious for many reasons why the architecture of an IoT system also needs to include a Physical Entity View. Firstly, the dimensions, the distribution and the properties of the Physical Entities have various implications. Examples of these implications are:

- **Devices**: the sensors/actuators needed and where are they situated; their relationship to the Physical Entity (directly mounted; touching; remote but in sight . . .), etc. Note that the device choice is influenced by the Physical Entity. In the recurring example, it is too expensive (in relation to the market price of the Physical Entity) to measure the temperature of each orchid. Instead, sensors that measure the air temperature are situated inside the cargo area. It is then assumed that the air temperature equals that of the orchids. In other words, the Physical Entity model also needs to include a sensing and/or an actuating model.
- **Information view**: what physical quantities are monitored by the sensors; how are the quantities related to each other, etc.? In the recurring example the quantity that is handled by the system is the air temperature in the cargo area of the truck.

   Secondly, in some use cases, the devices might be incorporated inside the Physical Entity, which can have a range of implications for the IoT system. For instance, if sensors are deployed inside a human body and the wireless sensor signal is to be relayed to an outside reader, we need to understand the in-body propagation characteristics of this signal. It may be the case that the strong attenuation caused by the body tissue calls for a scenario in which signal repeaters are deployed. This has implications for the communication aspect of the architecture ($\rightarrow$ functional view).

   Thirdly, the type of the Physical Entity – in combination with the application scenario – can have implications for the Trust, Security, and Privacy Perspective (see Sect. 8.2.3). Let us look again at the recurring example. Since orchids can be

very expensive, and since this can increase the likelihood of the truck being raided while, for instance, parked during a coffee break or overnight, it is paramount that the wireless signal emanated by the orchid monitoring system cannot be identified as such nor be deciphered. If this were to happen, it could, for instance, inform a burglar how many orchids are in the shipment.

Although the Physical Entity View is obviously very central to the IoT ARM, it is not covered in the IoT Reference Architecture. This apparent contradiction is attributed to the overwhelming range of Physical Entities in the IoT: they can range from the nano- and micrometre scale to truly macroscopic dimensions (e.g. glacier monitoring); they can be gaseous or liquid; they can be animate or inanimate or a mixture of both; they can be stationary or mobile. "Mobile" can include walking, running, moving on wheels, flying, coasting under water, flying through interplanetary space, and so on. Also, there is no ONE physical quantity to be monitored – in one use case it can be the temperature of orchids, in another the occupancy of a room (automated light switch), in another case blood sugar levels. This overwhelming range of Physical Entities provides for the generation of generic yet comprehensive viewpoints and thus models for the Physical Entity View. This lack of "least common denominator" is the reason why it was not possible to devise Physical Entity models at the reference architecture level and thus integrate them into the IoT ARM.

The user of the IoT ARM is advised to use his own domain understanding to devise the Physical Entity view. Where required, pertinent models (for instance, freshness vs. room temperature model for orchids) either need to be developed by the architecture team or they can be extracted from outside sources (literature, standards, etc.).

6.3.2 IoT Context View

As indicated in Fig. 6.1, the IoT Context View consists of two parts: the context view and the IoT Domain Model. The context view is an architecture view that is generated at the very beginning of the architecture process. It describes "the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)" (Rozanski and Woods 2011). To be more specific, the context view describes "what the system does and does not do; where the boundaries are between it and the outside world; and how the system interacts with other systems, organizations, and people across these boundaries" (Rozanski 2013). The concerns addressed by the context view are (Rozanski 2013):

- "System scope and responsibilities
- Identity of external entities and services and data used
- Nature and characteristics of external entities
- Identity and responsibilities of external interfaces

- Nature and characteristics of external interfaces
- Other external interdependencies
- Impact of the system on its environment
- Overall completeness, consistency and coherence"

Note that at least one of the concerns, i.e. "impact of the system on its environment" also applies to the Physical Entity, and investigating this concern thus requires input from the Physical Entity View (see Fig. 6.2).

A detailed example of a context view, including a context diagram and a description of the system components, can be found in Chap. 11.

Note that the context view focuses mainly on what lies outside the system and how the system interfaces to the outside world. This is sufficient for "generic" architecting processes, but for the IoT domain, not only do we know more about the system to be devised, we should actually also gather more information about the system at a very early stage in the architecting process. Why? Firstly, since IoT systems have many aspects in common by virtue of operation in the same domain, a lot of concepts are recurring concepts. One of the goals of the IoT ARM is to avoid "reinventing the wheel", namely to avoid discovering, analysing and naming the very same aspects every time an architecture is generated. In order to permeate the entire architecture description with this understanding, we prescribe its use early on in the architecting process. This has advantages not only for the architecture generation itself, but also for other usages, such as architecture reuse. If common concepts, semantics, structures and relationships are fused into the core of an architecture description, this makes it much easier to reuse aspects of the architecture description or even the entire architecture. This can, for instance, be interesting for architecture development within a technology roadmap. Also, trust, security, privacy and safety are contingent upon system borders and thus on the functionalities and hardware that reside inside and outside the system border. The IoT Domain Model readily comprises both the "inside" and the "outside" of a system, and thus provides a deeper insight into relationships between the system entities and also interactions with the "outside world". For all of these reasons, it is beneficial to conduct a domain model analysis before embarking on actions such as threat analysis and requirements engineering.

So what other reasons are there for expanding the context view "inward", namely also covering the system itself? Why not just add a view to the architecting, namely the IoT Domain View, to the architecture description? The main reason is that both models are complementary and need to be applied early on in the architecting process. This is why we chose to pair the two system views. Note that the context in the IoT Context View has an extended meaning to that in the "traditional" context view, where it alludes to the context in which the system finds itself in relation to its surroundings. The IoT Context View expands on this by also including the entities within the system and by setting each of these entities in relation – context! – to the other entities.

The IoT Domain Model, on the other hand, provides a semantic and ontological overlay for the context view in that it provides guidance on which entities make up an IoT system and how they relate to each other. It also helps to identify system boundaries, which is one of the main questions to be addressed in the context view.

For more information on the IoT Domain Model see Sect. 7.3, and for guidance on how to generate a concrete IoT Domain Model see Sect. 9.1.

Note, that since all are listed and characterised in the IoT Context View, this is also the natural place for where to address the roles of all entities. These roles can for instance, be categorised as permissions, prohibitions, and obligations. For more information on these categories the reader is referred to elsewhere in the literature (Raymond 1995). For a discussion of how these roles figure into the system composition see Sect. 7.5.2.1).

An exhaustive discussion of the context view is available in literature (Woods and Nick 2008), but in order to enable immediate usability of the IoT ARM, we provide a short summary below.

6.4 Requirements Process and "Other Views"

6.4.1 Requirements Process

So far, we have shed light on two of the views that constitute an IoT architecture: Physical Entity View and the IoT Context View. Now we will discuss the remaining mandatory activities for generating an architecture: the requirements process and the derivation of "other views" (see Fig. 6.2). Figure 6.3 illustrates the architecture activities in more detail. How exactly the IoT ARM contributes to each of these actions is covered in the next Section.

As indicated in Fig. 6.1 and discussed in the previous Section, the context view is expanded by the IoT Domain Model. Therefore, both the generation of the "traditional" context view (see Sect. 4.1) and the expansion of this view in the IoT Domain Model are included in the creation of an IoT context view. As also explained in Sect. 6.4.1, the Physical Entity View provides input for the generation of the IoT context view.

With the input from the Physical Entity View and the IoT Context View, we can conduct a threat analysis. This type of analysis identifies potential weaknesses of the system use case envisaged, and it also identifies design choices and in some cases even functionalities that mitigate the risks identified. This analysis also provides guidance for the requirements engineering action (the security risks that need to be addressed by requirements).

The requirements process consists of many intermediate steps. The requirements engineering action generates a list of references that belong to one of three types: view requirements (i.e. requirements that directly inform one of the architectural views), qualitative requirements and design constraints. Note that we categorise the Unified Requirements (see online at http://www.iot-a.eu/public/requirements) along different dimensions (functional requirement, non-functional requirement, . . .) in order to increase the usability of UNIs for users who are not familiar with the
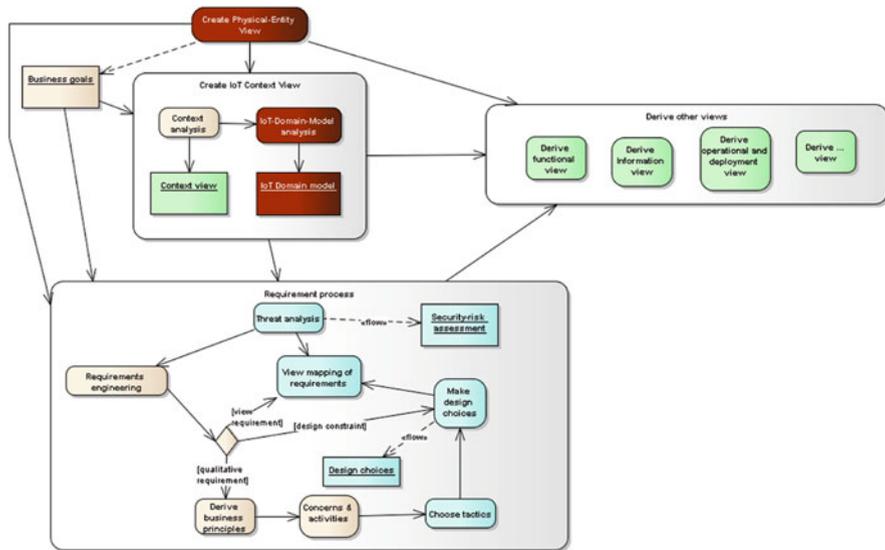
Fig. 6.3 IoT architecture generation (expansion of Fig. 6.2)

IoT ARM taxonomy of requirements. The translation of the UNI requirement types into IoT ARM process types is described Sect. 6.7.

In the prevailing approaches toward translating qualitative requirements into view-related requirements, we usually rely on a set of view requirements that is already available. An example of this type of approach is Quality-Function Deployment (Erder and Pureur 2003), which, amongst other things, is a central part of the ISO 9000 standards suite (ISO 2009). The assumption of an existing set of view requirements is a reasonable one for straight-forward product extensions or the design of simple systems, but for most IoT systems, this type of approach is not feasible. In other words, qualitative requirements cannot be translated directly into view requirements. In typical IoT systems, not only is complexity high, but there is often a plethora of options for achieving the desired performance of the system to be built. In other words, there are many sets of view requirements that meet the same set of qualitative requirements.

In order to overcome this design roadblock, we devised a step-by-step process through which view requirements can be inferred from qualitative requirements. The first step is to formulate the rationale of the qualitative requirements as business principles. For a detailed discussion of business principles, see (Rozanski and Woods 2011). This step is followed by identification of concerns and related activities. This action includes identifying each of the qualitative requirements with one or more architectural perspectives. The next step is to choose design tactics and then make design choices (covered in more detail in Sect. 6.4.1.8).

If the requirement is a design constraint, then it directly informs the design choice action.

From the design choices made, it is then possible to formulate implications for the functional view and other views (see Sect. 6.4.1.8).

If the requirement is of the view type, it can later be mapped directly onto the architecture description. We have found it very helpful to initially map functional view requirements onto the functional decomposition (see Sect. 8.1) throughout the requirements process. This makes it easier to track what parts of the system architecture are already covered by the requirements and whether more requirements are needed.

Salient inputs to the requirements process come, of course, from the Physical Entity View. This view, among others, provides the requirement engineer with information about special features of the "things" and the device-thing relationship (see Sect. 6.4). Another important source of information is the IoT Context View. It not only provides an overview of the system envisaged, but, thanks to the IoT Domain Model, it also provides the requirement engineer with information about the entities that are part of the system, what they are called, and how they relate to and interact with each other.

6.4.2 View Derivation

The remaining views are addressed in the activity "Derive other views". As shown in Fig. 6.3, this activity consists at least of the derivation of the functional view, the information view, the operational view and the deployment view. Where needed, other views can be addressed. Examples of such views are the concurrency view, the enterprise view and the engineering view (Wikipedia 2013b; view model). As indicated in Fig. 6.3, this activity is contingent on the requirements process and it is also guided by the Physical Entity View and the IoT Context View. For instance, the IoT Context View might indicate that, due to the different ownership of parts of the system, a communication firewall is needed ($\rightarrow$ functional view). In another example, the Physical Entity View might indicate that, due to the fragility of the Physical Entity, all devices attached need to be installed all at once ($\rightarrow$ deployment view).

In order to accommodate different architecting methodologies, we have detailed the dependence of each of the actions in Fig. 6.3 on each other in the crib sheet in Table 6.1. This Table provides an overview of IoT architecting activities and actions (left columns) and what relevant input one derives from other IoT architecting activities and actions (horizontal).

Figure 6.3 gives a detailed view of the actions taking place within each activity (Create context view; Requirement process; Derive other views).

- In Red: actions that are particular to the IoT-A architecting framework and that directly contribute to the architecture documentation;
- In Orange: actions that are not unique to the IoT-A architecting process, but that enjoy an emphasis in the IoT-A framework;
- In Green: other activities and documents that directly contribute to the architecture documentation;
- In Blue: actions that are not unique to the IoT-A architecting process, but that enjoy an emphasis in the IoT-A framework
- <<flow>>: information flow into a document.

Table 6.1 Overview of IoT architecting activities and actions (left columns) and what relevant input one derives from other IoT architecting activities and actions (horizontal). See Fig. 6.3 for a depiction of these activities and actions

		Create physical entity view	Create IoT context view	Requirement process							
			Domain-model analysis	Threat analysis	Requirements engineering	Choose tactics	Make design choices	Views mapping of requirements	Derive functional view	Derive information view	*Derive operation-al and deploy-ment view*
Create Physical Entity View	–		Overview of Physical Entities, properties one is interested in, and the basic relationships of the system entities and their properties.								
Create IoT Context View	*Domain-model analysis*	Defines what the Physical Entities in the IoT Context View are and how these entities relate to each other and what is to be done with them (sensing of properties, ..)									
Requirement process	*Threat analysis*	Overview of Physical Entities and the basic	Overview of entities in the system and								

Table 6.1 (continued)

	Create physical entity view	Create IoT context view	Requirement process								
			Domain-model analysis	Threat analysis	Requirements engineering	Choose tactics	Make design choices	Views mapping of requirements	Derive functional view	Derive information view	Derive operation-al and deploy-ment view
—	relationships of said Entities.	how they interact. Definition of system borders.									
Requirements engineering			Input for formulating requirements; overview of Physical Entities and the basic relationships of said Entities.	Input for formulating requirements; overview of entities in the system and how they interact. Definition of system borders.	Minimum list of security risks for which one has to formulate requirements. First input on functional strategies for mitigating said risks.						
Choose tactics											
Make design choices	Distribution of, property of, and physical connections between Physical Entities.	Distribution, grouping, and nesting of domain-model entities.	Minimum list of identified security risks with associated design choices.	Functional requirements → introduce complementary functionalities according to the design choices so that also qualitative	Chosen tactics.						

View mapping of requirements	requirements			View	requirements.		Mapping of design choices onto views.
Derive other views			requirements are covered.		requirements.		Information on life cycle, distribution, and hierarchy of information.
Derive functional view	Physical Entities and the basic relationships of said Entities.	Identified devices, resources, services, and Virtual Entities.	First input on functional strategies for mitigating said risks.	Functional requirements.	Design choices pertaining to the functional view.	Requirements that are (partially) mapped onto functions.	
Derive information view	Physical properties of interest.	Identified resources and Virtual Entities.	Impact of risks and identified mitigation strategies on information and data model.	Requirements related to information.	Design choices pertinent to information and data.		Identified FCs and the information to be exchanged between them.
Derive operational and deployment view	Physical Entities and the basic relationships and distributions of said Entities.	Identified devices, resources, services, and users.	Implications of identified security risks on operation and deployment.	Requirements related to information.	Design choices pertinent to deployment and operation.		Indications on operation from interacctions view.

Table 6.2 Overview of IoT architecting activities and actions

Architecting activity	Architecting action	Pertinent ARM module	Type of activity	Guidance/information provided in ARM module
Create IoT context view	Domain-model analysis	IoT domain-model (Sect. 7.3)	Create an IoT domain model of the envisaged IoT system. Use information provided in the context view for an identification of system boundaries and system scope	The IoT domain model including a thorough discussion of all its entities
		Reference manual pertaining to the IoT Domain Model (Sect. 9.1)		In-depth information on the entities in the IoT Domain Model together with examples of how to model these entities and entire systems
Requirement process	Threat analysis	Threat analysis (Sect. 4.1.7)	Perform a threat analysis for the envisaged system	List of system aspects to be taken into account during the threat analysis and what importance they carry
		Design choices (Sect. 4.1.8)	Link identified security risks to design choices for mitigation of said risks	Lists of design choices for mitigating security threats to IoT systems
	Requirements engineering	How to use unified requirements (UNIs) (Sect. 6.7)	Generation of system requirements	Insight of how the high-level UNIs can support the generation of system requirements
	Choose tactics	Perspectives (Sect. 8.3)	Mapping of requirements onto qualities (a.k.a. perspectives) and identification of suitable tactics for how to address said qualities. These tactics can be understood as architectural "lines of attack"	Minimum set of perspectives and associated tactics
	Make design choices	Design choices (Sect. 6.9)	By means of the tactics identified in the previous step, identify design choices for addressing the underlying qualitative requirements	Comprehensive taxonomy of tactics and associated design choices

View mapping of requirements	Non-functional views (Sect. 8.2.3 "Information View" and Sect. 8.2.4 "Deployment View")	Perform first mapping of non-functional view requirements (and, where expedient, of qualitative requirements) onto the information view and the deployment and operational views	Overview of IoT-specific aspects of these views and how to model them
	Functional view (Sect. 7. 2 "Functional Model" and Sect. 8.2.2 "Functional-View")	Perform first mapping of functional requirements (and, where expedient, of qualitative requirements) to FCs. Identify interfaces needed	Description of FGs and FCs
Derive other views	Derive functional view	Carrez et al. (2013)	Define basic functions, interfaces, and interactions for functional view	The Appendix Section provides exhaustive text on the FCs, related technical-use-case diagrams, descriptions of basic FC functions and high-level interfaces, as well as elementary interactions of FC and their basic functions (sequence charts)
	Chapter 10 "Interactions"	Define system-wide interactions of FCs	The Interactions Section provides examples of how such interactions can look like for a selected range of application scenarios
	Chapter10 "Interactions"	Derive interaction patterns for prevailing usage scenarios	As briefly discussed in Sect. 8.2.2, the interaction viewpoint is an integral part of the functional view, but the IoT ARM is situated at a far too high level of abstraction in order to allow the definition of the functional interaction sequences at the reference-architecture level. In order to mitigate this systemic shortcoming of the IoT ARM, we decided to discuss some few usage scenarios of IoT systems that can shed at least some light on how

Table 6.2 (continued)

Architecting activity	Architecting action	Pertinent ARM module	Type of activity	Guidance/information provided in ARM module
				such interaction patterns can look like, and they also provide inspiration for how such patterns might look like for the usages for which the concrete architecture is built
	Derive information view	Information view (Sect. 8.2.3 "IoT Information View")	Define information and data model for all FGs and also for the exchange with applications and devices (see Sect. 8.2.3)	The IoT Information View provides several pieces of general information. First, it elaborates on general information descriptions (description of Virtual Entities service descriptions; associations between services and Virtual entities). Second, it elucidates the information handling of FCs both from an interaction- and from a interaction-scenario point of view. Third, it discusses the information life cycle
Derive operational and deploy-ment view		Deployment & operational view (Sect. 8.2.4)	Define how to deploy and how to operate the system	This section discusses in depth how deployment of IoT systems. Since the deployment patterns vary a lot, this Section of course focuses on common topics and patterns. First, it identifies other views and device descriptions as viewpoints for the deployment. This aids in translating information in these views into actionable deployment knowledge. This Section also discusses the question of connectivity solutions and where to host the

Derive deployment view	Carrez et al. (2013)		services. It also touches upon the important questions of information storage and service resolution
Define APIs for the implemented FCs			Carrez et al. (2013) provides definitions of high-level interfaces for all the FCs in the functional decomposition in Sect. 8.2.2 "IoT Functional View". While these high-level interfaces are not equivalent to APIs, the information provided in Carrez et al. (2013) – (together with the sequence charts in the same Appendix) can be seen as a solid starting block for deriving APIs. More information can be found in the WP4 White Paper on Resolution Infrastructure Interface Binding (Bauer 2013), which describes a REST (Representational State Transfer) interface binding that was exemplarily defined based on the abstract interface specifications of the IoT Service Resolution and Virtual Entity Resolution FC found in Carrez et al. (2013)

This table gives an overview of Activities and actions when architecting an IoT system and shows what modules of the IoT-ARM document one can consult in order to support said activities and actions. The Table also provides outlines about the input/guidance that can be derived from said modules. For an overview also see Fig. 6.4

6.5 IoT ARM Contributions to the Generation of Architectures

After the previous detailed overview of the architecting actions related to the generation of an IoT architecture, we are now finally ready for a discussion of how the IoT ARM contributes to the generation of specific architectures. As already outlined in Sect. 6.2, we do not prescribe a particular methodology for the generation of the architecture. The choice of a particular methodology is contingent upon factors such as the organisational structure of the architecting team, its "architecture history", international standards or agreements that need to be adhered to, etc. Rather than prescribing a particular methodology and thus limiting its application range, the IoT ARM provides support and guidance for almost all of the actions and activities that are part of any architecting process (Fig. 6.4).

This figure gives a detailed view of the actions taking place within each activity and what parts of this document contribute to these activities and actions. The rectangular *dark-red boxes* represent sections in this document while <<flow>> represents information flow.

Table 6.2 discusses in more detail what each part of the IoT ARM contributes exactly to each of these actions and activities. This table also is intended to serve as a crib sheet for the architecting process.

6.6 Minimum Set of Functionality Groups

One question that we have often received concerns the least common denominator in terms of Functionality Groups of architectures that are derived from the IoT ARM. In other words: what Functionality Groups are part of any conceivable IoT ARM architecture?

The core aspects of IoT are things and communication. The things, i.e. Physical Entities (see Sect. 7.3) are accessed through devices, and data etc. pertaining to the Physical Entities is relayed by means of communication. Physical Entities are represented by Virtual Entities. Usually, the data is accessed via an application. Since we stipulate a service-oriented architecture framework in which the resources exposing data etc. about the Virtual Entities (and hence the Physical Entities) are exposed by IoT services, the minimum set of Functionality Groups is:

- Application Functionality Group
- IoT Service Functionality Group
- Communication Functionality Group
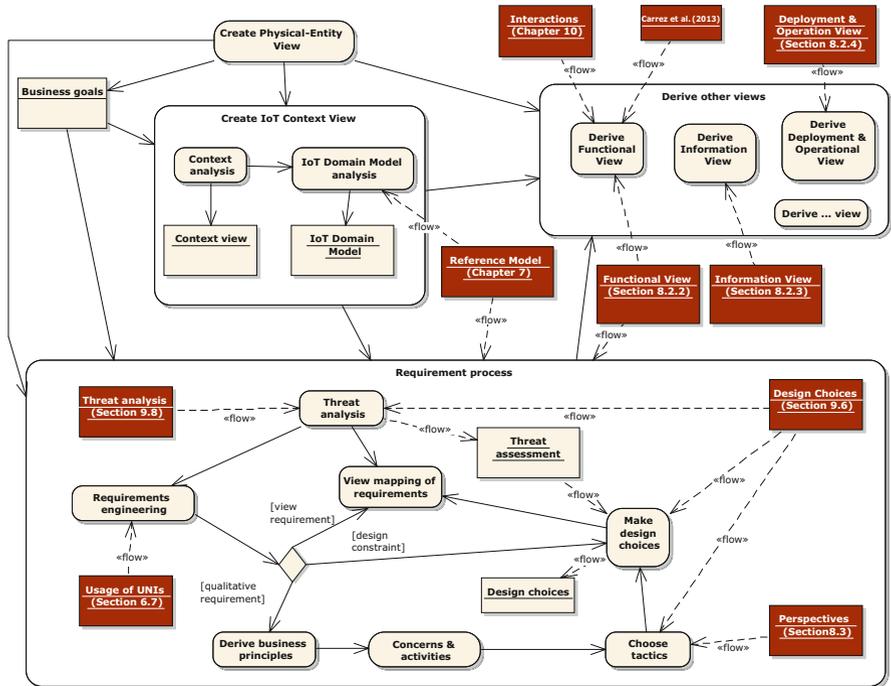- Device Functionality Group

Fig. 6.4 IoT architecture generation (expansion of Fig. 6.3) summarises the parts of the IoT ARM that are relevant to the IoT architecting process and to what particular action
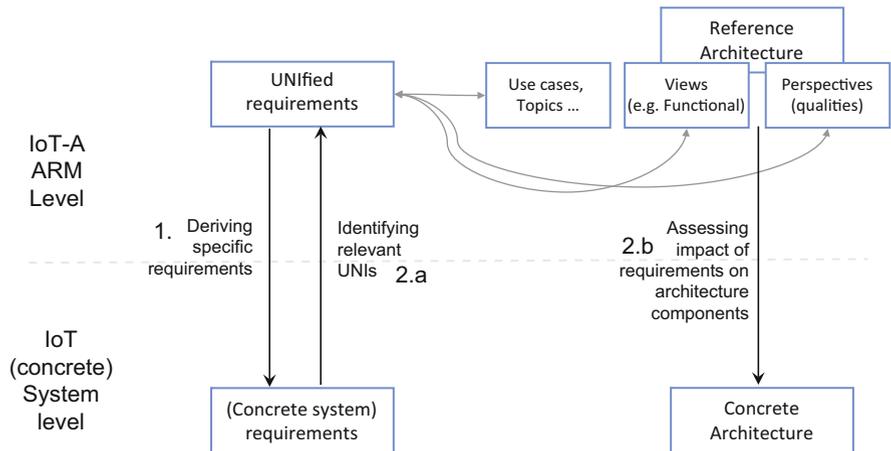

Fig. 6.5 Using IoT-A Unified Requirements and IoT ARM for concrete system architecture work

Note that this does not imply that other Functionality Groups (for instance, the Management Functionality Group) are optional. Rather, it means that for certain requirement sets these Functionality Groups are not needed.

6.7 Usage of Unified Requirements

6.7.1 Introduction

This section proposes guidelines to system architects on how to use the (already existing) Unified Requirement list (UNIs) during the Requirements process activity of their IoT architecture-generation process (Fig. 6.2). Such usage is by no means mandatory, as Requirement Engineering can be performed following the process described in Sect. 6.4 – however the UNIs list can serve as a helper tool to both the elicitation of requirements and to the system specification.

It is well known to system designers that requirement engineering is a crucial activity in system and software engineering. In the abundant documentation on the topic (e.g. Hull et al. 2011; Pohl 2010), one can distinguish three main steps where requirements play a role in designing complex systems: requirements elicitation (generally based on stakeholders input); deriving the system's specification from these requirements; and validating the implemented architecture.

As part of the work on the IoT Architectural Reference Model, UNIs were inferred and then published at http://www.IoT-a.eu/public/requirements. For more details on how these Unified Requirements were derived can be found elsewhere in the literature (Magerkurth et al. 2013). As these requirements do not apply to a concrete system, but rather to a Reference Architecture and a Reference Model applicable to all potential IoT systems, the reader needs to keep in mind a number of specifics before considering these Unified Requirements as input for the process of architecture translation:

- The Unified Requirement list should be seen as a basis and a living document. Although it tries to cover the whole spectrum of requirements families that could be applied to the IoT domain, it cannot be considered to be exhaustive, as, for instance, future regulation and legislation could impose requirements unforeseen at the time of publication. Additionally, Unified Requirements are often formulated on a quite high abstraction level (something largely avoided in concrete system's requirement engineering), resulting in requirements that are, for instance, mapped onto one or several views and possibly perspectives (again, something that concrete system designers tend to avoid);
- Formulation of requirements expressed by external or internal stakeholders (description field in the used Volere template) may sometimes apply directly to the IoT ARM (e.g. UNI.094 "The Reference Architecture shall support any IoT business scenario"), but in most cases they apply to a concrete system that can be implemented using the IoT ARM. In that latter case, they express characteristics on the system that the IoT ARM should enable to specify,

Table 6.3 Translation table for UNI requirement types from and to IoT ARM requirement types

UNI requirement type	IoT ARM requirement type	Indicated by
Design constraint	Design constraint	–
Functional requirement	View requirement	–
Non-functional requirement	View requirement	Mapping of UNI onto a view
	Qualitative requirement	Mapping of UNI onto one or more Perspectives

meaning they require to be interpreted by the reader/system designer to see how they apply to their own case – hence the wording "the system shall ..." generally used. Let us take for instance UNI.021 "The user shall be able to control the radio activity of the system": depending on the actual usage of radio communication, on the role of the user and on the importance of controlling the radio activity of the system in the concrete architecture, this requirement may be dropped, or specialised. In any case reinterpreting Unified Requirements is necessary (more on this in the following);

- Mapping to perspectives/views/functional groups and components is done on a lowest-common-denominator basis – e.g. it indicates which aspects are definitely impacted by a given Unified Requirement, but the reader should keep in mind that in certain (concrete system) specific cases, additional components may need to be considered. For instance, the Device Functionality Group is out of scope of the IoT ARM (see Figure "Functional-decomposition viewpoint" of the IoT Reference Architecture in Sect. 8.2.2.) and is therefore not listed in mapping of functional Unified Requirements, while it clearly needs to be considered when devising a concrete IoT system. Another instance is the lack of differentiation of the data plane vs. management plane in the IoT ARM, as this is a clear design choice (see Sect. 6.9).
- As pointed out in the ARM document, Sect. 6.4 and for the reasons explained there, the categorisation of the UNIs does not fully match that of the IoT ARM process and one needs to map the UNI categories onto that of the process in order to utilise the UNIs for the generation of architectures. Table 6.3 below provides this mapping information.

In a nutshell, the reader should keep in mind that the IoT ARM in general, and the Unified Requirement list in particular, should rather be seen as an inspirational than as a normative document.

6.7.2 Using Unified Requirements

IoT-A Unified Requirements (UNIs) can be used by system designers at two stages of their work: requirement elicitation and system specification.

6.7.2.1   Requirement Elicitation

UNIs can be used in a number of ways by system designers to identify "requirements topics" for their concrete system.

First, UNIs can be seen as seeds for deriving or instantiating concrete (precise) requirements from the broader, more abstract wording of Unified Requirements (Fig. 6.5- 1). For instance, UNI.018 reads "The system shall support data processing (filtering, aggregation/fusion, etc.) on different IoT-system levels (for instance device level)" (see Appendix). Based on this broad formulation, the system designer may derive his own requirements, identifying what kind of processing, on what kind of data, needs to happen where in his system.

Second, the mapping of the UNIs to Use Cases, facets of the IoT ARM (Models, Functional Groups, and Functional Components) or more informal categories can be used to filter and identify which topics and related UNIs should be considered by the system designer as potential candidates for instantiation on their own system. For example, using the web-based list, one can perform a global search on the word 'communication' (search all columns box), or filter all requirements categorised with the tag communication (Category column filter), or those which are sorted under the Communication Functionality Group (Functionality Group column filter) to see which UNIs in general apply to a given system.

6.7.2.2   System Specification

UNIs, and in particular their mapping to the IoT ARM, can also be useful to system designers during the specification phase. By identifying a UNI generalizing an already identified (concrete) system requirements (Fig. 6.5- 2.a), the various mapping on the IoT ARM enable the system designer to identify which IoT ARM components or more generally aspects are impacted by this requirement, and from there which concrete systems components or aspects need to be investigated (Fig. 6.5- 2.b). Figure 6.6 below presents this process using UML Activity diagram representation. Note that the "No corresponding UNI" case induces "regular" requirement engineering (i.e. without IoT ARM support).

For UNIs mapped on the Functional View, this enables the system designer to identify candidate functions in the concrete architecture that will be impacted by the overarching concern formulated in the UNI. For instance, UNI.623 reads "The system shall support location privacy". This requirement is mapped on the Security and Privacy Perspective, which means that the system designer should consider this Perspective when deriving her own system requirements (more on this below). This UNI is also mapped onto four Functional Components in three different Functionality Groups of the Reference Architecture (namely IoT Service, IoT Service Resolution for the IoT Service FG; Authorisation for the Security FG; and VE Resolution for the Virtual Entity FG). After identifying how these FCs are instantiated (or not) in a concrete system, the system designer can use such a mapping to derive where the considered requirement(s) impact the concrete architecture.
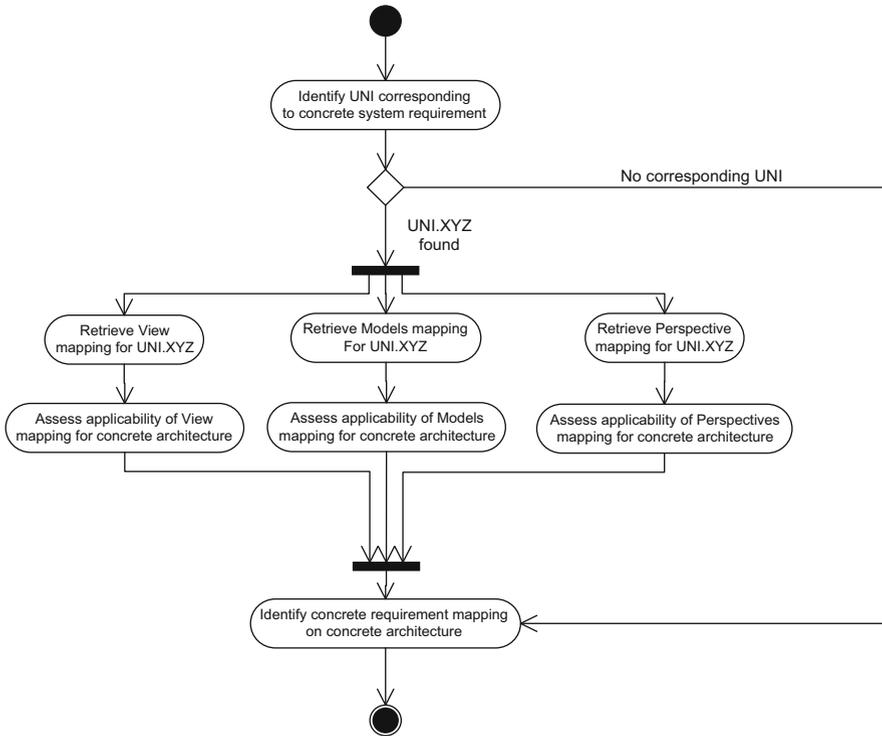
Fig. 6.6 How to use UNI to IoT ARM mapping to identify impacts of a given requirement on a concrete system architecture – activity diagram

Similarly, for UNIs assigned to quality aspects of the architecture (captured through ARM Perspectives), the mapping of UNIs onto design choices mapping (see Sect. 6.9 allows exploring perspectives and associated design choices that are impacted by a given UNI, and which therefore should be considered by the system designer. For instance, UNI.058 which reads "The system shall provide high availability" is mapped onto the Availability and Resilience ARM Perspective, and can be instantiated using two Design Choices (namely Cluster by location and Cluster by type of Resources). A corresponding concrete system requirement would typically provide more details (such as availability rate, etc.). After identifying which Perspectives apply (and how) to their concrete system, the system designer can use such a UNI-to-Perspective mapping to derive which quality aspects of the concrete architecture are impacted by the considered requirement(s).

6.8 Threat Analysis

As part of the setup of an IoT architecture, risk planning and resulting architectural decisions are of highest importance. The risk analysis carried out in this section aims therefore at assessing risks pertaining to the IoT, and at classifying them

according to the underlying mechanisms they apply to, the elements they affect, and the overall criticality they present.

Risk analysis traditionally begins with a definition of the elements that have to be protected. Then, an analysis of possible threats is conducted. How identified threats may actually affect elements to be protected, leads to the definition of risks. These risks have to be categorised, taking into account parameters such as criticality or probability of occurrence.

Various risk-analysis methods have been promoted in the literature, such as the French EBIOS (Ebios 2010) and OCTAVE (OCTAVE). The methodology for risk analysis that has been chosen in IoT-A, and that is used in this section, is based on Microsoft STRIDE/DREAD (Microsoft 2003). This choice has two reasons: first, this methodology is designed for assessing risks in the field of communications and information systems; second, it is mostly based on the analysis of architecture models and communications flows (instead of, for example, partly relying on experts interviews such as in EBIOS), which makes it a good fit for the ARM. The reasons for this are twofold. First, IoT, by it very name, encompasses information systems and communication. Second, no IoT-A-implementations are available at the time of writing. Therefore, the analysis has to centre on the Reference Architecture itself.

This section is organised as follows: first, a list of elements to be protected is provided. Then, the threats that may affect these elements (risk sources) are reviewed. The review follows the STRIDE classification. More details on STRIDE are provided below. The identified risks are then summarised and each risk is assessed in accordance with the DREAD methodology/metric.

This risk analysis is intended to be used as input for the derivation of architectures from the IoT ARM and for also for guiding the evolution of such architectures. By so doing one makes them more resilient against the most critical risks.

6.8.1   Elements to Protect

What elements need to be protected depends on the considered scenario. However, the IoT ARM was derived from the synthesis of a wide range of use-case areas, and identifying elements to be protected becomes rapidly very broad and multi-faceted. Instead, we decided to focus on the least common denominator of all use-case scenarios on which the IoT ARM is built. In other words, this analysis only looks at general elements to be protected, and this study is thus a good but non-exhaustive starting point for the study of a particular scenario to which the IoT ARM is going to be applied. The scenarios encompassed by the IoT ARM include:

- Transportation and logistics;
- Smart home;
- Smart city;

- Smart factory;
- Retail;
- eHealth;
- Energy (Smart Grid).

The following elements to be protected were identified:

- **Physical person**: This represents the human user. Threats affecting the human user are usually qualified as relating to 'safety' instead of 'security'. Such threats may arise if a critical service is diverted or made unavailable by an attacker. An example for this is a malicious service that returns erroneous information, or even information specifically shaped to create hazardous situations. The eHealth scenario is the most critical concerning such attacks. Notice that the level of this criticality of course depends on the degree of automation. It is likely that most critical decisions will still require the involvement of a human operator;
- **Subject's privacy**: This element represents all information elements that a subject (either a user or a device) does not explicitly agree to make publicly available, or whose availability shall be restrained to a controlled set of other subjects;
- **Communications channel**: The communication channel itself has to be protected. Common threats are attacks against the integrity of the data that are exchanged over the channel. Examples for such attacks are tampering and replay attacks. The communication channel shall also be protected against attacks aiming at the routing functionality of the underlying network (black hole, worm hole, depletion, etc.) (Mathur and Subbalakshmi 2007);
- **Leaf devices**: IoT-A leaf devices represent the wide variety of IoT elements that are interconnected by the common IoT-A infrastructure. Tags, readers, sensors, and actuators are examples for leaf devices. Various protection schemes relevant to their object class capabilities are to be implemented. These schemes need to ensure the integrity of the software, hardware, and the location of these devices;
- **Intermediary devices**: Intermediary devices provide services to IoT-A leaf devices and they also enable communication. A gateway designed to interconnect constrained and unconstrained domains is an example of such an intermediary device. Disabling or tampering critical intermediary devices can lead to denial-of-service attacks against the service infrastructure. Such attacks are within the scope of our analysis. However, attacks against specific intermediary devices that offer non-critical facilitating functions are outside the scope of our analysis and have thus to be considered case by case;
- **Backend services**: Backend services represent server-side applicative elements (for instance data-collection server communicating with sensor nodes). Compromising this software or the devices they are deployed on generally represents a critical threat against specific application systems and has to be prevented;
- **Infrastructure services**: Discovery, lookup and resolution services are very critical services as they provide worldwide fundamental functionalities to IoT systems. In the same way, security services (authorization, authentication,

identity management, key management, and trust and reputation) are essential for a secure interaction between subjects (as defined above);

- **Global systems/facilities**: This last category of elements to protect considers entire services in a global manner. For example, there might be a risk that an attack against the smart home scenario results in the complete disruption of the service, e.g. through the disruption of underlying communications between devices. The consequences of this resulting disruption can therefore be considered through this category.

6.8.2 Risk Sources

The risk sources are categorised following the STRIDE (Microsoft 2003) classification, which is a widely used way of classifying threats that relate to information systems. STRIDE stands for **S**poofing identity, **T**ampering with data, **R**epudiation, **I**nformation disclosure, **D**enial of service, and **E**levation of privilege. These categories are quickly summarised below – note, however, that real-world occurrences usually consist of a combination of these threats.

- **Identity spoofing** means that a peer illegitimately uses the identity of another peer. Spoofing attacks can happen with respect to all kind of identifiers, irrespective of whether they are used to designate physical persons, devices, or communication flows;
- **Data tampering** means that an attacker is able to alter the content of data exchanged between two or more peers. Data tampering may involve subtle attack schemes, wherein the attacker is able to trigger specific behaviours of recipients by finely modifying original data;
- **Repudiation** relates to attacks in which an attacker performs illegitimate actions and may afterwards deny having performed them, such that other nodes are unable to prove that the attacker actually behaved maliciously;
- **Information disclosure** means that information is disclosed to unauthorised peers. It is related to the existence of an authorisation model that defines for each information element a set of peers that are authorised to access it, possibly under some specific conditions;
- **Denial-of-service** attacks are carried out for disabling a service offered to legitimate users (as opposed, for example, to more subtle schemes wherein the attacked service can be altered, e.g. making a search service return false results, without the legitimate users being able to notice it);
- **Elevation of privilege** may occur in systems that feature different classes of users, each class being mapped to a specific set of rights. Illegitimate elevation of privilege occurs when an attacker manages to acquire rights that would normally only be granted to more privileged class(es). In the most critical case, an attacker may obtain administration rights for the entire system, or part of it, which means that the attacker may perform arbitrary actions on the elements the attacker has access to, thereby being able to destroy the system or entirely change its behaviour.

The risk sources considered here are restricted according to the following rules:

- Non-human risk sources either global (flood, lightning, fire, electrical, heat) or local (individual device failure) are not considered. Only human risk sources are. Note that a human forging a faked device identity in order to impersonate another device fits within the category of "human risk";
- Among human risk sources, only theft/loss and hacker-initiated attacks are considered. Technical staff errors or accidents are not considered. In other words we are only addressing malicious attacks and not involuntary attacks.

The STRIDE classification is used below in Table 6.4, immediately afterwards, on STRIDE classification] to identify risks, as intersections between a STRIDE item (column) and an element to protect (row).

6.8.3   Risk Assessment

Identified risks were assessed using the DREAD methodology based on (simplified) metrics. DREAD, defines scoring methodology and metrics that help to evaluate the criticality of an identified threat. DREAD stands for **D**amage potential, **R**eproducibility, **E**xploitability, **A**ffected users, and **D**iscoverability. It defines the criteria according to which a threat is evaluated. Each criterion is quantified at levels between 0 and 10. Eventually, the threat can be globally rated (sum of D, R, E, A, D ratings), or the threat can be described along with its individual ratings. The latter approach allows, obviously, for a more precise analysis. A simpler scheme for DREAD, used in what follows, consists of only three levels, viz. L (low), M (medium) and H (high) for each DREAD rating.

Note that a 'High' rating for Exploitability means that it is easy for an attacker to carry out an attack leading to the identified threat, whereas a 'High' rating in Discoverability means that it is difficult to discover the threat. This is to ensure a coherent approach, in which 'Low' ratings decrease the overall criticality of a risk, whereas 'High' ratings increase it.

The DREAD methodology and metric is used in Table 6.5, immediately afterwards, on DREAD assessment] for evaluating the risks identified in Table 6.4, the previous one, on STRIDE classification]. In addition to the DREAD rating, the Table 6.5 on DREAD assessment]also provides initial information on specific threats that may lead to the occurrence of the identified risk. In addition to this information, initial steps toward threat mitigation are provided. Furthermore, it links mitigation scenarios to the design choices (noted DC X.n) elaborated on in Sect. 6.9.

6.8.4   Discussion

Assessing the risks that relate to the Internet of Things and putting them in perspective with the Design Choices (see Design Choices) leads to interesting

Table 6.4 STRIDE classification (horizontal) of the identified risks broken down by the elements to be protected (vertical)

	Spoofing identity	Tampering with data	Repudiation	Information disclosure	Denial of service	Elevation of privilege
Physical person		Attack alters data so that wrong data is supplied to a critical monitoring system	Human users might use unattended electronic devices without leaving a digital trace		A service critical for user's safety is disabled	
Subject's privacy	User's identity is spoofed User is involved in transactions with a malicious peer			Attacker gleans knowledge of user private parameters Attacker gleans knowledge of user's location		
Communication channel		Alteration of the invocation of a service Alteration of the return value upon service invocation	Jamming wireless communication channels leads to local denial-of-service attacks that can be repudiated (no digital traces)	Attacker gains knowledge of sensitive exchange data	Attacker disrupts communications	Wrong authorisation information propagating from one server to another
Leaf devices	Loss or theft of physical device used for authentication	Attacker gains control of an actuator Attacker alters leaf-device content so that a user will eventually be		Disclosure of device-configuration information Device identification may divulge sensitive information, or may be linked so that it	Attacker physically disables local leaf device Attacker physically disables remote leaf device	

	Attacker changes the association between a Virtual Entity and the corresponding Physical Entity	Attacker alters sensor device so that monitoring of a Physical Entity fails … redirected to a malicious content		Loss or theft of device containing private information … exhibit information about usage patterns,	Attacker prevents proper communication to an actuator
Intermediary devices		Compromised intermediary devices alter traversing data	Intermediary devices behave maliciously and clients are not able to report the fact	Information re-routing by intermediary device so that it ends up at an unintended destination	Assisting intermediary devices are no longer usable
Backend Services	Usurpation of administrator role; Backend account hacked			Massive disclosure of collected data	Backend service is made unavailable
Infrastructure services	Attacker impersonates infrastructure services and compromises IoT functionalities and/or other dependent infrastructure services	Attacker poisons infrastructure databases and/or alters outgoing information		Disclosure of private services (existence & description); Disclosure of access policies; Disclosure of Identities and cryptographic material	Attacker denies legitimate users access to infrastructure services
Global systems/facilities				Massive disclosure of user information	Disruption of a global service

Table 6.5 DREAD assessment of the identified risks (see Table 6.4 STRIDE classification)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
Physical person	Attack alters data so that wrong data is supplied to a critical monitoring system	H/L/M/L/L → enforce **strong** security		Data-integrity protection provided as part of protocol security. DC S.16: cryptographic protocols DC S.19: integrity protection obtained from authentication enforcement at link layer
	Human users might use unattended electronic devices leaving no digital trace	L/L/H/L/L → enforce weak security		Addressable through proper (local/remote) user authentication scheme which is a feature of the Authentication Functional Component (see Sect. 8.2.2.7). DC S.1.3: ensure proper logging of authentication operations, e.g. through the use of a AAA (authentication, authorisation, and accounting) or a AAA-like system
	A service critical for user's safety is disabled	H/M/M/L/L → enforce **strong** security		Critical services have to be protected through redundancy of their key elements. Malicious actions are prevented through dedicated access-control policies (security management). Communication medium between user and critical service has to be made robust against DoS attacks at all OSI layers DC S.5: restrained service access DC A.16–17: autonomous security

User's privacy	User's identity is spoofed	L/H/L/M →enforce **strong** security	Credential theft	Robust user-authentication procedure preventing man-in-the-middle attacks, with proper credentials management policy provided by Authentication Functional Component (see Sect. 8.2.2.7)
				DC S.1: authentication over encrypted channel
				DC S.10 avoid common crypto credentials; avoid reliance on symmetric crypto
			Credential brute-forcing	
			Registration procedure that is vulnerable to man-in-the-middle attack	
	User is involved in transactions with a malicious peer	L/H/H/M/L →enforce **strong** security	Redirection to malicious content. The redirection may be caused by data tempering on communication channel or leaf node compromising (e.g. content of a tag is altered)	Trustworthy discovery/resolution/ lookup system. Trustworthiness of the entire system is enabled through its security Functional Components (especially Authentication and Trust and Reputation (see Sect. 8.2.2.7), as well as its global resilience to intrusions (security by design) Resolution security DC S.1: authentication over encrypted channel
	Attacker gains knowledge of user configuration	M/M/M/L/H →enforce **strong** security	User's private information leakage through user's characterisation as requiring certain data (and thus performing accordingly discovery, lookup, resolution of the corresponding services)	Enforcement of a robust pseudonymity scheme ensuring both anonymity and unlinkability of two successive data units; provided by the Identity Management Functional Component (see Sect. 8.2.2.7)

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
				DC S.10: encryption schemes, with a specific relevance of onion-routing-like encryption (best scheme with respect to anonymity support)
			User's private information leakage through user's characterisation as providing certain data. Traceability (this path, hence this user).	DC P.1: temporary identity, more easily changed for unlinkability, hence privacy
	Attacker gains knowledge of user's location	L/H/M/L/H →enforce weak security		User's location can be hidden through reliance on pseudonyms provided by the Identity Management Functional Component (see Sect. 8.2.2.7)
				DC P.1: temporary identity, more easily changed for unlinkability, hence privacy
Communication channel	Alteration of the sent invocation of a service	L/L/M/L/L →enforce weak security		End-to-end integrity protection of service-access signalling (data integrity protection is provided as part of protocol security)
				DC S.1.3: service-based data integrity
				DC S.19: integrity protection obtained from authentication enforcement at link layer
	Alteration of the return value upon service invocation	L/L/M/L/L →enforce weak security		End-to-end integrity protection of service-access signalling (data integrity protection is provided as part of protocol security)

Jamming wireless communication channels can lead to local denial-of-service attacks that can be repudiated	M/H/L/M/M ➔ enforce medium security	Service-based data integrity DC S.19: integrity protection obtained from authentication enforcement at a layer below the service Jamming denial-of-service attacks can be addressed through physical means: for instance, once the attack is detected localise and neutralise the jammer DC A.16–17: autonomous security *could* be enabled for detecting this attack
Attacker gains knowledge of sensitive, exchanged data	M/L/M/L ➔ enforce medium security	End-to-end confidentiality protection of exchanged data, offered through protocol security DC S.10: encryption schemes
Attacker disrupts communications	M/H/L/H/L ➔ enforce medium security	Various denial-of-service prevention schemes are available. Their applicability depends on the communication technology used (anti-jamming, enforced MAC, etc.). Schemes are offered through security-by-design of the communication stack DC A.16–17: autonomous security systems are generally able to deter denial-of-service attacks, however lightweight schemes are less powerful

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
	Wrong authorisation information propagating from one server to another	M/L/L/H/M ➔enforce medium security		Strong security for server-to-server communications that leverages individual's credentials (e.g. certificates) instead of group keys, and allows for revocation (security by design, adequate management policies)
Leaf device	Loss or theft of a physical device used for authentication	M/L/H/L/L ➔enforce weak security		Two-factor authentication, when applicable. This means that the gain of the physical device would not be enough for an attacker to pretend being a legitimate user and authenticate as such
				Cryptographic credentials should be themselves protected (PIN code, passphrase)
				DC S.1.3: authentication. Note that identification instead of authentication should *not* be applied
	Loss or theft of physical device containing private information	M/L/H/L/L ➔enforce medium security		Physical protection of stored credentials (e.g. security vault) – readability of a device only upon fulfilment of certain conditions (e.g. known reader)

Attacker changes the association between a Virtual Entity and the corresponding Physical Entity	M/L/M/H/L ➔enforce medium security	Wrong tag on a device	Secured discovery/resolution/lookup system
			A specific Design Choice for tamper-proof IDs is not provided for two reasons. First, one could realise it on a hardware-level by using tamper-proof hardware modules. Notice that hardware is out of scope for IoT-A (device level is not part of the RA). The second reason is that tamper-proof IDs can also be realised by a secure resolution system by means of Authentication and Authorisation which is already part of the RA and thus no Design Choice is needed
Attacker gains control of an actuator	M/M/M/L/M ➔enforce medium security	Compromising resolution system	Proper authorisation scheme as offered by the Authorisation Functional Component (see Sect. 8.2.2.7)
			End-to-end integrity protection, provided as part of protocol security
			DC S.5: prevent compromise through access restriction
			DC A.16–17: reactive (autonomous) security in case of compromise
Attacker alters leaf-device content so that a user will eventually be redirected to a malicious content	M/M/H/M/L ➔enforce medium security		Not specifically targeted. Addressable through a proper URI verification system on user device

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
	Attacker alters sensor device so that monitoring of a Physical Entity fails	L/M/L/L/H →enforce weak security		Not specifically targeted. Sensitive physical values may be monitored by a large number of sensors, or sensor integrity can be remotely verified
	Disclosure of device configuration information	L/L/L/L/H →enforce weak security		Not specifically targeted. Unlinkability between different actions of the same device, provided by the Identity Management Functional Component (see Sect. 8.2.2.7), will mitigate the criticality of this threat DC P.1: use of temporary identity to provide unlinkability
	Device identification	L/M/M/L/H →enforce medium security	Attacker bypasses in-place pseudonymity scheme and identifies a device as providing access to certain data	Adequate protection scheme requiring partial pre-knowledge of each other before a tag can be read by a reader (the tag will only answer to a "known" reader)
	Attacker physically disables leaf device (local)	L/H/H/L/L →enforce weak security	Tag destruction	Not specifically targeted. Typically addressable through physical investigation (identify the attacker through traces left by the physical attack; e.g. triangulation of a destructive electromagnetic pulse)

	Threat	Rating	Attack vector	Mitigation
	Attacker physically disables leaf device (remote)	M/H/L/H/L ➜enforce weak security	Tag destruction by remote electro-magnetic means	Not specifically targeted. Typically addressable through physical investigation
	Attacker prevents proper communication to an actuator	M/H/L/M/L ➜enforce medium security		Denial-of-service detection/reaction scheme (security by design)
Intermediary devices	Compromised intermediary devices alter data passing through	M/H/M/M/L		End-to-end security scheme provided by the Key Exchange and Management Functional Component (see Sect. 8.2.2.7), and enforced by the relevant protocol security function
		➜enforce medium security		Remote monitoring of intermediary devices can be another means of dealing with this threat (identification of compromised devices) DC S.10: end-to-end encryption DC A.16–17: autonomous security
	Intermediary devices behave maliciously and clients are not able to report the fact	M/M/L/M/H ➜enforce weak security		Remote monitoring of intermediary devices
				Depending on the malicious action performed by intermediary devices, client nodes may mitigate it by applying end-to-end security schemes (Key Exchange and Management Functional Component + protocol security) DC A.16–17: autonomous security

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
	Information re-routing by intermediary device	M/H/M/M/M ➔enforce medium security		End-to-end security scheme put in place by the Key Exchange and Management Functional Component (see Sect. 8.2.2.7), and enforced by the relevant protocol security function DC S.10: end-to-end encryption
	Assisting intermediary devices are no longer usable	L/M/H/H/L ➔enforce medium security	Exhaustion attacks	Denial-of-service detection/reaction scheme DC A.16–17: autonomous security
			Various specific attacks against the involved assistance mechanisms (e.g, no packet forwarding toward a routing service, replacing a received key fragment with garbage against a collaborative keying service...)	
Backend services	Administrator-role usurpation	H/M/L/H/L ➔enforce medium security	Administrator credentials disclosed/hacked/brute-forced	Not specifically targeted. Addressable through security management and credentials management policies
	Backend account hacked	M/M/L/H/M ➔enforce medium security		Not specifically targeted. Addressable through security management and credentials management policies
	Massive disclosure of collected data	H/M/L/H/L ➔enforce **medium** security		Not specifically targeted. Addressable through security management (databases)

Backend service becoming unavailable	L/M/M/H/L ➔enforce **medium** security	DoS detection/reaction scheme DC A.16–17: autonomous security
Attacker impersonates infrastructure services, compromising IoT functionalities and/or other dependent infrastructure services	H/M/L/H/M ➔enforce **strong** security	Prevention of impersonation techniques through proper use of authentication/authorisation procedures (enforced by the respective Authentication and Authorisation Functional Components (see Sect. 8.2.2.7) DC S.1.3: authentication
Attacker poisons infrastructure databases (records corruption/addition) or alters outgoing information	H/H/L/H/M ➔enforce **strong** security	Proper authorisation scheme provided by the Authorisation Functional Component (see Sect. 8.2.2.7) mitigates this attack. Enforcement of a trust model (Trust and Reputation Functional Component (see Sect. 8.2.2.7) protects against blind acceptance of erroneous data DC S.5: service access control. Although this does not allow identifying corrupted data, it may help identifying and excluding the attacker
Disclosure of private services (existence & description)	L/H/H/M/M ➔enforce medium security	Masking the belonging of multiple services to a single entity (unlinkability). This can be achieved by reliance on pseudonyms provided by the Identity Management Functional Component (see Sect. 8.2.2.7) DC P.1: mitigation through the use of temporary identifiers

(continued)

Table 6.5 (continued)

Element to protect	Risk	D/R/E/A/D rating	Examples of causes	Mitigation and relevant design choices (for the latter see Sect. 6.9)
	Disclosure of access policies	L/H/H/M/M →enforce medium security		Security management of infrastructure prevents global disclosure of access policies from the decision point to an unauthorised external attacker. Probe discovery of access policies by authorised, though compromised internal attackers are more subtle, and have to be dealt with through adaptive security (e.g., recognise a malicious pattern in the regular probing of security decision points) DC A.16–17: probing detection/reaction performed by autonomous security
	Disclosure of identities and cryptographic material	M/H/H/M/L →enforce **strong** security		Not specifically targeted – addressable through security management (databases)
	Attacker denies legitimate users access to Infrastructure Services	M/H/L/M/L →enforce medium security		Exclusion of the attacker, once identified as such through the Trust and Reputation security Functional Component (see Sect. 8.2.2.7)

Massive disclosure of user's personal information	H/L/L/H/L ➔enforce **strong** security	Secure storage of personal data with dedicated protection architecture (e.g. firewall diodes that let data flow in one direction only) and access control rules—this is part of security management
Disruption of a global service	H/M/L/H/L ➔enforce **strong** security	Reliance on all security Functional Components (see Sect. 4.2.2.7) + proper security management. This threat can also be addressed by multihoming. See **DC P.3** (Replication of instances of Functional Components locally) and **DC P.4** (Replication of instances of functional components in the cloud)

synthetic conclusions. First, we recognise in the risks and their mitigation mechanisms the well-known distinction between **internal attacks** and **external attacks**. This distinction implies the existence of a discrimination function that makes the system able to distinguish among authorised players (hence, able to launch internal attacks) and unauthorised players (restrained to external attacks). Second, it is also noticeable that some risks are not mapped to design choices – rather, they can be mitigated through dedicated context-dependent or local (entity-scope) security-by-design decisions. These concepts are elaborated on in what follows.

The distinction between internal and external attackers pertains to their ability to undergo an authorisation procedure, at the end of which only authorised players acquire some rights. These rights in turn enable the attackers to launch internal attacks. Note that this authorisation procedure may be characterised by more than the rejected/authorized two levels of granularity and define a full set of access policies. In this case, all but entirely rejected players are in position to launch internal attacks.

The defence against **external attacks** is traditionally based on two means: topological defence systems that almost *spatially* keep the attackers out of reach of the protected resources (e.g. firewalls) and cryptographic mechanisms (e.g. authentication or encryption algorithms) that *logically* prevent attackers to tamper with or otherwise access the protected resources.

- In the framework of IoT, special emphasis is put on one-to-one transactions wherein a service is accessed by a remote player. These transactions require a **secure transaction set up**. The service-access control involves in its most secure embodiments an authentication phase that can be based on various authenticating credentials. It has to be noted, though, that these authenticating credentials have to be mapped to an identity in order to fulfil their role. When the peer identity is not known prior to establishing a transaction, it has to be securely retrieved (resolved) from the resolution infrastructure. Likewise, the services themselves may need to be securely orchestrated;
- Upon successful authentication, **access control** has to be enforced in order to bind all data units exchanged between two players to their respective authenticated identities. This takes usually the form of an authentication procedure being implemented as an *authenticated key-exchange* (AKE) protocol, and all subsequent messages exchanged between the same two players are then integrity protected by the AKE-obtained session key. Various protocols exist for doing so: at the network layer, the *Host Identity Protocol Base Exchange* (HIP BEX) and *Internet Key Exchange* (IKE) are AKE protocols and IPsec is the corresponding secure data transport protocol. At the transport layer, TLS handshake is an AKE protocol for subsequent (D)TLS exchanges. Various service-specific protocols can of course also be used. Eventually, all risks mitigated by integrity protections should rely on specific cryptographically protected access-control schemes;
- In parallel with secure transaction set up and access-control-based integrity protection, protection against internal attacks requires a **coherent arrangement**

of the associated cryptographic primitives which have to be based on an assessment of the attacker profile and capabilities. Many design choices proposes different embodiments that provide different security levels. For example the perfect forward secrecy property is theoretically a more secure one. However, this additional security property would prove worthwhile only for an attacker able to (and interested in) accessing data exchanged in the past (hence possibly obsolete) but that the attacker would nevertheless have stored under an encrypted form. Clearly, most of attacker models and data criticality do not fit within this attack scenario. If one decides to envision it, though, the same attacker capabilities should be assumed for all other risks.

Protection against **internal attacks** is illustrated in the Table 6.5 on DREAD assessment by the reliance on **autonomous security** design choices (DC A.16,17). Classically, only behavioural analysis can allow identifying misbehaviours of an otherwise authorised node. Autonomous security can be instantiated under a wide variety of forms that pertain to the implemented functions in a given IoT infrastructure. Whenever behavioural patterns can be defined, deviations from these patterns can be detected and flagged as suspicious. More generically (and more easily), **logs** should be enabled as a rudimentary form of reactive security. Logs can be generated at various places in the network but will generally be aggregated at server-side, where they will be collected for further uses such as service management (e.g. dimensioning), lawful requirements or billing preparation. However, logging user activity or detecting identifying patterns within it countervents privacy. Autonomous security and privacy are in general mutually contradictory. Pseudonymity can be seen as an intermediary state, although pseudonyms are only worthwhile as long as they can be resolved to real identities at some point in the network. Choosing which scheme to favour is a question of high-level design choice. Diametrically opposed to privacy, non-repudiation plays a specific role that has to be reviewed here. In general, this security service, which ensures that an entity will not be in position of denying having performed a given transaction, is provided at service layer where both signature-based cryptographic primitives and transaction concept become relevant. Although the associated risk (repudiation) is part of the STRIDE classification, service-level non-repudiation was not considered in the previous section, being judged to be pertaining to policies, themselves associated to particular applications. In fact, services for which non-repudiation has to be provided are part of highly specific applications (e.g. inter-bank communications of aggregated banking transactions, or administration of highly-critical assets), which does not qualify them as generic mitigation means.

Finally, it is worth explaining why some identified risks are "not specifically targeted" in IoT-A, with no relevant technology being developed and no design choice being proposed. These non-targeted risks are of two sorts. Some of them are dependent on highly contextual physical parameters. They depend on the particularities of the communication technology that is put in place and, as such, exhibit highly diverse characteristics in terms of involved stakes. Accordingly, the existing mitigations can only be implemented at the physical layer with variable

costs in terms of, for instance, efficiency. The other non-targeted security risks pertain to in-entity security-by-design policies. For example, the protection of a given operating system or the choice to encrypt a user database fit into this category. As such, they cannot be qualified as being typical for the IoT environment.

6.9 Design Choices

6.9.1 Introduction

By following the architectural methodology according to (Rozanski and Woods 2011) it is recommended to apply the architectural perspectives to the views on an architecture in order to design systems that satisfy qualities like high performance, high scalability or interoperability. This step in the architectural methodology is similar to constructing the interrelationships between customer requirements and technical requirements in the 'House of Quality' matrix as applied in the Quality-Function Deployment (Erder and Pureur 2003) introduced in Sect. 6.4.

This section guides an architect by giving design choices for the architectural viewpoints defined in the Reference Architecture in Section 8.2 for each perspective listed in Sect. 8.3. Figure 6.7 illustrates that the perspectives 'Evolution & Interoperability', 'Performance & Scalability', 'Trust, Security & Privacy', and 'Availability & Resilience' are applied to the 'Functional View', the 'Information View' as well as the 'Deployment & Operation View' respectively.

While applying perspectives to views not every view is impacted by the perspectives in the same manner or grade. Rosanski and Woods distinguish between three grades of applicability (high, medium and low) for each perspective to each view. Table 6.6 illustrates the perspective to view applicability as presented in (Rozanski and Woods 2011).

In this section we focus mainly on the perspective and view pairs where the applicability of the perspective to the view is high. According to the Table 6.6 these pairs are the following:

None of the perspectives have a high impact when applied to the Operational View. This is an indicator for not considering the Operational View in the RA (Sect. 8.2) and therefore in this section respectively. The Concurrency View is not being considered in the RA Sect. 8.2 either, thus the applicability to this view, even with a high impact, is not followed up in this section.

Additionally, we do not present design choices for particular platforms (i.e. recommendations for specific hardware and software) as they would give the current status of available platforms at the time of editing this document only, but the recommendations could become obsolete soon after. Software architects are well advised to look for suitable platform solutions while designing their concrete architectures. Platforms that were researched during the project (Magerkurth 2011) are based on the OSGi framework (OSGi 2012). This framework specifies among others how software can be deployed in form of bundles and how the application lifecycle can be controlled remotely. The OSGi framework is a recommended
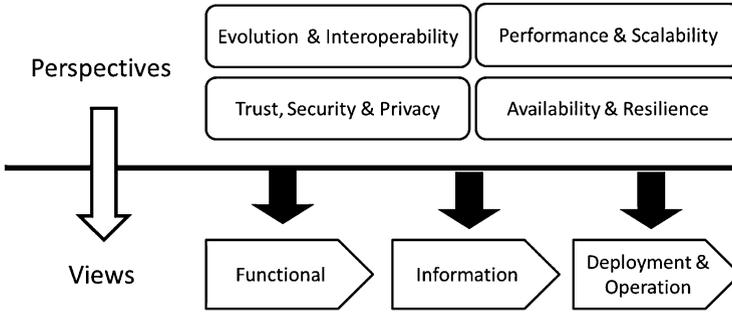
Fig. 6.7 Applying perspectives to views (Rozanski 2011; Fig. 6.4–1)

Table 6.6 Typical view and perspective applicability (Rozanski and Woods 2011)

Perspective / View	Security	Performance & Scalability	Availability & Resilience	Evolution
Functional	Medium	Medium	Low	**High**
Information	Medium	Medium	Low	**High**
Deployment	**High**	**High**	**High**	Low
Operational	Medium	Low	Medium	Low
Concurrency	Low	**High**	Medium	Medium

Table 6.7 Focus on high perspective to view ability

Architectural perspective	Architectural view
Evolution and interoperability	Functional
	Information
Availability and resilience	Deployment
Performance and scalability	Deployment
Trust, security and privacy	Deployment

design choice for the Deployment and Operation View. Based on the experience obtained in the project we recommend OSGi framework as a design choice for the Deployment and Operation Views with hardware platforms that provide support for OSGi. However, OSGi framework is not advisable for very constraint computing platforms.

According to Rozanski/Wood "a tactic is much more general and less constraining than a classical design pattern because it does not mandate a particular software structure but provides general guidance on how to design a particular

aspect of your system" (Rozanski and Woods 2011). Following Rozanski and Wood's definition this section picks up the tactics addressing the architectural perspectives listed in Sect. 8.3 and presents technology agnostic design patterns or other architectural solutions that are suitable to apply the tactics. Architects are then able to either implement the recommended design choices or to look for existing solutions that have implemented those choices.

6.9.2 Design Choices Addressing Evolution and Interoperability

The Evolution perspective addresses the fact that requirements change and software evolves sometimes rapidly. We identified a second, closely related, perspective namely Interoperability which plays a crucial role especially in IoT. The vision of the Internet of Things is still evolving. Many current technologies are not yet mature enough for operational use and there are many more technologies to come in the future. The Evolution and Interoperability Perspective is shown in Sect. 8.3.1. The tactics for evolution and interoperability are the key concepts of the IoT ARM and will be explained in Table 6.8.

Both, the Reference Model and the Reference Architecture are built to be extensible and to enable interoperability between Devices and Services. Therefore the activities listed in Sect. 8.3.1reflect the IoT-A approach in detail:

- **Characterize the evolution needs**: IoT-A has collected stakeholder and also internal requirements reflecting the actual and future needs in IoT systems (see IoT-A 2013);
- **Assess the current ease of evolution**: Also through the stakeholder workshops and in addition the use cases from WP7 and the state of the art analysis from WP1 and all technical work packages, the current status was collected;
- **Consider the evolution trade-offs**: The evolution trade-offs are heavily domain- and application-specific and are not part of the IoT-A work. Those trade-offs must of course be discussed when creating an architecture for a concrete application;
- **Rework the architecture**: The main result of IoT-A are the Reference Model and the Reference Architecture which were designed with interoperability in focus (see Sect. 7.5 and Chap. 8).

Moreover, Rozanski and Woods (2011) also introduce tactics to deal with interoperability and evolution. Here also the IoT-A Reference Model and Reference Architecture adapt the following tactics:

- **Create extensible interfaces**, **Apply design techniques that facilitate change**: IoT-A defines common entities, e.g. the IoT Domain Model, see Sect. 7.3, and entry points, e.g. the IoT Communication Model, see Sect. 7.6, which can be used to create IoT-A compliant systems;

Table 6.8 Tactics addressing evolution and interoperability

Desired quality	The ability of the system to be flexible in the face of the inevitable change that all systems experience after deployment, balanced against the costs of providing such flexibility
Tactics	Create extensible interfaces
	Apply design techniques that facilitate change
	Apply metamodel-based architectural styles
	Build variation points into the software
	Use standard extension points

Table 6.9 Tactics identified as not relevant for evolution and interoperability in IoT Systems

Tactic	Reason
Contain change	Not possible for public IoT-systems, new devices will participate in the systems
Achieve reliable change	Same as above
Preserve development environments	Due to the multiplicity of developers and technology providers, a common development environment will not exist

- **Apply metamodel-based architectural styles**: The IoT-A Reference Model and Reference Architecture define interoperability on architectural level. Especially the Domain Model, see Sect. 7.3, and the IoT Information Model, see Sect. 7.4, as metamodels are open for further extensions;
- **Build variation points into the software**, **Use standard extension points**: By using standardised protocols and gateways, even legacy devices are able to be linked to IoT-A systems.

Design Choices for Interoperability and Evolution cannot be named on this (application and domain independent) level. The IoT Reference Model and Reference Architecture are built with interoperability and evolution as the main drivers. To allow a system to evolve and to react to new technology and new requirements the following general remarks should be kept in mind:

- The IoT-A Reference Architecture is built out of modular blocks to allow changes and additions. When deriving the IoT-A work to a concrete architecture, this modularity and also the loose coupling between those blocks should be kept. This concept is also used in the 'Dispatcher' component (Hyttinen P ed et al. 2013) for the standardized processing of incoming requests without exposing the internal methods and functions;
- Not all of the systems functionality can be defined in advance. Therefore, some additional spaces and extensions points, e.g. for upcoming functionality, should be reserved. This can for example be done in interface definitions or data models, like the reserved bits in the TCP header definition. This allows the designers and architects to update the system and to adapt it to new requirements.

The tactics not considered as relevant are listed in Table 6.9.

Table 6.10 Tactics addressing performance and scalability

Desired quality	The ability of the system to predictably execute within its mandated performance profile and to handle increased processing volumes in the future if required
Tactics	Optimize repeated processing
	Replication
	Prioritize processing
	Distribute processing over time
	Minimize the use of shared resources
	Reuse resources and results
	Partition and parallelize
	Scale up or scale out
	Degrade gracefully
	Use asynchronous processing
	Reduce complexity
	Make design compromises

Table 6.11 Tactics and corresponding design choices for performance and scalability

Tactic	Impact on views		
	Functional	Information	Deployment and operation
Replication	Replication of functional components (DC PS.1)	Replication of gathered Information (DC PS.2)	Replication of instances of Functional Components locally (DC PS.3)
			Replication of instances of functional components in the cloud (DC PS.4)
Prioritize Processing	Functional component offer services for different priorities (DC PS.5)	Information holder for priority information necessary (DC PS.6)	Provide instances of different functional components for different priorities (DC PS.7)
			Priority-aware functional components with priority based processing and networking (DC PS.8)
Partition and parallelize	Multi-thread/multiprogramming aware Functional components (DC PS.9)	Information flow needs to be parallelizable (DC PS.10)	Location-aware deployment of functional components (DC PS.11)
			Deployment of functional components need to be according to data flow (DC PS.12)
Reduce computational complexity	No functional component (DC PS.13)	No Impact	Less functional component deployed (DC PS.15)
	Functional component with reduced capabilities (DC PS.14)		
Distribute processing over time	Design components to schedule processing (DC PS.16)	Information holder for deadline (DC PS.17)	No impact

(continued)

Table 6.11 (continued)

Tactic	Impact on views		
	Functional	Information	Deployment and operation
Minimize the use of shared resources	Design functional components to minimize use of shared resources (DC PS.18)	No impact	Minimize communication distances (DC PS.19) Deployment to minimize use of shared resources (DC PS.20)
Reuse resources and results	History aware functional components (DC PS.21)	Cache results which are likely to be reused (DC PS.22)	Storage of information locally (DC PS.23) Storage of information remotely (DC PS.24) Storage of information local and remotely (DC PS.25)
Scale up or scale out	Design functional Components in a replicable way (DC PS.26) Design function components so that they can use cloud support (DC PS.27)	No impact	Provision of further resources (DC PS.28) Use services in the cloud (DC P.29)
Degrade gracefully	Functional Components need to be able to restart (DC PS.29) Functional components with rollback functionality (DC PS.30)	Support of rollback points (DC PS.31)	Replication of components (DC PS.32) Redundancy of resources (DC PS.33)
Use asynchronous processing	Asynchronous-aware functional component (DC PS.35)	No impact	No impact

6.9.3 Design Choices Addressing Performance and Scalability

Performance and scalability are closely related. In the Internet of Things, with its anticipated billion or trillion nodes both performance and scalability will play a crucial role. In Sect. 8.3.1 the Performance and Scalability Perspective together with a set of tactics are presented. In the following we applied the tactics from the Performance and Scalability Perspective to our Design Choices. We furthermore evaluated their expected impact on the Functional, Information, and Deployment and Operation Views.

Not all tactics are explained in detail in this section. The tactic "Make Design compromises", for example, was omitted, as being too general and as the whole idea of the design choices it to make compromises. Additionally, as performance is something that is very dependent on both architecture and implementation it is

highly advisable to run through the corresponding activities like creating a performance model or conduct practical testing with measurements. The full list of activities are listed in Sect. 8.3.

6.9.3.1 Replication

The functional components (DC PS.1) and the information (DC PS.2) stored can be replicated to increase performance and scalability (DC PS.3). Having a single functional component is often against good scalability. The availability of information depends on the availability of the IoT device. Having instances of functional components and information available remotely (for example, in the cloud) usually increases both scalability and performance (DC PS.4). Nonetheless, in this case one needs to be enough connectivity and bandwidth provided.

6.9.3.2 Prioritize Processing

To be able to prioritize processing the functional components needs to be aware that it might be required to prefer one type of processing over the other. Therefore, the information model needs to be able to provide information that indicates priorities of processes, for instance high, normal, or low.. In terms of deployment the prioritized processing can be done with the help of the network stack (DC PS.7) or there can be different functional components for the different priorities (DC PS.8).

6.9.3.3 Partition and Parallelize

Partition and Parallelize aims towards increase both scalability, as well as, performance by making the functional components aware of multi-threading/multi-programming (DC PS.9). Furthermore the information needs to be partitionable (reduce interdependencies between information) (DC PS.10). The deployment can help a lot in partitioning, as in IoT access to IoT services are often locally distributed. This can be done either location aware (DC PS.11), or based on a data-flow model (DC PS.12).

As an example, the Virtual Entity resolution could be location-oriented, where a resolution server (RS) is responsible for indexing all connected things in a certain geographical area, called indexing scope. A Catalogue server then creates the Catalogue Index of every RS' indexing scope. A resolution request is redirected towards the RS whose indexing scope intersects the search scope of the request. Large-scale IoT systems are expected to have multiple administrative domains that must be handled by a federated resolution infrastructure. Different domains interact with each other by the means of a central domain directory or domain catalogue. Another possibility would be a federated infrastructure, in which Virtual Entities are clustered based on similarity. Dedicated places are in charge of the IoT Services

they offer and provide their descriptions as part of a distributed resolution framework. The framework is scalable and fault tolerant because of distribution.

6.9.3.4 Reduce Computational Complexity

Whenever possible the system can reduce the computational complexity, thus leading to a simpler system which needs less time and often energy. As an example, instead of a complex intrusion detection system, there could either be no intrusion detection at all (DC PS.13) or a less complex security by design (DC PS.14), e.g. a protocol stack with built in threshold-based protection against too many session initiations.

6.9.3.5 Distribute Processing Over Time

To reduce the number of resources needed it is often possible to distribute some processing tasks over time, when their results are not immediately necessary (DC PS.16). In case of hard real-time constrains this might not be always possible, but many system do not need real-time at all, or do only have soft real-time constraints. Distributing processing over time can help preventing the system from scaling or reduce the use of remote (over the web) services.

6.9.3.6 Minimize Used of Shared Resources

In many IoT systems the most scare and most expensive resource is bandwidth, especially in wireless battery powered systems. It is necessary to design the functional components accordingly and especially plan the deployment to avoid bottlenecks on the devices/resources.

6.9.3.7 Reuse Resources and Results

To be able to reuse resources and results the functional components need to be aware of a history for reuse (DC PS.21). The information model needs be aware of such caching mechanisms (DC PS.22). In terms of deployment the history can either be stored locally (DC PS.23), remotely (DC PS.24) or a combination of both (DC PS.25).

If the information history is stored locally (DC PS.23) the information history is stored on the IoT device that has produced the information over time. History information needs to be secured in the same way as the present information to avoid information leaks. If constrained IoT devices are used, then the storage size of information history as well as the information processing performance is limited: Having a local storage place for history information on each IoT Device requires less device performance and less effort to secure the history, but the single

information host is against good scalability. The availability of information history depends on the availability of the IoT device hosting the history.

DC PS.24 describes the case, where the information history is not stored on the IoT Device that has produced the information, but on a different IoT Resource, to which the information is uploaded to. The additional history resource needs to be secured too with either the same S&P policies as the original IoT Resource or different policies. A history resource in the cloud can perform better than IoT devices; the replication of information allows load balancing between history and present information which contributes to better scalability. The Information history still exists when the respective IoT device becomes unavailable.

Furthermore it is possible to combine the two aforementioned approaches (DC PS.25): The information history is stored on the IoT device that has produced the information as well as on a different IoT Resource replicating the information. History information that exceeds the capabilities of the hosting IoT device capabilities can be offloaded to high performance devices. This design choice contributes to high scalability as well as higher performance since the remotely stored history information is a replication of the locally stored information. Replicating information is cheaper to achieve by the device than retrieving 'fresh' information for every replication.

6.9.3.8   Scale Up or Scale Out

Scale up and scale out is one of the traditional ways to ensure scalability. Scale up (also known as vertical scalability) means providing more resources on a single system (DC PS.26/DC PS.28), scale out (also known as horizontal scaling) means providing more computing power by adding resources. In IoT it is usually not that easy to scale up or to scale out. One obvious possibility is, of course, to use cloud support (DC PS.27/DC PS.29). Migration in sensor networks is possible to some extend as well in a heterogeneous network.

6.9.3.9   Degrade Gracefully

Degrade gracefully is a property of a system, which allows it to continue operating properly even in the event of failure in one ore more components. The functional components need to be able to restart either completely (reset) or to rollback to a previous stable state. In case of hardware failures redundancy and replication allow to continue working even when a device/resource fails.

6.9.3.10   Use Asynchronous Processing

Asynchronous processing is usually intrinsic in IoT systems. All functional components should be prepared to do asynchronous calculations and synchronization needs to be planned accordingly.

Table 6.12 Tactics and corresponding design choices for trust

| Tactic | Impact on views | | |
	Functional	Information	Deployment and operation
Harden root of trust	The security policy defines how the root of trust may be accessed. (DC T.1)	No impact	Integration of IoT-A trust and reputation component (DC T.2)
	Secure implementation for protecting a root-of-trust based on hardware implementation (DC T.3)	No impact	Integration of a physically unclonable function (PUF) (DC T.4)
Ensure high quality of data	Protects data integrity and freshness by using a secure network encryption protocol (DC T.5)	Improvement of content dimension and intellectual dimension (DC T.6)	Integration of a secure network encryption protocol (DC T.7)
Infrastructural trust and reputation agents	Collects user reputation scores and calculates service trust levels (DC T.8)	Service description should include relevant aspects for what concerns trust evaluation (DC T.9)	Integration of IoT-A trust and reputation (DC T.10)
	Web of Trust system to establish the authenticity of the binding between a public key and its owner. (DC T.11)	No impact	Decentralized trust model (DC T.12)
Provide high system integrity	Evaluation of trust based on reputation (DC T.13)	No impact	Integration of a reputation framework for high integrity sensor networks (RFSN) (DC T.14)
Avoid leap of faith	Utilizes one-way hash chain to provide effective and efficient authentication (DC T.15)	No impact	Usage of lightweight authentication protocol (DC T.16)

6.9.4   Design Choices Addressing Trust

In Sect. 8.3.3.1 the Trust Perspective together with a set of tactics is presented.

In Table 6.12 all tactics together with their Design Choices are listed. A detailed description for each tactic follows the table.

6.9.4.1 Harden Root of Trust

The root-of-trust is the core component upon which the trust policy is based. The notion of a root-of-trust exists at multiple abstraction levels in a system, and can be software (less secure) as well as hardware (higher security). As an example for hardware realisation is RFID. The tags can be used to support anti-counterfeiting by using a security protocol based on public key cryptography. In this case their root-of-trust is based on a *Physically Unclonable Device* (PUF) (Verbauwhede and Schaumont 2007).

6.9.4.2 Ensure High Quality of Data

Information quality is improved in the technical dimension (e.g. timeliness and sampling). The suite of security protocols (SPINS) guarantees that an attack does not affect the remainder nodes in the network and thus preserves data integrity and freshness. In the context of the Information view it can be stated that data containing information is improved in terms of content dimension (e.g. accuracy or completeness) and intellectual dimension (e.g. reputation and trust). To reach this level of security a secure network encryption protocol must be implemented (Perrig et al. 2002).

6.9.4.3 Infrastructural Trust and Reputation Agents

The tactic "Infrastructural Trust and Reputation Agents for scalability" describes the presence of a Trust and Reputation component FC (Sect. 7.7.1). This impacts the information view as a Service Description should include relevant aspects for what concerns trust evaluation (type of deployment, tamper-proof features of hosting devices, authentication and authorization algorithms, etc. In case of peripheral devices the security of the deployment should be evaluated and asserted in the subject description. Furthermore the web of trust concept to establish the authenticity of the binding between a public key and its owner can be established. Its decentralized trust model is an alternative to the centralized trust model of a *Public Key Infrastructure* (PKI), which relies exclusively on a certificate authority (or a hierarchy of such).

6.9.4.4 Provide High System Integrity

To provide high system integrity the integration of Reputation framework for high integrity sensor networks (RFSN) can be considered (Ganeriwal and Srivastava 2004). It is capable of evaluating trust based on reputation and to act accordingly. Furthermore second hand information (experiences of other parties, e.g. nodes) about devices can be considered. It might be augmented by a Trust management system which calculates Trust values as a function of availability and packet forwarding.

Table 6.13 Omitted tactics for the trust perspective

Tactic	Reason
Ensure physical security and implement tampering detection	Pervasive deployment of IoT devices makes such devices accessible to malicious users
Consider device security in the global system design	Devices that are not tamper-proof can be compromised. Although this aspect is related to the deployment view, it has impacts on the design of the overall system and trust evaluation
Consider the impact of security/performance trade-offs on trust	This must be evaluated for each use case during the design phase by means of tests such as simulation. For that reason, no DC can be proposed
Use security imprinting	Out of scope for IoT-A since devices are not covered in the IoT Reference Architecture
Balance privacy vs. non-repudiation (accountability)	If system requirements include non-repudiation, these will necessarily impact the privacy feature of the designed system. Privacy can be granted by using an Identity Management. This component, run by a third party is trusted for what concerns both privacy protection and ability to track back malicious actions

6.9.4.5   Avoid Leap of Faith

The avoidance of leap of faith increases the overall security; however, it might limit the communication between certain parties as strong authentication is not feasible in each case (e.g. constrained devices). From a functional point of view one option can be a one-way hash chain to provide effective and efficient authentication. This feature can be implemented by using a Lightweight Authentication protocol (Lu and Pooch 2005).

For most of the tactics a design choice proposal is given, however for different reasons it is not possible to provide appropriate design choices for all tactics. The tactics not considered are presented in Table 6.13 with reasons for the omission.

6.9.5   Design Choices Addressing Security

In Sect. 8.3.3.2 the Security Perspective together with a set of tactics is presented. The Design Choices addressing security are presented in Table 6.14 showing the impact on architectural views by applying tactics relevant for security concerns.

6.9.5.1   Subject Authentication

For subject authentication two options are presented here. The first is the authentication over an encrypted channel while the other one is a crypto-based authentication solution over an open channel. The former uses the IoT-A Authentication FC (Sect. 7.7.2) while for the ladder a peer-to-peer communication is realised over an insecure channel.

Table 6.14 Tactics and corresponding design choices for security

Tactic	Impact on views		
	Functional	Information	Deployment and operation
Subject authentication	Authentication over encrypted channel (DC S.1)	No impact	Integration of IoT-A authentication FC (DC S.2)
	Crypto-based authentication over open channel (DC S.3)	No impact	Peer-to-peer authenticated communications over an insecure channel must be possible (DC S.4)
Use access policies	Policy-based service access (DC S.5)	Stored information must be managed in a way to support access control mechanisms (DC S.6)	IoT-A Authorisation FC component (DC S.7)
	Unrestricted access to service (DC S.8)	Stored information is not protected (DC S.9)	No impact
Secure communication infrastructure	End-to-end encryption (DC S.10)	Information transmission channel between device and application is secured (DC S.11)	IoT-A end to end communication FC, Network communication FC and key exchange and management FC (DC S.12)
	Hop-to-hop encryption (DC S.13)	Information transmission channel between device and application is secured (DC S.14)	IoT-A hop to hop communication FC, network communication FC and key exchange and management FC (DC S.15) (Sect. 7.7.2)
	Cryptographic protocols ensuring confidentiality, integrity, authentication of subjects (DC S.16)	Communication channel between two subjects is secured (DC S.17)	End-to-end security protocol to ensure wireless communication security (DC S.18)
Secure peripheral networks (link layer security, secure routing)	Link-layer encryption and authentication, multipath routing (DC S.19)	No impact	Integration of secure routing protocols in the Network Communication component (DC S.20)

6.9.5.2 Use Access Policies

The tactic of using access policies is a crucial aspect in IoT. Two main functional principles can be distinguished. The policy-based service access uses access control mechanisms to manage to access to information. Therefore the information must be managed accordingly so that it supports the used mechanism. This option can be realised by using the IoT-A Authorisation FC component (Sect. 7.7.2). The other possibility is to grant unrestricted access to services. This should be only done in those cases in which data security is not relevant.

6.9.5.3 Secure Communication Infrastructure

Securing the communication infrastructure focuses on delivering a secure and robust environment for the transmission of critical data. This can be obtained by using end-to-end or hop-to-hop encryption. In both cases the information transmission channel in which the information flow from a device to an application through an IoT service happens is completely secured. The end-to-end encryption uses therefore the IoT-A End to End Communication FC and Key Exchange and Management FC. Furthermore the Network Communication FC, which takes care of enabling communication between networks through Locators (addressing) and ID Resolution, is necessary (Sect. 7.7.2). For the hop-to-hop encryption the only difference is the usage of the IoT-A hop-to-hop Communication FC. For wireless communication security the implementation of an end-to-end security protocol which ensures confidentiality, integrity and authentication of subjects can also be considered (Perrig et al. 2004).

6.9.5.4 Secure Peripheral Networks (Link Layer Security, Secure Routing)

To secure peripheral networks a link-layer encryption and authentication combined with a multipath routing can be considered. This requires the integration of secure routing protocols in the Network Communication component (Karlof and Wagner 2003).

For most of the tactics a design choice proposal is given, however for different reasons it is not possible to provide appropriate design choices for all tactics. The tactics not considered are presented in Table 6.15 with reasons for the omission.

6.9.6 Design Choices Addressing Privacy

In Sect. 8.3.3.3 the Security Perspective together with a set of tactics is presented. The Design Choices addressing Privacy are presented in Table 6.16 showing the impact on architectural views by applying tactics relevant for Privacy concerns.

Table 6.15 Omitted tactics for the Security Perspective

Tactic	Reason
Harden infrastructural functional components	Infrastructural functional components are critical components that can compromise the whole system if compromised
Avoid wherever possible wireless communication	Wireless communication generally uses a shared medium for communication which, in turn, allows easy interception of link layer communication
Physically protect peripheral devices	Pervasive deployment of IoT devices makes such devices accessible to malicious users. While how to protect these devices is outside the scope of the IoT Reference Architecture (devices not covered!), this vulnerability must be taken into account in secure designs
Avoid OTA device management	No DC proposal possible as most of the devices connected in IoT must be managed over the air if at all possible

Table 6.16 Tactics and corresponding design choices for privacy

	Impact on views		
Tactic	Functional	Information	Deployment and operation
Pseudonymisation	Creation of a fictional identity (root identity, secondary identity, pseudonym or group identity) (DC P.1)	No impact	Integration of IoT-A identity management FC (DC P.2)
Avoid transmitting identifiers in clear	Encryption mechanisms for wireless connections (DC P.3)	No impact	Integration of a wireless security algorithm (DC P.4)
Minimize unauthorized access to implicit information	Access control management (DC P.5)	Stored Information must be managed in a way to support access control mechanisms (DC P.6)	IoT-A authorisation FC (DC P.7)
	Enablement of a scalable and secure key distribution between communicating subjects (DC P.8)	No impact	Encrypt communication with Resolution Components and with Services (e.g. KEM FC) (DC P.9)
Enable the user to control the privacy settings	Addresses privacy questions so that a user can operate anonymously (DC P.10)	No impact	IoT-A identity management FC (DC P.11)
Privacy-aware identification	Authentication of the responding host, the initiating host can stay anonymous (DC P.12)	No impact	Requires TLS and DTLS support (DC P.13)

6.9.6.1   Pseudonymisation

The tactic "Pseudonymisation" refers to a procedure by which fields that enable identification of a user within a data record or subject are replaced by one or more artificial identifiers. The purpose is to render the subject less identifiable and this way lower IoT user (e.g. customer or patient) objections to its use. This is functionally implemented by the creation of a fictional identity (e.g. root identity, secondary identity, pseudonym, or group identity) and can be realised by integrating the IoT-A Identity Management FC (Sect. 7.7.3).

6.9.6.2   Avoid Transmitting Identifiers in Clear

The transmission of identifiers in clear should be avoided in general. In a WSN, a base station is not only in charge of collecting and analysing data, but also used as the gateway connecting the WSN with outside wireless or wired network. In order to have a defence against local adversaries, the location information or identifier of the base station is sent in clear in many protocols. This information must be hidden from an eavesdropper, which can be done by traditional cryptographic techniques (encryption). One option for encrypting wireless connections is the integration of a wireless security algorithm proposed by (Peris-Lopez et al. 2007).

6.9.6.3   Minimize Unauthorized Access to Implicit Information

Unauthorized access to implicit information (e.g. deriving location information from service access requests) must be restricted at all events. Access control management as well as the enablement of a scalable and secure key distribution between communication subjects can be considered to achieve this objective. In the former case the information stored must be managed in a way so that the access control mechanism is supported. For deployment of this function the IoT-A Authorisation FC can be considered. For the secure key distribution the resolution components should be augmented by a Key Exchange Management component such as the one from IoT-A.

6.9.6.4   Enable the User to Control the Privacy Settings

Users should be given the opportunity to control their privacy settings. Hence, one option is the control of acting anonymously. This function can be realised by integrating the IoT-A Identity Management FC which creates a fictional identity (root identity, secondary identity, pseudonym, or group identity) along with the related security credentials for users and services to use during the authentication process.

Table 6.17 Omitted tactics for the privacy perspective

Tactic	Reason
Validate against requirements	Too general, no DC proposal possible
Consider the impact of security/performance trade-offs on privacy	This must be evaluated for each use case during the design phase. For that reason, no DC can be proposed
Balance privacy vs. non-repudiation (accountability)	This must be evaluated for each use case during the design phase. For that reason, no DC can be proposed

6.9.6.5 Privacy-Aware Identification

In human-to-thing and thing-to-thing interactions, privacy-aware identifiers might be used to prevent unauthorized user tracking. Similarly, authentication can be used to prove membership of a group without revealing unnecessary information about an individual. *Transport Layer Security* (TLS) and *Datagram Transport Layer Security* (DTLS) provide the option of only authenticating the responding host. This way, the initiating host can stay anonymous (Heer et al. 2011).

For most of the tactics a design choice proposal is given, however for different reasons it is not possible to provide appropriate design choices for all tactics. The tactics not considered are presented in Table 6.17 with reasons for the omission:

6.9.7 Design Choices Addressing Availability and Resilience

The Chapter in this document concerned with the Availability and Resilience Perspective (Sect. 8.3.4) lists tactics addressing the desired quality of the system to be designed as shown in Table 6.18.

In this Section design choices are presented that apply most of the tactics listed in Table 6.19. The tactics not considered here are given at the end of this Section with an explanation why they have been omitted. Table 6.19 presents for each tactic one or more architectural design choices together with their impact on the architectural views introduced in Chap. 8.

6.9.7.1 Use High Availability Clustering

For design choice 'VE Resolution location-oriented (DC A.1)' a resolution server (RS) is responsible for indexing all connected things in a certain geographical area, called indexing scope. A Catalogue server then creates the Catalogue Index of every RS' indexing scope. A resolution request is redirected towards the RS whose indexing scope intersects the search scope of the request. Large-scale IoT systems are expected to have multiple administrative domains that must be handled by a federated resolution infrastructure. Different domains interact with each other by the means of a central domain directory or domain catalogue. Communication between framework domains needs to be secured. The framework performs faster through a divided search space. Indexing scope can be adjusted according to usage load. The framework scales by adding more RSs. With this approach it is

Table 6.18 Tactics addressing Availability and Resilience

Desired quality	The ability of the system to be fully or partly operational as and when required and to effectively handle failures that could affect system availability
Tactics	Select fault-tolerant hardware
	Use high-availability clustering and load balancing
	Log transactions
	Apply software availability solutions
	Select or create fault-tolerant software
	Design for failure
	Allow for component replication
	Relax transactional consistency
	Identify backup and disaster recovery solution

impossible to retrieve things based on identifiers. Fault tolerance is achieved through data distribution and index data replication. The central domain directory is potential single point of failure. There is no theoretical limit on indexed things, but indexing scope is bound to geographic location (De 2012).

In design choice 'VE Resolution domain-oriented (DC A.2)' a domain-oriented VE Resolution approach organises the resolution framework in hierarchically organised domains similar to Domain Name System (DNS). The hierarchy is built according to the hierarchy of things captured by Virtual Entities from higher granularity to lower granularity, e.g. country → city → district → building → room. The resolution framework performs faster than an unclustered resolution solution through divided search space; its complexity is of $O(log\ n)$ in best case, and $O(n)$ in worst case, where n is the number of VEs hosted by the resolution framework. Load balancing is supported through replication, and a Resource can be member of different domains at a time. Fault tolerance is supported through distribution and redundancy; the framework evolves with the number of things connected (De 2012).

For design choice 'VE Resolution Semantic Web-oriented (DC A.3)' Semantic Web technologies are used to annotate Virtual Entity descriptions in a way machines can interpret them. This overcomes the need for exact syntactic match-making between resolution request and search terms in the resolution infrastructure. The search space of the resolution infrastructure is indexed by an unsupervised machine-learning technique and clustered through latent factors derived from the learning. This design is independent from the deployment of the resolution infra-structure. Distribution and replication is supported by this approach, but depends on implementation on how it is done. Semantic interoperability is achieved through shared ontologies, after extending ontologies the training model needs to be updated (De 2012).

A peer-to-peer infrastructure will maintain no centralised servers in design choice 'VE Resolution Peer-to-Peer-oriented (DC A.4)', all data is distributed in the network along with sophisticated retrieval and routing mechanisms. There are several approaches on how to distribute the data (pure, centralised indexing server, distributed hash tables). The latter approach is the recommended one for IoT

Table 6.19 Design Choices addressing availability and resilience

Tactic	Impact on views		
	Functional	Information	Deployment and operation
Use high availability clustering	VE resolution location-oriented (DC A.1) (De 2012)	Information model requires data type for defining scope for location of interest	VE resolution instances for each location cluster
	VE resolution domain-oriented (DC A.2) (De 2012)	Information model needs data type for defining types of resources	Resolution framework is organised hierarchically
	VE resolution semantic web-oriented (DC A.3) (De 2012)	Information model needs to be encoded according to semantic web standards	Search space of resolution framework needs to be indexed by certain machine-learning technique
	VE resolution peer-to-peer-oriented (DC A.4) (De 2012)	No impact	No centralised server needed
Use load balancing	Requires function that monitors load of components and triggers load balancing algorithm	Component descriptions need metric to measure current work load and defined load limits	Scaling out approach additional clones of components need to be available (DC A.5)
Log transactions	Apply circular logging strategy (IBM 2012) (DC A.6)	Information model needs concepts for transactions with unique identifiers	Storage for transaction logs needed
	Apply archive logging strategy (IBM 2012) (DC A.7)	Like above, but transactions need to be marked as either active or inactive additionally	External storage needed for large logs
Design for failure	Provide functionality to reserve spare resources and replace failed ones (Newtelligence 2012) (DC A.8)	No impact	Spare resources are kept on hold until an operating resource needs to be replaced, requires higher amount of resources
	Prefer service choreography FC over service orchestration FC (Sect. 8.2.2.3)	Identifiers in information model need to be unique for clones of FCs and across distributed system	No single FC or centralised FC (DC A.9)
	Provide FC that monitors latency (DC A.10)	Provide means to specify latency and timeout limits	No impact

Allow for component replication	Provide FC that implements state-machine (active) replication (Wikipedia 2013d) (DC A.11)	FC needs to be modelled as state-machine	To support F failures you must have at least 2F+1 replicas of the component
	Provide FC that implement transactional replication (Microsoft 2013) (DC A.12)	Also incremental changes of information can be replicated, leads to inconsistency, though, if not completed	Good performance, near real-time replication possible
	Provide FC that implements virtual synchrony (Wikipedia 2013e) (DC A.13)	Make replicated information indistinguishable non-replicated information	Very high performance
Relax transactional consistency (DC A.14)	Requires conflict resolution functionality	No impact	Needs more resources through replication
Identify backup and disaster recovery strategy — Preventive measures (DC A.15)	Requires data-replication functionality	Requires consistency among replicated data	Replicated and archived data are stored off-site in the cloud
Detective measures (DC A.16)	Requires monitoring of indicators for disastrous events	Requires modelling of disastrous events to be looked for and propagation of those	Disaster monitoring needs to be operated independently of components to be monitored
Corrective measures (DC A.17)	Requires restoring of previously back-upped configurations	Requires storage of configuration history	Disaster recovery needs to be operated independently of components to be recovered

Resolution infrastructures. Resolution requests result in traffic complexity of $O(n)$ in worst case and $O(log\ n)$ in best case, where n is the number of VEs managed by the resolution framework. The framework is stable and robust through distribution and redundancy (De 2012).

6.9.7.2 Load Balancing

The 'Scale out approach (DC A.5)' monitors the load of FCs during runtime and triggers offloading tasks to another less busy instance of the respective FC to avoid the FC being overloaded and therefore becoming a performance bottleneck or even out of function. The decision at what limit an FC is considered to be critically busy and to trigger off-loading to another instance is application specific, but the information model needs to provide some metric to specify those parameters for FCs.

Logging Transactions

'Circular Logging (DC A.6)' is a strategy that leads to overwriting old data when designated size of log is reached (IBM 2012). This approach does not support incremental backup strategy. Transactions need to be logged with unique id and status of their completion, indicating which functions need redoing and which need undoing. Apply this Design Choice if storage space for logs is restricted. This strategy provides better performance compared to archive logging.

   'Archive Logging (DC A.7)' keeps a complete archive of all transactions (IBM 2012). Recent transactions need to be flagged as active, older transactions as inactive. The archived logs grow over time so that external storage is needed on constraint devices. This strategy adds functionality for retrieving the external archive also for rollback and restore.

Design for Failure

The overall tactic can be further divided into more specific tactics that are presented as design choices here. The first sub-tactic is 'Acquiring more resources than needed and replace failed ones (DC A.8)'. By applying this tactic more resources are allocated for task execution than normally required. Besides allocating the resources essentially necessary spare resources are reserved that could execute the same task as the essential ones but are kept on hold. This is a precaution in case a resource fails during runtime and a spare resource can take over the task of the one that failed. Resource in this sense includes all computational resources, network resources and IoT Resources, meaning all FCs in the ARM. A typical FG that implements resource reservation is Service Organisation that is responsible for allocating IoT Services to service requests (see Sect. 8.2.2.3). Applying this tactic requires a higher number of resources essentially required.

   Another approach is to aim at having 'No FC or centralised FCs (DC A.9)'. The goal is to develop designs that avoid single points of failure, like centralised FCs or

FCs with just one instance. If a single FC fails no other instance was able to replace its functionality. By applying this tactic more than one instance of FCs are provided by the system so that their functionality can still be assured in case one instance becomes unavailable. For Service Organisation FG the decentralised Service Choreography FC can be preferred over Service Orchestration which requires a central orchestration engine (see Sect. 8.2.2.3). The decentralised choreography approach reduces the risk for a single point of failure.

To apply the design choice 'Treat Long Latency as potential failure (DC A.10)' the system design provides an FC that treats any long latency as a potential failure. For instance the round-trip-time for request-response-protocols is measured and a deadline is set as acceptable. After the deadline has passed the system treats the behaviour as potential failure and reacts in an appropriate manner, e.g., by querying another instance of the same FC.

Allowing Component Replication

The design choice 'State-machine (active) replication (DC A.11)' allows detection of faults by replicating service requests and comparing the service results to each other. If all results are identical no fault is assumed, if they are different it still needs to be analysed which of the results is faulty and which is correct (Wikipedia 2013d). To apply this technique some replication functionality needs to be implemented that multiplies the request to different instances of FCs. To assure fault detection 2F+1 replicas of the tested FC need to be held where F is the number of faults to be detected. The fault detection algorithm requires the tested FC to be modelled as state-machine.

'Transactional replication (DC A.12)' is used in server-to-server environments typically, in which incremental information changes need to be propagated to subscribers in nearly real-time (Microsoft 2013).

The choice 'Virtual synchrony (DC A.13)' is especially suitable for systems in which information evolves extremely rapidly. Applications are executed in process groups and the processes within the group update each other about execution progress by sending state updates. Implementing this technique requires functionality to join process groups, register event handler and send multicasts to group members. Consistency among information replicas can be achieved easily, thus virtual synchrony is suitable for systems with high evolution of information (Wikipedia 2013e).

Relaxing Transactional Consistency

To follow this tactic the 'BASE architecture (DC A.14)' can be applied. The 'BASE (**B**asically **A**vailable, **S**oft-state, **E**ventually consistent) architecture' is applicable in systems supporting distributed transactions with optimistic replication strategy. In this approach replicas of information are sent through a distributed system via transactions and 'eventual consistency' among the replicas is achieved by either the update reaches the replica or the replica retires from service (Wikipedia 2013f). BASE requires some conflict resolution functionality and additional system

resources in order to find failure in transactions. The approach is applicable for high performance designs.

Backup and Disaster Recovery Strategy

The following design choices should not be seen as alternative choices to apply one tactic; the three choices are rather three controls that can help to specify a disaster recovery plan for the system to be designed (Georgetown 2013). Therefore all three choices can be applied alongside.

The choice 'Preventive measures (DC A.15)' is aimed at preventing disastrous events, like data-loss, from occurring. To achieve this data is replicated to have identical copies in reserve in case the original data gets lost. Consistency among the data replicas needs to be assured by the design. To minimise risks the replicas are better stored at different locations that the original data, preferably in the cloud.

'Detective measures (DC A.16)' aim at detecting or discovering unwanted events by monitoring indicators for unwanted events, like measured values that exceed a certain range. This strategy requires an Information Model of those unwanted events together with their indicators that are used to detect the unwanted event. The event detection should be operated independent of the subsystem that is monitored to make sure the unwanted events can be detected.

The design choice 'Corrective measures (DC A.17)' is aimed at correcting or restoring systems after disastrous events have occurred. Assuming the previous two choices have been implemented, meaning the preventive methods have been applied and the disastrous event has been detected correctly, the system can be restored to working order again. Backups of system configurations that have worked correctly before are restored. A configuration history (Sect. 8.2.2.8) provides the functionality needed for restoring working configurations. The system correction process needs to be operated independently of the system to be restored.

Some of the tactics listed in Sect. 8.3 are not considered here because they are too specific to particular implementations:

- Select fault-tolerant hardware;
- Apply software availability solutions;
- Select or create fault-tolerant software.

6.9.8 Design Choices Conclusion

This section has presented design choices for architects who are driven by requirements for system quality capabilities like performance and scalability, evolution and interoperability, availability and resilience as well as aspects concerning trust, security, and privacy. An architect is guided by the presented design choices in supporting the targeted system quality attributes. In cases where the recommended design choice is one developed during the IoT-A project a reference is given where an architect can find more detailed information about the respective design choice.

The design choices listed in this section are as generic as possible addressing capabilities that are agnostic of particular functional requirements. The architect is still left with the choice which system capabilities are the most important ones for the system to be specified. In general trade-offs need to be made between for instance security and performance since security always involves more data and communication overhead that needs to be processed.

The optimal selection of design choices is dependent on the actual use case and therefore a one-fits-all complete solution cannot be given in this section. It rather needs to be made by architects according to their functional requirements which are not known in the context of this document.

What this document can provide instead is an example for a concrete architecture that is designed according to a sample use case. Architects shall find useful hints for applying the ARM to concrete architectures including a selection of appropriate design choices presents in this section. The sample concrete architecture is described in Chap. 11.

Chapter 7
IoT Reference Model

**Martin Bauer, Nicola Bui, Jourik De Loof, Carsten Magerkurth,
Andreas Nettsträter, Julinda Stefa, and Joachim W. Walewski**

7.1   Introduction

The first major contribution of the *IoT Architectural Reference Model* (IoT ARM) is
the IoT Reference Model itself. Besides models, the IoT Reference Model provides
the concepts and definitions on which IoT architectures can be built. This

M. Bauer (✉)
NEC Laboratories Europe, Software & Services Research Division, NEC Europe Ltd.,
Kurfürsten-Anlage 36,
Heidelberg 69115, Germany
e-mail: Martin.Bauer@neclab.eu; www.nw.neclab.eu

N. Bui
Consorzio Ferrara Ricerche, Via Savonarola 9, Ferrara 44122, Italy
e-mail: buincl@unife.it

J. De Loof
Alcatel-Lucent Bell N.V., Copernicuslaan 50, Antwerpen 2018, Belgium
e-mail: jourik.de_loof@alcatel-lucent.com

C. Magerkurth
SAP AG, Dietmar-Hopp-Allee 16, Walldorf 69190, Germany
e-mail: carsten.magerkurth@sap.com

A. Nettsträter
Fraunhofer Institute for Material Flow and Logistics IML, Joseph-von-Fraunhofer Str. 2-4,
Dortmund 44227, Germany
e-mail: andreas.nettstraetter@iml.fraunhofer.de

J. Stefa
Università Sapienza di Roma, P.le Aldo Moro 5, Rome 00185, Italy
e-mail: stefa@di.uniroma1.it

J.W. Walewski
Siemens AG, Otto-Hahn-Ring 6, Munich 81739, Germany
e-mail: joachim.walewski@siemens.com; www.siemens.com

Chapter introduces the IoT Reference Model as a precondition for working with the Reference Architecture that is introduced in Chap. 8.

The Reference Model consists of several sub-models that set the scope for the IoT design space and that address architectural views and perspectives discussed in Chap. 8. As already stated above, the primary and thus the key model is the IoT Domain Model, which describes all the concepts that are relevant in the Internet of Things. All other models and the IoT Reference Architecture are based on the concepts introduced in the IoT Domain Model. While certain models, such as the IoT Communication Model and the IoT Trust, Security, and Privacy Model might be less critical in certain application scenarios, the IoT Domain Model is mandatory for all usages of the IoT ARM. Therefore, it is advised to read Sect. 7.1.3 carefully, and at least to follow the information given in the Sect. 7.1.2 in order to get an overview of the different sub-models of the IoT Domain Model and how they relate to each other. Depending on the individual application of the IoT Domain Model, the Subsequent sections in this chapter provides details about the other models.

Next, we explain, who the sub-models in the IoT Reference Model relate and link to each other, and how they form an integrated reference model.

7.2   Interaction of All Sub-Models

The IoT Reference Model aims at establishing a common grounding and a common language for IoT architectures and IoT systems. It consists of the sub-models shown in Fig. 7.1, which we explain below. The yellow arrows show how concepts and aspects of one model are used as the basis for another.

The foundation of the IoT Reference Model is the IoT Domain Model, which introduces the main concepts of the Internet of Things like Devices, IoT Services and *Virtual Entities* (VE), and it also introduces relations between these concepts. The abstraction level of the IoT Domain Model has been chosen in such a way that its concepts are independent of specific technologies and use-cases. The idea is that these concepts are not expected to change much over the next decades or longer.

Based on the IoT Domain Model, the IoT Information Model has been developed. It defines the structure (e.g. relations, attributes) of IoT related information in an IoT system on a conceptual level without discussing how it would be represented. The information pertaining to those concepts of the IoT Domain Model is modelled, which is explicitly gathered, stored and processed in an IoT system, e.g. information about Devices, IoT Services and Virtual Entities.

The IoT Functional Model identifies groups of functionalities, of which most are grounded in key concepts of the IoT Domain Model. A number of these *Functionality Groups* (FG) build on each other, following the relations identified in the IoT Domain Model. The Functionality Groups provide the functionalities for interacting with the instances of these concepts or managing the information related to the concepts, e.g. information about Virtual Entities or descriptions of IoT Services. The functionalities of the FGs that manage information use the IoT Information Model as the basis for structuring their information.
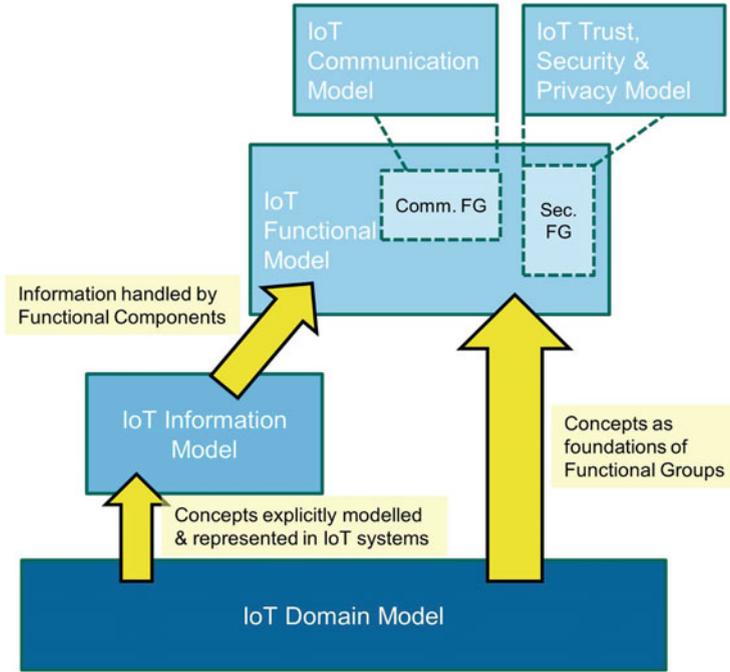
Fig. 7.1 Interaction of all sub-models in the IoT Reference Model. The sub-models are explained in the text body

A key functionality in any distributed computer system is the communication between the different components. One of the characteristics of IoT systems is often the heterogeneity of communication technologies employed, which often is a direct reflection of the complex needs such systems have to meet. The IoT Communication Model introduces concepts for handling the complexity of communication in heterogeneous IoT environments. Communication also constitutes one FG in the IoT Functional Model.

Finally, *Trust*, *Security and Privacy* (TSP) are important in typical IoT use-case scenarios. Therefore, the relevant functionalities and their interdependencies and interactions are introduced in the IoT TSP Model. As in the case of communication, security constitutes one FG in the Functional Model.

7.3 Domain Model

7.3.1 Definition and Purpose

The IoT-A project defines a domain model as a description of concepts belonging to a particular area of interest. The domain model also defines basic attributes of these

concepts, such as name and identifier. Furthermore, the domain model defines relationships between concepts, for instance "*Services* expose *Resources*". Domain models also help to facilitate the exchange of data between domains (The Consultative Committee 2006). Besides this official definition, and looking at our interpretation of it, our domain model also provides a common lexicon and taxonomy of the IoT domain (Muller 2008). The terminology definitions of IoT-A are provided online (Sect. 6.7).

The main purpose of a domain model is to generate a common understanding of the target domain in question. Such a common understanding is important, not just project-internally, but also for the scientific discourse. Only with a common understanding of the main concepts it becomes possible to argue about architectural solutions and to evaluate them. As has been pointed out in literature, the IoT domain suffers already from an inconsistent usage and understanding of the meaning of many central terms (Haller 2010).

The domain model is an important part of any reference model since it includes a definition of the main abstract concepts (abstractions), their responsibilities, and their relationships. Regarding the level of detail, the Domain Model should separate out what does not vary much from what does. For example, in the IoT domain, the device concept will likely remain relevant in the future, even if the types of devices used will change over time and/or vary depending on the application context. For instance, there are many technologies to identify objects: RFID, bar codes, image recognition etc. But which of these will still be in use 20 years from now? And which is the best-suited technology for a particular application? Since no one has the answers to such and related questions, the IoT Domain Model does not include particular technologies, but rather abstractions thereof.

Before we discuss the main abstractions and relationships of the IoT Domain Model in detail, let us go back to our recurring example that we introduced in Sect. 4.2 in order to get an understanding of what it means to formulate central concepts of a use case with the help of the IoT Domain Model.

Figure 7.2 shows an instance diagram of central aspects of the use case scene in Sect. 4.2. This example was cast in the language of the IoT Domain Model and then illustrated by use of UML. Information about UML can be found elsewhere in the literature (Fowler 2003) or by searching for terms such as "UML tutorial" on the web.

As we can see in Fig. 7.2, the important entities that are relevant for our use case are depicted with blocks of different colours. For instance, there is our truck driver "Ted" represented by as a yellow box (viz. instance), and the temperature sensor (that triggers an alarm after Ted had turned off the engine of the truck) is represented as a blue instance. Already at this stage we can easily deduct that there is some colour-coding involved that reflects an aspect of the respective entity. What these colours exactly stand for is discussed in detail in the next Sections. There is also a categorisation in textual form, as the entity name that we know from our recurring example is succeeded by an entity category such as Sensor in the case of the humidity or temperature sensors and *Human User* in the case of Ted. What these entity categories mean and how they relate to each other is discussed in detail in the next sections.
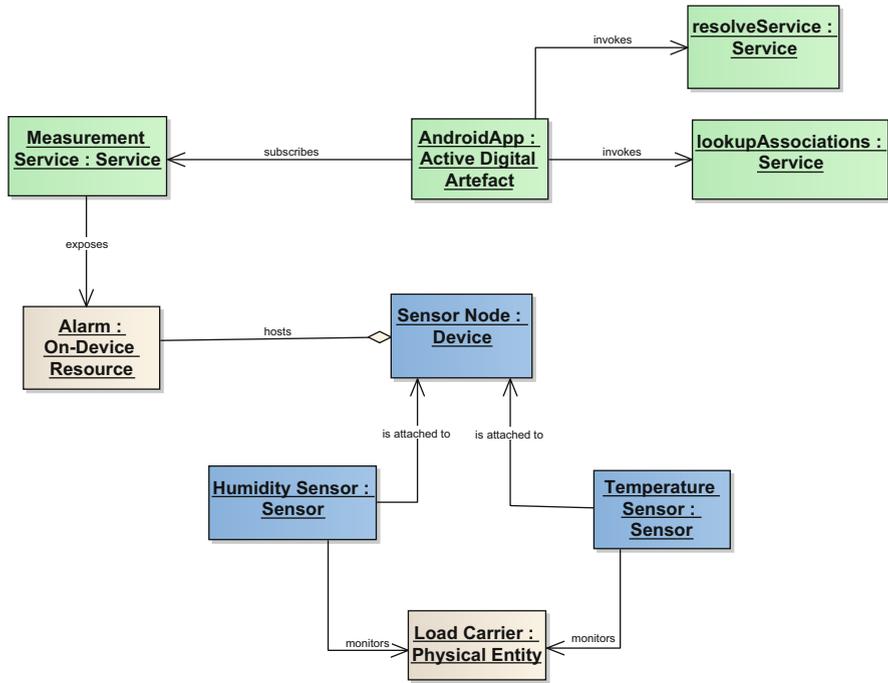
Fig. 7.2 Example, instantiated IoT Domain Model for the Red Thread Example (see Sect. 4.2)

In addition to the coloured boxes, the diagram also shows arrows with verbs that connect the boxes. If we look very closely to the arrows, we see that they have different terminators such as diamond shapes or traditional arrow shapes. These shapes illustrate different kinds of relationships between the objects that are connected by them. In a similar way as the category names and the colour coding of the objects are related to each other, the verbs indicate information about the relationships shown with the arrows. These are all concepts of the UML notation that will be discussed in the next section.

Even without understanding all of the concepts in detail, we can already understand that the IoT Domain Model helps us structuring an application scenario. We can use a concise graphical representation to show that for instance Ted, our truck driver, is a Human User that uses an Android application in order to subscribe to an Alarm service. This Android Application is an *Active Digital Artefact* (*ADA*). We do not yet know what this exactly means, but as the reader will progresses through this document and possibly other documents that make use of the IoT Domain Model, Active Digital Artefacts will come up again and again. By providing a standardised vocabulary for naming things that relate to the same abstract concepts, we facilitate and streamline communication of the IoT ARM users.

While several other parts of the IoT Reference Model, for instance the IoT Information Model, directly depend on the IoT Domain Model, and also several

views (as we will see in the next chapter), it should already be noted that the IoT Domain Model also takes a central role in the process of generating concrete architectures beyond merely providing a common language. As discussed in Chap. 6, Sect. 6.3, there is a special view called IoT Context View that is central in the process of generating concrete architectures. This view is an amalgam of the IoT Domain Model "traditional" context view. The latter is an architecture view that is usually generated at the very beginning of the architecture process. It describes "the relationships, dependencies, and interactions between the system and its environment (the people, systems, and external entities with which it interacts)" (Rozanski and Woods 2011).

7.3.2 Main Abstractions and Relationships

7.3.2.1 Interpreting the Model Diagram

This section describes the IoT Domain Model used in the IoT-A project. It was developed by refining and extending two models found in the literature (Haller 2010; Serbanati et al. 2011). The goal behind the IoT Domain Model is to capture the main concepts and the relationships that are relevant for IoT stakeholders. After a short introduction to the pertinent UML language (next Section), we expatiate the IoT terminology and concepts in Sect. 7.1.3.3. A discussion about guidelines and best practices on how to use the IoT Domain Model are provided in Chap. 9.

UML is used to graphically illustrate the model (Fowler 2003). Generalisation is used to depict an is-a relationship and should not be misinterpreted as sub-classing. Only the most important specialisations are shown, others are possible however. For example, not every *Device* can be characterised as a Tag, a Sensor, or an Actuator. The specialisations are, however, generally disjoint, if not noted otherwise.

Please note that generalisations/specialisations are modelled using a solid line with a large hollow triangle.

The notation indicates that class A is the Parent or super-class, while class B and class C are child or subclasses. Objects represented by class B and class C "inherit" the attributes of the object represented by class A (their parent), while having additional unique attributes of their own. This relationship is referred to as the is-a relationship – an object in class B or class C is-a type of class A (see Fig. 7.3).

This notation is not to be confused with an "aggregation or composition relationships". Rather, a terminating "open diamond" indicates an aggregation relationship, whereas a "filled diamond" indicates a composition relationship. The notation in Fig. 7.4 states that class A is an aggregation of (or contains) objects of class B and a composition of objects of class C. In other words, class A has-a class B and also class C is-part of class A. Aggregation and composition are rather similar, however the lifetime of objects of class C is determined by class A
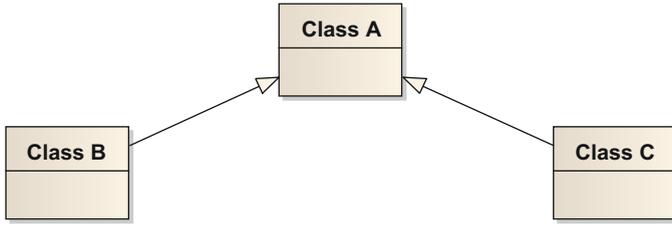
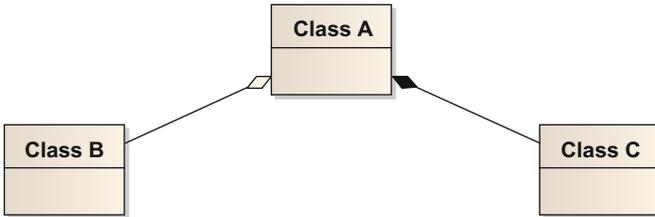**Fig. 7.3** UML generalization

Fig. 7.4 UML aggregation and composition

("A brain is part of a student" -> composition), whereas the lifetime of objects of class B is independent from class A ("A student has a school").

Finally, an "open arrow" is used to denote a "one-way" association. The notation shown in Fig. 7.5 indicates that every object in class A is associated with zero or more objects in class B, and that every object in class B is associated with exactly one object in class A. However more importantly, this notation indicates that a class A object will "know" class B objects with which it is associated, and that a class B object will "not know" the class A object with which it is associated, ref. Sensor and Physical Entity in Fig. 7.7.

The cardinalities ("asterisk", "1", etc.) are to be read as follows: from the source read the relation and the cardinality on the target gives the multiplicity with which the source can be in that relation with the target. For the inverse relation, the cardinality at the source is relevant. For example (see Fig. 7.7), a Tag identifies no or one (0..1) Physical Entity – whereas *a* Physical Entity may be identified by 0 or more Tags. A Virtual Entity may contain 0 or more other Virtual Entities, whereas a Virtual Entity can optionally be contained in at most one other Virtual Entity. Concepts depicting hardware are shown in blue, software in green, animate beings in yellow, and concepts that fit into either multiple or no categories in brown.

7.3.2.2   The Concepts of the IoT Domain Model

The most generic IoT scenario can be identified as that of a generic *User* needing to interact with a (possibly remote) *Physical Entity* (PE) in the physical world

Fig. 7.5 UML association

Fig. 7.6 Basic abstraction
of an IoT interaction

(see Fig. 7.6). In this short description we have already introduced the two key entities of the IoT. The User is a human person or some kind of a Digital Artefact (e.g., a *Service*, an application, or a software agent) that needs to interact with a Physical Entity.

In the physical environment, interactions can happen directly (e.g., by moving a pallet from location X to Y manually). In the IoT though, we want to be able to interact indirectly or *mediated*, i.e., by calling a Service that will either provide information about the Physical Entity or act on it. When a Human User is accessing a service, he does so through a service client, i.e., software with an accessible user interface. For the sake of clarity, the service client is not shown in Fig. 7.7. For the scope of the IoT Domain Model, the interaction is usually characterised by a goal that the User pursues. The Physical Entity is an identifiable part of the physical environment that is of interest to the User for the completion of her goal. Physical Entities can be almost any object or environment; from humans or animals to cars; from store or logistics chain items to computers; from electronic appliances to jewellery or clothes.

Physical Entities are represented in the digital world by a Virtual Entity. This term is also referred to as "virtual counterpart" in the literature (Römer et al. 2002), but using the same root term "entity" in both concepts clearer shows the relationship of these concepts. There are many kinds of digital representations of Physical Entities: 3D models, avatars, database entries, objects (or instances of a class in an object-oriented programming language), and even a social-network account could be viewed as such a representation, because it digitally represents certain aspects of its human owner, such as a photograph or a list of his hobbies. However, in the IoT context, Virtual Entities have two fundamental properties:

- They are Digital Artefacts. Virtual Entities are associated to a single Physical Entity and the Virtual Entity represents this very Physical Entity. While there is generally only one Physical Entity for each Virtual Entity, it is possible that the same Physical Entity can be associated to several Virtual Entities, e.g., a different representation per application domain. Each Virtual Entity must have one and only one ID that identifies it univocally. Virtual Entities are Digital Artefacts that can be classified as either active or passive. *Active Digital Artefacts* (ADA) are running software applications, agents or Services that may access other Services or Resources. *Passive Digital Artefacts* (PDA) are passive software elements such as database entries that can be digital
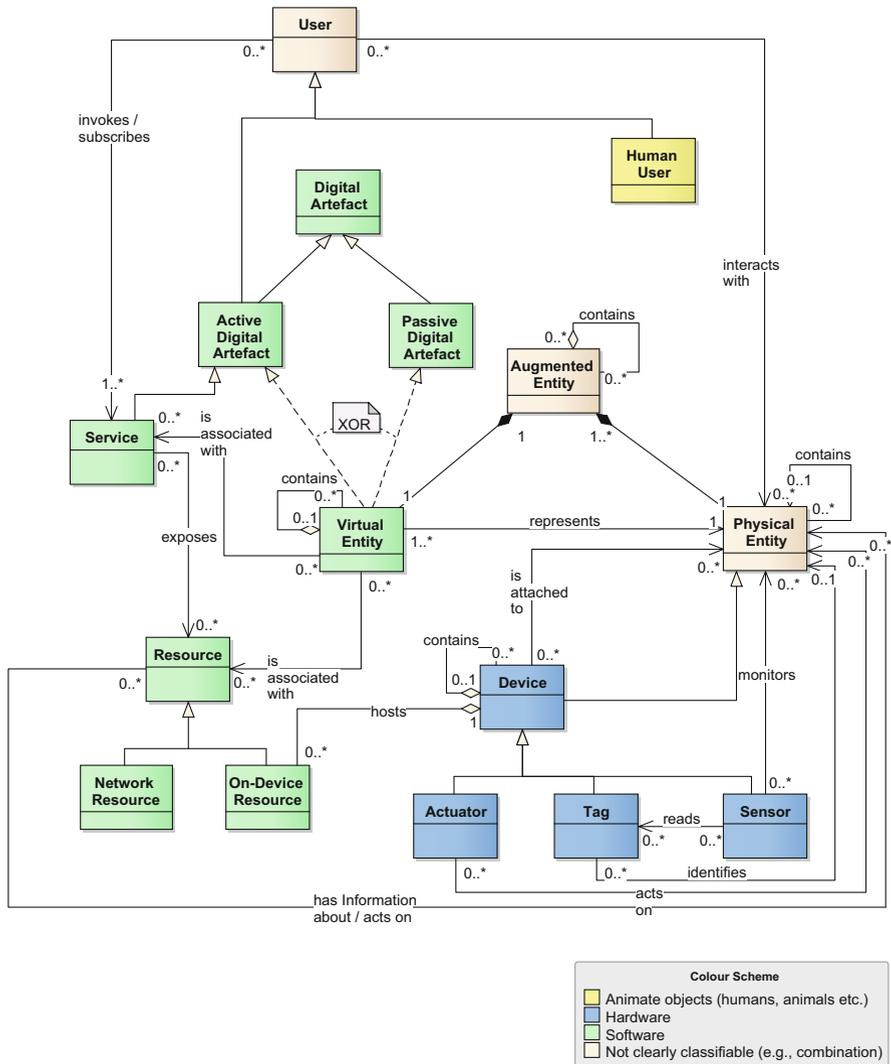
Fig. 7.7 UML representation of the IoT Domain Model

representations of the Physical Entity. Please note that all Digital Artefacts can be classified as either Active or Passive Digital Artefacts;

- Ideally, Virtual Entities are synchronised representations of a given set of aspects (or properties) of the Physical Entity. This means that relevant digital parameters representing the characteristics of the Physical Entity are updated upon any change of the former. In the same way, changes that affect the Virtual Entity could manifest themselves in the Physical Entity. For instance, manually locking a door might result in changing the state of the door in home automation

software, and correspondingly, setting the door to "locked" in the software might result in triggering an electric lock in the physical world.

At this point it should be noted that while Fig. 7.6, at first sight, seems to suggest only a Human User interacting with some Physical Entities, it also covers interaction between two machines: in this case, the controlling software of the first machine is an Active Digital Artefact and thus a User, and the second machine – or a Device in the terms of the IoT Domain Model – can be modelled as a Physical Entity. We introduce the concept of an Augmented Entity as the composition of one Virtual Entity and the Physical Entity it is associated to, in order to highlight the fact that these two concepts belong together. The Augmented Entity is what actually enables everyday objects to become part of digital processes, thus, the Augmented Entity can be regarded as constituting the "thing" in the Internet of Things.

It should be noted that there might be many types of users, as we have discussed before. A Human User is a specialisation of the general concept. However, different kinds of Users, such as maintenance people, owners, or security officers are plausible as well. It is also worth noting that we have not included different roles in the IoT Domain Model, for same reason that we have also not introduced different types of Users. Within the development of concrete architectures, it is very likely that the Users will take on different roles and these should be modelled accordingly. As the underlying taxonomies will vary with the use cases addressed, we do not prescribe a specific taxonomy here. Especially in the enterprise domain, where security roles are fundamental to practically every single IoT architecture, one common option for modelling roles can be found in (Raymond 1995). We will briefly revisit up this taxonomy within the context of the process management Section (see Sect. 7.1.5.2.1).

The relationship between Augmented, Physical and Virtual Entities is shown in Fig. 7.7, together with other terms and concepts that are introduced in the remainder of this section.

The relation between Virtual Entity and Physical Entity is usually achieved by embedding into, by attaching to, or by simply placing in close vicinity of the Physical Entity, one or more ICT Devices that provide the technological interface for interacting with, or gaining information about the Physical Entity. By so doing the Device actually extends the Physical Entity and allows the latter to be part of the digital world. This can be achieved by using Devices of the same class, as in the case of certain similar kinds of body-area network nodes, or by using Devices of different classes, as in the case of an RFID tag and reader. A Device thus mediates the interactions between Physical Entities (that have no projections in the digital world) and Virtual Entities (which have no projections in the physical world), generating a paired couple that can be seen as an extension of either one, i.e. the Augmented Entity. Devices are thus technical artefacts for bridging the real world of Physical Entities with the digital world of the Internet. This is done by providing monitoring, sensing, actuation, computation, storage and processing capabilities. It is noteworthy that a Device can also be a Physical Entity, especially in the context

of certain applications. An example for such an application is Device management, whose main concern is the Devices themselves and not the entities or environments that these Devices monitor.

From an IoT point of view, the following three basic types of Devices are of interest:

- **Sensors** provide information, knowledge, or data about the Physical Entity they monitor. In this context, this ranges from the identity of the Physical Entity to measures of the physical state of the Physical Entity. Like other Devices, they can be attached or otherwise embedded in the physical structure of the Physical Entity, or be placed in the environment and indirectly monitor Physical Entities. An example for the latter is a face-recognition enabled camera. Information from *sensors* can be recorded for later retrieval (e.g., in a storage of Resource);
- **Tags** are used to identify Physical Entities, to which *the* Tags are usually physically attached. The identification process is called "reading", and it is carried out by specific Sensor Devices, which are usually called readers. The primary purpose of Tags is to facilitate and increase the accuracy of the identification process. This process can be optical, as in the case of barcodes and QR codes, or it can be RF-based, as in the case of microwave car-plate recognition systems and RFID. The actual physics of the process, as well as the many types of tags, are however irrelevant for the IoT Domain Model as these technologies vary and change over time. These are important however when selecting the right technology for the implementation of a concrete system;
- **Actuators** can modify the physical state of a Physical Entity, like changing the state (translate, rotate, stir, inflate, switch on/off,. . .) of simple Physical Entities or activating/deactivating functionalities of more complex ones.

Notice though that Devices can be aggregations of several Devices of different types. For instance, what we call a sensor node often contains both Sensors (e.g., movement sensing) as well as Actuators (e.g., wheel engines). In some cases, Virtual Entities that are related to large Physical Entitie*s* might need to rely on several, possibly heterogeneous, Resources and Devices in order to provide a meaningful representation of the Physical Entity, c.f. in our Red Thread example, the values of several temperature Sensors are aggregated to determine the temperature of the truck.

Resource*s* are software components that provide data from or are used in the actuation on Physical Entities. Resources typically have native interfaces. There is a distinction between On-Device Resources and Network Resources. As the name suggests, On-Device Resources are hosted on Devices, viz. software that is deployed locally on the Device that is associated with the Physical Entity. They include executable code for accessing, processing, and storing Sensor information, as well as code for controlling Actuators. On the other hand, Network Resources are Resources available somewhere in the network, e.g., back-end or cloud-based databases. A Virtual Entity can also be associated with Resources that enable interaction with the Physical Entity that the Virtual Entity represents.

In contrast to heterogeneous Resources – implementations of which can be highly dependent on the underlying hardware of the Device – , a Service provides an open and standardised interface, offering all necessary functionalities for interacting with the Resources / Devices associated with Physical Entitie*s*. Interaction with the Service is done via the network. On the lowest level – the one interfacing with the Resource and closer to the actual Device hardware – , Services expose the functionality of a Device through its hosted Resources. Other Services may invoke such low-level Services for providing higher-level functionalities, for instance executing an activity of a business process. A typical case for this is the Service alerting "Ted" based on the temperature Service results in the "Red Thread" example.

Since it is the Service that makes a Resource accessible, the above-mentioned relations between Resources and Virtual Entities are modelled as associations between Virtual Entities and Services. For each Virtual Entity there can be associations with different Services that may provide different functionalities, like retrieving information or enabling the execution of actuation tasks. Services can also be redundant, i.e., the same type of Service may be provided by different instances (e.g. redundant temperature Services provided by different Devices). In this case, there could be multiple associations of the same kind for the same Virtual Entity. Associations are important in look-up and discovery processes.

The instance diagrams such as Fig. 7.2 are concrete instantiations of the IoT Domain Model, i.e. concrete architectures modelled with the concepts of the IoT Domain Model.

7.3.3   Detailed Explanations and Related Concepts

The IoT Domain Model as explained in the previous section is focusing on the main concepts at a high level of abstraction, capturing the essence of the IoT Domain. However, for easier understanding we provide here more detailed explanations.

7.3.3.1   Devices and Device Capabilities

From an IoT Domain-Model point of view, Devices are only technical artefacts meant to provide an interface between the digital and the physical worlds, i.e. a link between the Virtual Entities and the Physical Entities. For this reason, Devices must be able to operate both in the physical and digital world and the IoT Domain Model only focuses on their capability to provide observation and modification of the physical environment from the digital environment. If other properties of Devices were relevant, the Device would be modelled as an entity itself.

The hardware underlying the Devices is very important though and must have at least some degree of communication, computation and storage capabilities for the purposes of the IoT. Moreover, power resources are also very important, as they can

provide operational autonomy to the Devices. Many technologies and products are available and their capabilities vary noticeably. While these capabilities might not impact directly the IoT Domain Model, they are very important during the application-design phase, c.f. the Deployment and Operation view in Sect. 8.2.4 "Deployment & Operation view".

Communication capabilities depend on the type of data exchanged with the *Device* (identifier, identifier + data, sensor data, or commands) and the communication topology (network, reader-tag, peer-to-peer, etc.). These aspects are very important in the IoT context and have a large impact on energy consumption, data-collection frequency, and the amount of data transmitted. Communication capabilities indirectly impact the location of *Resources* (on-device or on the network). Please refer to the IoT Communication Model (Sect. 7.1.6) for a detailed discussion of this topic. Security features also impact communication capabilities, since they usually introduce a relevant communication overhead (c.f. Sect. 7.1).

Computation capabilities on the other hand have a huge impact on the chosen architecture, the implementable security features, and power resources of the Devices. They are also relevant for what concerns the availability of On-Device Resources and their complexity, as constrained Devices might not have sufficient computational resources.

The term storage usually refers to the capability of supporting the firmware/software running on the Device. This can be accomplished storing data provided by on-board sensor hardware or data gathered from other Services and needed for supporting a given Resource. Storage can range from none, as in the case of RFID technology to kilobytes in the case of typical embedded Devices or even more in case of unconstrained Devices.

7.3.3.2 Resources

Resources are software components that provide some functionality. When associated with a Physical Entity, they either provide some information about or allow changing some aspects in the digital or physical world pertaining to one or more Physical Entities. The latter functionality is commonly referred to as actuation. Resources can either run on a Device – hence called On-Device Resources – or they can run somewhere in the network (Network Resources). On-Device Resources are typically sensor Resources that provide sensing data or actuator Resources, e.g. a machine controller that effects some actuation in the physical world. They thus can be seen as a "bridge" between the digital and physical world. On-Device Resources may also be storage Resources, e.g., store a history of sensor measurements, but are limited by the storage capacity of the Device.

As Network Resources run on a dedicated server in the network or in the "cloud", they do not rely on special hardware that allows direct connection to the physical world. They rather provide enhanced Services that require more system resources than Devices typical for the IoT can provide. Such Resources can process

data, for instance they can take sensor information as input and produce aggregated or more high-level information as output. Also, Network Resources can be storage Resources, which typically do not suffer from the limitations of their on-device counterparts. Storage Resources can store information produces by Resources and they can thus provide information about Physical Entities. This may include location and state-tracking information (history), static data (like product-type information), and many other properties. An example of a storage Resource is an EPCIS repository (Electronic Product Code Information Services (EPC 1.0.13)) that aggregates information about a large number of Physical Entities. Notice that also Human Users can update the information in a storage Resource, since not all known information about an entity always is, or even can be, provided by Devices.

7.3.3.3 Services

Services are a widely used concept in today's IT systems. According to (MacKenzie et al. 2006), "Services are the mechanism by which needs and capabilities are brought together". This definition is very broad, and the Service concept in the IoT Domain Model is covering this broad definition – but Services *are* restricted to technical Services implemented in software (in contrast to general, non-technical services that e.g. a lawyer or a consultant provides). As such, Services provide the link between the IoT aspects of a system and other, non-IoT specific parts of an information system, like e.g. various enterprise systems; IoT-related Services and non-IoT Services can be orchestrated together in order to form a complete system.

As it has been pointed out in (Martín 2012), IoT-related Services need to be explained in more detail: IoT Services provide well-defined and standardised interfaces, hiding the complexity of accessing a variety of heterogeneous Resources. The interaction with a Physical Entity can be accomplished via one or more Services associated with the corresponding Virtual Entity. This association becomes important in the process of look-up and discovery. An IoT Service can thus be defined as a type of Service enabling interactions with the real world.

According to (Martín 2012), IoT Services can be classified according by their level of abstraction:

- **Resource-level Services** expose the functionality, usually of a Device, by accessing its hosted Resources. These kinds of Services refer to a single Resource. In addition to exposing the Resource's functionality, they deal with quality aspects, such as dependability, security (e.g., access control), resilience (e.g., availability) and performance (e.g., scalability, timeliness). Resources can also be Network Resources, i.e. the Resources do not necessarily reside on a Device in the sense of the IoT Domain Model (normal computers are not regarded as IoT Devices by the IoT Domain Model), but can also be hosted somewhere else. The concrete location of where the Network Resource is situated is commonly abstracted away by the Service;

- **Virtual Entity-level** *Services* provide access to information at a Virtual Entity-level. They can be Services associated to a single Virtual Entity that give access to attributes for reading attribute information or for updating attributes in order to trigger associations. An alternative is to provide a common Virtual Entity-level *Service* with an interface for accessing attributes of different Virtual Entities, as, for instance, the NGSI Context Interface (NGSI 2010) provides for getting attribute information of the Virtual Entities;
- **Integrated Services** are the result of a Service composition of Resource-level or Virtual Entity-level *Services* as well as any combinations of both Service abstractions.

7.3.3.4 Identification of Physical Entities

In order to track and monitor Physical Entities, they have to be identified. There are basically two ways for how this can be done, as is very well described in (Furness 2009): Using either natural-feature identification (classified as "primary identification") or using some type of Tags or labels (classified as "secondary identification") that are attached to the Physical Entity.

Both means of identification are covered in the IoT Domain Model. Tags are modelled as Devices that explicitly identify a Physical Entity. Natural-feature identification can be modelled, for example, by using a camera – a kind of *Sensor* – that monitors the Physical Entity and an additional Resource that does the natural feature extraction (i.e. a dedicated software component). The result of the natural-feature extraction can be used as search term for looking up the corresponding Virtual Entity.

RFID Tags are a prominent example in IoT. As they come with their own electronic circuitry it seems quite natural to classify RFID Tags as Devices in terms of the IoT Domain Model. The case is less clear-cut regarding the classification of a barcode label, however. As pointed out elsewhere (Haller 2010), classifying a barcode label as a Device seems a little far-fetched; regarding it as a "natural feature" of the Physical Entity it is attached to, seems to be more appropriate. However, as with many modelling questions, this is a matter of taste – the IoT Domain Model is not prescribing which variant to use.

7.3.3.5 Context and Location

As the IoT pertains to the physical world, the characteristics of the physical world play an important role. All elements of the physical world are situated within a certain context, and location is an essential aspect of this context. All concepts in the IoT Domain Model that refer to elements of the physical world, i.e., Physical Entities, Devices, and Human Users inherently have a location. This location may or may not be known within the IoT system.

The location of a Physical Entity can be modelled as an attribute of a Virtual Entity. This location can then be provided through Resources. In the case of a stationary Physical Entity, the Resource providing the location can be an On-Device (storage) Resource, in the case of a mobile Physical Entity the Resource could be a positioning system like GPS, or a tracking system like existing indoor location systems.

7.4 Information Model

The IoT Information Model defines the structure (e.g. relations, attributes, services) of all the information for Virtual Entities on a conceptual level, see also Sects. 7.1.3.2.2, 7.1.3.3.1 and 7.1.3.3.3. The term information is used along to the definitions of the DIKW hierarchy (see Rowley 2007) where data is defined as pure values without relevant or useable context. Information adds the right context to data and offers answers to typical questions like who, what, where and when.

The description of the representation of the information (e.g. binary, XML, RDF etc.) and concrete implementations are not part of the IoT Information Model.

The IoT Information Model details the modelling of a Virtual Entity. The Virtual Entity (VirtualEntity) has attributes with a name (resp. Attribute and AttributeName) and a type (AttributeType) and one or more values (Value) to which meta-information (MetaData) can be associated. Important meta-information is, e.g., at what time a value was measured (i.e. time stamp), the location where a measurement took place, or the quality of the measurement. Metadata can itself contain additional metadata, e.g. the unit in which the metadata is measured. The association (Association) between a Virtual Entity and a Service is detailed in the sense that is pertains to a certain Attribute of the Virtual Entity. The serviceType can be set either to INFORMATION, if the *Service* provides the attribute value to be read or to ACTUATION, if the *Service* allows the Attribute value to be set, as resulting of a corresponding change in the physical world.

7.4.1 Definition of the IoT Information Model

The diagram in Fig. 7.8 shows the structure of the information that is handled and processed in an IoT System. The main aspects are represented by the elements VirtualEntity, ServiceDescription and Association. A Virtual Entity models a Physical Entity and ServiceDescription describes a Service that serves information about the Physical Entity itself or the environment. Through an Association, the connection between an Attribute of a Virtual Entity and the ServiceDescription is modelled, e.g. the Service acts as a "get" function for an Attribute value.

Every Virtual Entity needs to have a unique identifier (identifier) or entity type (entityType), defining the type of the Virtual Entity representation, e.g. a human, a
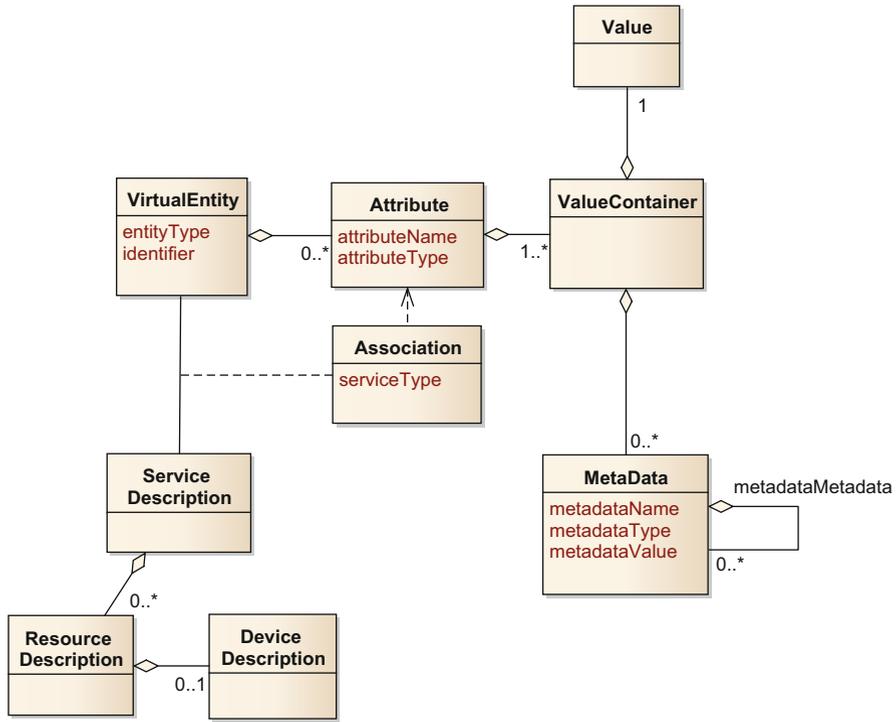
Fig. 7.8 IoT Information Model

car or a temperature sensor. Furthermore, a Virtual Entity can have zero to many different attributes (Attribute class in Fig. 7.8). The entityType of the VirtualEntity class may refer to concepts in an ontology that defines what attributes a Virtual Entity of this type has (see, for instance, [Group, W3C OWL]). Each Attribute has a name (attributeName), a type (attributeType), and one to *many* values (ValueContainer). The attributeType specifies the semantic type of an attribute, for example, that the value represents temperature. It can reference an ontology-concepts, e.g., qu:temperature taken from "Quantity Kinds and Units"-ontology (Lefort 2005). This way, one can for instance, model an attribute, e.g. a list of values, which itself has several values. Each ValueContainer groups one Value and zero to *many* metadata information units belonging to the given Value. The metadata can, for instance, be used to save the timestamp of the Value, or other quality parameters, such as accuracy or the unit of measurement. The Virtual Entity (VirtualEntity) is also connected to the ServiceDescription via the ServiceDescription-VirtualEntity association.

A ServiceDescription describes the relevant aspects of a Service, including its interface. Additionally, it may contain one (or more) ResourceDescription(s) describing a Resource whose functionality is exposed by the Service. The
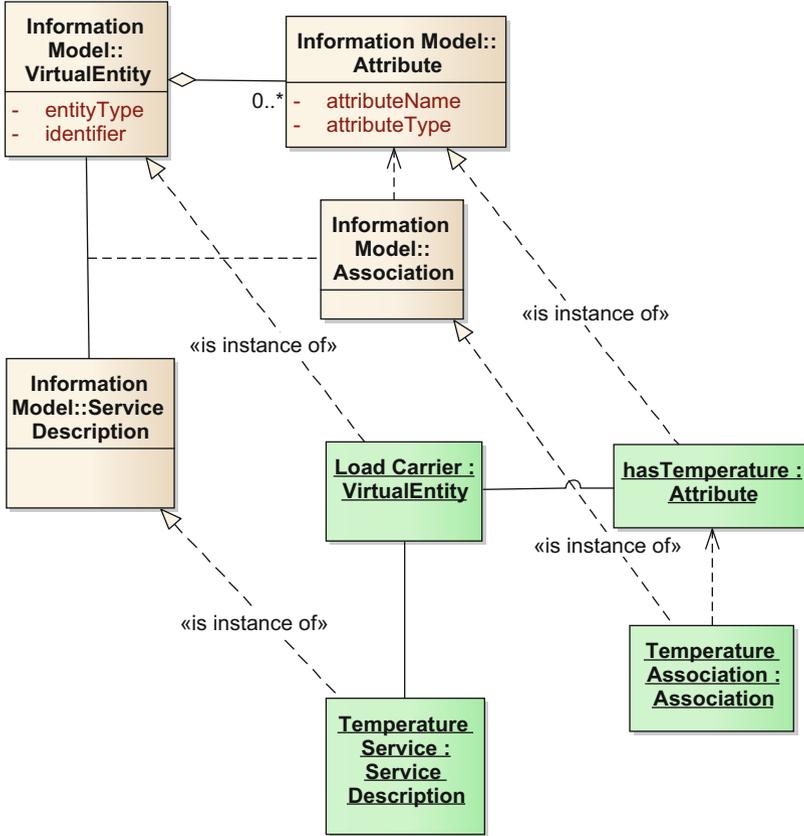
Fig. 7.9 Illustrating example for IoT Information Model

ResourceDescription in turn may contain information about the Device on which the Resource is hosted.

7.4.2 Modelling of Example Scenario

The IoT Information Model is a meta-model that defines the structure of key aspects of the information being managed in an IoT system. Therefore, unlike the Domain Model (see the recurring example in Sect. 7.1.3.1), it would typically not be directly instantiated (see information view Sect. 8.2.3 and the related Design Choices in Chap. 6 for this purpose). Nevertheless, for illustration purposes, we sketch here how the information relevant for our example scenario from Sect. 4.2 could be modelled (Fig. 7.9).

The element of interest for which we can get some information is the Load Carrier, which is therefore digitally represented by a Virtual Entity. Here, we show how the temperature aspect of the Physical Entity is modelled by the hasTemperature attribute of the Virtual Entity. This Figure also features a description of the service that is used to measure this temperature. What is finally needed is the connection between the hasTemperature attribute and the service that can provide its value. This is achieved by the Temperature Association.

7.4.3   Relation of Information Model to Domain Model

The IoT Information Model models all the concepts of the Domain Model that are to be explicitly represented and manipulated in the digital world. Additionally, the IoT Information Model models relations between these concepts. The IoT Information Model is a meta-model that provides a structure for the information being handled by IoT Systems. This structure provides the basis for all aspects of the system that deal with the representation, gathering, processing, storage and retrieval of information and as such is used as a basis for defining the functional interfaces of the IoT system.

Figure 7.10 shows the relation between the Domain Model concepts and the IoT Information Model elements. The main Domain Model concepts that are explicitly represented in an IoT system are the Virtual Entity and the service. The latter also comprises aspects of the Resource and the Device. As the Virtual Entity is the representation of the Physical Entity in the digital world, there is no other representation of the Physical Entity in the IoT Information Model.

7.4.4   Other Information-Related Models in IoT-A

Throughout IoT-A several other information- related models exist. Most of them are defined in the technical work packages WP2, WP3, WP4 and WP5. More information can be found in the respective deliverables (see below).

- **Entity model:** The Entity Model specifies which attributes and features of real word objects are represented by the virtual counterpart, i.e. the Virtual Entity of the respective Physical Entity. For every attribute specified in the entity model, services can be found that are able to either provide information about the attribute (sensing) or manipulate it, leading to an effect in the real world (actuating). More information about the entity model can be found in Sect. 7.1.3.2.2.
- **Resource model:** The Resource Model contains the information that is essential to identify Resources by a unique identifier and to classify Resources by their type, like sensor, actuator, processor or tag. Furthermore the model specifies the
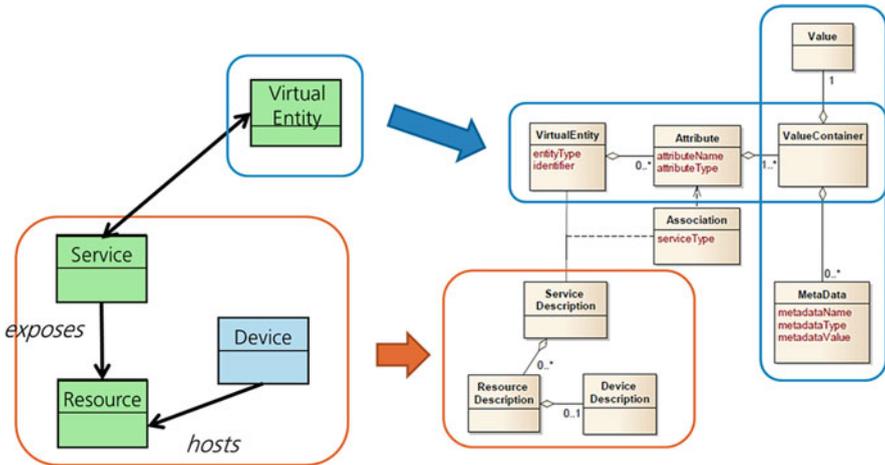
Fig. 7.10 Relation between the core concepts of the IoT Domain Model and IoT Information Model

geographic location of the Resource, the Device the Resource is hosted on (if so) as well as the IoT Services the Resource is exposed through. More information can be found in (Martin 2012) Sect. 3.3.

- **Service description model:** Services provide access to Resources and are used to access information or to control Physical Entities. An IoT Service accesses IoT Resources in order to provide information about attributes of entities or manipulates them leading to an effect in the real world. A Service Description describes a Service, using for instance a service description language such as USDL.[1] For more information see (Martin 2012) Sect. 4.6.3.
- **Event Model:** Event models are quite essential in today's IoT architectures, e.g. in the EPCglobal Information Services. Normally events are used to track dynamic changes in a (software) system, showing who or what has triggered it and when, where and why the change occurred. Event representation and processing is specified in Sect. 4.2 of (Voelksen 2013).

7.5 Functional Model

7.5.1 Functional Decomposition

In the IoT-A project, *Functional Decomposition* (FD) refers to the process by which the different *Functional Components* (FC) that make up the IoT ARM are identified and related to one another.

---

[1] http://www.internet-of-services.com/.

The main purpose of Functional Decomposition is, on the one hand, to break up the complexity of a system compliant to the IoT ARM in smaller and more manageable parts, and to understand and illustrate their relationship on the other hand.

Additionally, Functional Decomposition produces a superset of functionalities that can be used to build any IoT system. The output of Functional Decomposition is described in this document at two levels of abstraction:

- The Functional Model (purpose of this section);
- The Functional View (presented in Sect. 8.2.2).

The definition of the Functional Model is derived by applying the definition of a Reference Model found in (MacKenzie et al. 2006) to Functional Decomposition: "The Functional Model is an abstract framework for understanding the main *Functionality Groups* (FG) and their interactions. This framework defines the common semantics of the main functionalities and will be used for the development of IoT-A compliant Functional Views."

The definition contains the following concepts that need to be explained in more detail:

- **Abstract:** The Functional Model is not directly tied to a certain technology, application domain, or implementation. It does not explain what the different Functional Components are that make up a certain Functionality Group;
- **Functionality Groups and their interactions:** The Functional Model contains both the Functionality Groups and the interaction between those parts. A list of the Functionality Groups alone would not be enough to make up the Functional Model. Both the Functionality Groups and their interaction are mandatory;
- **Functional View:** The Functional View describes the system's runtime Functional Components, including the components' responsibilities, their default functions, their interfaces, and their primary interactions. The Functional View is derived from the Functional Model on the one hand and from the Unified Requirements on the other hand. Note that various Functional Views could be derived from the Functional Model. See also Sect. 8.2.2 for more detailed information on the functional view.

7.5.2 Functional Model Diagram

The Functional Model diagram is depicted in Fig. 7.11 and was derived as follows:

- From the main abstractions identified in the Domain Model (Virtual Entities, Devices, Resources and Users) the "Application", "Virtual Entity", "IoT Service" and "Device" FGs are derived;
- With regards to the plethora of communication technologies that the IoT ARM needs to support, the need for a "Communication" FG is identified;
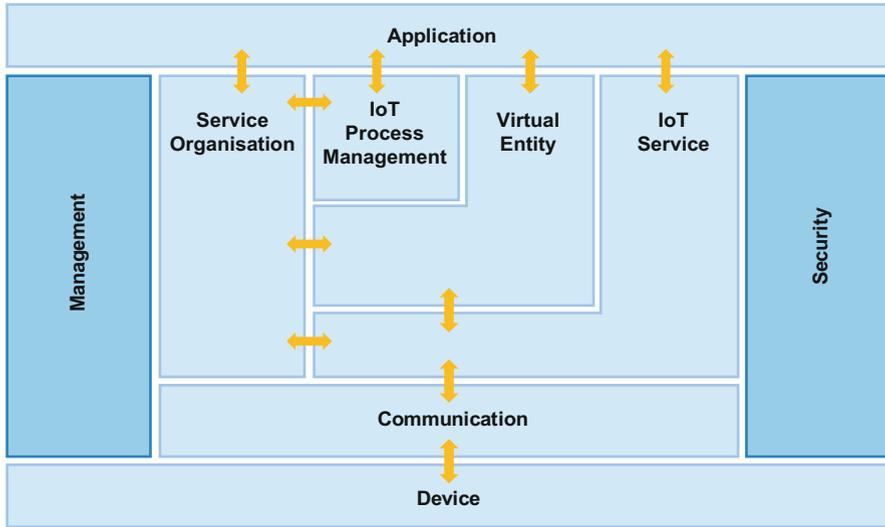
Fig. 7.11 Functional Model

- Requirements expressed by stakeholders regarding the possibility to build services and applications on top of the IoT are covered by the "IoT Process Management" and "Service Organisation" FGs;
- To address consistently the concern expressed about IoT Trust, Security and Privacy, the need for a "Security" transversal FG is identified;
- Finally, the "Management" transversal FG is required for the management of and/or interaction between the functionality groups.

The Functional Model contains seven longitudinal Functionality Groups (light blue) complemented by two transversal Functionality Groups (Management and Security, dark blue). These transversal groups provide functionalities that are required by each of the longitudinal groups. The policies governing the transversal groups will not only be applied to the groups themselves, but do also pertain to the longitudinal groups.

As an example: for a security policy to be effective, it must ensure that there is no functionality provided by a component that would circumvent the policy and provide unauthorised access.

Next, the interactions between the FGs are shown. As can be seen from Fig. 7.11, the Functional Model is a hierarchical model and the main interactions between the FG's are depicted with orange arrows. Since the transversal FGs (Management & Security) interface with most of the other FGs, their interactions with the other FG's are not explicitly depicted.

In the remainder of this section, each of the FGs will be described in more detail (with exception of the Application and Device FGs, since trying to capture their properties would be so generic that they do not add any value).

7.5.2.1 IoT Process Management

The IoT Process Management FG relates to the conceptual integration of (business) process management systems with the IoT ARM. The overall aim of this FG is to provide the functional concepts necessary to conceptually integrate the idiosyncrasies of the IoT world into traditional (business) processes. By so doing, enterprises can effectively utilise IoT sub-systems adhering to common standards and best practices, thus avoiding the overhead and costs of isolated and proprietary "Intranet-of-Things" island solutions.

In the IoT-A project, the IoT Process Management FG is addressed by WP 2. The IoT Process Management FG provides additions and extensions to industry standards, for instance BPMN 2.0. The additions feature IoT-specific aspects of (business) processes, such as the reliability or accountability of sensor data providing information about Virtual Entities or the required processing capabilities of Devices hosting certain Resources relevant for the real world. Applications that interact with the IoT Process Management FG via IoT-augmented process models can effectively be shielded from IoT-specific details of lower layers of the functional model, which greatly reduces integration costs and thus contributes to an increased adoption of IoT-A based IoT systems (Meyer et al. 2011).

One important aspect of IoT Process Management is its inherent closeness to enterprise systems. As it was already introduced in the IoT Domain Model Sect. 7.1.3, the IoT Process Management FG is where the business objects and processes are combined with the world of IoT, and especially here the modelling of processes must take into account not only the idiosyncrasies of the IoT domain, but also the specificities of the underlying business domain. The different roles of the business objects and users will be defined here. Again, as discussed in the IoT Domain Model section, we do not prescribe a specific taxonomy here. However, for pedagogical purposes we illustrate how this taxonomy looks like in the context of the RM-ODP context Enterprise View (see the discussion about RM-ODP (Raymond 1995) and roles in IoT Domain Model Sect. 7.1.3.2.2).

- Permission: what can be done? For instance a self-regulating ventilation system can be started by a central control system;
- Prohibition: what must not be done? For instance the ventilation system may not be shut down in its entirety if the outside temperature is above a pre-defined value and if humans are present in the building;
- Obligations: the central control system needs to save recorded environmental parameters for each room in the entire building (temperature, humidity, ventilation settings). Such records can, for instance, be required by national occupational-health laws.

When it comes to the practical realisation of the process management, these different policies will come into play when the respective business processes are modelled. In Chap. 5 we pick up the notion of enterprise views and illustrate how they factor into the requirements process.

The IoT Process Management FG is conceptually closely related to the Service Organisation FG and acts as a proxy to applications that integrate an IoT-A-compliant IoT system. Naturally, the IoT Process Management FG has a dependency on the Service Organisation FG, as a central concept in the execution of (business) processes is the finding, binding, and invoking of Services that are used for each process step. The IoT Process Management FG therefore relies on Service Organisation to map the abstract process definitions to more concrete Service invocations.

Applications can utilise the tools and interfaces defined for the IoT Process Management FG in order to stay on the (abstract) conceptual level of a (business) process, while, at the same time, making use of IoT-related functionality without the necessity of dealing with the complexities of IoT Services. In this respect, the IoT Process Management FG provides conceptual interfaces to the IoT ARM, that are alternatives to the more concrete Virtual Entity FG and Service Organisation FG interfaces.

7.5.2.2   Service Organisation

The Service Organisation FG is a central Functionality Group that acts as a communication hub between several other Functionality Groups. Since the primary concept of communication within the IoT ARM is the notion of the Service (see Domain Model Sect. 7.1.3), the Service Organisation FG is used for composing and orchestrating Services of different levels of abstraction. Within the IoT Reference Architecture, it effectively links the Service requests from high level FGs such as the IoT Process Management FG, or even external applications, to basic services that expose Resources (see Domain Model Sect. 7.1.3) (such as services hosted on a WSN gateway), and enables the association of entities with these services by utilising the Virtual Entity FG, so that a translation of high-level requests dealing with properties of entities (e.g., "give me please the temperature in the room 123") down to the concrete IoT services (e.g., "sensor service XYZ") can be realised. In order to allow for querying Virtual Entities or IoT Services that are associated with these entities, the Service Organisation FG is responsible for resolving and orchestrating IoT Services and also deal with the composition and choreography of Services. Service Composition is a central concept within the architecture, since IoT Services are very frequently capable of rather limited functionality due to the constraints in computing power and battery life that are typical for WS&ANs or embedded Devices comprising the IoT. Service Composition then helps combining multiple of such basic Services in order to answer requests at a higher level of Service abstraction (e.g. the combination of a humidity sensing Service and a temperature Service could serve as input for an air-conditioning). Service Choreography is a concept that supports brokerage of Services so that Services can subscribe to other Services available in the system.

As discussed in the previous section, the Service Organisation FG is closely tied to the IoT Process Management FG, since the Service Organisation FG enables (business) processes or external applications to find and bind Services that can be

used to execute process steps, or to be integrated in other ways with external applications. The Service Organisation FG acts as an essential enabler for the IoT Process Management FG. The Virtual Entities specified during the process modelling phase are resolved and bound to IoT Service FG needed for process execution.

7.5.2.3   Virtual Entity and IoT Service

The Virtual Entity and IoT Service FGs include functions that relate to interactions on the Virtual-Entity and IoT-Service abstraction levels, respectively. Figure 7.12 shows the abstraction levels and how they are related. On the left side of Fig. 7.12, the physical world is depicted. In the physical world, there are a number of Sensors and Actuators that capture and facilitate the change of certain aspects of the physical world. The Resources associated to the Sensors and Actuators are exposed as IoT Services on the IoT Service level. Example interactions between applications and the IoT system on this abstraction level are "Give me the value of Sensor 456" or "Set Actuator 867 to On". Applications can only interact with these Services in a meaningful way, if they already know the semantics of the values, e.g. if Sensor 456 returns the value 20, the application has to be programmed or configured in such a way that it knows that this is the outdoor temperature of the car of interest, e.g. Car MXD – 123. So, on this level no semantics is encoded in the information itself, nor does the IoT system have this information, it has to be a-priori shared between the Sensor and the application.

   Whereas interaction on the IoT Service level is useful for a certain set of applications that are programmed or configured for a specific environment, there is another set of applications that wants to opportunistically use suitable Services in a possibly changing environment. For these types of applications, and especially the Human Users of such applications, the Virtual Entity level models higher-level aspects of the physical world, and these aspects can be used for discovering Services. Examples for interactions between applications and the IoT system on this abstraction level are "Give me the outdoor temperature of Car MXD – 123" or "Set lock of Car MXD – 123 to locked". To support the interactions on the Virtual Entity level, the relation between IoT Services and Virtual Entities needs to be modelled, which is done in form of associations. For example, the association will contain the information that the outdoor temperature of Car MXD – 123 is provided by Sensor 456. Associations between Virtual Entities and IoT Services are modelled in the Information Model (Sect. 7.1.4).

Virtual Entity

The Virtual Entity FG contains functions for interacting with the IoT System on the basis of VEs, as well as functionalities for discovering and looking up Services that can provide information about VEs, or which allow the interaction with VEs.
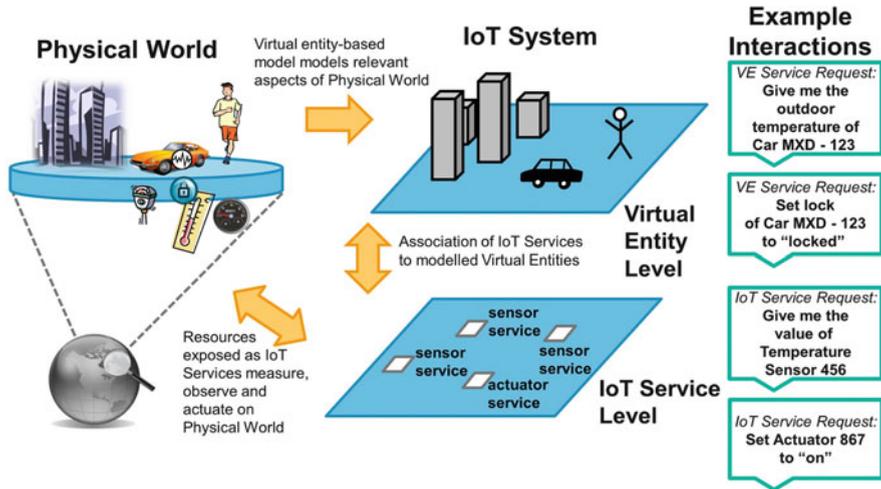
Fig. 7.12 IoT-Service and Virtual-Entity abstraction levels

Furthermore, it contains all the functionality needed for managing associations, as well as dynamically finding new associations and monitoring their validity. This need can be triggered by the mobility of Physical Entities represented by the Virtual Entities and/or Devices.

IoT Service

The IoT Service Functional Group contains IoT Services as well as functionalities for discovery, look-up, and name resolution of IoT Services.

7.5.2.4 Communication

The Communication FG abstracts the variety of interaction schemes derived from the many technologies (Device FG) belonging to IoT systems and provides a common interface to the IoT Service FG. It provides a simple interface for instantiating and for managing high-level information flow. In particular, the following aspects are taken into account: starting from the top layers of the ISO/OSI model it considers data representation, end to end path information, addressing issues (i.e. Locator/ID split), network management and device specific features.

The Communication FG can be customised according to the different requirements defined in the Unified Requirements list and, in particular, those related to communication specified within WP3. For instance, integrity and security can be enforced exploiting many different signature and encryption schemes at various ISO/OSI layers; reliability is achieved either by means of link layer

acknowledgements or end to end error correction schemes at the upper layers; quality of service is realised by providing queue management techniques; finally, in order to account for communication between different technologies, protocol translation and context passing functionalities are described.

7.5.2.5 Management

The Management FG combines all functionalities that are needed to govern an IoT system. The need for management can be traced back to at least four high-level system goals (Pras 1995):

- Cost reduction;
- Attending unexpected usage issues;
- Fault handling; and
- Flexibility.

Cost reduction: In order to control the cost of a system, it is designed for a maximum amount of users and/or use cases. "A way for the designer to deal with the requirements of multiple groups of users is to abstract from the differences in [the] requirements and [to] parameterise the design" (Pras 1995). Upon commissioning or start-up of the system, these parameters will be initialised by the Management FG.

Attending unexpected events: The IoT system is based on an incomplete model of reality – as literally all systems are. For example, even for the same type of user, unforeseen activity patterns in the physical world and thus unforeseen usage may arise may arise. For instance, errors are introduced into the system through explicit, erroneous management directives (Harrisburg, Chernobyl). Another example is that Devices can suddenly just die. The latter is most likely to become prevalent in the IoT, since the cost margins for IoT equipment and thus their reliability can be much lower than that for traditional telecommunications equipment (back-bone routers, etc.). The management FG can provide strategies and actions for the mitigation of impacts from unforeseen situations. Such impacts can be link failure, queue overload, etc. In order to better adapt to new situations, it is of course paramount that the Management FG has a good overview of the system state. To that end the management system provides supports collection.

Fault handling: This goal addresses the unpredictability of the future behaviour of the system itself. This is of special interest in complex IoT systems and also in IoT systems in which, for instance, the devices in an IoT system do not provide a model for their behaviour. The measures implied by this goal are:

- Prediction of potential failures;
- Detection of existing failures;
- Reduction of the effects of failures;
- Repair.

The first three measures can be achieved by comparing the current behaviour of system components with previous and/or expected behaviour.

Flexibility: The design of a system is based on use-case requirements. However, these use-case requirements are not static. Instead of designing a new system every time the requirements change, some flexibility should be built into the system. Due to this flexibility, the Management FG of the IoT system will be able to react to changes in the user requirements. This can take place during boot up, commissioning or also at run time.

All of the above goals rely on shared common functionality and repositories, as, for instance, a state repository. Other functionalities are:

- Management of the membership and accompanying information of a given entity to the IoT system. Such entity may be a Functional Component (FC), a Virtual Entity, an IoT Service, an application, a Device. The information considered may cover ownership, administrative domain, capabilities, rules, and rights;
- Retrieval of the list of members pertaining to a given property such as the ownership/administrative domain;

  Finally, some more examples for the above goals are provided:
- Enforcing rules attached to the usage of a certain entity e.g.

  - *Attending unexpected events:* A service needs temperature measurements every microsecond, but the rule file for the associated device says: maximum measurement frequency of this device is 100 Hz. The rule file also might say: no continuous operation of said device for more than 1 h (due to energy constraints);
  - *Fault handling:* A service wants to run a business process that would consume all IoT services and the VE lookup for more than a day. An example for this is a query for the geo-location of all temperature Sensors on planet Earth. The rule file may contain instructions about how many resources can be consumed by an application;
  - *Cost reduction:* Logging entity usage by a user for further processing (e.g. billing).

Besides the above, "traditional" goals of management, the Management FG also needs to address needs that arise when IoT systems can actuate and/or if the they are embedded in critical infrastructure. Examples for such situations are

- Bringing the entire system to an emergency stop, for instance a train;
- Turning the entire system into a sleep/energy-saving mode in order to relax to load on a failing Smart Grid.

7.5.2.6   Security

The Security Functionality Group (Security FG) is responsible for ensuring the security and privacy of IoT-A-compliant systems. It is in charge of handling the initial registration of a client to the system in a secure manner. This ensures that only legitimate clients may access services provided by the IoT infrastructure. The Security FG is also in charge of protecting private parameters of users. This is achieved by providing anonymity (ensuring that the user's identity remains confidential when she/he/it accesses a Resource or a service) and "unlink-ability" (ensuring that the user may make multiple uses of Resources or services without an attacker being able to establish links between those uses). This privacy support relies on fine-tuned identity management, which is able to assign various pseudorandom identifiers to a single user.

The Security FG also ensures that legitimate interaction occurs between peers that are statically authorised to interact with each other, or that are trusted by each other. This is achieved through the use of dedicated authorisation functions or through the reliance on a trust – and-reputation model, which is able to identify trustworthy peers in a privacy-capable and highly mutable architecture.

Finally, the Security FG enables secure communications between peers by managing the establishment of integrity and confidentiality features between two entities lacking initial knowledge of each other.

7.6   Communication Model

The IoT Communication Model aims at defining the main communication paradigms for connecting elements, as defined in the IoT Domain Model. We provide a reference set of communication rules to build interoperable stacks, together with insights about the main interactions among the elements of the IoT Domain Model. We propose a Communication Model that leverages on the ISO OSI 7-layer model for networks and aims at highlighting those peculiar aspects inherent to the interoperation among different stacks, which we will call in what follows, interoperability features. Further, the application of communication schemes, such as application layer gateways, transparent proxy, network virtualization, etc., to different IoT network types is discussed.

In particular, with reference to our Read Thread example of Sect. 4.2, the IoT Communication Model can be used to model how the monitoring Sensors of the truck can seamlessly interact with Ted's phone and how it can communicate with the store enterprise system.

The IoT Communication Model has multiple usages. For instance, it can guide the definition of the Communication Functional Components from which the Communication Functional Group is composed of. Finally, it can be used to derive the Communication best practices, as depicted in the following pictures (Fig. 7.13):
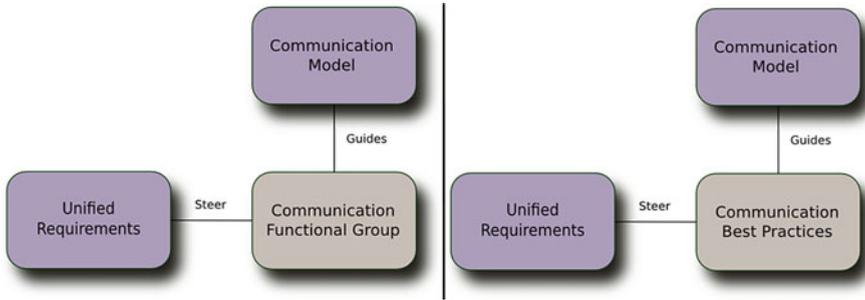
Fig. 7.13 IoT Communication Model usages: (*left*) using the CM together with the Unified Requirements to define the Communication FG; (*right*) deriving Communication Best Practices thanks to the CM and the Unified Requirements

7.6.1 *IoT Domain Model Element Communications*

For the IoT Communication Model, it is important to identify the communication-system elements and/or the communicating Users among those defined in the IoT Domain Model. One, if not the main peculiarity of the IoT is that Users can belong to many disjoint categories: Human Users, Services or Active Digital Artefacts. While the same picture is emerging in today's Internet usage, the percentage of human-invoked communication will be even lower in the IoT. Moreover, entities can be physical, digital, or virtual. While a Physical Entity cannot directly take part in communication, it can exploit Services associated to its virtual counterpart.

The communication between these users needs to support different paradigms: unicast is the mandatory solution for one-to-one connectivity. However, multicast and anycast are needed for fulfilling many other IoT-application requirements, such as data collection and information dissemination, etc.

With reference to our "Red Thread" and the IoT Domain Model section, the main communicating elements are: the Mote Runner Node (Device), the Alarm Service (Service), the AndroidApp (Active Digital Artefact) and Ted (Human User).

This section provides insight and guidance on the interactions between elements of the IoT Domain Model. In particular, per possible communicating entity pair, a discussion about the relevant layer of the IoT Communication Model will be provided.

7.6.1.1 User-Service / Service-Service Interactions

As shown in Fig. 7.14, the IoT Domain Model entities involved in this interaction are mainly two: User and Service (circled in solid red lines). For instance, in our recurring example this interaction models the truck driver, Ted, who needs to interact with the AndroidApp in order to receive alarms. However, a Service may
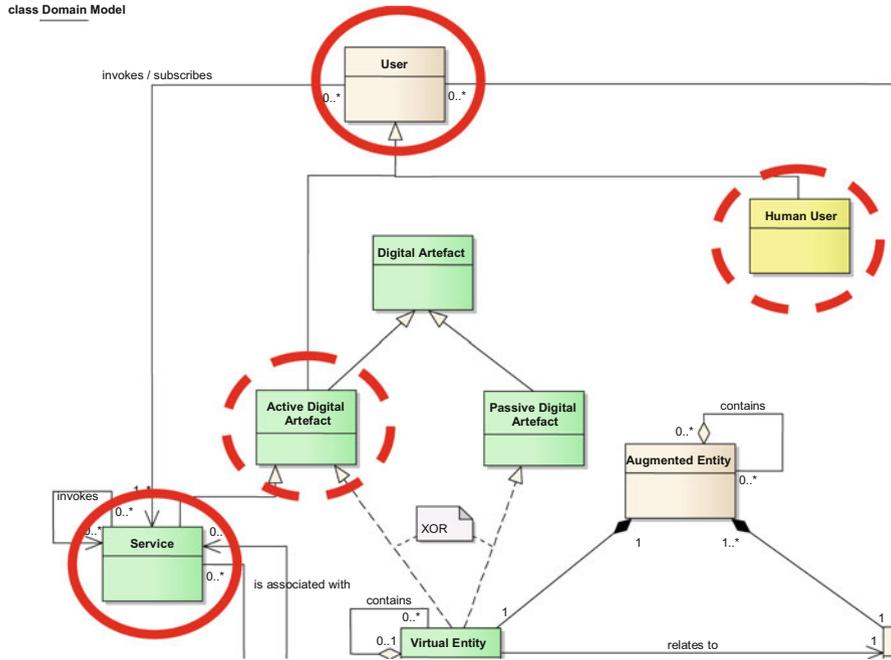
Fig. 7.14 DM entities involved in a User-Service / Service-Service interaction (zoom of the whole IoT Domain Model in Fig. 7.7)

also assume the user role when invoking another Service, thus Users can either be Human User or Active Digital Artefacts.

This interaction is straightforward, as it is identical to typical Internet interactions between Users and Services. In fact, in most of the application scenario the User-Service connection can be established using standard Internet protocols.

However, two main exceptions to this general assumption apply when two Services communicate one to each other and one or both of the communicating elements belong to a constrained network.

The latter case, which applies when Services are deployed on constrained Devices such as Sensor nodes and when the User of a given Service is deployed on a constrained Device, requires for the use of constrained communication protocols (see Rossi 2012). Finally, when the two elements belong to different sub-networks, gateway(s) and/or proxy(ies) must be deployed for ensuring successful end-to-end communication. To this extent, as a general rule, if a Service is constrained, or if it needs to provide access to constrained Users, it must be accessible with constrained protocols (e.g., 6LoWPAN, UDP, CoAP, etc.).

Fig. 7.15 IoT DM entities involved in a Service / Resource / Device interaction (This is a zoom of the complete IoT DM highlighting the interested parts, see also Fig. 7.7)

7.6.1.2 Service / Resource / Device Interactions

Figure 7.15 illustrates the entities of the IoT Domain Model involved in the interactions among Services, Resources and Devices. These interactions can be exemplified with the communication among the Alarm Service, the Alarm Resource and the Mote Runner Node of the recurring example. This Figure also illustrates the interconnections of these entities.

The complexity of this interaction is due to variety of different properties that a Device can have; in particular, a Device can be as simple and limited as a Tag and as complex and powerful as a server (Tag Terminal Type (TT3) and unconstrained Terminal Type (TT1), respectively, in (Rossi 2012)). In fact, while powerful Devices can easily support the needed software to host and access Services and to expose the needed Resources for other Services to interact with, simpler Devices

may only be able to provide access to their own Resources and the simplest may even not be powerful enough even to support this. In the latter two cases, Resources and Services must be provided somewhere else in the network, by some other more powerful Device(s).

Thus the IoT Communication Model helps to model and to analyse how constrained Devices can actively participate in an IoT-A compliant communication and to study possible solutions, such as the usage of application layer gateways, to integrate legacy technologies.

7.6.2   Communication Interoperability Aspects

The model we are going to propose in this section takes its roots from the ISO/OSI (ISO 1994) stack, the US Department of Defense 4 layer model (DoD4) (Darpa 1970) and, the Internet stack, but it puts its focus on IoT specific features and issues. All the previous models have a great value, going beyond any discussion, but simply they have not been conceived with the IoT issues and features in mind.

In Fig. 7.16 we can see the Internet and the DoD4 stacks. It is evident how they map onto each other, thus in what follows we will address the 4 layers Internet model only.

The 4-layer Internet stack abstracts from the underlying Physical Layer; in fact its lowest layer is represented by the Link Layer. This choice is indeed the right one for the Internet, as the Link Layer is not visible from the Application Layer, and the same can be applied to fully homogeneous networks, since applications can be totally agnostic of the underlined common physical technology. However, the Physical Layer rises to a great importance when talking about the IoT; in fact the IoT is characterized by a high heterogeneity of hardware and technologies and the necessity of making different system interoperable. Moreover, this is a clear statement on the fact that IP is conceived in order to be implemented on top of any hardware networking technologies, while in the IoT there exist technologies that do not dispose of the needed resources to manage a complete IP compliant communication. Thus, solutions such as 6LoWPAN, are needed to extend IP communication to constrained networks.

Moreover the main objective of the 4-layer Internet model is to let Internet applications communicate, having intermediate devices understanding the communication at IP level, without meddling with upper layers. This model is wonderful in its simplicity, but this simplicity is one of the reasons why it is unsuitable for the IoT, since it is not able to address the aforementioned interoperability issues.

Obviously this dates back to the beginning of the Internet, when developing an Application Protocol for each application was best practice. While in principle that is a reasonable approach, even in the current Internet we can perceive how it is misleading. We are not here to discuss pros and cons of developing an Application

Fig. 7.16 Four layers
Internet stack (*left*) and
DoD4 stack (*right*)
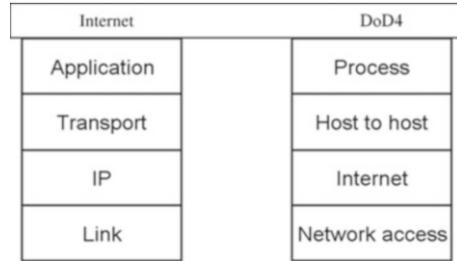
Protocol for each application but we have just to notice this is not a common practice anymore, with the majority of applications leveraging on HTTP or even on more specific HTTP constructs like REST protocols. So nowadays it is crucial to have a distinction between Applications and the Application Protocols.

Another major issue of the 4 layers Internet model arises from the lack of expression for the so called security layers, the two major examples being SSL (IETF 2011)/TLS (IETF 2008) and IPsec (IETF 1998).

The main reference point for communication system developers is the ISO/OSI stack, and, although its validity as an implementation guideline is out of question, it fails to depict the overall complexity of IoT systems as it is meant to represent single technology stacks.

After the considerations on the model discussed so far we felt necessary a different approach for highlighting the peculiar features of IoT communication, which are not directly evident using the ISO/OSI model alone.

The model, as depicted in Fig. 7.17 on the left-hand side, stresses the relevance of the wide range of interoperability aspects that characterise IoT systems. In fact, instead of focusing on a specific realisation of the communication stack, the IoT Communication Model provides a transversal approach from which one or more communication stacks can be derived: in fact a single interoperability aspect can be used to describe the interactions of stacks belonging to different communicating systems. Once a system is modelled according to the IoT Communication Model it is easy to derive a set of ISO/OSI interoperable stacks in order to provide the needed interoperability features.

Below, the different interoperability aspects are described:

- **Physical aspect**: This interoperability aspect concerns the physical characteristics of the communication technologies used in the system. It is similar to the OSI Physical Layer. This is necessary in order to neither exclude any available technology, and to prevent emerging solutions from being integrated into the Reference Model. This aspect does not force the adoption of any specific technology, but it uses the adopted technologies as a base to model the remaining of the system. In fact, as per the recurring example the Mote Runner Node can communicate using some low-power radio transceiver such as ZigBee, while the AndroidApp can be hosted in an IoT-Phone connected to the Internet either via WiFi or 3G cellular networks;

Fig. 7.17 The interoperability aspects of the IoT Communication Model (*left*) compared to the ISO/OSI communication stack (*right*): dashed lines have been used for the IoT aspects to make them graphically different from stack layers

IoT Aspects

Data

End to end

Network & ID

Link

Physical

ISO Stack

Application

Presentation

Session

Transport

Network

Data Link

Physical
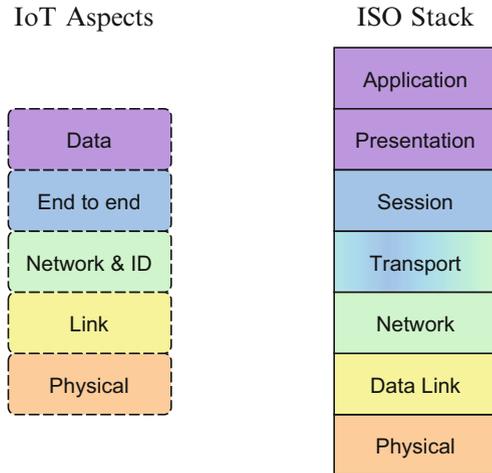
- **Link aspect**: In order to address the heterogeneousness of networking technologies represented in the IoT field, the Link aspect requires special attention. In fact, most networks implement similar, but customised communication schemes and security solutions. In order for IoT systems to achieve full interoperability, as well as the support of heterogeneous technologies and a comprehensive security framework, this layer must support solution diversity. At the same time, it needs to provide upper layers with standardised capabilities and interfaces. Therefore, this layer needs to abstract a large variety of functionalities, enabling direct communication. IoT systems do not have to restrict the selection among data link layers, but must enable their coexistence;
- **Network and ID aspect**: This interoperability aspect combines two communication aspects: *networking*, which provides the same functionalities as the correspondent OSI layer; and *identifiers*, which are provided using resolution functionalities between locators and IDs. In order to support global manageability, interoperability, and scalability, this aspect needs to provide a common communication paradigm for every possible networking solution. This is the narrow waist for the Internet of Things. The difference between identifiers (unique descriptors of the Digital Artefact; either active or passive), and locators (descriptors of the position of a given IoT element in the network), is the first convergence point in the IoT Communication Model. Thus, this interoperability aspect is in charge of making any two systems addressable from one another notwithstanding the particular technologies they are adopting. In the case of our recurring example the AndroidApp must be able to receive alarms generated by the alarm Service, which in turns, must receive information from the Mote Runner Device: in order for this to be possible the system must ensure that the correct identifiers are supported by all the communicating technologies or can be resolved via appropriate methods;

- **End-to-end aspect**: this aspect takes care of reliability, transport issues, translation functionalities, proxies/gateways support and parameter configuration when the communication crosses different networking environments. By providing additional interoperability aspects on top of those of the Network and ID aspect, this aspect provides the final component for achieving a global M2M communication model. Connections are also part of the end-to-end scope. Also, Application Layer aspects are taken care of here. Moreover Application Protocols in the IoT tend to embed confirmation messages, and congestion control techniques require being more complex than what is achievable in the Transport Layer in the legacy models. With reference to the recurring example, this aspect will take care of modelling the overall communication between the Alarm Service and the Mote Runner Node and between the AndroidApp and the Alarm Service;
- **Data aspect**: the topmost aspect of the IoT Communication Model is related to data definitions and transfers. While the Information Model provides a high-level description for data of IoT systems, the purpose of this aspect is to model data exchange between any two actors in the IoT. As described in the IoT Information Model (see Sect. 7.4), data exchanged in IoT can adopt many different representations, ranging from raw data to complex structures where meta-information is added to provide context specific links. Finally, to make this possible, the data aspect needs to model the following characteristics (Rossi 2013): (1) capability of providing structured attributes for data description; (2) capability of being translated (possibly by compression/decompression) the one to each other, e.g. CoAP is translatable to HTTP by decompression or XML is translatable to EXI by compression, IPv4 is translatable to IPv6 by mapping; (3) constrained device support. For instance, in the recurring example, the raw data produced by the Mote Runner Sensors shall be converted into machine-readable formats in order for the Alarm Service to be able to interpret and use them.

7.6.3 Composed Modelling Options

Actual networks may need more than a single communication stack that can be arranged in several configurations: in particular, here we will analyse how two of the most popular configurations can be modelled according to the IoT Communication Model. In the following we will refer to (1) *gateway configuration* as the composition of two or more protocol stacks that are placed side by side across different media, and (2) *virtual configuration* as the composition of two or more protocol stacks, one on top of the other.

7.6.3.1   Gateway Configuration

In this configuration, the IoT Communication Model describes the overall communication behaviour of the system so that any two communicating element can be seen seamlessly connected.

Figure 7.18 provides a graphical example of the modelling of three protocol stacks in gateway configuration. In this example, the two end-point Application Layers can communicate thanks to the gateways which maps the underline stacks.

In particular, the first gateway (on the left of the figure) bridges the communication between an Ethernet and a WiFi network, while the second (on the right), in addition to the bridging functionality between WiFi and ZigBee, adds a translation functionality for converting IP to 6LoWPAN, TCP to UDP, HTTP to CoAP and vice versa.

This gateway configuration may be used in the recurring example to let the Mote Runner Node communicate using ZigBee technology with the Alarm Service deployed in a server farm thanks to the two gateways.

While the actual configuration of the different protocol stacks is out of the scope of the model, the overall behaviour of the system can be modelled according to the five interoperability aspects described above.

7.6.3.2   Virtual Configuration

In this configuration the IoT Communication Model aims at describing the overall communication behaviour of a system, where the actual communication path is virtualised by tunnelling the communication using a second protocol stack.

Figure 7.19 exemplifies the modelling of a system behaviour using a virtual configuration: here, there is an inner communication path composed of an Ethernet network and a WiFi network using a bridging block and an outer communication path that is independent of the inner path and allows for the two application layers to communicate. Such a scheme is usually realised using virtual private network solutions.

7.6.4   Channel Model for IoT Communication

This model aims at detailing and modelling the content of the channel in the Shannon-Weaver model in the context of the IoT domain. This model does not pretend to capture every possible characteristic of IoT technologies, but provides a common ground to be used to compute overall system performance and for benchmarking. Further models have to be considered in order to account for more specific physical aspects.
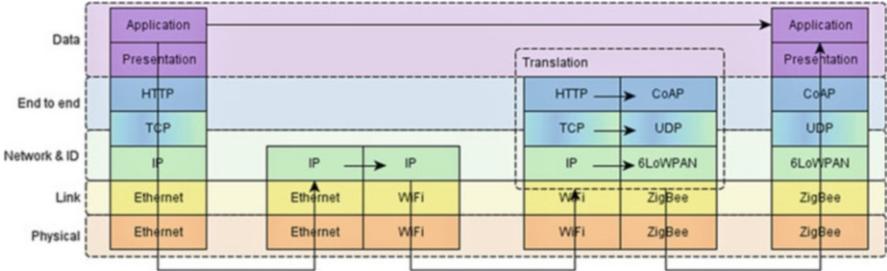
Fig. 7.18 Gateway configuration for multiple protocol stacks, aligned according to the IoT aspect model (see Fig. 7.17)
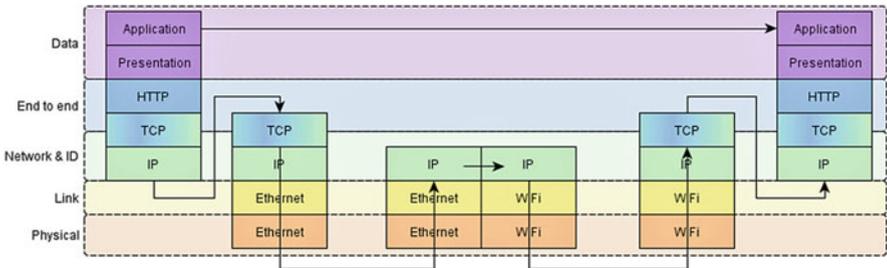
Fig. 7.19 Virtual configuration for multiple protocol stacks

Figure 7.20 depicts end-to-end abstraction of a packet delivery between distant Devices. The information source can be abstracted as a resource in the IoT Domain Model, and the transmitter as a Device; while the receiver and destination pair can be mapped as a Device-Service pair.

Following this abstraction, and pushing it forward, we focus on the channel modelling. In the IoT context, the channel can assume a multiplicity of forms.

Please notice that the following abstraction is useful in order to have an abstract description but when it comes to apply the Shannon-Hartley theorem it is crucial to remember this theorem has to be applied independently to each link composing the path between the sender and the receiver: $C_I=B_I \log(1+S_I/N_I)$, where C_I is the channel capacity, B_I is the channel bandwidth, S_I/N_I is the signal-to-noise ratio (or the carrier-to-noise ratio in case of modulated signals), each of them related to the I-th link. This channel capacity metric is concave and it can be aggregated according the following rule: $C_{i,k}=\min(C_{i,j},C_{j,k})$, where $C_{i,k}$ is the aggregated capacity from i to k, while $C_{i,j}$ is capacity of the link from i to j and $C_{j,k}$ is the capacity of the link between j and k.

Given two adjacent channels, which require to be connected by the means of a gateway, their aggregated capacity is extremely useful in order to dimension the gateway itself. Nonetheless, assuming you cannot control the routing on the Internet the scope is limited to the portion of links of which you know the characteristics, or for a link of which you can suppose to know the lower bound. A valid assumption
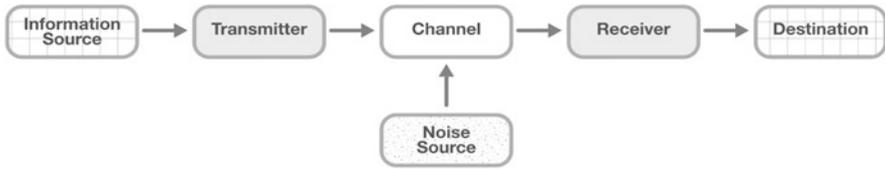
Fig. 7.20 Schematic diagram of a general communication system

will be anyway that the aggregated capacity cannot be bigger than the capacity of the known links, providing a strong tool to avoid over-dimensioning the gateways. Indeed, this is extremely useful when designing a constrained network and its ingress and its egress.

It is important to point out that there is a distinction between the channel model in the current Internet and that of the IoT. The former is depicted in Fig. 7.21 below, where the Internet block acts as a black-box summarising every channel transformation that may happen between the two gateways.

To proceed in modelling the channel in IoT it is important to give a definition of what we call constrained and unconstrained networks:

- **Unconstrained networks** are characterised by high-speed communication links (e.g., offering transfer rates in the Mbit/s range or higher) like, for instance, the wired Internet of today. Link-level transfer latencies are also small and mainly impacted by congestion events in the network rather than by the physical transmission technology;
- **Constrained networks** are characterised by relatively low transfer rates, typically smaller than 1 Mbit/s, as offered by, e.g., IEEE 802.15.4. These networks are also characterised by large latencies. These are due to several factors including: (1) the involved low-bitrate physical layer technology and (2) the power-saving policy of the terminals populating these networks, which may imply the periodic power off of their radios for energy-efficiency reasons.

According to this *heterogeneous networks* can be seen as the combination of *constrained* and *unconstrained networks* linked together via gateways and/or proxies.

The picture is much different in the IoT. As can be seen in the scenarios depicted in (Rossi 2013), in the simplest IoT case the channel consists of a single constrained network (e.g. a WSAN island), as depicted in Fig. 7.22.

In a slightly more complicated case, the IoT channel can consist of several constrained networks, which can rely on different network technologies (see Fig. 7.23).

A different case consists of a channel embodied by a constrained network and an unconstrained one (see Fig. 7.24).

An additional case consists of a channel formed by two constrained networks intermediated by an unconstrained network. A common implementation of this case us the most important in the IoT: the one involving two constrained networks linked by the Internet (see Fig. 7.25).

Fig. 7.21 Channel model for the current Internet

Fig. 7.22 IoT channel for a single constrained network

Fig. 7.23 IoT channel for communication over two constrained networks

What makes IoT very peculiar is the nature of the constrained networks it relies on. Such networks are formed by constrained Devices and the communication between the Devices can:

1. Be based on different protocols;
2. Require additional processing in the gateways.

It is important to point out that the characteristics of each network can have a noticeable impact on the overall end-to-end communication.

In the previous section we tackled the channel capacity using the Shannon-Hartley theorem and the min operation in order to aggregate multiple hops. Obviously the channel capacity is not the only important metric when modelling the IoT communication.

7.7 Trust, Security, Privacy

IoT systems integrate in a seamless way physical objects, data, and computing devices into a global network of information about 'smart things'. In this scenario, services act as bridges through which these 'smart things' interact with each other in an automated way and with as less human intervention as possible. Towards our aim to provide a Reference Architecture for IoT systems, it becomes thus mandatory to discuss potential security issues and define a security model for our architecture. On the way to our goal we proceed as follows: we identify a few separate classes of security properties that we deem important for an IoT system and provide, for each class, tools and mechanisms that serve as solid foundations upon which we can build complex solutions that guarantee those properties.

Fig. 7.24 IoT channel for communication constrained to unconstrained networks

Fig. 7.25 IoT channel for communication over two constrained networks intermediated by the Internet

Considering the multi-faceted entities that a IoT system is made of, we spot the following necessary properties: Trust, Security, Privacy, and Reliability. In the remainder of this chapter we discuss these properties separately and delineate, for each of them, a reference model within the framework of our architecture.

7.7.1 Trust

An important aspect of IoT systems is the fact that they deal with sensitive information (e.g. patients' electronic health records). The entities and services therein recurrently process, store, retrieve, and take decisions upon this type of data. In this scenario, enforcing trust – compliance to an expected functional behaviour – on all entities, protocols, and mechanisms an IoT system is made of becomes a "must".

Within this project, we focus on Trust at application-level. In particular, we aim at defining a Trust Model that provides data integrity and confidentiality, and endpoint authentication and non-repudiation between any two system-entities that interact with each other.

Trust Model Mandatory Aspects

Describing all possible trust-model archetypes is out of the scope of this document. Nonetheless, we list hereafter a few and basic aspects that we believe to be mandatory for defining a Trust Model for IoT systems:

- The Trust-Model **domains**: In complex systems that include multi-faceted entities, like, e.g., the IoT, a model that equally shapes the Trust of all components is difficult, if not impossible, to define. For this reason, various domains within the system should be determined, with every domain defining a specific set of subjects to which certain aspects of the trust model apply;
- Trust-**evaluation mechanisms**: They define a coherent and safe methodology for computing the trustworthiness degree of a subject within the system. Evaluation mechanisms are based on information previously collected on the given subject. Depending on the application scenario, this information can be obtained

by direct experiences with the subject, witness information on the subject coming from other members of a community, social-network analysis providing sociological information on the subject and so on. A trust-evaluation mechanism must take into account the source of the information on which the trust value is being computed, i.e. the trustworthiness of the source itself, and carefully weight its information accordingly in computing the final trust value;

- **Behaviour policies**: They regulate the ways two subjects within the same Trust Model domain interact according to their trustworthiness value. They define how subjects that use the system may interact with other subject. E.g., if a wireless sensor A is asked to handle a multi-hop message coming from a sensor B with a very low trust value, Sensor A might decide, according to the behaviour policies defined by the Trust Model, to not accept the message from Sensor B. Though it is not recommended, a Trust Model can define specific behaviours for interacting with subjects whose trust-value cannot be computed within that model;

- **Trust anchor**: It is a subject trusted by default (possibly after authentication) by all subjects using a given Trust Model, and exploited in the evaluation of third parties' trustworthiness. In the IoT environment the trust anchor can either be local to a given subnetwork – running on a node in the same peripheral network, e.g. a gateway – or a global and centralised device that is deployed on the Internet;

- **Federation of trust**: It delineates the rules under which trust relationships among systems with different Trust Models can be defined. The federation of trust is essential in order to provide interoperability between subjects that use different Trust Models. The federation of trust becomes particularly important within an IoT system deployed on a large scale, where the coexistence of many different Trust Models it is very likely;

- **M2M support**: The interaction among autonomous machines is deemed very common in IoT systems. Prior dynamically identifying and accessing resources of one-another, these machines should be able to autonomously, according to the specifics in the Trust Model, evaluate the trustworthiness of each-other.

7.7.2 Security

Now that we have discussed the fundamental aspects that will be included in our Trust Model, in this section we provide a generic overview of the Security reference model in our architecture.

Our Security reference model is made of three layers: the Service Security layer, the Communication Security layer and the Application Security layer. The former, described in details in (Gruschka and Gessner 2012), includes the following components: Authorization, Identity Management, Trust and Reputation, Authentication, and key exchange and management. In the remaining of this section we detail the two last layers.

7.7.2.1   Communication Security

IoT systems are heterogeneous. Not only because of the variety of the entities involved (data, machines, sensors, RFID, and so on), but also because they include Devices with various capabilities in terms of communication and processing. Therefore, a Communication Security Model must not only consider the heterogeneity of the system, but it also should guarantee a balance between security features, bandwidth, power supply and processing capabilities (Rossi 2012).

Here we work under the assumption that the IoT device space can be divided into two main categories: constrained networks (NTU) and unconstrained networks (NTC) (see Networks and communication entities, Chap. 2 in (Rossi 2012)). The domain of constrained devices contains a great heterogeneity of communication technologies (and related security solutions) and this poses a great problem in designing a model encompassing all of them. Examples for such communication technologies can be found in (Rossi 2012)).

To mitigate the aforementioned heterogeneities we could provide a Communication Security Model with a high degree of abstraction. However, it would be useless, as it would lack the specifics needed in the moment of implementing a specific IoT architecture. As in the Communication Model (see Sect. 7.1.6), we address the problem by introducing profiles that group heterogeneous Devices into groups characterised by given specifications.

Figure 7.26 above depicts our approach to lower-layer security in IoT. We exploit gateways:

On the edge between the domains of unconstrained and constrained devices, gateways have the role of adapting communication between the two domains. Gateways are unconstrained devices; therefore, they can be exploited to boost up the security of constrained devices by running on their behalf energy-hungry and complex security mechanisms. In addition, gateways can also be used in order manage security settings in peripheral NTC networks.

We enable these functionalities in the gateways by extending them with the following features:

- Protocol adaptation between different networks (by definition);
- Tunnelling between themselves and other nodes of the NTU domain. (Optional; impacts on trust assessment);
- Management of security features belonging to the peripheral network (Optional);
- Description of security options related to traffic originated from a node attached to the gateway. (Authentication of source node, cryptographic strength, . . .);
- Filtering of incoming traffic (i.e. traffic sent to one of the nodes attached to the gateway or vice-versa) according to network policies, user-defined policies, and destination-node preferences (Optional).
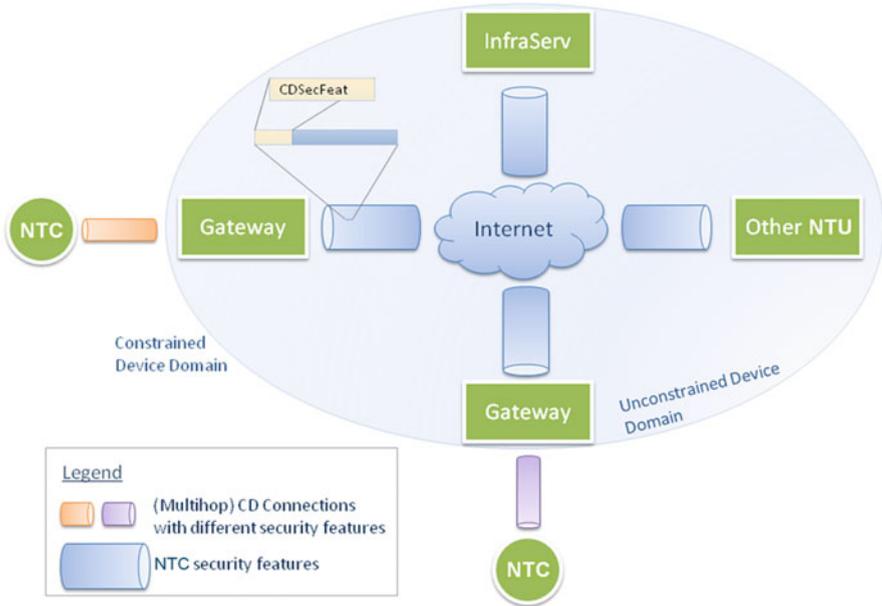
Fig. 7.26 *NTC* Constrained Device Network, *NTU* Unconstrained Device Network, *CDSecFeat* Constrained device security feature. The CDSecFeat implementation leverages the extension of the functionalities of gateway devices

7.7.2.2 Application Security: System Safety and Reliability

IoT systems include, without any doubt, a wide range of application scenarios: from home-security to structure monitoring of bridges, buildings, and so on, and from surveillance systems to health monitoring. Most of these scenarios must be reliable: a single failure in the system can lead to tragic consequences. This is why, besides from security and privacy mechanisms that guarantee trustworthiness of the system as a whole, it becomes important to assure also system safety.

System safety is application specific: for an electricity system safety includes assuring that no harm is done in case of a short circuit. For an elevator system safety would include making sure that the elevator does not start moving when the elevator doors are opened. Nonetheless, there is a common approach to achieve fail-safe systems made of two phases. The first, called the hazard identification phase, aims at detecting all possible risks that could possibly lead to severe accidents. The second phase includes the system design according to the fail-safe philosophy: systems are designed in a way that the far majority of failures will simply result in a temporary or total loss of service, so to avoid major damage/accidents. An example of a safe-fail system is the security belt sensor in smart-cars: If the driver does not fasten it, the car does not start.

While we believe that the classical fail-safe approach to system design can assure safety in IoT systems, with respect to hazards inside the system (e.g. the

security belt within the car, the short circuit within the electricity system and so on), we also believe that often, the safety of the system depends on issues that originate outside the system. The following scenario gives a representative example of outside-the-system hazards: A bulldozer aiming at bringing down a tree damages (by chance) the foundations of a building nearby. Even though the damage is not visibly spottable right away, at the first slight earthquake it makes the building crumble down by thus costing human lives.

Clearly, in these cases, threat analysis plays an important role. Despite from considering only system-insider hazards, the system designer should carefully examine the 'outside world' of the system in order to identify potential outside hazards. Only after a meticulous analysis of all possible threats (both insiders and outsiders) proceed with the system design following the fail-safe philosophy.

Lastly, another group of vicious threats imposed to safety, or rather, reliability of IoT systems are terroristic. These can either aim at bringing down large automatic systems e.g. a city or country wide electricity system, internet connectivity, border security monitoring system and so on, or targeting directly the users (e.g. by wirelessly reprogramming pacemakers of patients[2]). In the former case, the attack consequences could be limited by including intrusion/failure detection mechanisms (e.g. heart-beat protocols) coupled with redundancy that brings the targeted service up in a short-time period after the attack. In the second case, however, this type of solution might not work well: If the pacemaker of a patient is stopped, even though an alarm might be raised in the IoT system, the patient's life would most probably end in a short time.

7.7.3 Privacy

Due to the variety of the entities that handle user-generated data in IoT, guaranteeing data privacy becomes mandatory in these systems. For this reason we include in our reference model also a Privacy Model, the aim of which is to describe the mechanisms – e.g. access policies, encryption/decryption algorithms, security mechanisms based on credentials, and so on – that prevent data of a subject (either user or entity) to be used improperly.

According to (Weber and Weber 2010), a privacy friendly system should guarantee the following properties:

- The subject must be able to choose sharing or not sharing information with someone else;
- The subject must be able to fully control the mechanism used to ensure their privacy;

---

[2] According to a report published at www.secure-medicine.org, peacemakers can be wirelessly hacked in, and reprogrammed to shut down or to deliver jolts of electricity that would potentially be fatal to patients.

- The subject shall be able to decide for which purpose the information will be used;
- The subject shall be informed whenever information is used and by whom;
- During interactions between a subject and an IoT system, only strictly needed information shall be disclosed about the subject, and pseudonyms, secondary identity, or assertions (certified properties of the end-user) shall be used whenever possible;
- It shall not be possible to infer the subject's identity by aggregating/reasoning over information available at various sources;
- Information gained for a specific purpose shall not be used for another purpose. E.g., the bank issuing a credit card should not use a given client's purchase information (logged so to keep track of that client's account) to send him advertising on goods similar to his purchases.

To provide the above properties the IoT-A privacy model relies on the following functional components: Authentication FC, Trust and Reputation FC.

Table 7.1 below briefly summarizes how these components mitigate some of the privacy threats to privacy, further discussed in the threat analysis performed in IoT-A (see Appendix).

Central to the Privacy Model is the Identity Management Functional Component. A description of this FC is provided in deliverable (Gruschka and Gessner 2012).

In our system, any subject (service or user) is univocally mapped to its root identity. However, a subject might require to be provided with multiple secondary identities by the Identity Manager. The set of multiple identities associated to a unique subject is denoted with identity pool (see Fig. 7.27). Secondary identities can then be used, for privacy or usability purposes, when the subject interacts with the IoT system. However, the system does log the identities (either secondary/pseudo or root identities) of the subjects it interacts with so to mitigate possible Repudiation. The Identity Management FC provides a mapping functionality that maps (to requesters with the required credentials) root identities to secondary identities/pseudonyms.

The second corner-stone functionality for ensuring privacy is *Authentication* (AuthN component).

Its functionality is to bond a subject to its identity (root identity) or to certify properties/roles of the subject, or both. If the subject is a user, examples of possible certified properties can be:

- Has age over 18 years old;
- Has valid driving license;
- Has certification level x.

Similarly, certified roles can be:

- Management;
- Operational;
- Maintenance,...

Table 7.1 Example of privacy threats mitigation within IoT-A

Threat	Result	Mitigation
Identity spoofing	User's identity is spoofed	Robust user authentication procedure preventing man-in-the-middle attacks, with proper credentials-management policy provided by an **Authentication FC**
	User is involved in transactions with a malicious peer	Trustworthy discovery / resolution / lookup system. Trustworthiness of the entire system is guaranteed through its security components (especially **Authentication FC** and **Trust and Reputation FC**) as well as its global robustness (security by design)
Information disclosure	Attacker gains knowledge of user's private parameters	The **Identity Management FC** enforces a robust pseudonymity scheme that ensures anonymity and unlinkability
	Attacker gains knowledge of user's location	User's location can be hidden through reliance on pseudonyms provided by **Identity Management FC**
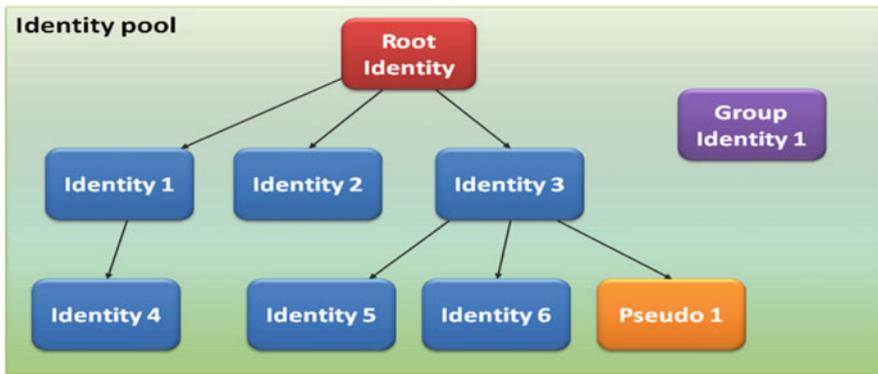
Fig. 7.27 Example of an identity pool

So, in our system, a given subject can be granted access to an IoT resource according to the subject's identification, or according to the subject's certified properties/roles. This enables subjects to still get access to the system yet not revealing their identity.

The AuthN component proposed by IoT-A offers the Authenticate functionality, the profile of which is:

assertion: Authenticate (UserCredential)

where UserCredential is any kind of information used by the Authenticate functionality to check the identity of the party to be authenticated (e.g. username – password pair, PIN code, retinal identification and so on).

assertion (following definition of (Gruschka and Gessner 2012)) is the information that guaranties the occurrence of an authentication of a user client at a particular time using a particular method of authentication. The assertion is further used by the Authorisation (AuthS) component in order to decide upon granting or denying access to a resource.

Finally, the AuthN component provides also *Authorisation* (AuthS): It is the process by which access to information or an IoT Resource is granted to a subject, according to an access policy and for a specific type of action. In order to guarantee user-privacy, the end-users should be in control of access policies relating to their personal data.

The profile of the Authorise function is:

Boolean: Authorise (Assertion, Resource, ActionType),
where Assertion is the result of Authentication, Resource represents the resource to be accessed, and ActionType represents the action to be performed upon the resource.

As mentioned earlier, there are various models of authorisation, property-based access control and assertion-based access control (Gruschka and Gessner 2012). Both are supported by IoT-A through abstract APIs (Gruschka and Gessner 2012).

Identity Management FC, Authentication FC, and Authorisation FC guarantee privacy within the IoT system. Nonetheless, if the data within the IoT system's database is stored as clear text, nothing prevents hackers from tampering with the database and accessing the data. To protect the user against these types of attacks, we believe that the data should be encrypted before storing it in the database.

7.7.4   Contradictory Aspects in IoT-A Security

In distributed systems, including IoT-like ones, one has often to trade-off between security properties. In particular, trust and privacy, are considered as being two contradictory properties. From one side, we want to build a system which is trustworthy. I.e., every entity in that system can prove, according to either trust-building mechanisms or to certificates distributed by some authority, its own trust value. From the other side, we want the system to provide, to each entity, the privacy that it requires, without forcing it to disclosure more personal information that it wants to. This tension between security and privacy emerges also in our reference model. Indeed, the trust-evaluation mechanisms for example not couple well with the many pseudonyms an entity might present to protect its privacy in various scenarios. Indeed, a given malicious entity can fool the system by presenting, within a given context, the pseudonym with the highest trust value built so far. It becomes thus very important to strongly bind, somehow, the trust value of an entity with its root ID. But, from the other side, this imposes problems to the privacy of the entities: If the trust-value has to be calculated on the fly, based on

certificates given to that entity in the many interactions it has had in the past, all bound to its root ID, the entity can be easily traced inside the IoT, even though it presents different pseudonyms. A solution to this problem is to make the trust value be recalculated, each time an interaction occurs, by a unique, trustworthy system component which is also able to bind various pseudonyms to root IDs. This solution does guarantee correct trust values for all entities in the system, yet preserving their privacy. However, it has two major drawbacks: (1) The unique component would become a huge bottleneck in the system; (2) It would become a single point of failure: By compromising it (or tampering with it) an attacker would be able to de-anonymize all entities in the system, or even change trust-values to his liking, by boosting trust-values of malicious entities, and lowering the trust value of others.

For the above reasons, we believe that within the IoT-A system we should opt for a mechanism which trades-off trust for privacy: Subjects are allowed just one trust-value, valid for a certain number of pseudo-identities, and included in a trust-certificate signed by the AuthN component. The trust value is then updated each time the subject interacts in the system, by the counter-part of this interaction. The trust value is to be used for sensitive interactions and/or access to sensitive system resources, data, and services, within which the subject is thus required to present one of the pseudonyms bound to the trust-certificate. This way, a subject cannot fake its unique trust value, which is, from the other side not bound to its pseudonyms "trust-free" – the ones through which the subject can access less system's resources, data, and services, that do not require proof of trust values.

7.8   Conclusion

In this Section we introduced the foundation of the IoT ARM, the IoT Reference Model. The IoT Reference Model defines the basic concepts, models, terminology, and relationships in the IoT ARM. It demonstrates our thinking, rationale and design space for structuring the domain of the Internet of Things. It also proposes the Functional Groups that we deem relevant for IoT architectures, as outlined in the IoT Functional Model (see Sect. 7.1.5).

Within the IoT Reference Model, the IoT Domain Model was discussed in great detail, as the IoT Domain Model defines the language, the concepts, and the entities of the IoT world and how they are related to each other. This is confirmed by the fact, as we learn in Section (sec: Chap. 6 "IoT Context View"), that the IoT Domain Model plays a prominent role in IoT-A-guided system architecting. As we will see in Chap. 12 when we perform a reverse mapping analysis with the concepts defined in other projects and standards related to the Internet of Things, the definition of a common understanding is crucial for developing interoperable architectures and systems. This common understanding permeates every aspect of the architecture, and will be a key aspect for the widespread acceptance of a future IoT systems and standards. In that respect it is most important to carefully study the concepts of the

IoT Domain Model, as it is the foundation of the other models presented in this chapter and of the IoT Reference Architecture that will be discussed in Chap. 8.

While a common language and common terminology is the precondition for all other models, this chapter also provided the other models crucial for the development of IoT architectures, most importantly the IoT Information Model that relates to important aspects of information in an IoT system and will be detailed in the IoT Information View in Sect. 8.2.3 that discusses information on a higher level of detail.

The IoT Functional Model discussed in this chapter defines several Functional Groups that pick up the IoT concepts and entities introduced in the IoT Domain Model and it relates them to common functionalities present in an IoT architecture. Just as for the IoT Information Model and View, the IoT Functional Model will be further detailed with concrete functional components in Sect. 8.2.2.

Finally, Communication and Security models, as well as techniques of system safety and reliability where introduced that address these issues in IoT. The security and the communication model constitute Functionality Groups in the IoT Functional Model, and will be picked up again in the IoT Reference Architecture (see Sects. 8.2.4 and 8.3.3).

What we have also addressed in this chapter, is the application of the common IoT use case introduced in Sect. 4.2 to several models in order to facilitate getting acquainted with the concepts defined in the respective model by tying their understanding together with a common, "Red Thread". We hope that this application of the use case helps with understanding the different models. We are aware of the complexity of the IoT Domain Model and the Trust, Security, and Privacy issues, but this complexity is inherent in the domain of the Internet of things itself. It is however crucial to really understand the models introduced in this chapter, before moving on.

The next Chap. 8, the IoT Reference Architecture, builds upon this foundation and details it even further, so that concrete IoT-compliant architectures can be derived. The section uses several ways of projecting the IoT Reference Architecture, and it also presents several "Views" that complement the different models presented in this section. For instance, we propose Functional Components, which relate to the IoT Functional Model and the IoT Communication Model, in the Functional View (see Sect. 8.2.2) that we discussed in this chapter. We also provide an Information View, which tightly relates to the IoT Information Model discussed in this chapter.

Chapter 8
IoT Reference Architecture

**Martin Bauer, Mathieu Boussard, Nicola Bui, Jourik De Loof,
Carsten Magerkurth, Stefan Meissner, Andreas Nettsträter, Julinda Stefa,
Matthias Thoma, and Joachim W. Walewski**

M. Bauer (✉)
NEC Europe Ltd., NEC Laboratories Europe, Software & Services Research Division,
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
e-mail: Martin.Bauer@neclab.eu; www.nw.neclab.eu

M. Boussard
Alcatel-Lucent Bell Labs France, Route de Villejust, 91620 Nozay, France
e-mail: mathieu.boussard@alcatel-lucent.com; www.alcatel-lucent.com

N. Bui
Consorzio Ferrara Ricerche, Via Savonarola 9, 44122 Ferrara, Italy
e-mail: buincl@unife.it; www.unife.it

J. De Loof
Alcatel-Lucent Bell N.V., Copernicuslaan 50, 2018 Antwerpen, Belgium
e-mail: jourik.de_loof@alcatel-lucent.com; www.alcatel-lucent.com

C. Magerkurth • M. Thoma
SAP AG, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany
e-mail: carsten.magerkurth@sap.com; www.sap.com; Matthias.thoma@sap.com;
www.sap.com

S. Meissner
University of Surrey, Stag Hill, GU2 7XH Guildford, UK
e-mail: s.meissner@surrey.ac.uk; www.surrey.ac.uk

A. Nettsträter
Fraunhofer Institute for Material Flow and Logistics IML, Joseph-von-Fraunhofer Str. 2-4,
44227 Dortmund, Germany
e-mail: andreas.nettstraetter@iml.fraunhofer.de; www.iml.fraunhofer.de

J. Stefa
Università Sapienza di Roma, P.le Aldo Moro 5, 00185 Rome, Italy
e-mail: stefa@di.uniroma1.it; www.cattid.uniroma1.it

J.W. Walewski
Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
e-mail: joachim.walewski@siemens.com; www.siemens.com

In this chapter we present our IoT Reference Architecture. This IoT Reference Architecture is, among others, designed as a reference for the generation of compliant IoT concrete architectures that are tailored to one's specific needs. For other usages of the IoT Architectural Reference Model see Chap. 3.

The IoT Reference Architecture is kept rather abstract in order to enable many, potentially different, IoT architectures. Guidance on how to use all the parts of the IoT Reference Architecture can be found in Chaps. 5, 6, 9, 10, and 11.

Both in devising this chapter and in presenting the outcomes of our deliberations, we are adhering to the framework of architectural views and perspectives, as described in the software engineering literature and standards (for more details see (Rozanski and Woods 2011)). The use of well-known concepts makes it easier for architects from other domains to feel comfortable in the IoT world and this framework was thus a rather natural choice. To be more precise, we used the definitions of views from (Woods 2008), as well as their architectural-perspective catalogue. We adopted both according to IoT-specific needs. One has to be careful though, about the definition of views and viewpoints as these differ between authors. Nonetheless, there are no conceptual differences to traditional approaches and someone with a background in designing any kind of system should not have a steep learning curve. Notice though that architectural views and perspectives were originally defined for concrete architectures and not for reference architectures. Views that are very use-case dependent, for instance the IoT Physical Entity view and the context view, are therefore not covered here. For a more detailed discussion of this aspect see Chap. 5. Furthermore, since a reference architecture covers a wide range of use cases, it is of course void of use-case-specific details (for instance usage patterns and the related interactions of the system's functional components), such aspects are not covered in the IoT Reference Architecture but have to be attended during, for instance, the architecture-generation process.

The structure of the chapter is as follows: First, we give a short overview on architectural views and perspectives. We then go on with presenting views that constitute the IoT Reference Architecture. The functional view and its viewpoints are described in great detail. At the time of writing there was indeed so much information at hand that we decided to only present an overview of the functional view here and to cover, for instance, the detailed definitions of the functional components of the functional-decomposition viewpoint in Carrez et al. (2013). Next, the information view is introduced as well as the deployment and operational view. The remainder of the chapter is then devoted to architectural perspectives. We describe four architectural perspectives (evolution and interoperability; performance and scalability; trust, security, and privacy; and availability and resilience). How architectural perspectives influence the architecting process is not covered here but in Chap. 6.

8.1 Short Definition of Architectural Views and Perspectives

A system architecture, and thus by default, a reference architecture, needs to answer a wide range of questions. Such questions can, for instance, address:

- Functional elements.
- Interactions of said elements.
- Information management.
- Operational features.
- Deployment of the system.

What the user of an architecture expects, is an architectural description, viz. "a set of artifacts that documents an architecture in a way its stakeholders can understand and that demonstrates that the architecture has met their concerns" (Rozanski 2005b). Instead of providing these artifacts in a monolithic description, one often chooses to delineate them by so-called architectural views. The idea behind so doing is to focus on system aspects that can be conceptionally isolated. Architectural views make both the derivation of the architecture and its validation easier. The above bullet-point list provides examples of such views. A more detailed discussion of views and how we adapted them to the reference-architecture realm is provided in the next section.

In the past it has been found that views are unfortunately not enough for describing system architectures and that many stakeholder aspirations are rather of a qualitative nature (Rozanski and Woods 2011). Such qualitative aspirations cut across more than one view. Such aspirations are referred to architectural perspectives, of which privacy is but one example. A more detailed discussion of architectural perspectives is provided in Sect. 8.8.

The joint use of architectural views and perspectives in architecture descriptions is described in more detail in the pertinent literature (Rozanski and Woods 2011).

8.2 Architectural Views

Views are used during the design and implementation phase of a concrete system architecture. They are defined in the following way:

> A view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders (Rozanski and Woods 2011).

A view is composed of viewpoints, which aggregate several architectural concepts in order to make the work with views easier. The IEEE standard 1471 defines viewpoints as follows:

> A viewpoint is a collection of patterns, templates, and conventions for constructing one type of view. It defines the stakeholders whose concerns are reflected in the viewpoint and the guidelines, principles, and template models for constructing its views (IEEE Architecture Working Group 2000).

Some typical examples for viewpoints are:

- **Functional view**: functional-decomposition viewpoint; interaction viewpoint; interface viewpoint;
- **Information view**: information-hierarchy viewpoint; semantics viewpoint; information-processing viewpoint; information-flow viewpoint.

8.2.1 Usage of Views and Perspectives in the IoT Reference Architecture

As mentioned in the introduction to this chapter, the IoT Reference Architecture is use-case- and application- independent and is therefore not compatible to the concept of views and viewpoints one-by-one. But the idea behind the concept is nevertheless helpful and was thus adopted for the use within the IoT Reference Architecture. As discussed above the following views were left out from the IoT Reference Architecture but are discussed in Chap. 5:

- Physical Entity View and
- Context View.

Concerning the Functional View, of the above three viewpoints, interactions are not covered in the IoT Reference Architecture, since the number of arrangements of the Functional Components and also their invocation is practically infinite. Instead, we chose to cover some typical – but yet high-level – interaction patterns (see Sect. 8.5).

The same is true for the deployment and operational View. However, there are aspects to both that are practically invariant over the IoT domain and these aspects are covered in Sect. 8.7. Also, what is an aspect of the deployment view in one architecture can be an aspect of the operation view in another architecture. Situating these aspects in either or is contingent on, among others

- Requirements (usability; institutional rules and traditions; . . .) and
- Design choices made (commission on manufacturing floor; shipment and installation by experts; operation by experts).

Fig. 8.1 Functional view
process
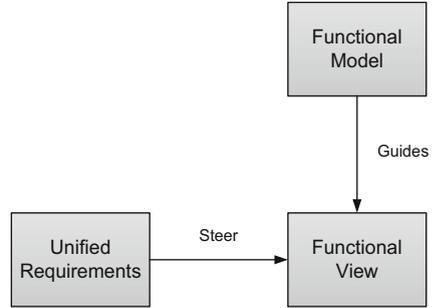
The following sections present the IoT Functional View, IoT Information View, and the IoT Deployment and Operational view of the IoT Reference Architecture.

8.2.2 Functional View

8.2.2.1 Devising the Functional View

The functional view is defined by applying the methodology defined in Chap. 5 to functional decomposition as can be seen in Fig. 8.1.

In a first step, the Unified Requirements are mapped to the different Functionality Groups of the IoT Functional Model.

Next, clusters of requirements of similar functionality are formed and a Functional Component for these requirements defined.

Finally, the Functional Components are refined after discussion with the technical work packages.

The viewpoints used for constructing the IoT Functional View are hence:

1. The Unified Requirements;
2. The IoT Functional Model.

Once all Functional Components are defined, the default function set, system use cases, sequence charts and interface definitions are made, which all can be found back in Carrez et al. (2013).

The Functional View diagram is depicted in Fig. 8.2 and shows the nine functionality groups of the Functional Model. Note that:

- The Application FG and Device FG are out-of-scope of the IoT-A Reference Architecture and are coloured in yellow;
- Management FG and Security FG are transversal FGs and are coloured dark blue.

For each of the Functionality Groups, the *Functional Components* (FC) are depicted.
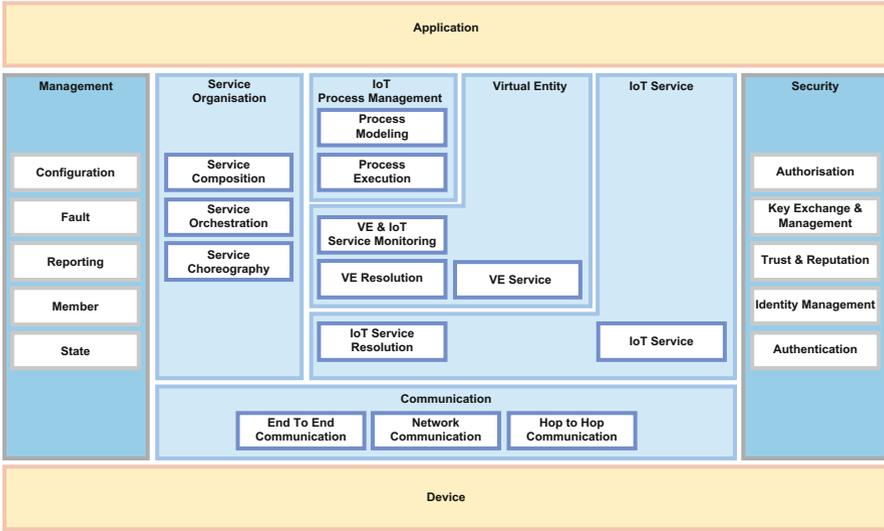
Fig. 8.2 Functional-decomposition viewpoint of the IoT Reference Architecture

In the following sub-sections, the FC's of each FG will be described in more detail.

The Functional View presented in this chapter will give a description of the Functional Components, but will not describe the interactions taking place between the Functional Components.

The reason is that these interactions are typically depending on Design Choices which are not made at this level of abstraction.

Chapter 10 will go more into detail and depict some typical interaction scenarios.

In addition to the description in this chapter, more detailed information such as requirement mapping, system use cases, interaction diagrams and interface definitions can be found in Carrez et al. (2013).

8.2.2.2   IoT Process Management

The IoT Process Management FG relates to the integration of traditional process management systems with the IoT ARM. The overall aim of the FG is to provide the functional concepts and interfaces necessary to augment traditional (business) processes with the idiosyncrasies of the IoT world.

The IoT Process Management FG consists of two Functional Components (see Fig. 8.3 below):

- Process Modelling;
- Process Execution.

Fig. 8.3 IoT process
management
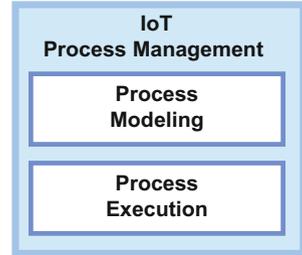
The **Process Modelling FC** provides an environment for the modelling of IoT-aware business processes that will be serialised and executed in the Process Execution FC.

The main function of the Process Modelling FC is to provide the tools necessary for modelling processes using the standardised notation, i.e. using novel modelling concepts specifically addressing the idiosyncrasies of the IoT ecosystem (Meyer 2012).

The **Process Execution FC** executes IoT-aware processes that have been modelled in the Process Modelling FC described above. This execution is achieved by utilising IoT Services that are orchestrated in the Service Organisation layer.

The Process Execution FC is responsible for deploying process models to the execution environments: activities of IoT-aware process models are applied to appropriate execution environments, which perform the actual process execution by finding and invoking appropriate IoT Services.

The Process Execution FC also aligns application requirements with service capabilities. For the execution of applications, IoT Service requirements must be resolved before specific IoT Services can be invoked. For this step, the Process Execution FC utilises components of the Service Organization FG.

Finally, the Process Execution FC can run applications: after resolving IoT Services, the respective services are invoked. The invocation of a service leads to a progressive step forward in the process execution. Thus, the next adequate process based on the outcome of a service invocation will be executed.

8.2.2.3 Service Organisation

The Service Organisation FG (see Fig. 8.4) is the central Functional Group that acts as a communication hub between several other Functional Groups. Since the primary concept of communication within the IoT ARM is the notion of a Service, the Service Organisation is used for composing and orchestrating Services of different levels of abstraction.

The Service Organisation FG consists of three Functional Components:

- Service Orchestration;
- Service Composition;
- Service Choreography.

Fig. 8.4 Service
organisation
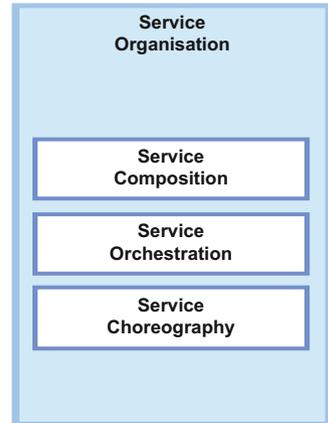
The **Service Orchestration FC** resolves the IoT Services that are suitable to fulfil service requests coming from the Process Execution FC or from Users.

Its only function is to orchestrate IoT Services: resolve the appropriate services that are capable of handling the IoT User's request. If needed, temporary resources will be set up to store intermediate results that feed into Service Composition or complex event processing.

The **Service Composition FC** resolves services that are composed of IoT Services and other services in order to create services with extended functionality. The Functional Component has two main functions: (1) support flexible service compositions and (2) increase quality of information.

To support flexible service compositions, the Service Composition FC must provide dynamic resolution of complex services, composed of other services. These combinable services are chosen based on their availability and the access rights of the requesting user.

Quality of information can be increased by combining information from several sources. For example, an average value – with an intrinsically lower uncertainty – can be calculated based on the information accessed through several resources.

The **Service Choreography FC** offers a broker that handles Publish/Subscribe communication between services. One service can offer its capabilities at the FC and the broker function makes sure a client interested in the offer will find the service with the desired capabilities.

Also service consumers can put service requests onto the Choreography FC while a suitable service is not available at the time when the request was issued. The service consumer will get notified as soon as a service became available that fulfils the service request issued before.

Fig. 8.5 Virtual entity
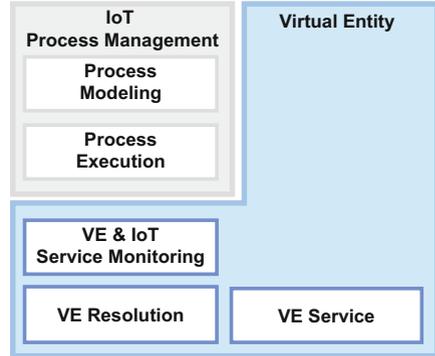
8.2.2.4   Virtual Entity

The Virtual Entity FG (see Fig. 8.5) contains functions for interacting with the IoT System on the basis of VEs, as well as functionalities for discovering and looking up services that can provide information about VEs, or which allow the interaction with VEs. Furthermore, it contains all the functionality needed for managing associations, as well as dynamically finding new associations and monitoring their validity.

The Virtual Entity FG consists of three Functional Components:

- VE Resolution;
- VE & IoT Service Monitoring;
- VE Service.

The **VE Resolution FC** is the Functional Component which provides the functionalities to the IoT User to retrieve associations between VE's and IoT Services.

This includes the discovery of new and mostly dynamic associations between VE and associated services. For the discovery qualifiers, location, proximity, and other context information can be considered. If no association exists, the association can be created.

The User can also subscribe or unsubscribe to continuous notifications about association discovery that fit a provided specification of the VE or of the Service. In case of a notification, a callback function will be called.

Similar, the User can subscribe or unsubscribe to notifications about association lookup.

The VE Resolution FC also allows to lookup VE-related services, i.e. search for services exposing resources related to a VE.

Finally, the VE Resolution FC allows managing associations: insert, delete and update associations between a VE and the IoT Services that are associated to the VE.

The **VE & IoT Service Monitoring FC** is responsible for automatically finding new associations, which are then inserted into the VE Resolution FC. New

associations can be derived based on existing associations, Service Descriptions and information about VE's.

The functions of the VE & IoT Service Monitoring FC are to assert static associations, i.e. create a new static association between VE's and services described by the provided association, discover dynamic associations, i.e. create a new dynamic or monitored association between VE's and Services, update the association and delete the association from the VE Resolution framework.

Finally, the **VE Service FC** handles with entity services. An entity service represents an overall access point to a particular entity, offering means to learn and manipulate the status of the entity. Entity services provide access to an entity via operations that enable reading and/or updating the value(s) of the entities' attributes. The type of access to a particular attribute depends on the specifics of that attribute (read only/write only or both).

A specific VE service can provide VE history storage functionality, to publish integrated context information (VE context information – dynamic and static), VE state information, VE capabilities.

The two functions currently defined for the VE Service FC are to read and set an attribute value for the entity.

It is not required to have an explicit register for Virtual Entities, but the VE Resolution FC could be extended to be used in this way. The important aspect is to agree on how to assign identifiers to Virtual Entities. For modelling any other aspect of the Virtual Entity, a Virtual Entity service can be used that gives you access to all information about a Virtual Entity. This can be current sensor information, as well as historic information. Historic information would typically be stored in a database, which can be modelled as a Network Resource (see Sect. 7.3.3).

8.2.2.5 IoT Service

The IoT Service FG (see Fig. 8.6) contains IoT services as well as functionalities for discovery, look-up, and name resolution of IoT Services. It consists of two Functional Components:

- IoT Service;
- IoT Service Resolution.

An IoT Service exposes one Resource to make it accessible to other parts of the IoT system. Typically, IoT Services can be used to get information provided by a resource retrieved from a sensor device or from a storage resource connected through a network. An IoT Service can also be used to deliver information to a resource in order to control actuator devices or to configure a resource. Resources can be configurable in non-functional aspects, such as dependability security (e.g. access control), resilience (e.g. availability) and performance (e.g. scalability, timeliness).

Fig. 8.6 IoT service
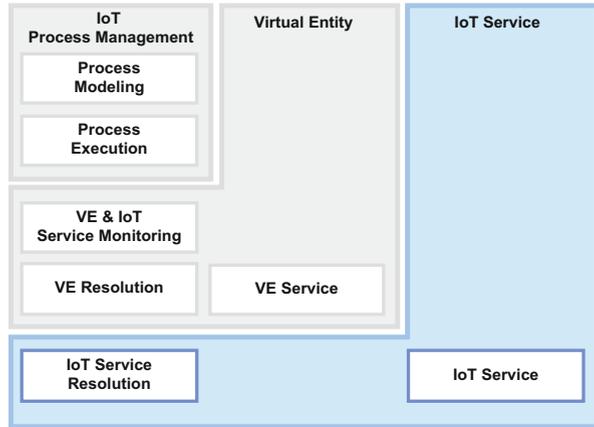
IoT Services can be invoked either in a synchronous way by responding to service requests or in an asynchronous way by sending notifications according to subscriptions previously made through the service.

A particular type of IoT Service can be the Resource history storage that provides storage capabilities for the measurements generated by resources.

The main functions of the **IoT Service FC** are to (1) return information provided by a resource in a synchronous way, (2) accept information sent to a resource in order to store the information or to configure the resource or to control an actuator device and (3) subscribe to information, i.e. return information provided by a resource in an asynchronous way.

The **IoT Service Resolution FC** provides all the functionalities needed by the user in order to find and be able to contact IoT Services. The IoT Service Resolution also gives services the capability to manage their service descriptions (typically stored in a database as one entry), so they can be looked up and discovered by the user. The user can be either a Human User or a software component.

Service Descriptions are identified by a service identifier and contain a service locator that enables accessing the service. Typically they contain further information like the service output, the type of service or the geographic area for which the service is provided. The exact contents, structure and representation depend on design choices taken, which is left open at the Reference Architecture level. Examples for service models (structure) and a service description representations can be found in (Martín D2.1 2012).

The functionalities offered by the IoT Service Resolution FC in brief are:

- Discovery functionality finds the IoT Service without any prior knowledge such as a service identifier. The functionality is used by providing a service specification as part of a query. What can be queried based on a service specification depends on what is included in the service description. As described above, this may include the service output, the service type and the geographic area for which the service is provided. The representation of the service specification will

also be linked to the service description, e.g. if the service description is represented in RDF, a service specification based on SPARQL would be appropriate;

- Lookup is a functionality which enables the User to access the service description having prior knowledge regarding the service identifier;
- Resolution function resolves the service identifiers to locators through which the User can contact the Service. A service locators are typically also included in the service description, the resolution function can be seen as a convenience function that reduces the amount of information that has to be communicated, especially if the service description is large and the contained information is not needed;
- Other functionalities provided by the IoT Service Resolution FC are the management of the service descriptions. IoT Services can update, insert or simply delete the service descriptions from the IoT Service Resolution FC. It is also possible that these functions are called by the functional components of the Management FG and not by the IoT Services themselves.

8.2.2.6 Communication

The Communication FG (see Fig. 8.7 below) is an abstraction, modelling the variety of interaction schemes derived from the many technologies belonging to IoT systems and providing a common interface to the IoT Service FG.

The Communication FG consists of three functional components:

- Hop To Hop Communication;
- Network Communication;
- End To End Communication.

The **Hop To Hop Communication FC** provides the first layer of abstraction from the device's physical communication technology. The functional component is an abstraction to enable the usage and the configuration of any different link layer technology.

Its main functions are to transmit a frame from the Network Communication FC to the Hop To Hop Communication FC and from a Device to the Hop To Hop Communication FC. The arguments for the frame transmission can be set; examples of arguments include: reliability, integrity, encryption and access control.

The Hop To Hop Communication FC is also responsible for routing a frame. This function allows routing a packet inside a mesh network such as for instance 802.15.4 (mesh-under routing). Note that this function is not mandatory for all implementations of the Hop To Hop Communication FC. It is required only for meshed link layer technologies.

Finally, the Hop To Hop Communication FC allows to manage the frame queue and set the size and priorities of the input and output frame queues. This function can be leveraged in order to achieve Quality of Service.

Fig. 8.7 Communication

The **Network Communication FC** takes care of enabling communication between networks through Locators (addressing) and ID Resolution. The FC includes routing, which enables linking different network address spaces. Moreover different network technologies can be converged through network protocol translations.

The functions of the Network Communication FC are to transmit a packet from the Hop To Hop Communication FC to the Network Communication FC and from the End To End Communication FC to the Network Communication FC. The arguments for the packet transmission can be configured and examples of arguments include: reliability, integrity, encryption, unicast/multicast addressing and access control.

The Network Communication FC enables as well network protocol translation where it allows translating between different network protocols. Examples would be to translate IPv4 to IPv6 and ID to IPv4. Note that this function is necessary to implement a Gateway.

In case a packet needs to be routed, the Network Communication FC allows finding the next hop in a network. It also allows dealing with multiple network interfaces. The function is not mandatory for all implementations of the Network Communication FC. It is required only on devices with multiple network interfaces.

Another function of the Network Communication FC is to resolve the locator-to-ID where it allows getting a locator from a given ID. The resolution can be internal based on a lookup table or external via a resolution framework.

Finally, the Network Communication FC can manage the packet queue and setup the size and priorities of the input and output packet queues. This function can be leveraged in order to achieve QoS.

The **End To End Communication FC** takes care of the whole end-to-end communication abstraction, meaning that it takes care of reliable transfer, transport and, translation functionalities, proxies/gateways support and of tuning configuration parameters when the communication crosses different networking environments.

The End To End Communication FC is responsible to transmit a message from the Network Communication FC to the End To End Communication FC and from (IoT) Service to the End To End Communication FC. The arguments for the message can be configured and examples include: reliability, integrity, encryption, access control and multiplexing.

A second function of the End To End Communication FC is to cache and proxy. The Cache and Proxy function allows to buffer messages in the End To End Communication FC.

Fig. 8.8 Security

Another function of the FC is to translate end-to-end protocol. The Translate End To End Protocol function allows to translate between different End To End Protocols. An example would be to translate HTTP/TCP to COAP/UDP. Note that this function is necessary to implement a Gateway.

A last function of the FC is to pass the context of protocol translation between gateways. The context could be related to addressing, methods specific for a RESTful protocol, keying material and security credentials.

8.2.2.7 Security

The Security FG (see Fig. 8.8) is responsible for ensuring the security and privacy of IoT-A-compliant systems.

It consists of five functional components:

- Authorisation;
- Key Exchange & Management;
- Trust & Reputation;
- Identity Management;
- Authentication.

The **Authorization FC** is a front end for managing policies and performing access control decisions based on access control policies. This access control decision can be called whenever access to a restricted resource is requested. For example, this function is called inside the IoT Service Resolution FC, to check if a user is allowed to perform a lookup on the requested resource. This is an important part of the privacy protection mechanisms.

The two default functionalities offered by the Authorization FC are firstly, to determine whether an action is authorized or not. The decision is made based on the

information provided from the assertion, service description and action type. Second functionality is to manage policies, such as adding, updating or deleting an access policy.

The **Authentication FC** is involved in user and service authentication. It checks the credentials provided by a user, and, if valid, it returns an assertion as result, which is required to use the IoT Service Client. Upon checking the correctness of the credentials supplied by a newly joining node, it establishes secured contexts between this node and various entities in its local environment.

The two functionalities provided by the Authentication FC are (1) to authenticate a user based on provided credential and (2) to verify whether an assertion provided by a user is valid or invalid.

The **Identity Management FC** addresses privacy questions by issuing and managing pseudonyms and accessory information to trusted subjects so that they can operate (use or provide services) anonymously.

Only one default function is attributed to this FC: to create a fictional identity (root identity, secondary identity, pseudonym or group identity) along with the related security credentials for users and services to use during the authentication process.

The **Key Exchange and Management** (KEM) **FC** is involved to enable secure communications between two or more IoT-A peers that do not have initial knowledge of each other or whose interoperability is not guaranteed, ensuring integrity and confidentiality.

Two functions are attributed to this FC:

- Distribute keys in a secure way. Upon request, this function finds out a common security framework supported by the issuing node and a remote target, creates a key (or key pair) in this framework and then distributes it (them) securely. Security parameters, including the type of secure communications enablement, are provided.
- Register security capabilities. Nodes and gateways that want to benefit from the mediation of the KEM in the process of establishing secure connections can make use of the register security capabilities function. In this way the KEM registers their capabilities and then can provide keys in the right framework.

The **Trust and Reputation Architecture FC** collects user reputation scores and calculates service trust levels.

Again, two default functions are attributed to the FC:

- Request reputation information. This function is invoked at a given remote entity to request reputation information about another entity. As input parameters, a unique identifier for the remote entity (subject), as well as the concrete context (what kind of service) is given. As a result a reputation bundle is provided;
- Provide reputation information. This function is invoked at a given remote entity to provide reputation information (recommendations or feedback) about another entity. As input parameters, a unique identifier for the entity to be

assessed (subject), as well as the concrete context, the given score and a timestamp are given. As a result, the corresponding reputation is provided.

8.2.2.8 Management

Section 7.5.2.5 provides a high-level discussion for the role and the goals of the Management FG, but it does not specify how to functionally parse this group. For guidance on this question we turned to FCAPS, which offers a comprehensive high-level framework for network management (Flextronics 2005). It was, among others, incorporated into an ITU-T recommendation (ITU-T 1997) and it has already been considered for Smart-Grid applications, which are just one example for IoT (Greenfield 2009). The letters F C A P S stand for the functionalities **F**ault, **C**onfiguration, **A**ccounting (Administration), **P**erformance, and **S**ecurity.

Of these functionalities, Fault, Configuration, and Performance cover all the important goals of the Management FG. In this document we choose to make Security a separate functionality group in order to emphasise its importance for IoT. FCAPS was designed with telecommunication applications in mind, while subscriber-based services will be just one of many business models for the IoT. Therefore accounting functionalities will be covered by primary services. However, for administration purposes we introduce the functional components State FC and Member FC. Performance functionality is related to the monitoring of the state of the system and to the adaptation of its configuration, and is therefore incarnated into the Fault, State and Configuration Functional Components. (see Fig. 8.9) illustrates how the high-level goals motivating the creation of a Management FG (see Sect. 7. 5.2.5) map onto the chosen functional components (Table 8.1).

IoT systems differ from pure networking solutions in that they also offer low-level services and support for business administration. An IoT system is thus much more complex than a communication system, and we chose to make the management of FG-specific FCs part of that very FG, while the Management FG is responsible for cross-functionality-group task (see Appendix UNI.703). In other words, it is responsible for the composition and tracking of actions that involve several of the "core FGs" (i.e. not including Device and Application FG). The requirement grounding for the Management FG is based on the extrapolation of a number of communications requirements to system-wide management and behaviours (these requirements can be found in the description of the individual functional components). In addition, if the interaction of the Application and/or Device FG necessitates the composition and tracking of at least two core FGs, such actions are also candidates for the sphere of responsibility of the Management FG.

By exclusion, the following management activities are thus out of the scope of the Management FG. First, activities that only pertain to a single functionality group. An example for this is the management of authorisations in the Security FG. Second, the management of interactions between functionality groups that do not require "external" intervention. An example for the latter are requests between two FGs that can be managed by the requesting FG itself.

Fig. 8.9 Management

Table 8.1 Mapping of the high-level roles of the Management FG (see Sect. 7.5.2.5) onto functional components

High-level goals	Management FCs				
	Fault	Configuration	Reporting	Member	State
Cost reduction	X	X	X	X	
Attending unforeseeable usage issues	X	X	X	X	X
Fault handling	X	X	X	X	X
Flexibility		X	X	X	X

The Management FG (see Fig. 8.9) consists of five Functional Components:

- Configuration;
- Fault;
- Reporting;
- Member;
- State.

The **Configuration FC** is responsible for initialising the system configuration such as gathering and storing configuration from FC's and Devices. It is also responsible for tracking configuration changes and planning for future extension of the system.

As such, the main functions of the Configuration FC are to retrieve a configuration and to set the configuration:

- The retrieve configuration function allows to retrieve the configuration of a system, either from history (latest known configuration) or from the system (current configuration, including retrieval of the configuration of one or a group of Devices), enabling tracking of configuration changes. The function

can also generate a configuration log including descriptions of Devices and FCs. A filter can be applied to the query;

- The set configuration function is mainly used to initialise or change the system configuration.

The goal of the **Fault FC** is to identify, isolate, correct and log faults that occur in the IoT system. When a fault occurs, the respective functional component notifies the Fault FC. Such notification triggers, for instance, are the gathering of more data in order to identify the nature and severity of the problem. Another action can encompass bringing backup equipment on-line.

Fault logs are one input used for compiling error statistics. Such statistics can be used for identifying fragile functional components and/or devices. Also, "performance thresholds can be set in order to trigger an alarm." (Wikipedia 2012a). Performance data is provided by the State FC.

The Fault FC contains functions to handle a fault, to monitor a fault and to retrieve a fault.

The role of the function that handles a fault is to react to fault detection by generating alarms, logging faults, or applying corrective behaviours. Generated alarms can be disseminated to other FCs. This function can also analyse faults and, if requested, start an action sequence that tackles the fault, possibly interfacing with the changeState() function of the State FC. This usually includes command messages sent to other FCs. This function can also set the system back to a previous state by calling the setConfiguration() function in the Configuration FC. One of the actions this might entail is setting back the system to a previous configuration.

Faults can also be monitored by the Fault FC. This function is mainly used in subscription mode where it monitors the errors of the system and notifies subscribers of matching events.

Finally, the Fault FC provides access to the Fault History. For this access, a filter function can be applied.

The **Member FC** is responsible for the management of the membership and associated information of any relevant entity (FG, FC, VE, IoT Service, Device, Application, User) to an IoT system.

It is typically articulated around a database storing information about entities belonging to the system, including their ownership, capabilities, rules, and rights.

This FC works in tight cooperation with FCs of the Security FG, namely the Authorisation and Identity Management FCs.

The Member FC has three default functions: the continuous monitoring of members, the retrieve member function which allows retrieving members of the system complying with a given filter and also allows to subscribe to updates of the membership table fitting a specified filter (e.g. to be notified of all updates to entities belonging to a given owner) and finally the update member function which allows to update member metadata in the membership database and to register or unregister member metadata in the membership database.

The **Reporting FC** can be seen as an overlay for the other Management FCs. It distils information provided by them. One of many conceivable reporting goals is to

determine the efficiency of the current system. This is important since by "collecting and analysing performance data, the [system] health can be monitored." (Wikipedia 2012a). Establishing trends enables the prediction of future issues. This FC can also be utilised for billing tasks.

There is only one default function for the FC: retrieve a report. This function generates reports about the system. Can either return an existing report from the Report History, or generate a new on through calls on the other Management FCs.

The **State FC** monitors and predicts state of the IoT system. For a ready diagnostic of the system, as required by Fault FC, the past, current and predicted (future) state of the system are provided. This functionality can also support billing. The rationale is that Functions/Services such as Reporting need to know the current and future state of the system. For a ready diagnostic of the system one also needs to know its current performance.

This FC also encompasses a behaviour functionality, which forces the system into a particular state or series of states. An example for an action for which such functionality is needed is an emergency override and the related kill of run-time processes throughout the system. Since such functionality easily can disrupt the system in an unforeseen manner this FC also offers a consistency checks of the commands issued by the `changeState` functionality in the State FC.

The functions of the State FC are to change or enforce a particular state on the system. This function generates sequence of commands to be sent to other FCs. This function also offers the opportunity to check the consistency of the commands provided to this function, as well as to check predictable outcomes (through the `predictState` function).

A second function is to monitor the state. This function is mainly used in subscription mode, where it monitors the state of the system and notifies subscribers of relevant changes in state.

Other functions of the FC are to predict the state for a given time, to retrieve the state of the system through access to the state history and to update the state by changing or creating a state entry.

8.2.2.9 Mapping of Functional View to the Red Thread Example

In this section, the "Red Thread" example will be mapped on the Functional View and the main Functional Components used for the example are highlighted as can be seen in Fig. 8.10:

In Fig. 8.10, Functional Components which are used only once, such as during the instantiation of the process model or configuration of devices are indicated in light yellow.

Functional Components which are used at runtime of the use case are indicated in orange.

The example of this section can be described only at a high level, since a concrete architecture and implementation are needed to go into further detail. Also the design choices of the concrete architecture need to be considered.
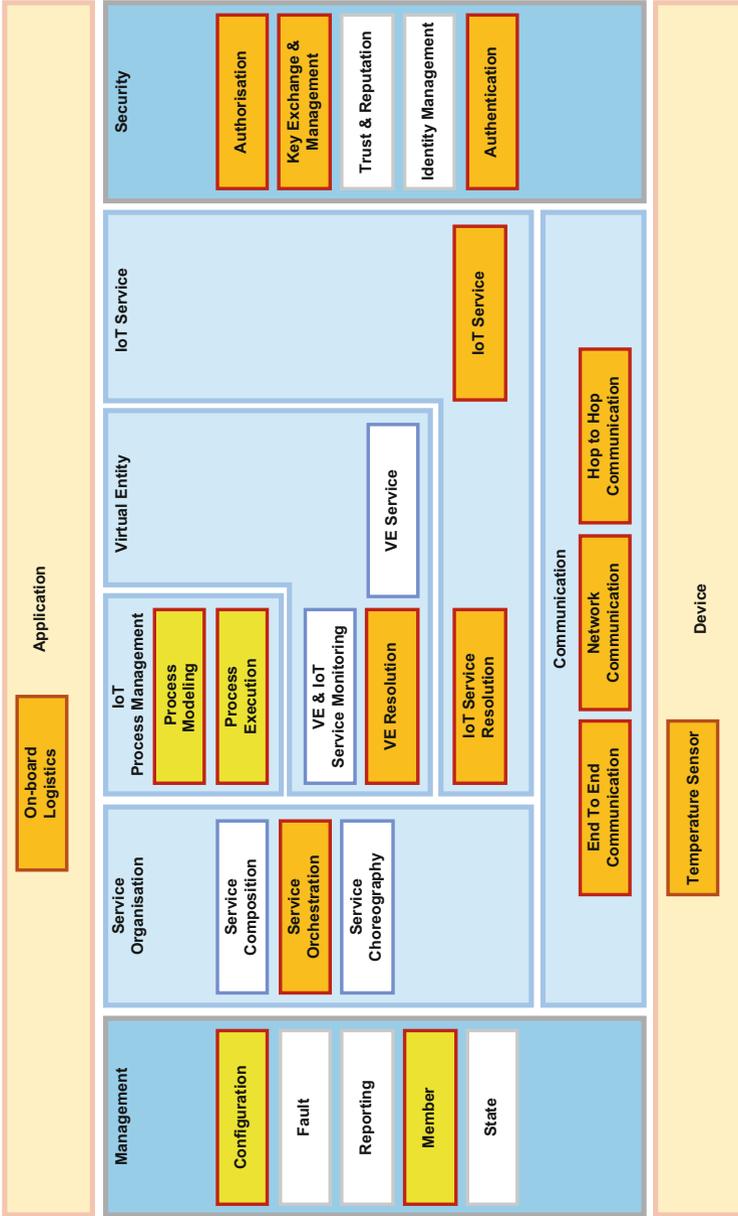
Fig. 8.10 Red-Thread example

In this example, the embedded sensors (Temperature Sensor) continuously measure the environmental conditions within the truck. The measurement data is available to Ted's IoT-Phone (On-board Logistics Application) since the IoT-Phone is subscribed to the service exposing the measurement data

(IoT Service). In order to subscribe to the data, the association between the service exposing the data and the Load carrier needs to be resolved (VE Resolution and IoT Service Resolution). The communication from sensor to IoT-Phone makes use of the network protocol stack of the IoT Communication Model (End To End Communication, Network Communication, Hop to Hop Communication, Key Exchange & Management). All transactions take place in a secure way, meaning that no operations are allowed unless authentication (Authentication) took place and explicit authorisation is obtained for the particular operation (Authorisation).

It is beyond the scope of this section but an illustration of the adaption of the ARM to a specific case and implementation can be found in (Meyer et al. 2013).

8.2.3   Information View

One of the main purposes of connected and smart objects in the IoT is the exchange of information between each other and also with external systems. Therefore the way how to define, structure, store, process, manage and exchange information is very important. The information view helps to generate an overview about static information structure and dynamic information flow.

Based on the IoT Information Model, this view gives more details about how the relevant information is to be represented in an IoT system. As we describe a reference architecture as opposed to a specific system architecture, concrete representation alternatives are not part of this view.

Going beyond the IoT Information Model, the information view also describes the components that handle the information, the flow of information through the system and the life cycle of information in the system.

The current version of the Information View focuses on the description, the handling and the life cycle of the information and the flow of information through the system and the components involved. Given the current level of detail, we will provide a viewpoint only for modelling the type system of Virtual Entities.

8.2.3.1   Information Description

Description of Virtual Entities

The Virtual Entity is the key concept of any IoT system as it models the Physical Entity or the Thing that is the real element of interest. As specified in the IoT IM, Virtual Entities have an identifier (ID), an `entityType` and a number of attributes that provide information about the entity or can be used for changing the state of the Virtual Entity, triggering an actuation on the modelled Physical Entity. The modelling of the `entityType` is of special importance. The `entityType` can be used to determine what attributes a Virtual Entity instance can have, defining its semantics. The `entityType` can be modelled based on a

Driver		Worker		Manager		Box		Pallet	
+	licenseNumber	+	name	+	groupName	+	ID	+	ID
+	name	+	workPlace	+	name	+	stackable		

Fig. 8.11 Example for flat entityType model

flat type system or as a type hierarchy, enabling sub-type matching. Figure 8.11 shows a flat `entityType` model for aspects of the red thread scenario with boxes and pallets as concrete load carriers. Figure 8.12 shows a hierarchical `entityType` model for the same scenario. Here more abstract `entityTypes` have been introduced like `Human` and `LoadCarrier`. The `entityType` `Human` has an attribute *name*, which is inherited by all sub-types, i.e. by `Driver`, `Worker` and `Manager`.

For modelling `entityType` hierarchies, ontologies or UML class diagrams can be used. Of course, this choice is related to the design choice on how the overall Virtual Entity information is represented.

Viewpoint for Modelling entityType Hierarchies

EntityTypes are similar to classes in object-oriented programming, so UML class diagrams as shown above are suitable for modelling entityTypes. As shown in Fig. 8.12 the generalization relation can be used for modelling sub-classes of entityTypes, creating a hierarchy of several entityTypes inheriting attributes from its super-classes. Alternatively, ontology languages like OWL[1] also provide the means for modelling classes and sub-classes, so they can also be used for modelling type hierarchies. This is especially useful, if information in the IoT system is to be modelled using ontologies.

Service Descriptions

Services provide access to functions for retrieving information or executing actuation tasks on IoT Devices. As a basis for finding and interacting with services, services need to be appropriately described, which is done in the form of Service Descriptions. Service Descriptions contain information about the interface of the service, both on a syntactic as well as a semantic level, e.g. the required inputs, the provided outputs or the necessary pre-conditions as well as post-conditions. Furthermore, the Service Description may include information regarding the functionality of the resources, e.g. the type of resource, the processing method or algorithm etc., or information regarding the device on which the resource is running, e.g. it's hardware

---

[1] http://www.w3.org/TR/owl2-overview/
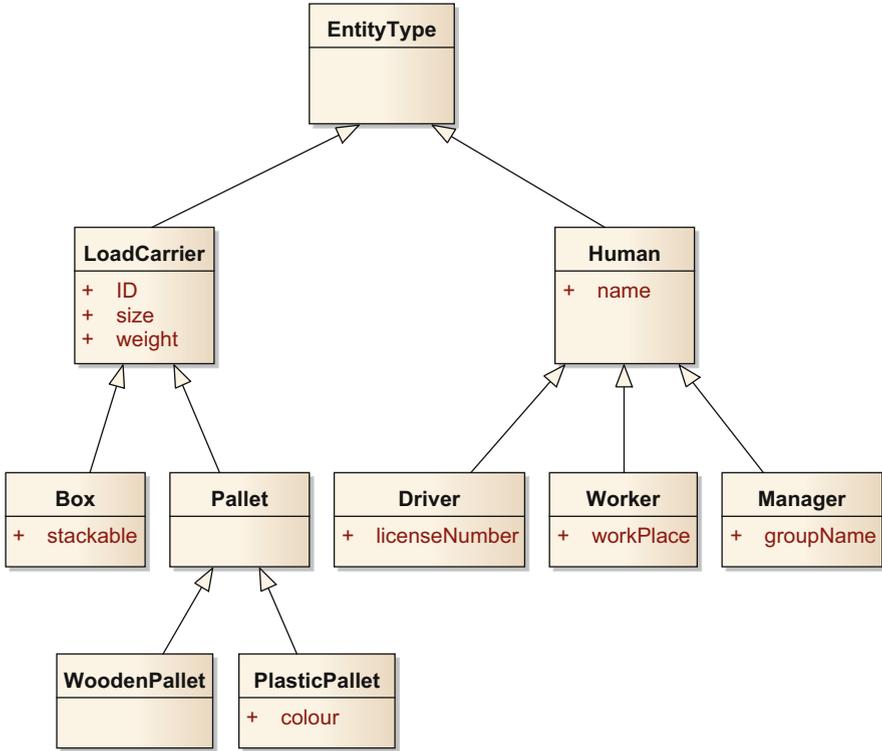
Fig. 8.12 Example for hierarchical entityType model

or its geographical location. Different specification languages for describing services are available, so again, there are different design choices.

Associations Between Virtual Entities and Services

Services can provide information or enable actuation, but the services themselves may not be aware of e.g., which Virtual Entities can provide what information or can enable what kind of actuation. This information is captured by associations that relate to the Virtual Entity and the Service. The association includes the attribute of the Virtual Entity for which the Service provides the information or enables the actuation as a result of a change in its value.

8.2.3.2   Information Handling

Information in the system is handled by IoT Services. IoT Services may provide access to On-Device Resources, e.g. sensor resources, which make real-time

information about the physical world accessible to the system. Other IoT Services may further process and aggregate the information provided by IoT Services/Resources, deriving additional higher-level information. Furthermore, information that has been gathered by the mentioned IoT Services or has been added directly by a user of the IoT system can be stored by a special class of IoT Service, the history storage. A history storage may exist on the level of data values directly gathered from sensor resources as a resource history storage or as a history storage providing information about a Virtual Entity as a Virtual Entity history storage.

IoT Services are registered to the IoT system using Service Descriptions. Service Descriptions can be provided by the services themselves, by users or by special management components that want to make the service visible and discoverable within the IoT system. The IoT Service Resolution is responsible for managing Service Descriptions and providing access to Service Descriptions. In detail, the IoT Service Resolution provides an interface for discovering Service Descriptions based on service specifications given by the requestor, for looking up a Service Description based on the identifier of a service and for resolving a service identifier to a service locator. The latter can also be seen as a convenience function as the Service Description also contains the currently valid service locator.

Associations can be registered with the VE Resolution by services that know for what Virtual Entities they can provide information. The registration can be done by users, by special management components, or by the VE & IoT Service Monitoring component. The VE & IoT Service Monitoring component automatically derives the Associations based on information existing in the system, including Service Descriptions and other associations.

8.2.3.3   Information Handling by Functional Components

The following section describes how information is handled and exposed by the functional components in an IoT-system and shows the information flows between the functional components.

Before going into detail Fig. 8.13 shows the information flow through the Functional Components based on the recurring example from Sect. 4.2. From the actuator on device level the temperature information is transferred to the IoT Service and afterwards to the VE Service. The VE Service itself is described in Sect. 7.4.2. From the VE Service the temperature value is transferred to the AndoidApp via the Subscribe/Notify-pattern.

General Information Flow Concepts

There are four message exchanges patterns considered for information exchange between IoT-A functional components. The first message exchange pattern is the Push-pattern, the second one is the Request/Response-pattern; the third one is the
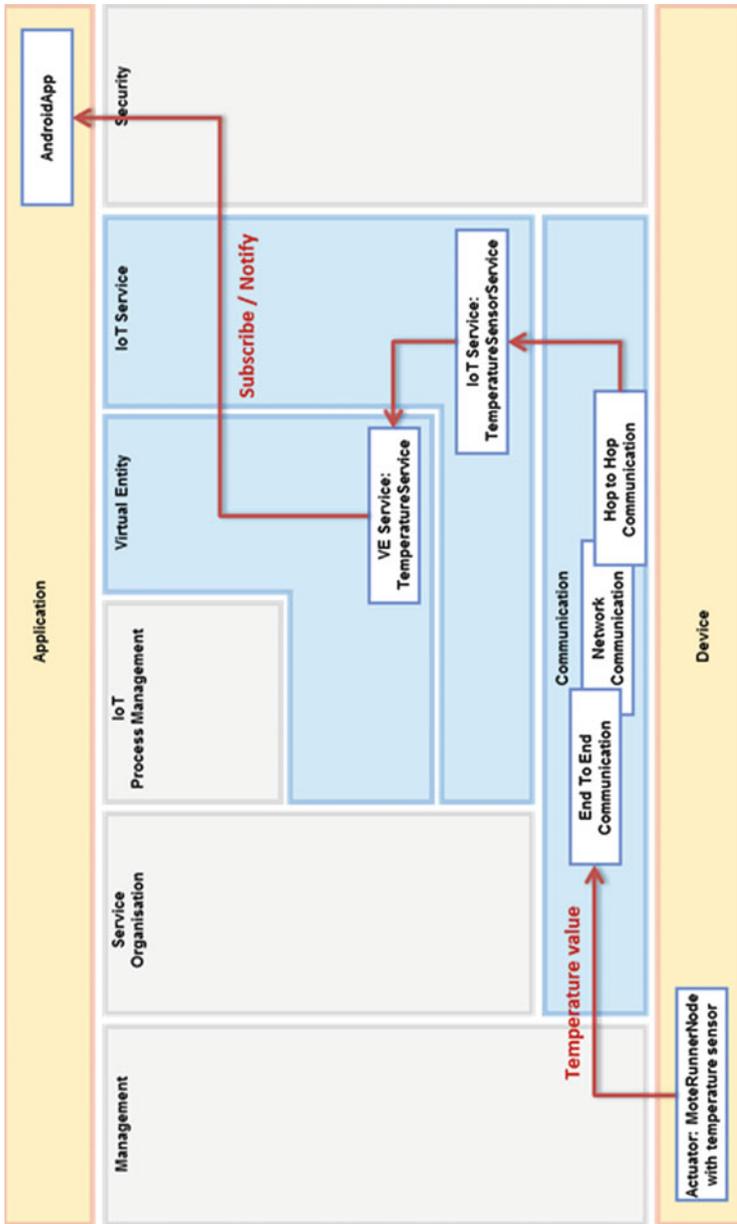
Fig. 8.13 Information flow based on the recurring example

Fig. 8.14 Push-pattern
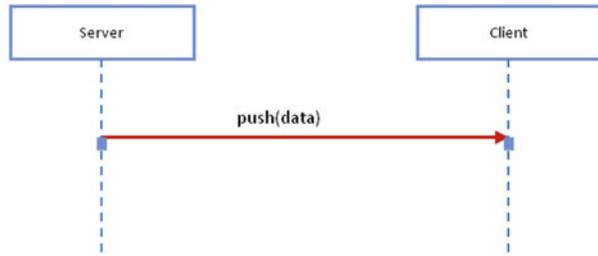
Subscribe/Notify-pattern, and the fourth one is the Publish/Subscribe-pattern. All patterns are explained in the following.

Push

The Push-pattern (see Fig. 8.14) is a one-way communication between two parties in which a server sends data to a pre-defined client that receives the data. The server hereby knows the address of the client beforehand and the client is constantly awaiting messages from the server. The communication channel in this pattern is pre-defined and meant to be applied in scenarios in which the communication partners do not changed often. For example the server can be a constrained device that sends data to a gateway dedicated to this device. The gateway is listening constantly to the device and is consuming the data received from this device.

The Request/Response-pattern (see Figs. 8.15 and 8.16) is a synchronous way of communication between two parties. A client sends a request to a server. The server will receive the request and will send a response back to the client. The client is waiting for the response until the server has sent it.

The server needs some time to prepare the response for the client. In the meanwhile another client might send a request. When the server is still busy with preparing the response for the first client it cannot produce the response for the second client. The second client will be placed into a queue until the server is ready to prepare its response. Such scenario might lead to unacceptable response times.

Subscribe/Notify

The Subscribe/Notify-pattern (see Figs. 8.17 and 8.18) allows an asynchronous way of communication between two parties without the client waiting for the server response. The client just indicates the interest in a service on the server by sending a subscribe-call to the server. The server stores the subscription together with the address of the client wants to get notified on and sends notifications to this address whenever they are ready to be sent.

One advantage of the Subscribe/Notify-pattern over the Request/Response-pattern is the non-blocking behaviour of the subscribe method. The clients can
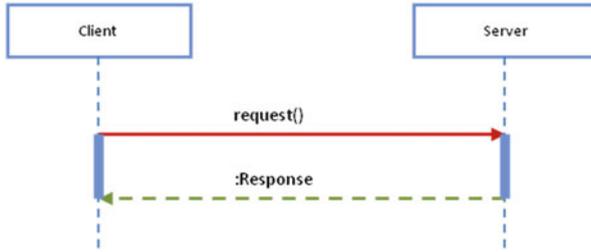
Fig. 8.15   Request/Response-pattern for one client
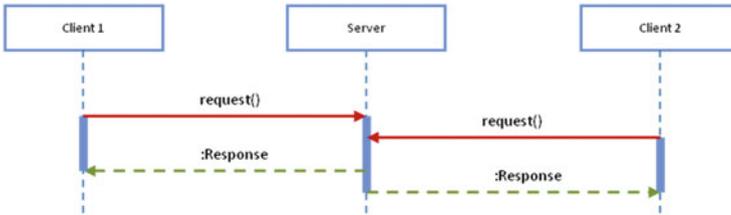
Fig. 8.16   Request/Response-pattern for clients
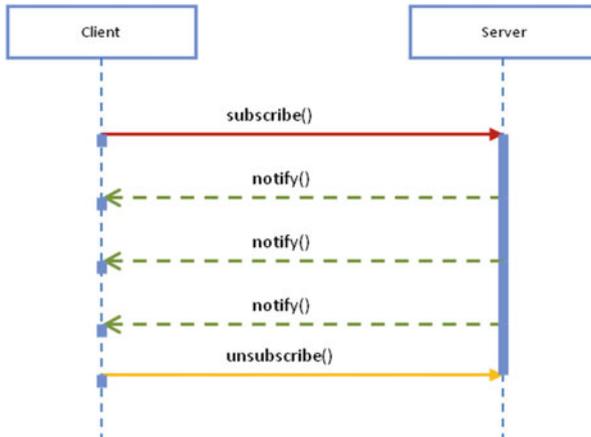
Fig. 8.17   Subscribe/Notify-pattern for one client

continue with other task and need to process the notification only when it arrives. Another big advantage on the server side is that notifications can be multiplied and sent off to clients if the clients have subscribed to the same kind of notifications. To implement the Subscribe/Notify-pattern a server is required that is more powerful compared to the one required for the Request/Response-pattern. The server has to keep records about its subscribers and the kind of subscriptions if it allows several of them.
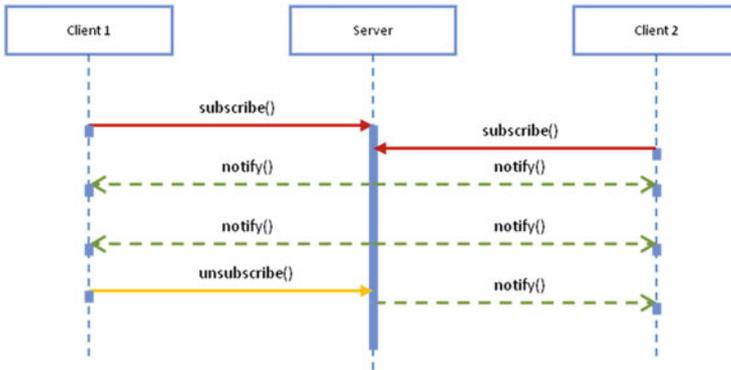
Fig. 8.18 Subscribe/Notify-pattern for two clients

Publish/Subscribe

The Publish/Subscribe-pattern (see Figs. 8.19 and 8.20) allows a loose coupling between communication partners. There are services offering information and advertise those offers on a broker component. When clients declare their interest in certain information on the broker the component will make sure the information flow between service and client will be established.

Services can publish information to the broker regardless how many clients are interested in this information; if no client has subscribed to it the broker does not forward the notification to any client, if more clients have subscribed to the same information the broker will multiply the information and send out notification to each subscriber.

Information Flow Through Functional Components

User Requests Information from IoT Service

Figure 8.21 shows the information request from a user to an IoT Service and the corresponding response.

User Gets Information from Virtual Entity-Level Service

Virtual Entity-level service provides access to Virtual Entity information, augmenting sensor information with entity information (entityId, entityType or several attributes), thus changing the abstraction level. Figure 8.22 shows the Subscribe/Notify-pattern, which can be used to get updates about an Attributes value.
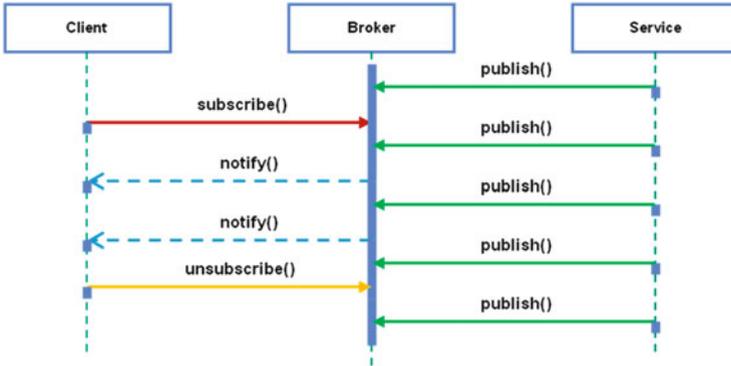
Fig. 8.19 Publish/Subscribe-pattern
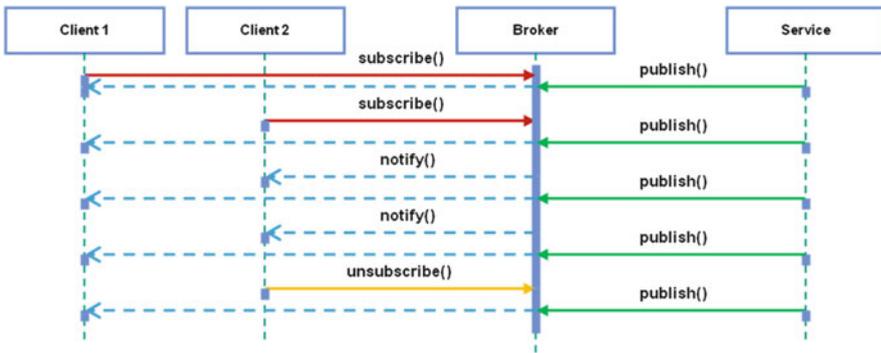
Fig. 8.20 Publish/Subscribe-pattern two clients

Service Gets Sensor Value from Device

The Sensor Device in Fig. 8.23 pushes an updated sensor value using the Functional Component Flow Control & Reliability to an IoT Service. Besides the Push-pattern Request/Response and Subscribe/Notify-pattern are possible. Figure 8.24 shows a similar situation but the information is pushed up to the VE Service.

Sensor Information Storage

Figure 8.25 shows the special case of using an information storage device which stores additional, e.g. historic, values. The IoT Service DataStorage requests values and the StorageDevice sends the corresponding response. The storage policy of the Storage Device is application-specific, e.g. stores values only for certain duration, stores values with reduced granularity over time or in an averaged or aggregated form. Such a storage device can also be used from the VE Service level.
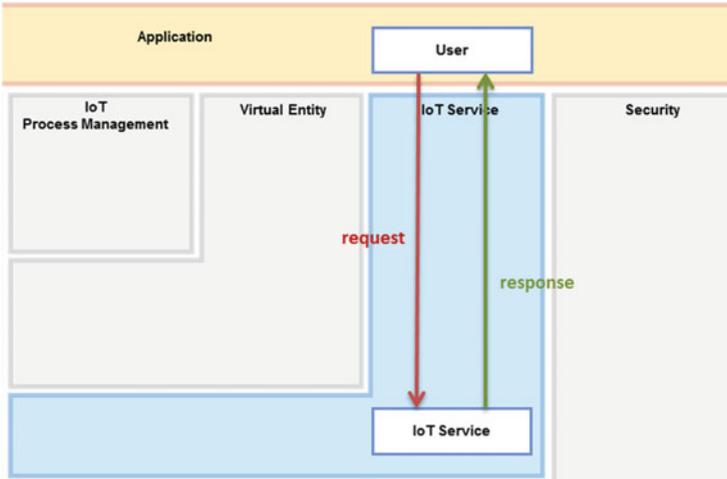
Fig. 8.21 User requests IoT service
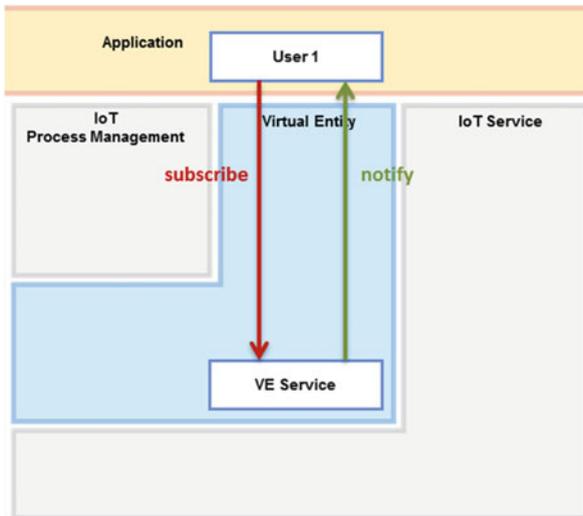
Fig. 8.22 User subscribes for updates of VE-attribute

IoT Service Resolution

The Functional Component IoT Service Resolution hosts the Service Descriptions that are needed for looking up and discovering IoT Services. Thus the resolution component offers methods to insert, update, and delete Service Descriptions (see Fig. 8.26) according to the availability of IoT Services. The methods are meant to be invoked by the IoT Services itself, e.g. upon their deployment, dynamic change of location due to mobility or their undeployment from the system. It is also
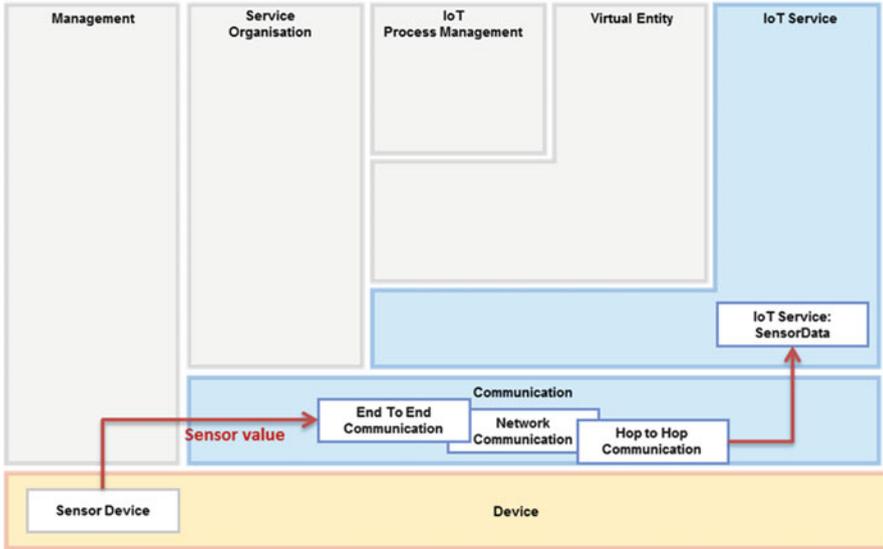
Fig. 8.23 Information flow from sensor device to IoT service using the push-pattern
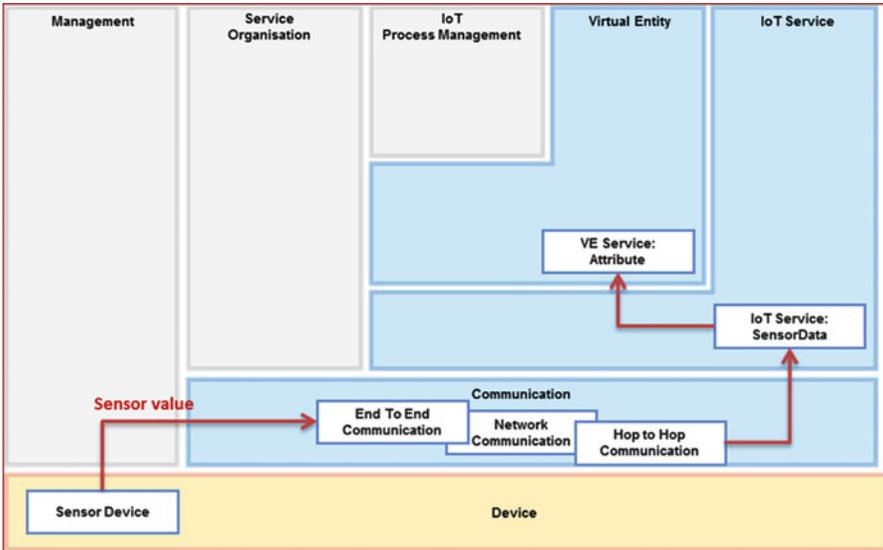
Fig. 8.24 Information flow from sensor device to VE service using the push-pattern

possible for the Service Management component to invoke these methods in order to maintain the system. For deleting a Service Description its Service ID needs to be given.
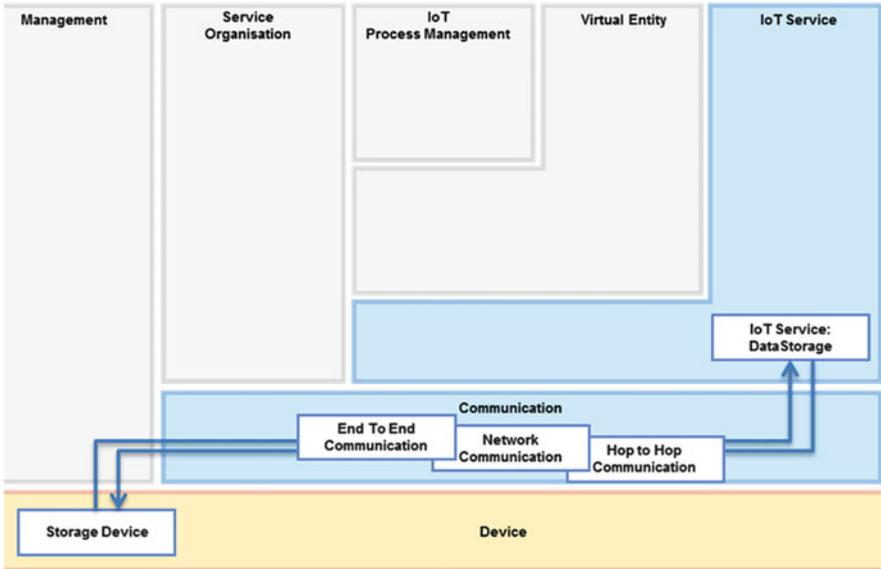
Fig. 8.25 Usage of sensor information storage device
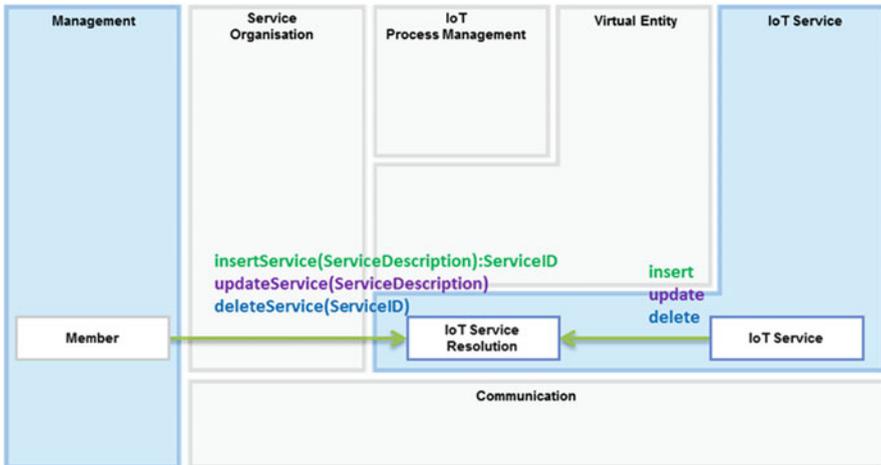
Fig. 8.26 Insert, update, and delete Service Description

The IoT Service Resolution component offers three methods to find IoT Services (see Figs. 8.27 and 8.28):
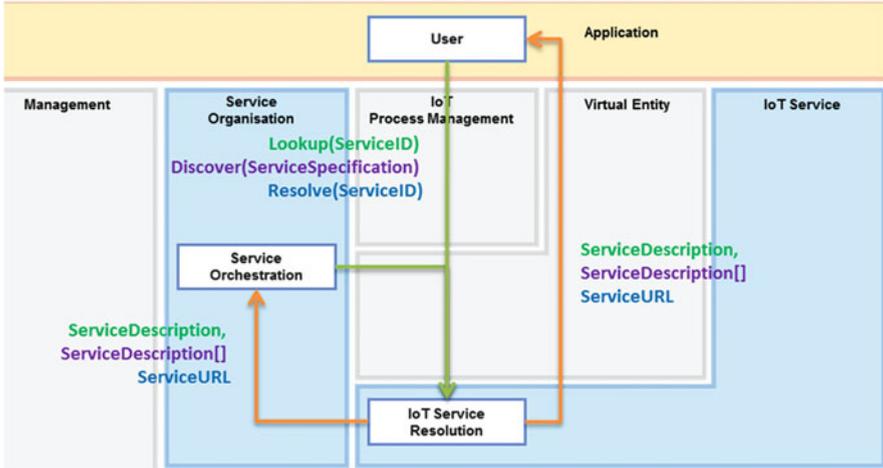
Fig. 8.27 Request lookup, discover, and resolve IoT Services
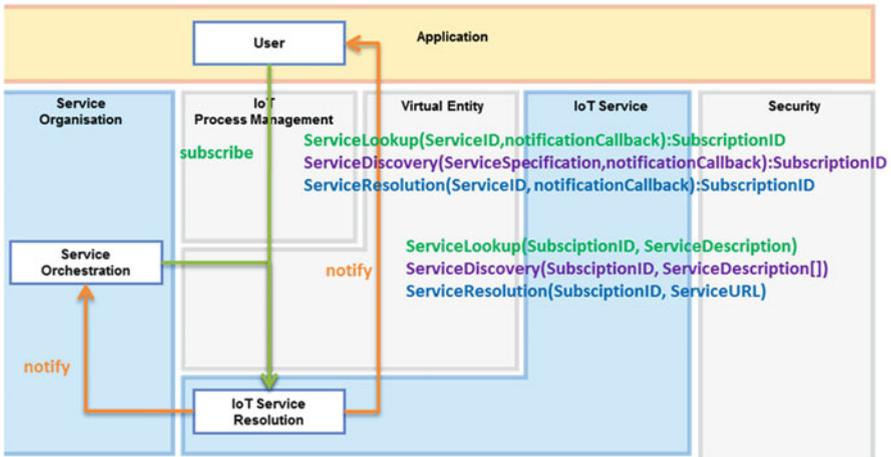
Fig. 8.28 Subscribe to lookup, discover, and resolve IoT Services

1. Look-up of Service Description based on service identifier;
2. Discovery of Service Descriptions based on service specification;
3. resolution of service identifier to service locator (contained in Service Description).

Figure 8.27 shows the different methods in a Request/Response manner, the component also offers similar functionality realised as Subscribe/Notify-pattern. The information flow is similar to the one according to Request/Response, but additionally identifiers for subscriptions and locators for call-back interfaces are exchanged as shown in Fig. 8.28.
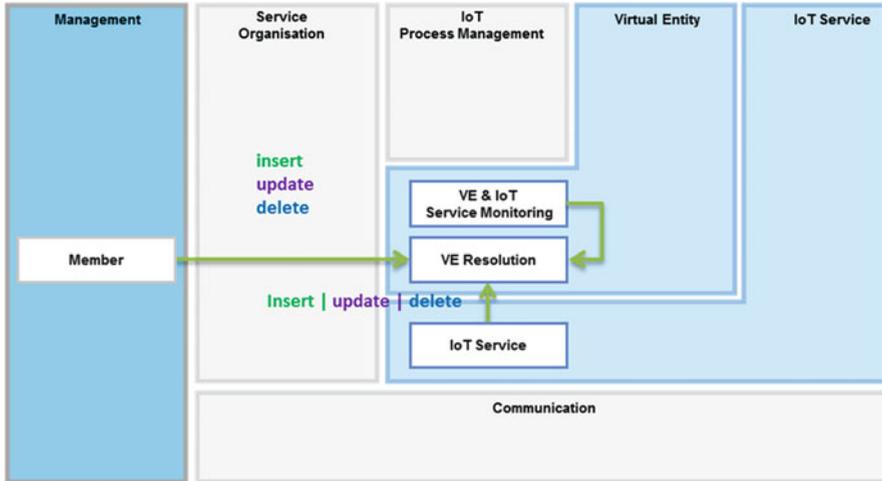
Fig. 8.29 Insert, update, and delete Association

VE Resolution

Associations between Virtual Entities and IoT Services are inserted into VE Resolution by IoT Services, the Service Management components or the VE & IoT Service Monitoring. They can later be updated and eventually deleted, e.g., when the IoT Service is undeployed. The message exchange is shown in Fig. 8.29.

The VE Resolution component allows retrieving of associations between Virtual Entities and IoT Services based on VE identifier and VE service specification through a lookup request as well as discovery of Associations based on VE specification and VE service specification as depicted in Fig. 8.30.

The VE Resolution component provides a information flow while applying the Subscribe/Notify-pattern. With this identifiers for subscriptions and locators for call-back interfaces are exchanged additionally as shown in Fig. 8.31.

8.2.3.4 Information Life Cycle

Information provided by sensor resources is transient in nature and may not even be measured or observed without a specific request. Information stored by a storage resource may be permanently stored there or have an expiry date after which the information is to be removed. For this purpose a storage resource may have to implement mechanisms that remove such information on a regular basis. It is also possible to adapt the granularity of information that is stored over time, i.e., for a certain time interval all the information is stored, for a further time interval only a fraction of the information is kept whereas the rest is discarded. Such a scheme may
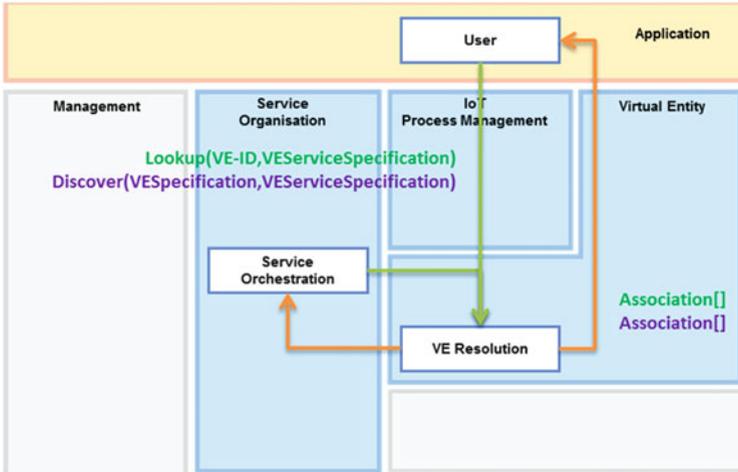
Fig. 8.30 Request lookup and discover Associations
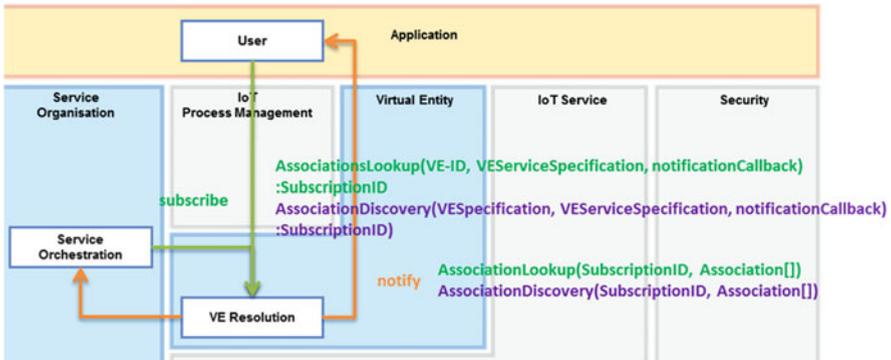

Fig. 8.31 Subscribe to lookup and discover Associations

allow the definition of multiple such time intervals and also requires specific underlying mechanisms that can implement the scheme.

To avoid keeping Service Descriptions of services that no longer exist, a time-out mechanism needs to be implemented by the IoT Service Resolution. After the time-out has been reached without a renewal of the Service Description, the Service Description should automatically be removed. This in turn requires that the components originally providing the Service Description renew the registration of the Service Description before the time-out is reached. The same applies for associations stored by the VE Resolution.

8.2.4 Deployment and Operation View

Connected and smart objects in the IoT can be realized in many different ways and can communicate using many different technologies. Moreover, different systems may need to communicate the one to each other in a compliant way. Hence the Deployment and Operation view is very important to address how actual system can be realized by selecting technologies and making them communicate and operate in a comprehensive way.

The Deployment and Operation view aims at providing users of the IoT Reference Model with a set of guidelines to drive them through the different design choices that they have to face while designing the actual implementation of their services. To this extent this view will discuss how to move from the service description and the identification of the different functional elements to the selection among the many available technologies in the IoT to build up the overall networking behaviour for the deployment.

Since a complete analysis of all the technological possibilities and their combination falls beyond the scope of this view, this section will identify those categories that have the strongest impact on IoT systems realization. In particular, starting from the IoT Domain Model, we found three main element groups (see Fig. 8.32): Devices, Resources, and Services highlighted in red, blue and yellow, respectively. Each of them poses a different deployment problem, which, in turn, reflects on the operational capabilities of the system.

In particular, the viewpoints used in the Deployment and Operation view are the following:

1. The IoT Domain Model diagram is used as a guideline to describe the specific application domain; to this extent UML diagrams can be used to further detail the interaction among the many elements composing the target application;
2. The Functional Model is used as a reference to the system definition; in particular it defines Functional Groups such as IoT Services and Connectivity groups which are fundamental for a correct definition of the system;
3. Network connectivity diagrams can be used to plan the connectivity topology to enable the desired networking capability of the target application; at the deployment level, the connectivity diagram will be used to define the hierarchies and the type of the sub-networks composing the complete system network;
4. Device Descriptions (such as datasheets and users manuals) can be used to map actual hardware on the service and resource requirements of the target system.

First of all, devices in IoT systems include the whole spectrum of technologies ranging from the simplest of the radiofrequency tags to the most complex servers. The unifying characteristics are mainly two-fold: on the one hand, every device is connected with one another forming a part of the IoT; and, on the other hand, every device is "smart", even though with different degree of complexity, in that it provides computational capabilities. These two characteristics are the subject of the first choices a system designer has to make. Note that, for a given device to be
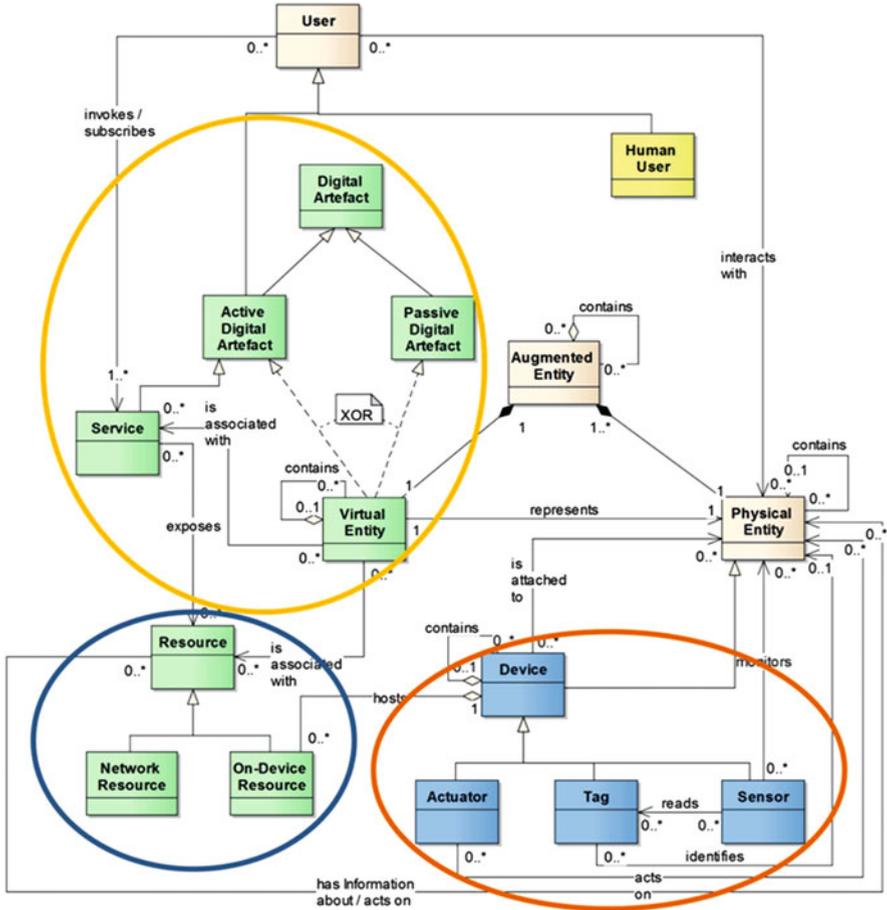
Fig. 8.32 Domain model elements grouped according to their common deployment aspects

fully interoperable in an IoT-A compliant system, it must respect the functionality definitions of the Functional Model. However, legacy systems that do not fully support the FM, may implement wrappers and adaptation software to comply to the model.

Selecting the computational complexity for a given device is somewhat intrinsic to the target application. However, choosing among the different connectivity types is not as straightforward as different choices may provide comparable advantages, but in different areas. For the same reason, it is possible to realize different systems implementing the same or similar application from the functional view which are extremely different from the deployment and operation view. In this section, we will simply detail the main options for device connectivity; further details about deployment configurations can be found in the Reference Manual (see Chap. 9). The impact of those configurations onto the architectural perspectives described in

Sect. 8.8 is discussed in Chap. 6. The following list provides a few of the typical technologies that can be found in IoT systems:

- Sensor & Actuator Networks;
- RFIDs and smart tags;
- WiFi or other unconstrained technologies;
- Cellular networks.

As a consequence of the coexistence of different communication technologies in the same system, the second choice the system designer must account for is related to communication protocols. In particular, connectivity functionalities for IoT system are defined in this document in Communication FG of the FM; in addition, in order to better understand the application, it is important to describe it within the Functional View. Although, IoT-A and WP3 in particular suggest a communication protocol suite aimed at the interoperability among different technologies with IP as the common denominator, the system designer may be forced to make suboptimal choices (Rossi 2012, 2013). In particular, we identified the following possibilities:

1. **IoT protocol suite**: This is the main direction supported by this project and providing the best solution for interoperability;
2. **Ad-hoc proprietary solutions**: Whenever the performance requirements of the target application are more important than the system versatility, ad hoc solutions may be the only way to go;
3. **Other standards**: Depending on the target application domain, regulations may exist forcing the system designer to adopt standards, different from those suggested by the IoT protocol suite, that solved a given past issue and have been maintained for continuity.

After having selected the devices and their communication methods, the system designer has to account for services and resources, as defined in the IoT Service FG section. These are pieces of software that range from simple binary application and increasing their complexity up to full blown control software. Both in the case of resources and for services the key point here is to choose where to deploy the software related to a given device. The options are as follows:

1. **On smart objects**: This choice applies to simple resource definitions and lightweight services, such as web-services that may be realized in few tens or hundreds of bytes;
2. **On gateways**: Whenever the target devices are not powerful enough to run the needed software themselves, gateways or other more capable devices have to be deployed to assist the less capable ones;
3. **In the cloud**: Software can be also deployed on web-farms. This solution improves the availability of the services, but may decrease the performance in terms of latency and throughput.

Note that this choice has to be made per type of resource and service and depending on the related device. As an example, a temperature sensor can be deployed on a wireless constrained device, which is capable of hosting the

temperature resource with a simple service for providing it, but, if a more complex service (for instance, when the Service Organisation FG is called in) is needed, the software has to deployed on a more powerful device as per option 2 or 3.

On the same line, it is important to select where to store the information collected by the system, let their data be gathered by sensor networks or through additional information provided by users. In such a choice, a designer must take into consideration the sensitiveness (e.g.: is the device capable of running the security framework), the needed data availability and the degree of redundancy needed for data resiliency. The foreseen options are the following:

1. **Local only**: Data is stored on the device that produced it, only. In such a case, the locality of data is enforced and the system does not require complex distributed databases, but, depending on the location of a given request, the response might take longer time to be delivered and, in the worst case scenario, it may get lost;
2. **Web only**: No local copy is maintained by devices. As soon as data is sent to the aggregator, they are dispatched in databases;
3. **Local with web cache**: A hierarchical structure for storing data is maintained from devices up to database servers.

Finally, one of the core features of IoT systems is the resolution of services and entities, which is provided by the Entity and Service Resolution FCs, respectively and is in charge of semantically retrieving resources and services, discovering new elements and binding users with data, resources, and services. In particular, this is performed adopting the definitions of the Virtual Entity FG. This choice, while one of the most important for the designer, has only two options:

1. **Internal deployment**: The core engine is installed on servers belonging to the system and is dedicated to the target application or shared between different applications of the same provider;
2. **External usage**: The core engine is provided by a third party and the system designer has to drive the service development on the third party APIs.

Differently from the other choices, this is driven by the cost associated to the maintenance of the core engine software. In fact, since it is a critical component of the system, security, availability and robustness must be enforced. Hence, for small enterprises the most feasible solution is the external one.

8.2.4.1 Deployment Example

Coming back to our "Red Thread" example, this section analyses the system deployment for the "Transport monitoring with Smart Load Carriers" scenario.

First of all, we need to define the purpose of the application(s), the functionalities and their requirements for a correct operating behaviour and the data that needs to be treated.

Purpose: the application measures several environmental parameters of the load carrier such as the light, the temperature and the humidity of the truck and monitors the status of the several installed devices.

Functionalities:

- **Monitoring**: the application needs to provide the users with means to access information gathered by many sensors installed in the truck;
- **Controlling**: the application needs to provide users with means to modify the behaviour of the many installed devices;
- **Alarm**: the application needs to provide users with means to configure alarms to be triggered when a given condition is verified (e.g.: the temperature rises over a threshold value).

Requirements:

- **Lifetime**: all the installed devices must operate unassisted for more than 2 years;
- **Robustness**: a maximum data loss of 5 % of the information is tolerated and no command nor alarm loss can be tolerated;
- **Responsiveness**: a maximum delay of 10 s is tolerated when issuing a command and for alarm reporting. A maximum delay of 15 min is tolerated for data reporting in steady state condition.

Data: all the information managed by the system is not sensitive and does not require for high security.

As a second step, the system integrator must define the Virtual Entities and the Services to be used in the application. To keep the example simple, we will define a single Service and a single Virtual Entity only. The service will be in charge of monitoring the sensing units and to provide users with interface to access the data. We will call this service "Monitoring service". For what concerns the Virtual Entity we choose to represent a room in the house as a Virtual Entity, which is connected to the room Physical Entity and with the resources provided by the Sensors (Device) installed in the truck.

Basically, the application can be simply implemented by allowing the Service to query the Resources of the associated Virtual Entities periodically. However, many possibilities are left to the integrator for the actual deployment of the application.

Resources: it is clear that Resources must provide a connection between the sensing Devices and the Service, but the actual software harmonizing the Sensor behaviour with the service language can be run either on the sensing Device itself, in a gateway device connecting the house network with the external network, or directly in the cloud. The most versatile solution is to run the Resource software directly on the Device in order to enable any other Service to query directly the Device for the needed information; however, depending on the actual hardware capabilities, the other two solutions can be considered.

Service: it must be possible to access the monitoring service from anywhere there is an Internet connection, and, in particular, from within the house. Note that, users using the service from within the house may be less tolerant to delays. A typical service deployment in this case is to have two paired services providing the

same monitoring functionality: one is running in a local server and is able to directly query the devices in order to fetch up to date information, the second is running in the cloud and provides accessibility from the Internet. Note that the local service is also maintaining an information database of the data gathered in the house; database, which is only accessed by the service in the cloud.

Finally, the system integrator must make decisions about connectivity and data management: since the time requirements of the application are quite loose, low power devices can be chosen and low data rate connection can be selected for the sensing devices.

The first and foremost requirement is the addressability of every Service/ Resource regardless of the Device hosting it. This can be achieved by supporting IP addressing and its compressed version defined by 6LoWPAN is currently the most feasible way to implement this in constrained devices. In addition, to make Resources and Services unambiguously addressable, unique identifier must be provided. To this extent many solutions have been proposed, but, in order to obtain the widest interoperability, it is preferable HTTP mappable solutions, such as CoAP. In such a way it is possible to implement very simple Services on the most constrained Device by providing web-service like interaction capabilities to every resource and functionality offered.

However, if the above baseline solution is not realizable, it is important to mimic its behaviour as close to the source device is located. To this extent Resources, Services or both can be deployed on other devices such as aggregator servers, gateways and proxies of the network. In such a way, it is the more powerful Device providing Resource and Service in the correct format that will interact with Services and Users on behalf of the final Device; also, this device must ensure the synchronization between the mimicked functionalities and their actual counterparts. This workaround allows for the integration of any possible technologies in the IoT, however it does not grant the full compliance to all the IoT-A unified requirement list.

However, in order to make the sensing devices interoperable with both, the local and the cloud services, connectivity gateways or proxies must be considered. A few possible realizations are the following:

- Cabled sensors with Ethernet/xDSL gateway

  - Pros: reliable, possibility to use the same cable for connectivity and power.
  - Cons: high installation costs.

- Wireless sensors (802.15.4) with Ethernet/xDSL gateway

  - Pros: low cost, easy and cheap installation, moderate robustness, good lifetime.
  - Cons: may suffer from data losses.

- Low power WiFi sensors with WiFi/xDSL gateway

  - Pros: moderate costs, easy gateway implementation, easy and cheap installation, higher data rate is possible.
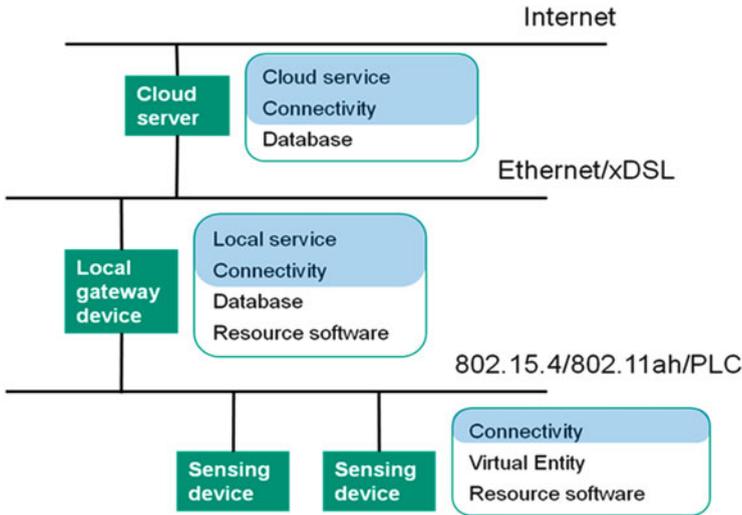 - Cons: shorter lifetime than 802.15.4.

Fig. 8.33 Transport monitoring example with possible deployment choices highlighted

The figure (Fig. 8.33) shows the deployment example above, highlighting the several physical devices involved (dark green), the different network type involved (solid horizontal lines) and the software installed per device (white/cyan rounded boxes, cyan is for mandatory parts while cyan is for optional elements).

Although this example is quite simple, it can be used as a building block for more complex scenarios. In particular it is important here to understand how to separate the different networks in the system, where to deploy each functionality and which connectivity type to use per sub-network.

8.3 Perspectives

Architectural decisions often address concerns that are common to more than one view, or even all of them. These concerns are often related to non-functional or quality properties. We are following the approach described by Rozanski and Woods (2011), that suggests special perspectives to address these aspects of a concrete architecture. They emphasize the importance of stakeholder requirements just like we do within our project. Therefore we are adopting their definition of perspective for usage within IoT-A:

> An **architectural perspective** is a collection of activities, tactics, and guidelines that are used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the system's architectural views (Woods and Rozanski 2005).

where a quality property is defined as:

> A **quality property** is an externally visible, non-functional property of a system such as performance, security, or scalability (Rozanski and Woods 2011).

The stakeholder requirements clearly show a need of addressing non-functional requirements. Based on them, we identified the perspectives which are most important for IoT-systems:

- Evolution and Interoperability;
- Availability and Resilience;
- Trust, Security and Privacy and
- Performance and Scalability.

As these requirements are requiring some kind of quality for a real system, the perspectives aim more on the concrete system architecture, than at a Reference Architecture.

We got 18 requirements concerning the Evolution and Interoperability perspective, 15 concerning Availability and Resilience, more than 20 related to Trust, Security and Privacy, and 17 more related to Performance and Scalability. As can be seen from the definition above there is a close relationship between Perspectives, Views and Guidelines.

We will generally follow the structure as suggested by Rozanski and Woods, to describe the perspectives, but adjusted according to our needs. Each perspective contains the following information:

Desired Quality	The desired quality that the perspective is addressing
IoT-A Requirements	The IoT-A requirements presented in Appendix this perspective addresses
Applicability	The Applicability of the perspective, e. g. the types of systems to which the perspective is applicable
Activities	A set of possible activities that are suggested to achieve the desired qualities. We are reusing existing literature, as well as, our own identified best practices here.
Tactics	Here we list Architectural Tactics, which an architect can use when designing the system.

An architectural tactic is defined as follows:

> An **architectural tactic** is a design decision for realizing quality goals at the architectural level.

It can already be seen from the definition of tactic that there is a close relationship to the design decisions as outlined in Chap. 6. We therefore will list high level design choices as architectural tactics whenever feasible.

We think that taking advantage of perspectives makes a lot of sense for every software architect, even more in the IoT-domain where a lot of Quality parameters have to be taken into account. Perspectives provide a framework for reusing knowledge: It is absolutely necessary to apply a systematic approach to ensure that a certain system fulfils the required quality properties. The use of Perspectives, combined with Views and Guidelines is a step towards that. In the Guidelines chapter in Sect. 9.4 we present a suggested usage of the perspectives in conjunction with Design Choices.

8.3.1 Evolution and Interoperability

The Evolution and Interoperability perspective addresses the fact that requirements change and software evolves sometimes rapidly and need to interoperate not only with today's technologies, but also needs to be prepared to interoperate with later technologies. Interoperability therefore plays especially in IoT a crucial role. The vision of the Internet of Things is still evolving itself. There are, for example, not yet all used technologies mature enough, and there are for sure many more technologies to come in the future (Table 8.2).

8.3.2 Performance and Scalability

This perspective addresses two quality properties which are closely related: Performance and Scalability. Both are, compared to traditional information systems, even harder to cope with in a highly distributed scenario as we have it in IoT (Table 8.3).

8.3.3 Trust, Security and Privacy

This chapter addresses fundamental properties of IoT systems for what concerns their relation to the user. They are inter-related and, often, the evaluation or the improvement of one of these qualities is necessarily related to the others.

Table 8.2 Evolution and interoperability (adopted from (Rozanski and Woods 2011)), extended with IoT specific aspects

Desired Quality	The ability of the system to be flexible in the face of the inevitable change that all systems experience after deployment, balanced against the costs of providing such flexibility
IoT-A Requirements	UNI.003, UNI.010, UNI.012, UNI.023, UNI.042, UNI.047, UNI.048, UNI.071, UNI.093, UNI.094, UNI.096, UNI.417, UNI.422, UNI.432, UNI.509, UNI.701, UNI.712, UNI.720
Applicability	Important for all systems to some extent; more important for longer- lived and more widely used systems. IoT systems are expected, as an emerging technology, to be highly affected by evolution and interoperability issues
Activities	Characterize the evolution needs
	Assess the current ease of evolution
	Consider the evolution trade-offs
	Rework the architecture
Tactics	Contain change
	Create extensible interfaces
	Apply design techniques that facilitate change
	Apply metamodel-based architectural styles
	Build variation points into the software
	Use standard extension points
	Achieve reliable change
	Preserve development environments

8.3.3.1   Trust

Trust in the IoT environment is a fundamental yet difficult to obtain quality. As the security one, this quality is highly dependent on the computation and communication performance of the system. In the frame of IoT moreover, M2M subjects must be enabled to evaluate this quality in order to obtain autonomous systems (Table 8.4).

8.3.3.2   Security

Security is an essential quality of an IoT system and it is tightly related to specific security features which are often a basic prerequisite for enabling Trust and Privacy qualities in a system (Table 8.5).

8.3.3.3   Privacy

There are usually different concepts conveyed with the term privacy, some being more from the technical side and some more from the legal perspective, without forgetting ethical aspects (Table 8.6).

Table 8.3 Performance and scalability (adopted from (Rozanski and Woods 2011)), extended with IoT specific aspects

Desired Quality	The ability of the system to predictably execute within its mandated performance profile and to handle increased processing volumes in the future if required
IoT-A Requirements	UNI.008, UNI.026, UNI.027, UNI.028, UNI.066, UNI.089, UNI.101, UNI.102, UNI.234, UNI.511, UNI.512, UNI.615, UNI.706, UNI.708, UNI.711, UNI.716, UNI.717
Applicability	Any system with complex, unclear, or ambitious performance requirements; systems whose architecture includes elements whose performance is unknown; and systems where future expansion is likely to be significant. IoT systems are very likely to have unclear performance characteristics, due to their heterogeneity and high connectivity of devices
Activities	Capture the performance requirements
	Create the performance models
	Analyze the performance model
	Conduct practical testing
	Assess against the requirements
	Rework the architecture
Tactics	Optimize repeated processing
	Reduce contention via replication
	Prioritize processing
	Consolidate related workload
	Distribute processing over time
	Minimize the use of shared resources
	Reuse resources and results
	Partition and parallelize
	Scale up or scale out
	Degrade gracefully
	Use asynchronous processing
	Relax transactional consistency
	Make design compromises

8.3.4 Availability and Resilience

As we are dealing with distributed IoT systems, where a lot of things can go wrong, the ability of the system to stay operational and to effectively handle failures that could affect a system's availability is crucial (Table 8.7).

8.4 Conclusion

The chapter has given an overview about the current state of the IoT Reference Architecture that is proposed to be applied to any IoT-architecture. The IoT Reference Architecture abstracts from domain specific use cases; it rather focuses on the domain agnostic aspects that IoT Architectures may have in common. It does

Table 8.4 Trust perspective (extension of concepts originally found in (Rozanski and Woods 2011))

Desired Quality	A complex quality related to the extent to which a subject expects (subjectively) an IoT system to be dependable regarding in all the aspects of its functional behaviour
IoT-A Requirements	UNI.062, UNI.065, UNI.099, UNI.407, UNI.408, UNI.602, UNI.604, UNI.605, UNI.620, UNI.622
Applicability	Relevant to the systems that share the use of resources with subjects that are not a priori trusted
Activities	Capture trust requirements
	Perform risk analysis
	Check interoperability requirements and their impact on trust between heterogeneous subjects
	Define trust model
	Consider risks derived from malicious or unintentional misuse of IoT systems[a]
Tactics	Harden root of trust
	Ensure physical security and implement tampering detection
	Ensure and check data freshness
	Consider the impact of security/performance trade-offs on trust
	Use (trusted) infrastructural Trust and Reputation Agents for scalability
	Use security imprinting
	Check system integrity often
	Balance privacy vs. non-repudiation (accountability)

[a]For example, simulating traffic by broadcasting car-to-infrastructure signals or inducing emergency maneuvers in ships or planes by simulating adverse environmental conditions. Generally, it is possible to make a fictional situation credible if the assumption that Physical and Virtual Entities are always and securely synchronized is overlooked

not mean that every IoT-architecture has to implement every feature listed here, but in this report we have covered functional as well as non-functional aspects that are important to support in today's IoT-solutions on one hand and that are important to the stakeholders we have interviewed on the other hand. Following our architectural methodology we presented several views and perspectives of the IoT Reference Architecture.

The Functional View describes the functional building blocks of the architecture and the Deployment and Operation View explains the operational behaviour of the functional components and the interplay of them.

The Information View shows how the information flow is routed through the system and what requests are needed to query for or to subscribe to information offered by certain functional components.

The perspectives listed in this chapter tackle the non-functional requirements IoT-architectures might have. The perspectives are categorised according to the non-functional requirements that have been extracted from the unified requirements (UNIs) presented in Appendix. As a result of the requirement analysis we have categorised the required system attributes into the four perspectives "Evolution and Interoperability", "Performance and Scalability", "Trust, Security and Privacy", and "Availability and Resilience".

Table 8.5 Security perspective (adopted from (Rozanski and Woods 2011), extended with IoT specific aspects)

Desired Quality	Ability of the system to enforce the intended confidentiality, integrity and service access policies and to detect and recover from failure in these security mechanisms
IoT-A Requirements	UNI.062, UNI.407, UNI.408, UNI.410, UNI.412, UNI.413, UNI.424, UNI.502, UNI.507, UNI.604, UNI.609, UNI.611, UNI.612, UNI.617, UNI.618, UNI.624, UNI.719
Applicability	Relevant to all IoT systems
Activities	Capture the security requirements
	Check interoperability requirements for impacts on security processes between heterogeneous peers
	Conduct risk analysis
	Use infrastructural Authentication components that support more Identity Frameworks for scalability and interoperability
	Use infrastructural or federated Key Exchange Management to secure communication initiation and tunnelling between gateways for interoperability
	Use an Authorization component to enable interoperability with other systems
	Define security impact on interaction model
	Address all aspects of Service and Communication Security
	Integrate the trust model and support privacy features
	Identify security hardware requirements
	Consider performance/security trade-offs
	Validate against requirements
Tactics	Use an extended Internet Threat Model for which takes into account specific IoT communication vulnerabilities
	Harden infrastructural functional components
	Authenticate subjects
	Define and enforce access policies
	Secure communication infrastructure (gateways, infrastructure services)
	Secure communication between subjects
	Secure peripheral networks (data link layer security, network entry, secure routing, mobility and handover)
	Avoid wherever possible wireless communication
	Physically protect peripheral devices or consider peripheral devices as available to malicious users in the attacker model
	Avoid Over-The-Air device management; if necessary secure properly

For each of the perspectives we list a number of tactics to achieve the desired attribute of the system, e.g. anonymous usage. The tactics are state-of-the art methodologies commonly used in today's systems architectures.

In Chap. 6 we present examples of Design Choices for the respective tactics listed in the perspectives section as example solutions for non-functional architectural requirements. The Design Choices will help the architect with selecting suitable solutions for non-functional architectural problems to focus on the domain-specific functional aspects.

Table 8.6 Privacy perspective (adopted from (Rozanski and Woods 2011), extended with IoT specific aspects)

Desired Quality	Ability of the system to ensure that the collection of personally identifying information be minimized and that collected data should be used locally wherever possible
IoT-A Requirements	UNI.001, UNI.002, UNI.410, UNI.412, UNI.413, UNI.424, UNI.501, UNI.606, UNI.611, UNI.623, UNI.624
Applicability	Relevant to all IoT systems
Activities	Capture the privacy requirements
	Conduct risk analysis
	Evaluate compliancy with existing privacy frameworks.
Tactics	Use an Identity Management component that supports pseudonymization
	Avoid transmitting identifiers in clear especially over wireless connections
	Minimize unauthorized access to implicit information (e.g. deriving location information from service access requests)
	Validate against requirements
	Consider the impact of security/performance trade-offs on privacy
	Enable the user to control the privacy (and thus security and trust) settings
	Balance privacy vs. non-repudiation (accountability)

Table 8.7 Availability and resilience (adopted from (Rozanski and Woods 2011), extended with IoT specific aspects)

Desired Quality	The ability of the system to be fully or partly operational as and when required and to effectively handle failures that could affect system availability
IoT-A Requirements	Uni.040, UNI.050, UNI.058, UNI.060, UNI.064, UNI.065, UNI.092, UNI.230, UNI.232, UNI.233, UNI.601, UNI.604, UNI.610, UNI.616, UNI.718
Applicability	Any system that has complex or extended availability requirements, complex recovery processes, or a high profile (e.g., is visible to the public)
Activities	Capture the availability requirements
	Produce the availability schedule
	Estimate platform availability
	Estimate functional availability
	Assess against the requirements
	Rework the architecture
Tactics	Select fault-tolerant hardware
	Use high-availability clustering and load balancing
	Log transactions
	Apply software availability solutions
	Select or create fault-tolerant software
	Design for failure
	Allow for component replication
	Relax transactional consistency
	Identify backup and disaster recovery solution

Chapter 9
The IoT ARM Reference Manual

Martin Bauer, Nicola Bui, Christine Jardak, and Andreas Nettsträter

Whereas we explained the process of creating an IoT architecture with the support of the IoT ARM in Chap. 6 [Process Chapter] and gave an example how a concrete architecture can be defined based on different models and views of the IoT ARM in Chap. 11 [Concrete Architecture Chapter], we now provide reference manuals with guidelines how to use the IoT Domain Model, the IoT Information Model, the IoT Communication Model and the Perspectives when creating a concrete architecture.

Starting with the IoT Domain Model.

9.1 Usage of the IoT Domain Model

This section is intended for architects who want to apply the IoT Domain Model on a specific use case. We discuss typical instantiations of the IoT Domain Model. These model cases can be used as basic patterns when doing concrete modelling.

M. Bauer (✉)
NEC Laboratories Europe, Software & Services Research Division, NEC Europe Ltd,
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
e-mail: Martin.Bauer@neclab.eu; www.nw.neclab.eu

N. Bui
Consorzio Ferrara Ricerche, Via Savonarola 9, 44122 Ferrara, Italy
e-mail: buincl@unife.it; www.unife.it

C. Jardak
Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
e-mail: christine.jardak@siemens.com; www.siemens.com

A. Nettsträter
Fraunhofer Institute for Material Flow and Logistics IML, Joseph-von-Fraunhofer Str. 2-4,
44227 Dortmund, Germany
e-mail: andreas.nettstraetter@iml.fraunhofer.de; www.iml.fraunhofer.de

9.1.1 Identification of Main Concept Instances

Similar to the identification of stakeholders and actors in standard software engineering practices, the IoT Domain Model is used in a first step of the architectural design process in order to:

1. Identify Physical Entities and related Virtual Entities;
2. Identify Resources (at least from a functionality perspective);
3. Identify Devices (or device options);
4. Identify Services;
5. Identify Users.

The identification of Resources and Devices is used together with the IoT Communication Model to define the communication paradigms and how these devices and resources interact. This is comparable to interaction models in standard software engineering practices. The Services to be used and where they should be deployed are analysed and finally the Users of these services are identified.

9.1.2 Modelling of Non-IoT-Specific Aspects

It is important to understand that the IoT Domain Model is not attempting to be a domain model for all types of ICT systems. Rather, it focuses on the IoT-specific parts. When modelling a complete system, many of the aspects to be covered are not IoT-specific. For these aspects, the IoT Domain Model will provide only little help.

For example, the Service concept in the Domain Model is primarily focused on modelling IoT Services that directly or indirectly expose Resources; however, the Service concept also can be used to provide a link to general services in the ICT domain.

9.1.3 Identifiers and Addresses

Identifiers and addresses are logically two different concepts, which unfortunately however are often confused in practice, in particular in the discussions about IoT (Haller 2010). While in some cases the address might be used in the role of an identifier, it is important to distinguish between these terms.

Identifiers are used to identify something, for example a Physical Entity. In this case, the identifier is an attribute of the related Virtual Entity. Examples include URIs (Uniform Resource Identifiers as used on the Web, e.g. foo://example.com/building1/room3), EPCs (Electronic Product Codes, e.g. 01.23G3D00.8886A3.365000A03)

(EPC Tag Data Standard) and uIDs (uCode Identifiers,[1]
e.g. 0123456789ABCDEF0123456789ABCDEF).

Addresses, on the other hand, are means for locating, accessing or communica-
tion with something, e.g., a service or a device. Addresses manifest themselves as
attributes of the corresponding concepts, i.e., attributes of a service or a device.
Examples include IPv6 or MAC addresses.

As mentioned above, there are cases in which it can make sense to use addresses
as identifiers, e.g. when the address uniquely identifies the Physical Entity. For
example, a street address is good identifier for a building, but not for a human being.
An e-mail address on the other hand provides a unique way of identifying people.

Modelling Option 1	An address can be used as an identifier for a Physical Entity (and the corresponding Virtual Entity) if it uniquely identifies it.

Overall, identification and addressing are very important aspects of IoT systems.
When designing an IoT system the different options should be evaluated and
decided on early in the process, but as the decision depends on various
requirements, assumptions and even technology choices, we cannot give specific
recommendation on the reference model level.

9.1.4   Granularity of Concepts

In the IoT Domain Model, concepts like Device, Resource, and User have
specialisations. Pertinent examples for Devices are Sensors and Actuators. When
modelling a concrete scenario, one can use either the general concepts or their
specialisations; the IoT Domain Model does not prescribe anything. For example,
instead of using a concrete concept like Sensor it is also possible to use a more
general concept like Device. However, the specialisations are more precise and are
therefore preferable where they apply. In other words, if at the time of modelling it
is not (yet) clear what type of device is used, then just use Device.

Modelling Rule 1	Model as precisely as possible based on the domain model concepts at the time of modelling. Use the more concrete, more fine-granular concepts and instances whenever possible, but only to the granularity that appears reasonable for the given purpose.

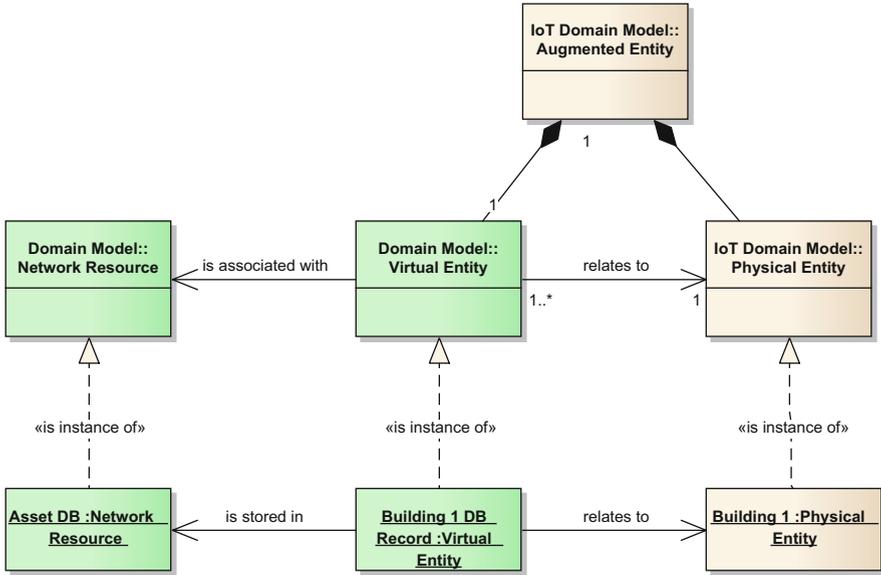[1] http://www.uidcenter.org/spec#UID-00010.

Fig. 9.1 Data-base pattern as an example for an augmented entity

9.1.5 Common Patterns

9.1.5.1 Augmented Entities

As described in Sect. 7.3.2.2, Augmented Entities are the composition of a Physical Entity with its related Virtual Entity. In many cases though, the Augmented Entity is of little practical relevance and will have no concrete instantiation, as the example in Fig. 9.1 shows. In this figure, a typical pattern is shown for how Physical Entities are mapped to data base records: In a data base of assets (a Network Resource in terms of the IoT Domain Model), a data base record (Virtual Entity, and also a Passive Digital Artefact) is stored for every building (Physical Entity).

> **Modelling Option 2** ⋮ The Virtual Entity for a given Physical Entity can be a data base record stored in a Network Resource.

A different case is truly smart objects, i.e., intelligent devices that have embedded logic seemingly able to act autonomously. In this case, the Augmented Entity is the smart object itself, and the associated Virtual Entity is an Active Digital Artefact, namely, the embedded logic (e.g., the software agent).

Figure 9.2 shows an example of a smart object: an *Unmanned Aerial Vehicle* (UAV). The body of the UAV can be considered the Physical Entity, while the
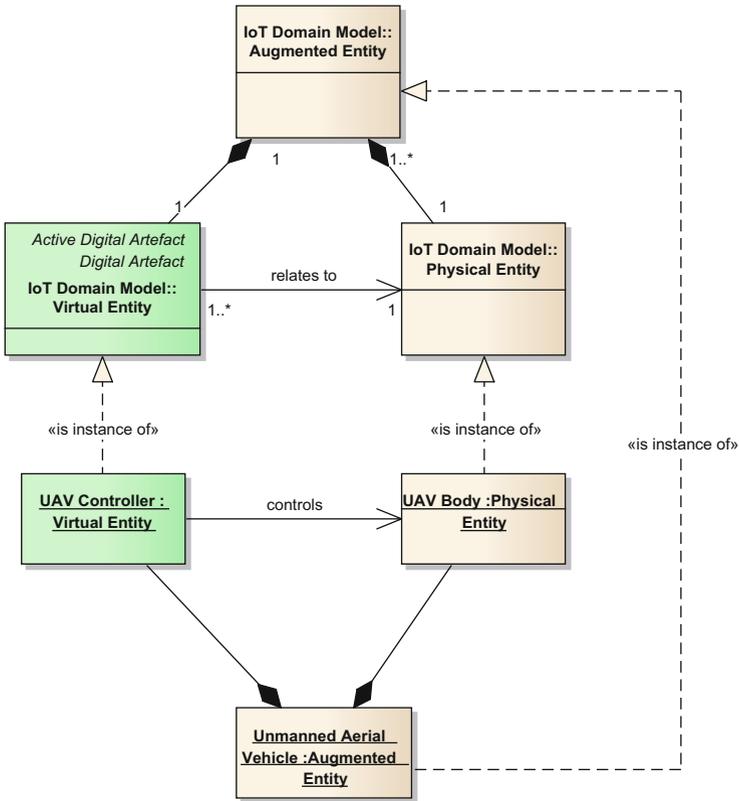
Fig. 9.2 Smart-object pattern. UAV: unmanned aerial vehicle

UAV controller is the related Virtual Entity. Together they form the Augmented Entity, the smart object.

Modelling Rule 2	When modelling an autonomous object, an Augmented Entity is used, consisting of a device (Physical Entity) and its software controller (Virtual Entity).

Finally, the question often arises if something should be modelled as a Physical Entity or not. While possibly every real-world object could be modelled as a Physical Entity, this does not make sense. Not every sand corn needs to be represented in an IoT system. Hence we can deduce:

Modelling Rule 3	Only model something as a Physical Entity if it is relevant in the IoT system so that the representing Virtual Entity is also modelled.

9.1.5.2 Multiple Virtual Entities

In order to understand the case of multiple Virtual Entities, we take the example of a customer buying a new car. The customer visits the exhibition of an automobile manufacturing company and buys a new car. He then registers it under his name at the department of motor vehicles. In order to protect himself from unexpected financial expenses resulting from traffic collisions, he decides to buy a car insurance. In this small scenario we notice that the same car, which is the Physical Entity, is registered at three stakeholders: the manufacturer, the vehicle-registration department, and the insurance company. As depicted in Fig. 9.3 each of the three stakeholders maintains a unique entry in its database identifying the car. These entries are multiple Virtual Entities representing the same car.

In practice, the number of Virtual Entities depends on the systems and domains, where the Physical Entity is represented and of course also which stakeholders are involved. We note that the characteristics of the Physical Entity change and, therefore, many of the Virtual Entities need to be maintained and kept up-to-date. Notice that the IoT Domain Model does not explicitly spell out any requirements on the maintenance of single and multiple Virtual Entities.

9.1.5.3 Smart Phones and Other Mobile User Devices

Smart phones are a very common element in many IoT-related scenarios. They are on the one hand Devices containing a multitude of sensors, but they also host apps (Active Digital Artefacts), Services, and Resources. Figure 9.4 shows this in exemplary fashion: John's smart phone is used as a Device to track the location of John, its owner. The GPS sensor is embedded in the phone itself. It is thus embedded sensor hardware. Its data is made accessible through the related On-Device Resource and the location service that exposes it. An app can be used to display the location information.

Note that in this example (see Fig. 9.4), both the service as well as the application are shown to be hosted on the phone itself. While this depicts a common case, other instantiations are possible.

Instead of a smart phone other mobile user devices could be used, e.g. tablets or PDAs. The general modelling would be the same.
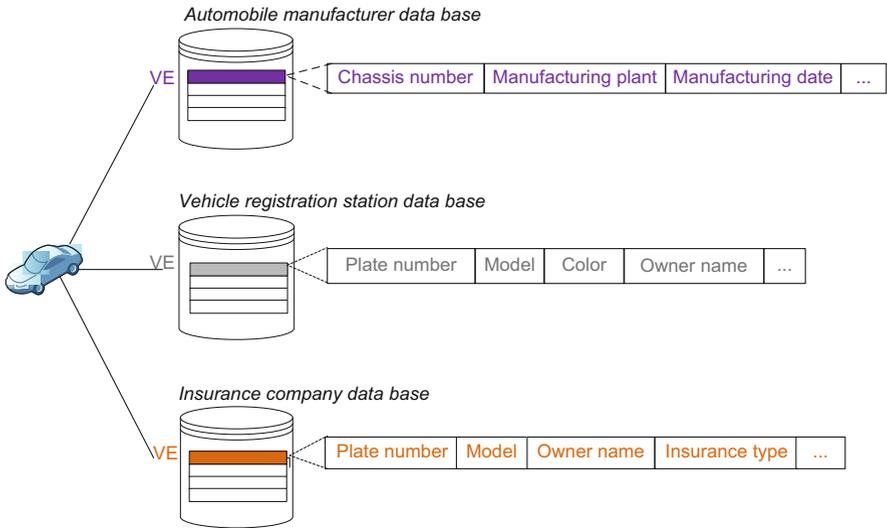
Fig. 9.3 Multiple virtual entities (data-base entries) for a single physical entity (car)
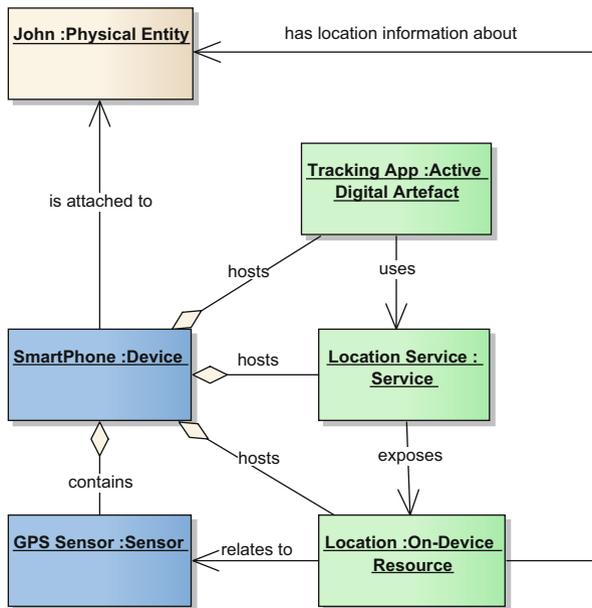

Fig. 9.4 Exemplary modelling of a smart phone that is used as tracking device

9.1.5.4 IoT Interactions

The IoT paradigm enables mediated interactions between Users and the physical world. This complements the direct interactions in the physical world that are

possible between Human Users and Physical Entities. It also enables the digital world, i.e. Active Digital Artefacts, to interact with the physical world.

9.1.5.5 Simple Mediated Interactions

A common case is that a User needs to access a Resource exposed through a Service in order to attain a given goal. Such goals may range from observing a Physical Entity by using a Sensor, to modifying its state by leveraging an Actuator device. We differentiate the following cases:

- **Retrieving information**: In this case a user would invoke a Service for retrieving some information. There are different options for the Service to get this information, which may be pull or push based. In case the Resource pushes the information, the Service would cache the information and provide it on request
- **Subscribing for information**: In the subscription case, the User subscribes to the Services and asynchronously receives notifications. After subscription, the Resource (e.g., on a Device) will detect the events of interest according to the specification provided by the user. The Service providing access to the Resource will then forward the event to the interested User. In an alternative implementation, the Service is performing the event detection by processing all the raw data from the Resource;
- **Actuation**: In the case, the User wants to control some aspect of the physical world mediated through the IoT system, it would call an Acutation service. In this case, the Service would interact with the Resource which would trigger the Actuator to execute the actuation.

9.1.5.6 M2M Interaction

Machine-to-Machine (M2M) communication is a technological approach for enabling meaningful information exchange between networked machines that show a certain degree of smartness. The term *machine* is generally related to an autonomous application while the smartness is related to the capability of controlling its own behaviour and communicating. This reflects the capability of making decisions on the basis of information retrieved from outside the system and being able to receive and execute commands. This approach is very relevant to the IoT and a specific definition of IoT Machine can be provided. In the terms of the IoT Domain Model, we define an IoT Machine as a composition of:

- An **Augmented Entity** whose Virtual Entity component is an Active Digital Artefact. In this way, it can start interactions (being a User, it can invoke Services) and can control the behaviour of the machine;

- One or more **Resources and the underlying Devices** which are used by the Active Digital Artefact to monitor/control the Physical Entity. Note that, because Resources are internal functionalities and the Active Digital Artefact is generally co-located on the same hardware, the interaction can happen even without the use of Services;
- The **Services** that are used for exposing Resources.

The example shown in Fig. 9.5 shows how a car interacts with a road barrier in order to speed up the passage through the barrier, i.e. that the barrier is removed as early as possible to enable the passage of the car. The incoming car is modelled as IoT `Machine1`, the automated barrier operator as IoT `Machine2`. The `Machine1 Controller`, an instantiation of an Active-Digital-Artefact Virtual-Entity, will access as a User (Active Digital Artefact can be Users) `Service2` and will require the activation of the barrier. `Service2` provides access to `Functionality2` (Resource) related to `Machine2` and thus, by accessing `Service2`, the car can retrieve the information about the barrier status which is needed in turn to decide whether it needs to slow down or can pass through without danger.

As M2M is about the communication-based interaction between machines, it is important to clarify that IoT Machines can also interact with non-IoT Machines. For example, an IoT-Machine could need certain information provided by an autonomous web application, a non-IoT Machine, in order to make decisions.

However, as the controlling program of `Machine1` is a User according to the IoT Domain Model, it can also communicate with other Machines by calling appropriate embedded Services on another Machine, as shown in a simplified way in Fig. 9.6.

Object identification and tracking with RFID. The term "Internet of Things" was originally coined by the MIT Auto-ID Centre around 1999 (Ashton 2009), the precursor to what is now known as EPCglobal. EPCglobal is a standardization organization set up for achieving the worldwide adoption of the *Electronic Product Code* (EPC). It is based on RFID technology and the sharing of related information over the Internet. Due to its importance, it is worthwhile to map one of the most common scenarios of EPCglobal to the IoT Domain Model: the tracking of goods – pallets, cases, etc. – throughout the supply chain, from the manufacturer via distribution centres to the retail stores. A reverse mapping of EPCglobal onto the ARM can be found in Sect. 12.9.

A first thing to note is that we often have a hierarchy of Physical Entities and the related Virtual Entities. A large boxed pallet is identified by a shipping company as `PE5` with its corresponding Virtual Entity `VE5`. As depicted in Fig. 9.7, the large boxed pallet contains multiple other cases that are identified as `(PE1, VE1)`, `(PE2, VE2)`, `(PE3, VE3)`, and `(PE4, VE4)`.

We note that the granularity of identifying PEs contained in other PEs is not defined by the IoT Domain Model, since it intimately depends on the application. In this example, if the large box contains four boxes of similar goods, e.g., shoes, the interest of the shipping company usually stops at identifying `PE5` and thus tracking
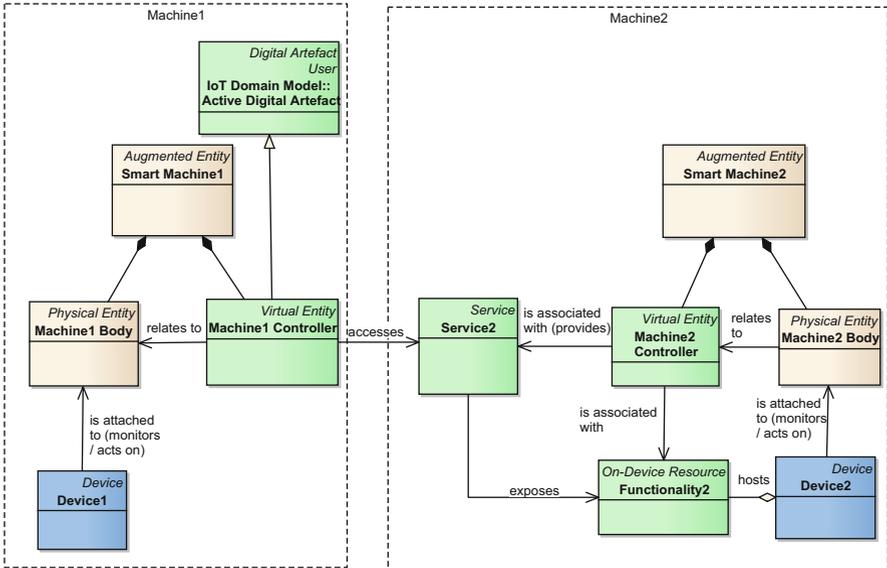
Fig. 9.5 IoT domain model instantiation for a M2M communication scenario
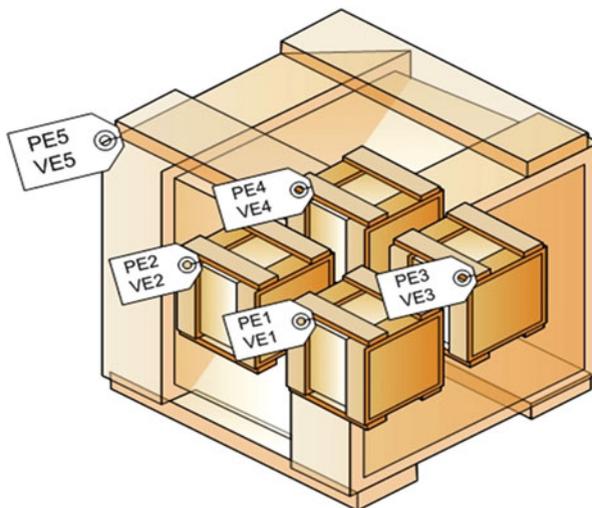

Fig. 9.6 M2M communication

Fig. 9.7 Shipping box containing multiple packets. The VE-to-PE mapping is exemplified by paper tags
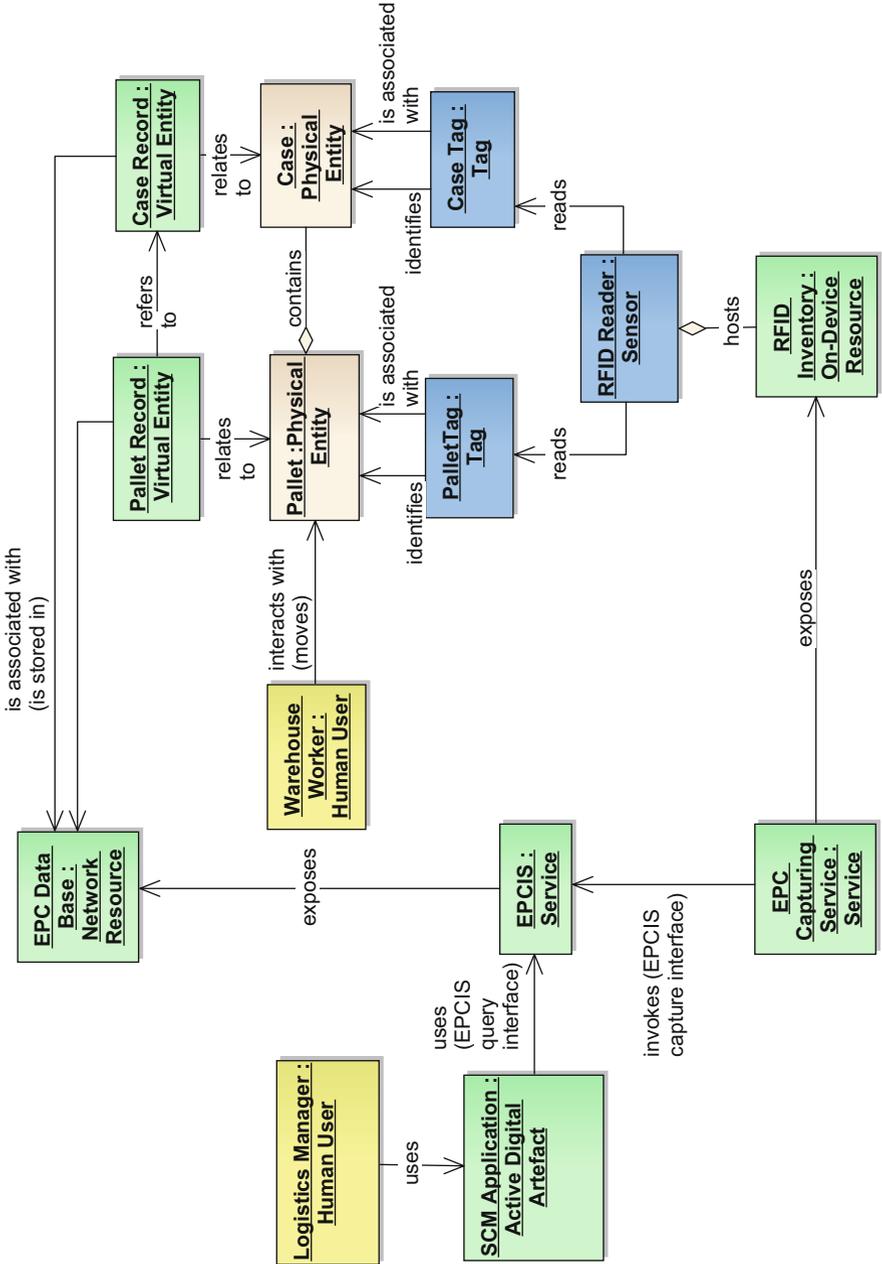
Fig. 9.8 Domain modelling of a typical EPC-based RFID scenario (pallet containing cases)

it by using `VE5`. Now if each of the four boxes contains different goods, e.g., shoes, hats, earrings, and bags, it might be of interest for the shipping company to additionally identify the four boxes as `PE1`, `PE2`, `PE3`, and `PE4`. The aim behind this higher granularity is to facilitate the process of sorting out the goods after delivery by checking `VE1`, `VE2`, `VE3`, and `VE4`.

The result of the whole mapping of the RFID logistics scenario, for only the pallet plus everything it contains, is depicted in Fig. 9.7.

In this example, the Virtual Entities take the form of database records (Fig. 9.8) stored in a Network Resource, the EPC Database. This database is exposed for querying and updating through the EPCIS service (EPC Information Service).

The logistics manager, a Human User, can use the SCM application in order to view the status of the tracked items (pallets and cases). The SCM application is invoking the EPCIS query interface in order to get the necessary data.

Both pallet and cases have RFID tags attached that identify them. A RFID reader – a type of sensor – reads the EPCs on the tags and hosts a resource that makes the RFID inventory data accessible. A special service, the EPC Capturing Service, is exposing this resource and is updating the EPC Data Base by invoking the EPCIS capture interface of the EPCIS service. The EPCIS capture interface and the EPCIS query interface are standardized and defined by EPCglobal (EPC 1.0.13).

In principle other technologies for identification, e.g. visual ones like bar codes could be used. In this case, there is no hardware Device of type Tag involved and the Sensor would be a camera or barcode reader. The `identifies` relation (as in the IoT Domain Model) would then be directly between the Sensor and the Physical Entity. The other aspects would be modelled in the same way.

Finally note that also physical interactions with the pallet can take place: a warehouse worker – a Human User – moves around the pallet.

9.1.6 Examples for IoT Domain Model Concepts

In this section we give examples on different concepts in the IoT Domain Model. For each concept we discuss a practical example and, where applicable, we highlight the dependency of the concept on other concepts and also provide some general information.

9.1.6.1 User

A User interacts with a Physical Entity, physically or mediated through the IoT system. In the case of a mediated interaction, a User invokes or subscribes to a Service.

Application

- **Example**: A WSN installed in a wine cellar monitors environmental factors such as temperature, humidity, and light intensity. These factors play an essential role in defining the quality of the final wine product. Therefore, the winegrower has an intelligent application running on his smart phone. The application allows him to periodically visualize the status of the cellar. In this example, the application is a user and the cellar is a Physical Entity.
- **Note**: An application is one kind of Active Digital Artefact.

Human User

- **Example**: The employee in a supermarket loads the fridge with meat instead of cheese. Therefore, he regulates the temperature of the fridge accordingly. In this example, the employee is a Human User and the fridge is a Physical Entity.
- **Note**: The case of multiple Human Users for one Physical Entity is possible as well. We take the example of the safe in a bank. For security reasons, more than one high-ranked employee is required to identify themselves simultaneously at the safe in order to be able to open it. In this example the eligible employees are Human Users and the safe is the Physical Entity.

9.1.6.2 Physical Entity

A Physical Entity is a discrete, identifiable part of the physical environment which is of interest to the User for the completion of his goal. In the following different kinds of Physical Entities are discussed.

Environment

- **Example**: An optical fog sensor measures the density of water particles in the air that limit visibility. This sensor is used for traffic-control purposes, where it is often installed on the side of roads for monitoring visibility impairment through fog. The information about the fog is sent to a traffic management system where it is analyzed. In this example the near surrounding above the road is the Physical Entity.

Living Being

- **Example**: A WSN for agricultural monitoring. The network targets to report on the growth of fruits. To this end growth monitors are deployed. They are equipped with fruit-growth sensors as depicted in Fig. 9.9. In this example, the fruits are Physical Entities that are living beings.

Fig. 9.9 Growth fruit
sensor © 2010 Phyto-Sensor
Group
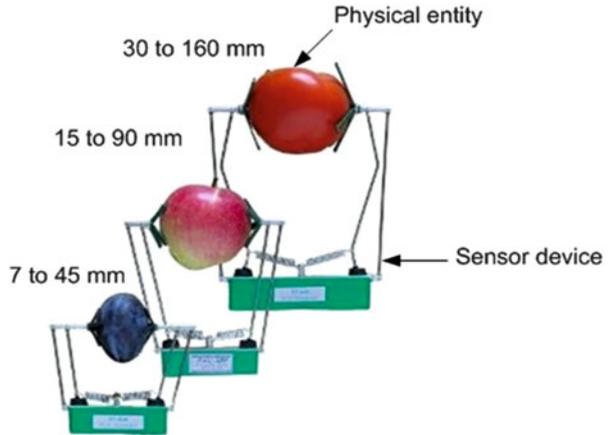
Structural Asset

– **Example**: Equipping bridges with electrochemical fatigue sensors that reveal
  flaws in metal (Phares 2007). This works much the same way as an electrocar-
  diogram tests the human heart. First, bridge inspectors identify parts of the
  bridge that are more susceptible to cracks. Second, they equip these areas with
  electrochemical fatigue sensors. Third, they apply a constant electrical current
  that runs between the sensors and the bridge. By monitoring the amplitude of the
  current passing through the metal, sensors can detect cracks. In this example, a
  susceptible area of the bridge is a structural-asset Physical Entity.

9.1.6.3   Resource

Resources are software components that provide information about or enable the
actuation on Physical Entities. We explain two examples for Resources, one
illustrating an On-Device Resource and the other a Network Resource.

On-Device Resource

– **Example**: TinyOS is an event-based OS for embedded networked sensors (Levis
  and Gay 2009). TinyOS provides predefined software components that manage
  the access and control of i.e., local LEDs, radio, or sensors. In this example, the
  software components are On-Device Resources.

Network Resource

– **Example**: HBase[2] is an open-source, distributed, column-oriented database. HBase offers a set of functionalities that allow the management of distributed information. In this example the HBase software libraries and components are -Network Resources.

9.1.6.4   Service

A Service provides a well-defined and standardised interface, offering all necessary functionalities for interacting with Physical Entities and related processes. Often it exposes a functionality provided by a Resource to the overall IoT system.

Interacting Services

– **Example**: A system for home-patient monitoring. The system is composed of a body sensor network (BSN) attached to the body of the patient. Bioelectric chips monitor the status of the patient and require no direct involvement from a human being. As depicted in Fig. 9.10, the intelligence of the system resides not only in the hardware but also in three main services. First, the BSN monitoring service that evaluates the readings of the bioelectric chips i.e., a blood pressure. Second, the automatic service call, which alerts the relatives of the patient whenever his situation deteriorates. Third, another automatic service call that alerts the ambulance. The diagram in Fig. 9.10 shows the conditions to be fulfilled for one service to invoke another service.
– **Note**: A service demanding high processing and storage capabilities can be divided into multiple subservices running on different machines that invoke each other. In comparison to the original service, each of these subservices requires less storage and processing capabilities. Therefore, a trade-off exists between the number of subservices and the power consumption of the hosting machines. Distributed subservices induce an inter-communication overhead that increases the power-consumption of the hosting machines. This trade-off should be taken into consideration when dealing with low-power communicating devices (Polastre et al. 2005).

Service Associated with a Virtual Entity

– **Example**: Services can be associated with Virtual Entities and these associations are stored and can be discovered in the IoT system. The management of these associations can be handled in a centralized database or in a highly distributed fashion as in a peer-to-peer system, depending on the characteristics of the underlying system.

---

[2] http://hbase.apache.org/

Fig. 9.10 Interacting services for a home-patient monitoring scenario
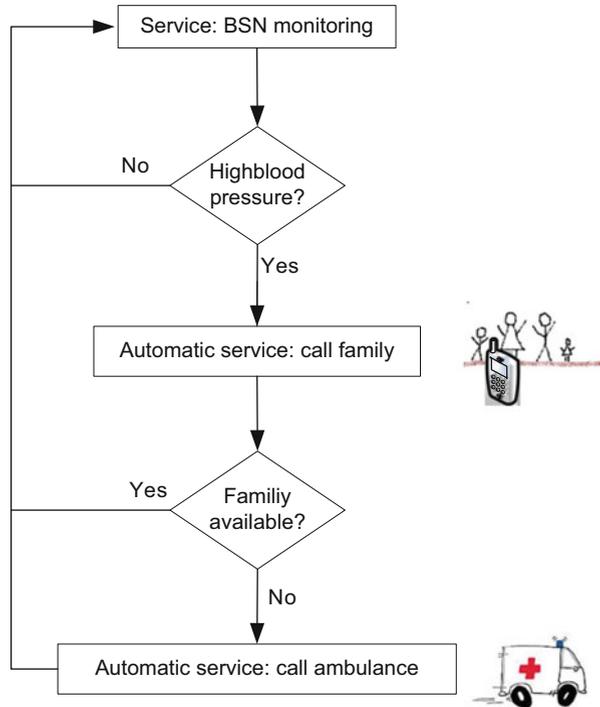
Service Accessing a Resource

– **Example**: A service for monitoring air pollution. Sensor nodes are semi-randomly distributed in a city and measure the percentage of CO in the air. A remote server runs software that periodically queries the readings from the sensor nodes, analyses the readings, and monitors the evolution of the air pollution. In this example, the monitoring software is a service that accesses multiple resources. The latter are the components and functions running on sensor nodes, and these components allow operations such as reading from the sensors.

9.1.6.5 Device

Devices are technical artefacts, i.e. hardware, for bridging the real world of Physical Entities with the digital world of the Internet. Often a Device hosts Resources, which represent the software counterpart.

Devices

– **Example**: Typical devices are sensors, like temperature, noise or light sensors, but also more complex ones like cameras – or actuators, like switches, door openers or more complex ones like air conditioning systems.

Hierarchical Devices

– **Example**: As depicted in Fig. 9.11, a Telos node contains three types of integrated sensors (photodiode, humidity and temperature), several expansion pins to mount external sensors, and three integrated LEDs (Polastre 2005). Two views of the node exist: The node as a whole may be seen as a single device or it can be seen as a composition of multiple sensors and actuators acting as individual devices.
– **Note**: A device can be seen as a single unit as well as a composition of multiple devices. This granularity of modulating a device is not specified in the IoT Domain Model due to the fact that it is application dependent.

9.1.6.6 Deployment Configurations

Figure 9.12 shows a range of deployment configurations for resources, services, and users. In Fig. 9.12 (a) resource, service, and the user (application) are running on the same device. This is a configuration in which we have a powerful device, and the interaction with the user is local. In Fig. 9.12 (b) the service of the user is running somewhere else, e.g., in the cloud, and the interaction is thus not local. The API used between the service client and the service, however, is the same. In Fig. 9.12 (c) the service is not running on the device, but in the cloud. This is a typical configuration for a constrained device that may not be able to expose a user interface across the network. For example, due to energy constraints or other limiting factors, such a device may sleep most of the time and is therefore not able to always handle user requests. The interface between the service and the resource may be very specific and proprietary.

Network-based resources are not shown in Fig. 9.12, as they can be regarded as being hidden behind cloud-based services.

Of course, in a real IoT system all these different configurations may be realized at the same time and there could be interactions between users and services from the different configurations.

Fig. 9.11 Telos ultra-low power wireless module © 2008 University of California, Berkeley
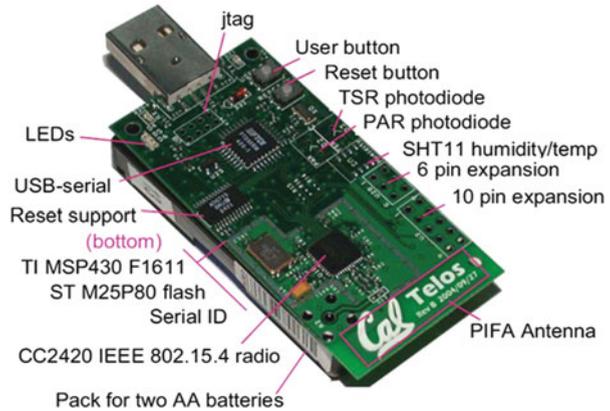
9.1.7 Generating a Specific IoT Domain Model

As discussed in Sect. 6.3, the IoT Domain Model is an integral part of the IoT architecting process. In the following we provide a six-step process that supports the generation of use-case specific IoT Domain Models. In the following, we illustrate the answers with examples from the recurring example that we introduced in Sect. 4.2.

In order to proceed with the modelling of a system, its usage from the perspective of each User needs to be analysed. For each of the Users identified, the architect needs to answer six simple questions, and create suitable instance diagrams from the Domain Model based on the answers.

Q1: What does a User invoke or subscribe to?

   A1: The answer determines the Service(s) that the user invokes or subscribes to – In the recurring example the user subscribes to the alarm service (using an Android app).

Q2: Which part of the environment does the User want to interact with?

   A2: The answer determines the PE(s) – in the recurring example the user wants to be kept informed about the status of the load carrier.

Q3: What is used to identify/monitor this PE in the physical world?

   A3: The answer determines the Device(s) – in the recurring example, the load carrier can be identified with an RFID, a humidity sensor and a temperature sensor can monitor relevant state information.

Q4: What is used to identify the PE in the digital world?

   A4: The answer determines the VE(s) – in the digital world the identifier provided by the RFID can be used for the VE modelling the load carrier.

Fig. 9.12 Various deployment configurations of devices, resources, and services
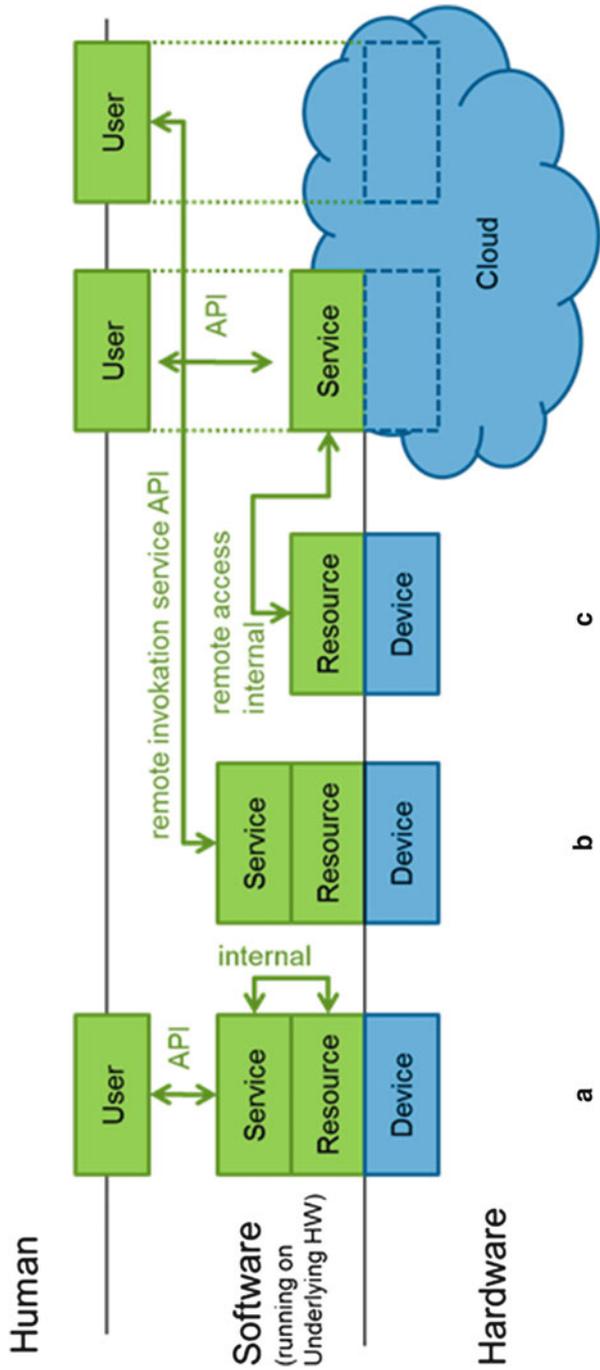
Q5: What software can provide information or allow changing aspects related to PE?

 A5: The answer determines the Resource(s) – the Alarm Resource can trigger a notification if the temperature or the humidity are no longer within the required range.

Q6: What exposes this Resource and/or makes it accessible?

 A6: The answer determines Resource-level Service(s) – the Alarm Service exposes the Alarm Resource.

9.2 Usage of the IoT Information Model

The IoT Information Model cannot be instantiated directly like the IoT Domain Model. Moreover the IoT Information Model defines an abstract framework or meta-model that is technology agnostic and restricted to a minimum. The model is just enough to accommodate the relationships defined in the IoT Domain Model and to model the key concepts that are used as a basis for defining interfaces of functional components. Thus only the skeleton of an information model is provided in the ARM that IoT-A compliant architectures will have in common. A common model on the other hand can serve as a bridge between more specific -but different -information models to be used in concrete architectures.

   The way to work with the IoT Information Model is split into three steps (see also Fig. 9.13 below):

1. Use the IoT Information Model, viz. meta-model, as a basis explaining the common information structure and the core elements defined in the IoT Domain Model, like Virtual Entities, Attributes and Services;
2. Generate a domain-specific information model out of the meta-model, which defines a minimal set of attributes and Services for your application domain. Attributes which every VE needs to have (e.g. `EntityId` or `EntityType` as in Chap. 7 Fig. 9.10 "IoT Information Model") are defined but not necessarily described in detail. Additionally the Service Descriptions can already be defined as interfaces with input and output parameters.
3. Several (different) representations of the domain-specific model can be generated, implementing the defined Attributes and Services. The use of different representations is useful when there are different implementation-specific requirements, like binary storage of information for constrained sensor nodes and XML storage for the backend server storage.

   The IoT Information View in Sect. 8.2.3, and especially the Sect. 8.2.3.1, give some examples how the concrete modelling of the domain-specific model can look like. An additional example for an information model used to model events is shown in (Voelksen 2013; Sect. 4.2).
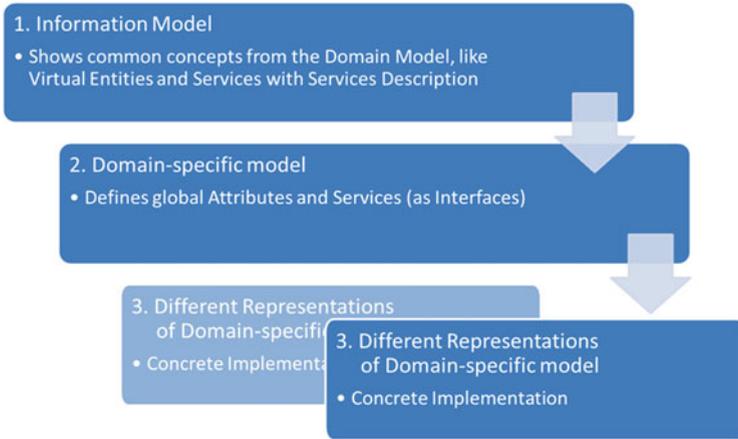
Fig. 9.13 Three steps to use the IoT information model

9.3 Usage of the IoT Communication Model

Extending the analysis performed using the other models the IoT Communication Model is used in the architectural design process to:

1. Identify homogeneous sub-systems and their capabilities and constraints;
2. Identify suitable protocol stacks and network topologies to be merged in a common system view;
3. Define gateways and other bridging solutions.

9.3.1 Guidelines for Using the IoT Communication Model

Since the IoT Communication Model aims at providing an overall framework for communication in IoT systems, it requires well-defined domain and information definitions. This can be achieved following the examples of the previous sections. Starting from those, it is possible to identify all the sub-systems, the complete system is composed of, where we define homogeneous sub-system as a set of system elements sharing the same communication technology and sharing similar hardware capability.

Once the sub-systems have been defined, it is possible to analyse capabilities and constraints for each of them. By capabilities and constraints we intend communication specific parameters such as data rate, delays, medium reliability (channel errors) and technology specific parameters such as the available memory, computational power and supported functionalities.

Modelling Rule 4	Identify homogeneous sub-systems from the complete domain model and determine their capabilities and constraints.

Subsequently, it is possible to analyse communication requirements deriving from both services in the domain definition and interaction patterns from the information model. The main goal of the IoT Communication Model is to identify a set of interoperable protocol stacks and topologies with the following characteristics:

1. Each stack must grow from a specific communication technology;
2. Interoperability shall be enforced in the lowest possible layer of stack;
3. The combination of identified stacks and topologies must satisfy all the requirements.

Modelling Rule 5	Use existing standard communication mechanisms and related protocols whenever possible. If this is not possible then each of the sub-system is the starting point for building a protocol stack which is both technology specific and interoperability prone.

This rule enforces technology optimizations and ensures feasibility in all the subsystems.

Modelling Rule 6	Interoperability shall be enforced in the lowest possible layer.

This rule enforces simplicity and interoperability, because it avoids stack duplication and makes it possible to reuse the same protocols (and their implementations) horizontally in the system. Usually, the most effective interoperability point is the Network & ID aspect in the IoT Communication Model (or the Network layer in the ISO/OSI model) as it is the lowest common point in the stack which is not technology specific and, thus, it can be the same across different sub-systems.

For such a reason the selection of the protocols governing the Network & ID aspect is of paramount importance, since they must satisfy the requirements from services and respect the technology constraints.

The next aspect in the IoT Communication Model is the end-to-end aspect: this considers every possible interaction path among any couple of sub-systems. Again, technology dependent constraints and service dependent requirements will push the system architect in two opposite directions and often there is no single rule for all

the systems: in fact, even though the Network & ID aspect is capable of making two sub-systems communicate to one another, it is the end to end aspect which harmonizes the overall system behaviour. For such a reason, this is the place where gateways and proxies are mostly needed.

Modelling Rule 7	In order to allow seamless interaction between sub-systems, gateway and proxies shall be designed for the whole system.

Finally, the Data interoperability aspect of the IoT Communication Model, which accounts for the highest layer of the ISO/OSI communication stack, considers the remaining aspects of data exchange, compression and representation. Although application layer gateways can always be designed to map two different data representations, it is not always advisable to do so. Most often adopting a compressed format which fits constrained network capabilities provides two advantages, (1) simpler network interactions, and (2) lower traffic.

9.4 Usage of Perspectives

Perspectives are used to help an architect in designing software architectures. Their purpose is threefold (Rozanski and Woods 2011):

1. Standardised store of knowledge about a given quality property;
2. As a guide for architects not experienced with a given quality property, and
3. As a memory aid for the experienced architecture.

The actual use of perspectives in an architectural design process is shown in Fig. 9.14. Within the IoT-A project we extended the use of perspectives by adding another layer: the Design Choices catalogue. Design Choices which are very concrete usages of the IoT Reference Architecture applied to Functionality Groups and Functional Components. An architect can consider solutions provided by the Design Choices when creating the initial candidate architecture and later on when he is modifying the architecture to resolve the problems with unacceptable quality properties.

The architect designing should always keep the desired use of the system into account. For example, the architect designing the system used in the "Red Thread" example from Sect. 4.2 would go through the scenarios with a specific "hat" for all perspectives. He would first extract the non-functional requirements (e. g. the reliability needs of the sensors, security concerns) and then, once he has a candidate architecture, use the perspectives to ensure that on all the non-functional requirements have been taken care of. He would, for example, have a specific look at the safe storage of the sensor history and select a Design Choice which ensures that it cannot be altered. The perspectives then will help him making sure
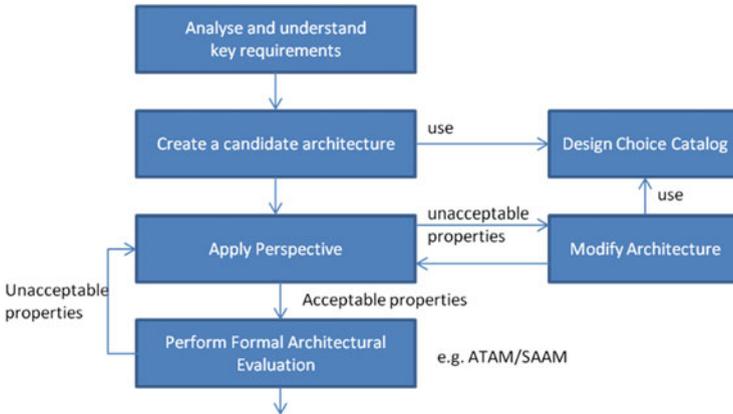
Fig. 9.14 Using perspectives (Adopted from (Rozanski 2011))

that still all other requirements are fulfilled, and if not, at least can help making the trade-offs explicit.

Applying a perspective is more than a review process: the outcome of applying a perspective is cross-view changes to the architecture. As an additional outcome of the perspectives there might be additional documents like performance analysis data etc.

Chapter 10
Interactions

Martin Bauer, Mathieu Boussard, and Stefan Meissner

As discussed in Sect. 8.2.2 and found in the literature, the functional view of a concrete architecture typically consists of three viewpoints: functional decomposition (viz. the logical structure), interfaces, and behaviour. Despite its significantly more abstract nature, we provide an analysis of these viewpoints for the IoT Reference Architecture in Sect. 8.2.2 and in Carrez et al. 2013: Annex C. However, only rudimentary interaction analysis is presented in the latter section, focusing mostly on technical use cases within a single FG.

Nevertheless, as can be appreciated by looking at already existing IoT systems, the operation of such systems generally involves sequences of FC interactions from all FGs. To help the reader better understand how common system-wide scenarios can be realised using the IoT ARM, and further apply this knowledge to their concrete architecture, this section provides the reader with an analysis of interactions between FCs across different FGs for some selected scenarios.

As explained earlier, the very nature of the IoT ARM is to cover all usage domains and architectures that can be derived from it – therefore it is not feasible to describe every possible FC interaction sequence for every possible scenario and architecture combination. Furthermore, instantiating a given scenario implies in most cases taking some clear Design Choices, before one can illustrate them in terms of FC interactions.

M. Bauer (✉)
NEC Laboratories Europe, Software & Services Research Division, NEC Europe Ltd., Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
e-mail: Martin.Bauer@neclab.eu

M. Boussard
Alcatel-Lucent Bell Labs France, Route de Villejust, 91620 Nozay, France
e-mail: mathieu.boussard@alcatel-lucent.com

S. Meissner
University of Surrey, Stag Hill, GU2 7XH Guildford, UK
e-mail: s.meissner@surrey.ac.uk

However, in order to provide the reader at least with a general understanding of how such interactions can look like, we provide analyses for a few usage scenarios. The scenarios presented in the following sub-sections address some of the most representative system-wide general use cases, identifying relevant FCs and proposing an analysis of possible Design Choices when applicable.

The scenarios presented in this section are:

- Management-centred scenarios dealing with modification of the IoT system through

  - Configuration of the system when adding a device
  - Changing the device configuration

- Service-centred scenarios

  - Discovering relevant services using IoT Service Resolution and VE Resolution

Interworking of Service Choreography and IoT Services in the context of Complex Event Processing

10.1 Management-Centric Scenarios

This section presents the analysis of a "management-centric" scenario, namely the auto-configuration of an IoT system when adding a device or group of devices to the system. This scenario also encompasses the system-triggered configuration of such device(s) through the Management FG. Although the Device FG is out of the ARM (see Sect. 7.5.2), system designers have no choice but to consider this FG in the specification of their concrete systems. In particular, the interactions the FCs of the Device FG have with the entire system to make devices usable should be defined (we'll consider actual devices as distinct FCs here, but a device ensemble could be modelled likewise by a Device Group FC depending on chosen design choice).

In this section, we describe the interactions taking place across the Device, Management, Security, Communication, and IoT Service FGs for two management-centric scenarios, namely (i) what happens when a device is added to the IoT system and made available to its components, and (ii) what happens when a device configuration is changed within the system.

10.1.1 Configuration of the System When Adding a Device

From a general point of view, such addition can happen automatically, semi-automatically or in a manual fashion (which is a clear design choice). Common examples of these three different design choices are:

- For automatic joining, typically the handover between cells in a GSM network; Plug & Play solutions such as those supported by protocols like UPnP or Bonjour;
- For semi-automatic joining, the connection to a private network with a firewall, where a network administrator needs to manually grant access through inclusion of the requester's MAC address in a white list;
- For manual joining, any system where the complete compulsory information is manually inserted by an administrator (possibly including physical intervention).

The automated addition of devices is commonly addressed in concrete IoT systems through the usage of Plug&Play solutions (or a mix thereof). Extended to networked systems, Plug & Play conceptually covers addressing and more generally communication, resource description and discovery, registration and look-up as well as possibly secure and trusted access (see e.g. (Houyou et al. 2012)). Semi-automatic would e.g. imply equivalent discovery mechanisms but wait for approval of a human system manager to actually make the new device available to the rest of the system. Finally, some systems may not imply any automation at all – human system engineers perform static provisioning of necessary device information and trigger the addition of the device to the system when the physical deployment is performed. Regardless of the selected design choice, a number of actions need to take place, which are depicted below.
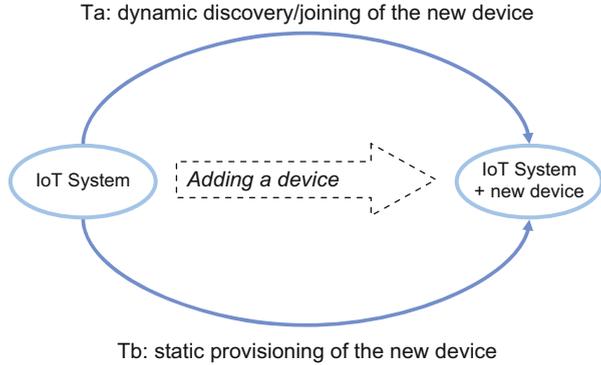
When considering an IoT system, the goal is to go from state A (system in initial state) to state B (system + new component in a state where this new component is actually potentially usable by the rest of the system components). In the following, we describe how the system might make this transition for two of the identified design choices (automatic or manual), as the semi-automatic case can be inferred as a mix of these two cases. The transition from state A to state B is caused by two types of triggers (Ta and Tb below):

- **Ta**: automatic design choice trigger (dynamic discovery/joining of the new device); the device is discoverable, e.g. actual (dynamic) appearance of the device in the range of the system (e.g. turning on the device, mobile device getting in range of the (e.g. radio) system);
- **Tb**: manual design choice trigger; the system is told to (statically) add a resource (or this specific resource); such a request can be issued within the system or by a human user.

Figure 10.1 below illustrates these two possibilities.

Triggers of type Ta) rely primarily on network-level mechanisms (e.g. joining network when requested from the incoming resource, or discovery when polled by the existing system communication gateway) that are specific to the concrete system implementation choices (e.g. Bluetooth discovery mechanisms), or to service-level mechanisms over a pre-existing network connectivity (e.g. UPnP over an IP-based local network), or a mix of both. From a concrete system perspective these mechanisms have to be supported both by the new resource and the system. Further aspects are discussed below.

Ta: dynamic discovery/joining of the new device

IoT System   Adding a device   IoT System
+ new device

Tb: static provisioning of the new device

Triggers of type Tb) are typically issued from a management function either upon explicit human (system manager) provisioning the configuration data for the new device (e.g. using a management console), or from another (non-management related) functionality in the system towards the management function (e.g. the IoT Service Resolution FC asking Management FG to add a new resource when it can't find one already available matching a given request)

From the Reference-Architecture point of view, the steps of the process when going from state A to state B shall include the following activities (represented in Fig. 10.2). Note that all the following steps should involve the necessary security measures for access control, namely authentication, authorization through respectively the Authentication and Authorization FCs of the Security FG:

- Update of Management FG components: in particular the Member FC's UpdateMember() interface should be called (see (Carrez et al. 2013); 7.4 Member)) – as the corresponding entry does not exist in the Member Database, it actually makes an "add" rather than an "update"). Other Management FCs can be impacted as well. The Configuration FC may retrieve and store the configuration of the new components, i.e. the resource and the collateral updates to existing components in the IoT Reference Architecture. The State FC may reflect the change of state of the system incurred by the addition/updates of these components. The Fault FC may have to correct related alarms, for instance if the former absence of the newly added devices incurred an alarm on the system. For instance in an IoT system controlling water level in a river with actuators offline due to maintenance, which raises an alarm in the Fault FC: as actuators go back online after maintenance, the system detects their re-appearance; the State FC is updated, and the Fault FC restarts regulation services as a consequence of the clearing of the alarm;
- As is the normal way to make use of a device through its associated IoT Resource, which itself is exposed through an IoT Service FC, the IoT Service FG needs to be updated, by creating a new IoT Service to represent this new IoT Resource (if necessary), and by updating the IoT Service Resolution FC (through its insertService() or updateService() interface);
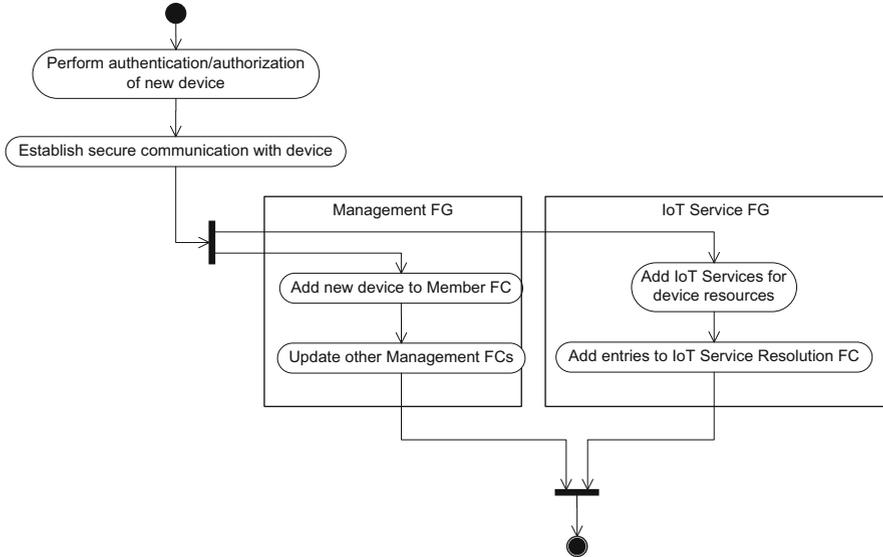
Fig .10.2 Adding a new device to the system – activity diagram

- If not already available (e.g. the IoT Service is newly created, or was not bound to the actual resource so far), the communication link between the IoT Service and the actual resource from the Device FG needs to be established, taking into account the specificities of the resource (e.g. intermittent availability) and desired communication patterns (cf Information View Section).

10.1.2   Changing the Device Configuration

In this section, we discuss how a device or a group of devices can be configured using different FCs of the IoT Reference Architecture. Such process involves the following steps depicted in Fig. 10.3:

- As a pre-requisite, the communication link from the Device FG to the various IoT Resources should be established, relying on Communication and Security functions;
- The request for a configuration change is issued by a human system manager through a management console, or by a FC. It results in a call to the Management:Member FC (step 1) through the `retrieveMember()` interface followed by a call to Management:Configuration FC (step 2) through the `setConfiguration()` interface. Naturally, such calls are subject to access control through the Security: Authentication and Security: Authorization FCs – not represented);
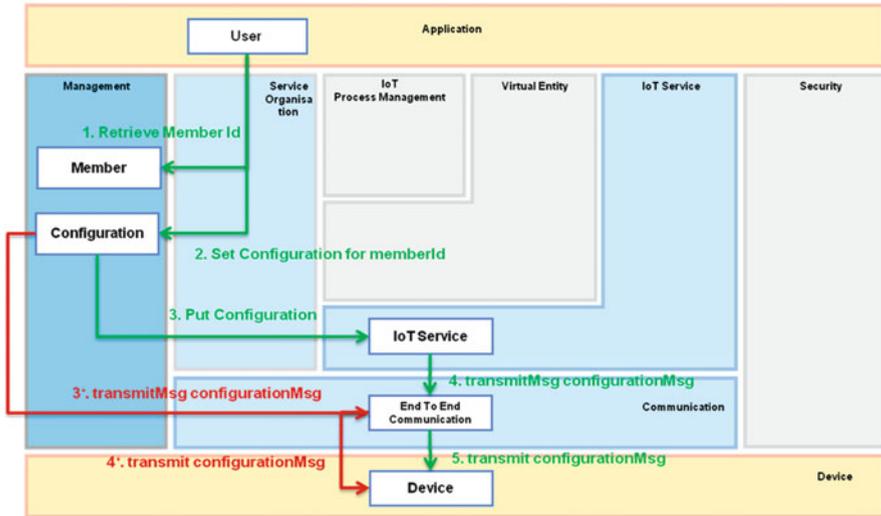
Fig. 10.3 Device configuration update interactions

- Depending on design choices made (e.g. whether the configuration of the associated resource is part of the related IoT Service or not), the actual configuration update on the device can be realized by either communicating directly with the device (e.g. sending a configuration message, steps $3', 4'$) or through the IoT Service associated with the resource (steps 3–5).

Please note that `prepareConfigurationMsg()` and `transmit()` methods are related respectively to the preparation of a usable configuration message to be transmitted by the End-to-End Communication FC, and to the actual reception of data on the Device itself, which are both out of scope of the IoT Reference Architecture, and therefore only shown here as an illustration. Underlying interactions with Security FCs and between End To End Communication FC and other FCs of the Communication FG are not shown (see ((Carrez et al. 2013); 5) on Communication FG).

10.2 Service-Centred Scenarios

10.2.1 Discovering Relevant Services Using IoT Service Resolution and VE Resolution

In existing, small-scale IoT scenarios, applications are often hard-coded or configured with respect to the sensors and actuators they are going to use. If we think of truly large-scale IoT scenarios, this is not going to be possible.

Applications should work in any environment, where the necessary infrastructure is available. This means that the necessary sensors and actuators first need to be found. The Functional Components responsible for this are the IoT Service Resolution and the VE Resolution. Due to the heterogeneity of the underlying hardware, and in order to make the functionality accessible in the whole IoT domain, it is desirable to provide a higher abstraction level than the hardware-level interface of the sensor. Therefore, the ARM offers a service abstraction level and a virtual entity abstraction level for the interaction with the IoT system. The IoT Service Resolution is the functional component responsible for discovering IoT Services based on a service description, which would typically include the service area; the VE Resolution is responsible for discovering the IoT Services associated to VEs, which can either provide information about the represented PEs or enable actuation on them.

In the following we look at a traffic scenario, but the interactions shown also apply to a large number of other scenarios. We have modelled the roads in form of road segments, where each road segment is a VE, and for each road segment, there is an associated sensor-based service that provides the road condition, e.g. whether the road there is dry, wet or icy. Figure 10.4 depicts the scenario.

To get to this scenario, the assumption is that either the IoT Services themselves or a management component, e.g. the Member FC, have registered each service together with their service area within the IoT Service Resolution. This is depicted as "Insert Service Description" steps (1)/ (1′) in Fig. 10.5. For the service areas only a few examples are shown. To simplify the discovery, road segments are modelled as Virtual Entities and associations between the road segments and the services are introduced. The respective IoT Services have service areas overlapping with the area of the road segment. The associations may be explicitly introduced by a service management component, e.g. the Member FC. (see Fig. 10.5 (2)) or they may be automatically discovered by the VE & IoT Service Monitoring component (see Fig. 10.5 (2′)), e.g. as the result of applying some rule on existing service descriptions from IoT Service Resolution and existing associations from Virtual Entity Resolution. The VE & IoT Service Monitoring component would then insert the newly created Association into the Virtual Entity Resolution component.

Now that the relevant information is available in the IoT Service Resolution and Virtual Entity Resolution, a car acting as a User that is driving along the road could then discover the services that provide information about the road conditions in the direction in which it is driving. Such a scenario is depicted in Fig. 10.6. The car would specify the geographic scope based on its current position and the driving directions, possibly taking map information into account. The scope is then used discover the associations between the upcoming Road Segments and the sensor services providing the respective road condition. Based on the service identifier, which is part of the associations, the service descriptions can be retrieved, so that the service can be accessed (Fig. 10.7).

The Application first discovers associations from the Virtual Entity Resolution looking for Virtual Entities of type road segment, with attribute road condition, within the geographic scope specifying an area that covers the road in the driving
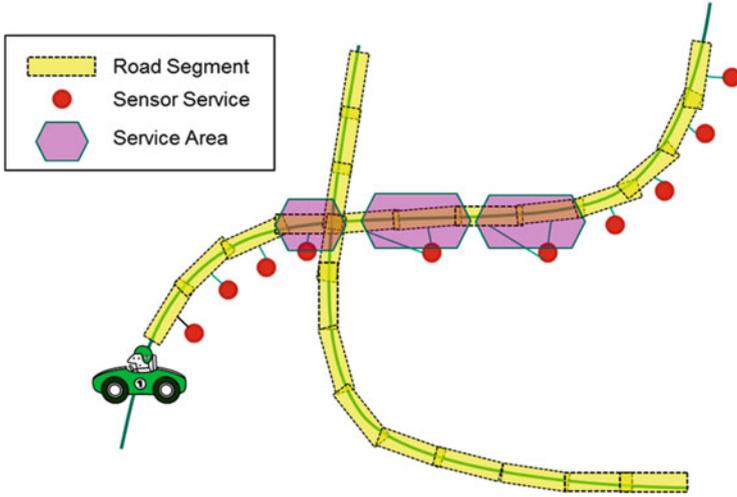
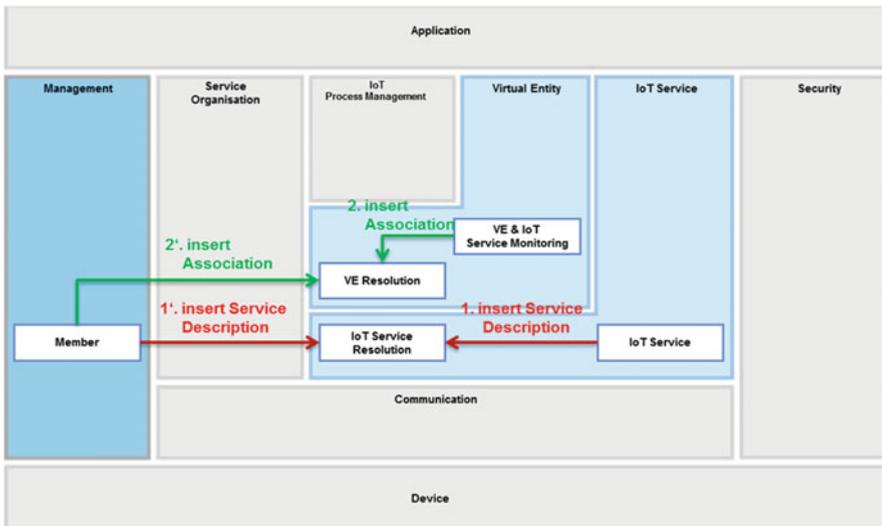**Fig. 10.4** Road condition scenario

Fig. 10.5 Insertion of service descriptions and associations

direction (see Fig. 10.4 (1)). The returned associations contain the identifiers of the services that can provide the respective information. Based on these service identifiers, the service descriptions are looked up from the IoT Service Resolution (see Fig. 10.4 (2)). The returned service descriptions contain the information needed by the application to contact the respective IoT Services (see Fig. 10.4 (3)).
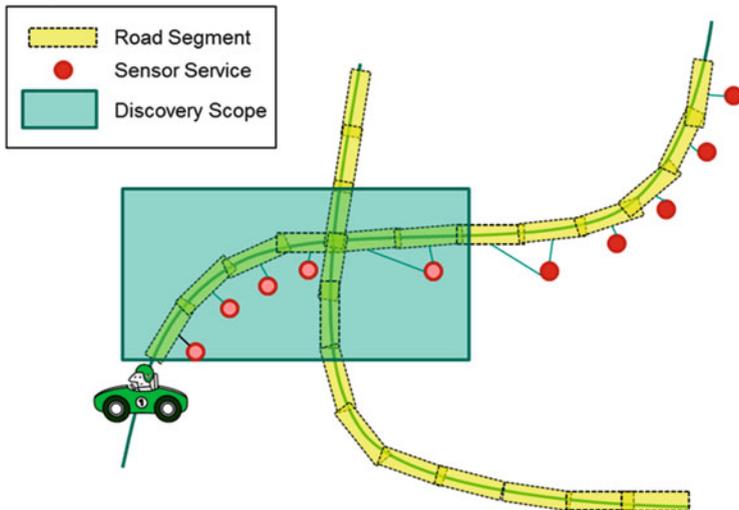
Fig. 10.6 Discovery of services providing information about the road conditions for the road segment in the direction the car is driving
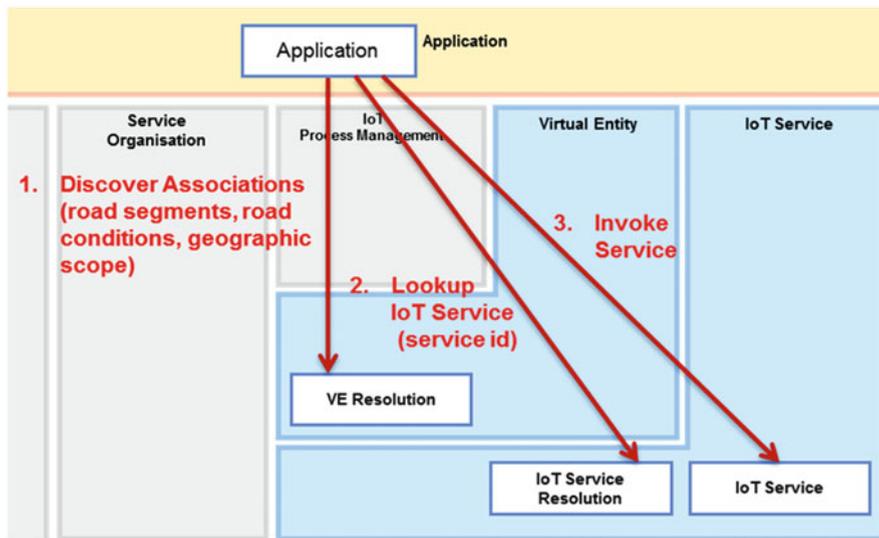

Fig. 10.7 Discovery and invocation of services providing the road conditions based on a geographic scope
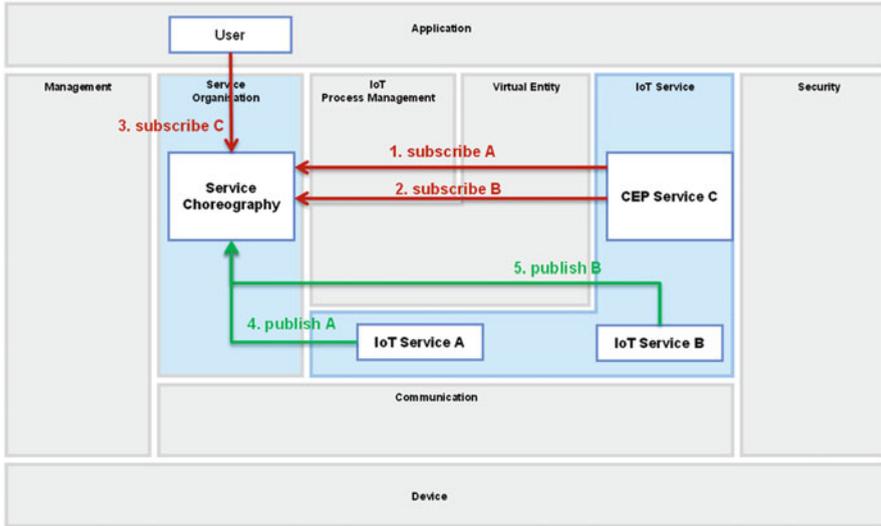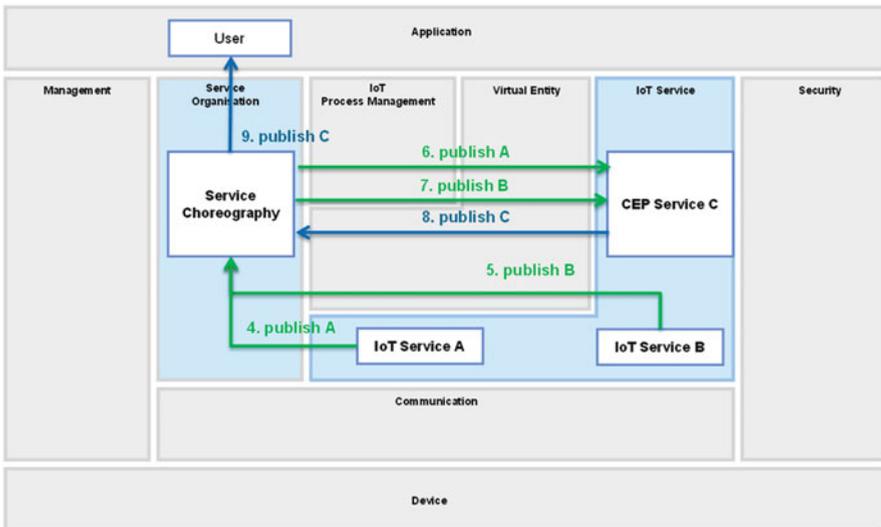
Fig. 10.8 Interactions `CEP Service C` subscribe

Fig. 10.9 Interactions `CEP Service C` publish

10.2.2   *Managing Service Choreography*

The FG Service Organisation contains the FC Service Choreography that supports Publish/Subscribe-functionality for IoT Services. In contrast to the IoT Service Resolution FC (Sect. 8.2.2.5) the Service Choreography FC contains a broker that can find suitable services for service requests given by potential service consumers. The service requests declaring an interest in certain IoT Service functionality are stored within the broker even if a suitable service is not available at the time the service request was given to the FC. As soon as a suitable service becomes available the broker receives the information the services publishes and forwards the information to the service consumer. On the other hand services can advertise their capabilities at the broker to await usage of potential service users. IoT Services can also publish information to the broker even if no service consumer is present.

In case multiple service consumers are interested in the information one particular service provides, the broker distributes the information to all subscribers (Sect. 8.2.3).

This Publish/Subscribe functionality allows using IoT services for CEP. In the scenario depicted in Fig. 10.8 the Design Choice has been made to provide CEP functionality as IoT Service, identified as CEP Service C. Such CEP services compute complex event based on simple events produced by other IoT Services (IoT Service A and B in Fig. 10.9). For this CEP Services need to subscribe to the IoT Services publishing the simple events (steps 1 and 2 in the figure below).

When the simple events are published to the Service Choreography FC (steps 4 and 5) the broker forwards the events to the CEP Service C (step 6 and 7). The CEP Service C is then able to process them to a complex event that is again published to the Service Choreography FC as illustrated in step 8 in Fig. 10.9

Since the user has subscribed to get notified if and when the complex event occurs (step 3 in Fig. 10.8) the Service Choreography FC publishes the event notification to the User as depicted in step 9 in Fig. 10.9.

Chapter 11
Toward a Concrete Architecture

Christine Jardak and Joachim W. Walewski

11.1 Objective and Scope

This Section serves to illustrate how the IoT ARM can be used for the generation of concrete architectures. This goal is pursued by applying the IoT ARM to a concrete use case and application scenario. This Section serves thus as a complement to Sect. 6.3. Notice that we are not providing all the details that would usually be part of an architecture description, rather, the idea is to illustrate aspects of the architecture actions elaborated on in Sect. 6.3.

> Throughout this Section we provide summaries of how the description provided here illustrates statements made elsewhere in the document, for instance Sect. 6.3. In such summaries we occasionally also discuss how complementary actions to those laid out in Sect. 6.3 can enhance the architecting process. All such meta-commentary is set apart in light-grey boxes like this one.

The targeted use case of this architecture is a combination of *Pay-By-License-plate* (PBL) parking and *Recognise-By-License-plate* (RBL) parking enforcement. The core idea of such a system is to use the license plate of a car as a unique identifier for on-street parking. Upon purchase of a time-parking permit, the customer provides the license-plate number of her car for identification. This parking feature shall be available to time parkers and residents. Examples for time parkers are tourists, and locals from a suburb who visit the city centre for shopping, restaurant visits, etc. Residents are defined as denizens of a municipality,

C. Jardak (✉) • J.W. Walewski
Siemens AG, Otto-Hahn-Ring 6, 81739 Munich, Germany
e-mail: christine.jardak@siemens.com; joachim.walewski@siemens.com

and they are purchasing a parking permit for an extended period of time for on-street parking in the vicinity of their residence. By using the license-plate number as ID for the parked car, paper copies of the parking permit do not longer have to be placed on the dash board of the parked cars. In such a system, the license plate is also used by the parking enforcement for checking the permit of the car against a data base provided by the parking service itself. More information on PBL and RBL can be found elsewhere in the literature (Digital Payment Technologies 2013; Genetec 2013). In the remainder of this Section we refer to this envisaged system as a PBL system.

It should be also noted that the entire system is to be designed in a way that it can be made part of a version update of an already existing central system that manages municipal on-street parking lots.

> Notice that scopes usually are part of the business goals. Depending on the complexity of the use case such description can be rather complex and long. Besides describing the goal of the system, the description also needs to include a sketch of how one intends to achieve this goal. Without a spelled-out approach, it is impossible to generate an architecture.
>
> Also notice that due to resource and time constriction we were not able to dedicate the same level of attention to all the steps in the architecting process as laid out in Sect. 6.3. In particular, no Functional Decomposition, Interactions, nor interface definitions are provided. Also, neither the Deployment nor the Operational Views are touched upon.

11.2   Physical Entity View and IoT Context View

11.2.1   Physical Entity View

> This Section relates to Sect. 6.3 and Chap. 6 Figure 3. In the referenced Section, the content and the importance of the Physical-Entity View are discussed. Here, we provide a concrete example of the PE View for the PBL system presented in the previous Section. Notice that this view can be much more complex for other use cases. For instance, if the state of the Physical Entity is going to be inferred from a wide range of measured physical quantities, one not only needs to catalogue these quantities (viewpoints!), but also their range and how these ranges translate into the

(continued)

(continued)

qualitative states that are to be inferred from the measured quantities. An illustrative use case for this is the Red-Thread example (see Sect. 4.2), viz. the transport of orchids. One needs a rather fine-tuned model of the orchids in order to infer their current condition from environmental quantities such as air temperature and humidity and the duration for which the orchids have been exposed to these conditions.

As briefly described in Sect. 11.1, the thing at the core of the IoT system is the car. More specifically, the entity of interest is the **parked** car. Therefore, the Physical Entity in the IoT Domain Model (see Sect. 7.3) is the parked car. An example of the Physical Entity is shown in Fig. 11.1.

Notice that the parking lot itself is not the Physical Entity but the car. That this is the case is not an intrinsic property of the Physical Entity, rather of what the business goals behind the envisaged architecture are, and how they will be achieved (the aforementioned approach).

As described in Sect. 11.1, the goal of the envisaged IoT system is to implement one service for both time and resident parkers, and the car's license plate was chosen upfront as the unique identifier for both use cases. The parking lot becomes an entity of interest when, for instance, the parking enforcement enquires whether a parked car is authorised to park at that specific location. However, since this is only one of the envisaged use-case scenarios (see below) where the parking lot could qualify as the Physical Entity, the parked car and not the parking lot is chosen. This does not imply that there can only be one Physical Entity per IoT system. Rather, one Physical Entity type turns out to be sufficient in order to meet the system goal as described in Sect. 11.1.

Notice that the process provided in Chap. 6 indicates that the Physical View is contingent on the business goals: once the goals are chosen the Physical Entity can be identified together with the properties about the Physical Entities that are of interest for the IoT system. This dependency is illustrated in the above example.

Notice that since only the license plate is used to identify the parked car, the envisaged system can readily encompass the parking of motor bicycles and the like. What is paramount is that it is a vehicle that is identifiable through its license plate.

Fig. 11.1 A car parked in the Gatwick North Terminal Flightpath long stay car park (Whittington 2010)

As shown in Chap. 6 Figure 3, both the Physical View and the business goals inform the IoT Context View. In the next section we illustrate this inter-relatedness for the PBL architecture.

11.2.2 IoT Context View

As already stated in Sect. 11.1, the envisaged system is to be integrated with an existing system for the control of parking-payment systems, which we refer to as Control Centre. In other words, the system envisaged is an extended version of the existing system. Future extensions are very likely.

The context diagram of the PBL system is shown in Fig. 11.2.

As described in Sect. 6.3.2, the context view describes "the relationships, dependencies, and interactions between the system and its environment" (Rozanski and Woods 2011). While we describe some inner structure of the envisaged system, viz. an enhanced version of the Control Centre, this level of detail is not mandatory. What is mandatory though is to provide an information about (Rozanski 2013).

- System scope and responsibilities
- Identity of external entities and services and data used
- Nature and characteristics of external entities
- Identity and responsibilities of external interfaces
- [Nature and characteristics of external interfaces]
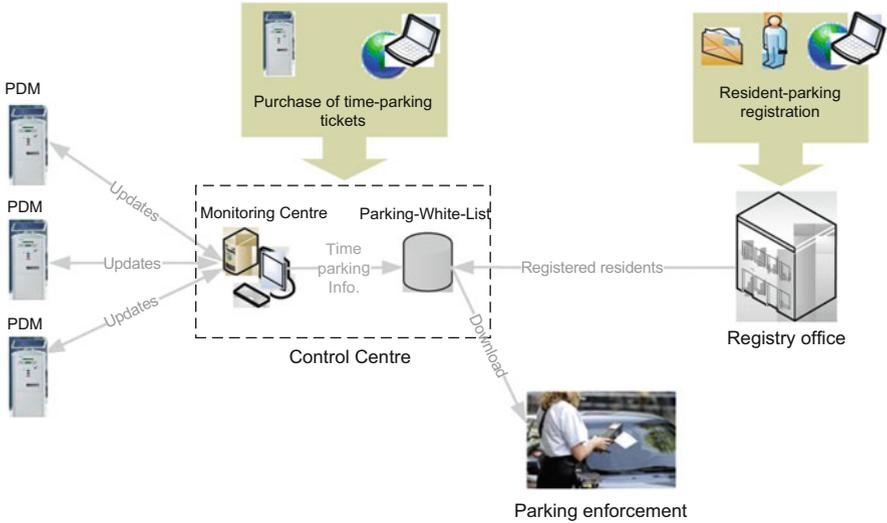- Other external interdependencies

Fig. 11.2 Context diagram of the PBL IoT system. The dashed box indicates the border of the Control Centre (© comeo.de)

(continued)

- [Impact of the system on its environment]
- [Overall completeness, consistency, and coherence]

Note that for the sake of brevity of this section we will ignore the aspects between brackets.

The nature of external interfaces was not addressed since this information was not available at the time of writing. Besides the mission statements in Sect. 11.1 we cannot yet predict how this system impacts its environments. This question can often only be addressed when the system is implemented and tested. The overall completeness, consistency, and coherence of the system was not addressed here, since, due the simplicity of the use case, and the strong boundaries put onto it by the business model (for instance, off-street parking is excluded), we felt that this item is fulfilled by default. Also notice that system scope already was provided in Section 0 In a regular architecture description, the scope is part of the business goals. There is thus a natural overlap of context view and business goals. In case the business goal already contains the full information about system scope and responsibilities,

Table 11.1 Types of parkers and the services to be offered

Type	Description	Services to be offered
Resident parker	Lives in the vicinity of the parking lot used. Needs to park on a frequent but not necessarily on a daily basis. Purchases a subscription for this type of parking	PBL on a subscription basis. Subscription shall be possible via walk in at the local Registry Office. Other access modalities include mail, email, web services, and calls
Time parker	Needs to park for a limited time interval on a location that is typically not in the vicinity of the driver's residence. Envisaged usages encompass short-time city parking (for instance for shopping) but also extended-stay parking at, for instance, airports	PBL on a pay-by-need basis. This type of permits shall be purchasable at PDMs, but also through web services (for instance, a smart-phone application)

(continued)

this information does of course not need to be repeated in the context view, but can rather be cross-referenced. Notice that the context view can be kept rather descriptive, but this cannot be done at the expense of completeness.

The Control Centre, which is the focus of the IoT architecture to be devised, is seen at the centre of Fig. 11.2. Also shown are purchase/transaction operations by the time parkers and the resident-parkers. The two types of parkers and what services shall be offered to each of them are summarized in Table 11.1. Figure 11.2 also contains on-street *Pay-and-Display Machines* (PDMs), parking enforcement, and the registry office. The latter maintains a database on resident parkers (name, address, permit purchased, etc.). What is inside and outside the scope of the IoT architecture to be generated is summarised in Table 11.2.

Notice that Table 11.1 can alternatively be part of the business goals (description of end customers and the services to be offered to them).

As shown in Chap. 6 Figure 3, the IoT Context View consists of two parts, the context view and the IoT Domain Model. In many cases it will be easier to construct the context view first, since (a) one does not yet need to understand the inner workings of the envisaged IoT system, and (b) the context view

Table 11.2 Overview of what components and interfaces in the context diagram (see Fig. 11.2) are part of the architecture to be devised

Within scope of the system architecture	Outside the scope of the system architecture
Control Centre; interfaces to PDMs, parking enforcement, and Registry Office. Parker; car	PDMs; web services for interacting with Control Centre (online time-parking tickets) and with Registry Office (Resident Parking Registration); Enforcement system

(continued)

focuses on the interfaces and what lies outside of the IoT system. The amount of detail on "outside interfaces" and the outside itself is usually much less than that of the IoT system itself.

11.2.2.1 Business Goals Revisited

As already mentioned the envisioned IoT system extends and improves existing car parking system. In the following we provide more information about the actors and devices involved, and also how their functionalities and roles are going to change due to the envisaged system enhancement. Such a detailed discussion is valuable not only from a mission-statement point of view, but also from an IoT-Domain-Model point of view, since it provides valuable additional information about the entities that form the IoT Domain Model, and how these entities interact.

Notice that contrary to the partition prescribed in Chap. 6 Figure 3, business goals and the IoT Context View (and the Physical Entity View) can of course be provided in one contiguous part of the architecture description. Such an aggregate presentation can make sense since all three (as in the example provided here) are characterised by a strong interdependence (chicken-and-egg problem!)]. If these two/three descriptions are indeed bunted together this needs of course to be clearly flagged in the table of content of the architecture description.

In this section, we shed more light on the planned improvement of the parking system by comparing the current functionalities of the entities in the context diagram with how they are going to look like after the planned improvement.

Pay-and-Display Machines (PDM)

Today: Parking Ticket Identification

PDMs are mounted on the side of public roads and have a major task of managing on-street parking places (Wikipedia 2013g). They allow a driver to buy a time-limited parking permit for a defined geographic region of on-street parking lots. After paying the parking fee, the PDM prints out the corresponding parking ticket. The driver is tasked to place the parking ticket visibly on the dashboard of her car.

Enhancement: Pay-by-License Plate

Our target is to simplify on-street parking by allowing the driver to head toward the nearest PDM, to type in the license plate number of her car, and to pay the parking fee. In this scenario she does not need to place a printed parking permit on the dashboard of her car. Instead the information entered (license plate) and the information about the permit (begin, end, zone) is communicated from the PDM to the Control Centre (see Fig. 11.2), where it is stored in a Parking-White-List database. The information stored in the database can be accessed by the parking-enforcement authority operating in the pertinent precinct of the municipality (see the below entry on the Registry Office).

Control Center

Today: PDMs Monitoring Centre

The Control Centre is a monitoring centre for multiple PDMs. It supplies PDMs with new parameter data such as current parking fees. It also monitors the PDM transaction data, cash-box status, and it provides statistical data (e.g., # of tickets sold) and status messages about the monitored PDMs to the users of the Control Centre.

A Control Centre is not always managed by the municipality itself. In many cases the management of the system is outsourced to a private company, but the municipality remains the owner of the data.

> Notice that ownership of the system parts (Control Centre) and also of the out-of-system parts (Registry Office, parking enforcement …) has direct implications for the requirements engineering (functional view, information view, security-risk analysis …).

Enhancement: Connection to Web, and to the Registry Office

Instead of buying a paperless time-parking permit from a PDM, drivers are given the possibility to execute the purchase online. To do so, a driver logs in to the corresponding webpage or installs an app on her smart device. The driver provides her license plate number, the parking zone, the parking time interval, and finalises her purchase by paying parking fee. The aforementioned information is then stored in the Parking-White-List database.

Registry Office

Today: Registering Residents and "Sticky" Permits

A registry office maintains a municipal database containing information on the current residence of persons and their permits. Residents can register for a parking permit by, for instance, providing pertinent information on the Registry-Office's Internet website; by sending the information by mail; or even by visiting the Registry Office in person. A standardised permit is handed to registered resident parkers. These permits are usually fixed to the car (glued to the inside of the windshield, etc.).

   Enhancement: The municipality offers "immaterial" permits. Here, the license-plate number is used for the identification of cars that are allowed to park on a resident-parking term. Information about these cars, viz. their license-plate numbers, the zones where they are allowed to be parked, and when they are allowed to be parked are provided to the Control Centre, which stores this information in a Parking-White-List database.

Parking-White-List Database

The purpose of this database is to maintain a parking white list, i.e. a list of cars -identified by their license-plates that are permitted to park within the region managed by the Central System. Besides the license-plate number, the white list also provides the geographical region, where the pertinent cars can be parked, and also when parking commences and when it ends.

Today: No Parking-White-List Database

While statistics about, for instance, how many permits have been purchased from the PDMs (see, for instance (Island Group 2012) can be retrieved from the database, no identification about the parked cars is performed in the current system.

Enhancement: Parking-White-List in Control Centre

A Parking-White-List for time-parkers and resident-parkers is made part of the Control Centre. The content of the Parking-White-List is gathered from three sources: PDMs, web services, and the Registry Office (see Fig. 11.2). The first two sources provide the Parking-White-List with information updates about time-parkers that have booked via PDMs or via the Internet. Information about resident parkers is provided by the Registry Office.

Enforcer/Handheld

Today: Controlling Parking Tickets and Resident Parking Permits

The task of the enforcer is to control whether a car is authorised to be parked in a specific zone and at a specific time. In the most common scenario, the enforcer is equipped with a handheld device that is capable of printing a paper ticket to be left at the vehicle. The handheld ensures that a variety of checks are executed on the data in order to eliminate invalid entries such as misspelled street names. The entered data is then transferred from the handheld either overnight or immediately via, for instance, GPRS to a back-end office, where the information about issued parking-violation tickets is stored.

Enhancement: Controlling the License Plate Number Only

The task of the enforcer changes in a sense that she does not need to struggle with badly visible parking tickets and resident parking permits in order to read and enter the data in the handheld. Instead, she scans the license-plate number of a parked car with her handheld. In the next step, she uses the handheld for checking the license-plate number together with the geographical location of the parked car, against the Parking-White-List. Notice that in this enhanced scenario, neither time-parkers nor resident-parkers need to place any permit visibly in their car.

11.2.3   IoT Domain Model as an Expansion of the Context View

As discussed in Sect. 6.3, in the IoT-A architecting process, the IoT Domain Model is generated in order to enrich the standard context view with IoT-specific context and with more details about the inner workings of the system. The latter is important for, among others, the requirements process
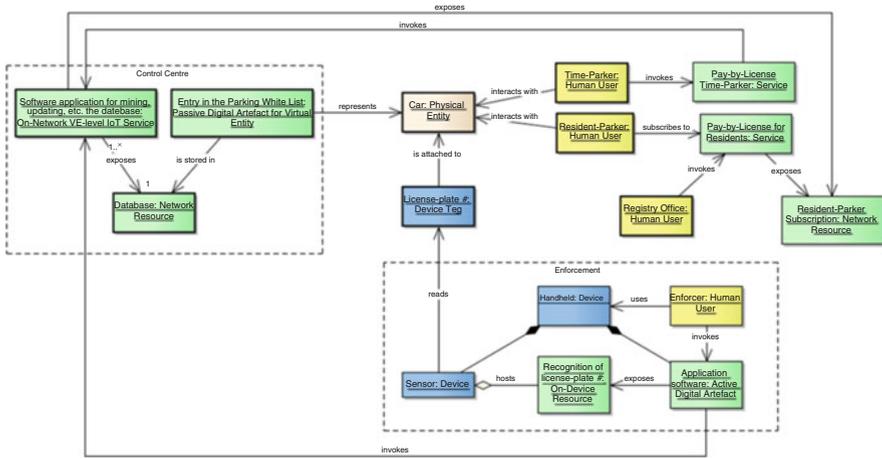
(continued)

Fig. 11.3 IoT Domain Model of the PBL system

(continued)

(see Sect. 6.4). Such a domain model also stipulates entity names and relationships to be used in the requirement process and for the derivation of other architectural views (functional view, information view, deployment view . . .).

The IoT Domain Model for the PBL system is shown in Fig. 11.3.

Notice that major input for Fig. 11.3 was derived from the previous Section on business goals.

Next we describe the steps we took for deriving the IoT Domain Model depicted in Fig. 11.3. After that, we discuss the particularities of the entities in the IoT Domain Model.

The previous figure provides an enriched and IoT-specific viewpoint to the context. This relates to the context diagram in Fig. 11.2; The legend reads as follows:

- In Yellow: human users;
- In Green: software;
- In blue: hardware;
- In beige: concept that fits in none of the previous categories.

- Classes with thick boundary lines: part of the architecture description. The architecture also covers all associations originating from or terminating at the Control Centre.
- Dashed boxes: system borders. Notice that only the Control Centre is within the system scope of the generated architecture.

11.2.3.1   Modelling Steps

System Users

The human/institutional system users can usually readily be inferred from the business goals and the context view (see above). In the following text we summarises the available information about the users so that we next can apply the IoT-Domain-Model mapping exercise in Sect. 9.1.7 (see Sect. 9.8).

Generally, the users of a system are interested in its functionalities. Our system has three functional outputs: it introduces a simple parking procedure, which is PBL; it allows to easily identify illegal parkers by means of RBL; and it increases parking revenues and public order due to quick spotting and processing of parking violations.

Who is interested in these functional outputs? By answering this question, we can determine the different categories of system users. In our use-case, the PBL is an interesting functionality for parkers, the RPL for enforcers, and the increase of the parking revenues and public order is obviously of a high interest for the municipality, e.g. the registry office. In the following, we define the users of each category.

- **Parkers**: human users. We distinguish between two types parkers: A resident-parker and a time-parker (see Table 11.1). The first is a resident that would like to have an affordable and easy solution to park his car on the street in his neighbourhood. The second is a driver that needs to park his car on a street for a limited period of time in order to accomplish a local activity. The time parker departs after the local activity is completed.
- **Enforcer**: The enforcer in this case, could be the human that uses the handheld or, being more granular, it could even be the application software that runs on the handheld and which is used by the enforcer. The IoT DM is flexible in terms of the granularity of modelling. We decided to model the enforcer as a user.

- **Registry office**: The Registry Office is a municipal office that can be considered as a system user. For the sake of clarity, we note for the reader that other offices, such as the public-order office and the police can be also be modelled as system users. Here, we only model the Registry Office as a user, since the other entities are not part of the business model. However, in a future extension of the system, these entities could of course come into the scope of the PBL Service and would then be added to the IoT Domain Model. This user provides the system with the newest information of the subscribed cars to the PBL Service.

Notice that the system users can be identified with a similar question as for the Physical Entity. In the latter case one asks the question what physical entity the system needs to interact with in order to fulfill its business goal. In the case of system users one asks who is interested in the output generated from system. This output encompasses of course also information inferred from interacting with the Physical Entity.

Procedure Application

In this section we model the different parts of our system (see Fig. 11.3) by applying the six-step procedure, in Sect. 9.1.7 to each of the four system-users: resident-parker, time-parker, parking enforcer, and the Registry Office. This six-step procedure yields six answers (A1 to A6), which are discussed below.

Resident-Parker

In order for a resident-parker to use the parking PBL facility, he needs to subscribe to it. Hence, we model this facility as Service (A1). The resident-parker is interested in parking her car. Therefore, we model the car as the PE (A2). The car is identified in the physical world by a license plate number. The latter is modelled as a Device of type Tag (A3). In the digital world, the car of a resident-parker is identified with an entry in a white list. We model an entry in a white list as a VE of type Passive Digital Artefact (A4). Entries of a white list are stored in a database that allows accessing the entries in read and write modus. This database is therefore, modelled as a Network Resource (A5). A software application is responsible for mining the

database white list, for instance for verifying whether a specific car is allowed to park in a given city zone, or if it is in unauthorised. This application software is modelled as an On-network Resource (A6). After the resident-parker successfully registers to the Resident PBL Service, her information needs then to be inserted in the white list database of parkers. This results into one Service invoking the other one as depicted in Fig. 11.3.

Time-Parker

Having the time-parker as an additional user of the system adds only one new part to the already described entities in the IoT Domain Model. A time-parker needs to subscribe to the time-parker PBL system. Here, we also model the functionality provided by the PBL system as a Service (A1). The remaining answers steps, viz. A2 to A6, are exactly the same as the ones for the resident-parker.

Enforcer

The enforcer, i.e. the traffic warden, invokes the application on the handheld (A1). The enforcer is interested in a parked car, which we have already modelled as a PE (A2). The car is identified by its license plate number, which we have already modelled as a Device Tag (A3). The handheld has a sensor that reads the license plate number to identify the car. The type of this sensor depends on the deployment and can be, for instance, an RFID reader or a camera. In any case, we model the handheld as a generic device and the sensor as a Sensor Device (A3) that reads the Device Tag. A car which is allowed to park, is identified in the digital world with an entry in a white list. We have already modelled this entry as a VE of type Passive Digital Artefact (A4). The handheld runs software that computes the sensor readings in order to identify the license plate. For example, in case of a Device camera, this software processes the images taken for a license plate. We model this software as an On-device Resource (A5). This Resource is then directly accessible by the user. Therefore, we do not have a Resource-level Service.

Registry Office

In order to feed the system with the newest information of the registered cars, the registry office invokes software to maintain/query the database. This is done by

invoking the same service as resident parkers, but with different user rights (right to delete entries; right to change payment status, etc. We have already modelled this software as an On-network Service (A1). Answers (A2) to (A6) are also the same as for the resident parker.

11.3   Requirement Process and "Other Views"

11.3.1   Requirement Process

As discussed in detail in Sect. 6.4, the requirements process generates view requirements. Major inputs into this activity are

- Business goals
- Physical Entity View
- IoT Context View

All three of them have already been discussed in greater detail above, and we are now progressing to the requirements-engineering step.

11.3.2   Requirements

Notice that we do not prescribe any particular requirement-engineering process for how to generate requirements. Rather, the IoT ARM offers a set of aids that ease the translation of requirements into architecture features. For the generation of the requirements a wealth of engineering approaches and aids is described in the pertinent literature. Just one example are the Volere requirements templates (Volere 2013).

An abridged list of requirements is provided in Appendix [requirements for concrete architecture].

Notice that for the sake of brevity, the list in Appendix [:requirements for concrete architecture] only contains an illustrative list of requirements that

(continued)

(continued)

shed light on the IoT ARM supported architecting process. In praxis, unabridged requirement lists can readily contain several hundred requirements. Most of the view requirements are related to the fact that this architecture is an upgrade to an existing system (see Sect. 11.1).

In this section we are not simply repeating the requirements in the Appendix, rather we discuss where and how they enter the architecting process.

As explained in Sect. 6.4, we organise requirements along three disjunct topics:

- View requirements
- Design constraints
- Qualitative requirements

The type of each requirement is listed in the second column to the left in Appendix [requirements for concrete architecture]. Let us have a look at each of the requirement types.

Notice that one does not create requirements ex nihilo, rather they are based on business principles (as indicated in the rationales of the requirements in Appendix [requirements for concrete architecture]). Also, the IoT-A Unified Requirements (see Appendix [requirements]) can be consulted for generating requirements for a concrete architecture (see Sect. 6.7).

11.3.2.1   View Requirements

Examples for view requirements are BPL #5 and #14, viz.

PBL #	Requirement type	Description	Rationale	View	Perspective	Functionality group	Functional component
5	View	Communication with the pay-and-display machines shall adhere to the OCIT2 standard.	This is the standard used in the current systems. It readily accommodates the new information types to be exchanged with the Control Centre (car ID, permit details). Also, it is an international standard, so it does not conflict with PBL #1. (business principle)	Functional	None	Communication	End to end communication
14	View	Parking white list is provided by the system.	In order to facilitate all the new envisaged usages (PBL for time parker and resident parker, as well as RBL for the parking enforcement) the Control Centre needs to maintain and provide access to a parking white list.	Information	None	Virtual entity	Virtual entity repository

As already stated above, the IoT ARM does not offer any specific support in deriving requirements (besides the inspiration provided by the Unified Requirements; see Sect. 6.7), rather the IoT ARM provides support in mapping the requirements onto the IoT-ARM concepts. This is exemplified in Section View requirements by the yellow-coloured columns. These columns are populated during the initial mapping of the requirements onto concepts used in the IoT ARM. The core concepts used in the IoT ARM are views and perspectives, and these are shown to the far left.

What view these requirements map onto is indicated in the fifth column from the left, viz. the view column. Here we have an example for a functional-view and an information-view requirement. Both of them can – already at this stage- be mapped onto the functional decomposition that was introduced in Sect. 8.2.2. PBL #5 can be mapped onto the End to End Communication FC in the Communication FG, while PBL #14 can be mapped onto the Virtual Entity FG. Mapping requirements at this early stage speeds up the population of the various architecture views with concrete goals.

Notice that PBL #14 is not mapped onto any of the FCs listed in Sect. 8.2.2, rather onto a new FC, i.e. a Virtual Entity repository. This reflects a design choice made, viz. to not include the Parking White List in the VE Resolution FC, rather in its own FC. One of the main reasons behind this design decision is the evolvability of the system. By keeping the white list apart from the Virtual Entity Resolution FC, it is easier to extend and change the system during future version iterations. This mapping is thus actually attributable to several of the qualitative requirements, viz. PBL #4, #11, #13, who all address the evolvability of the PBL system. See more on this in the below Section on qualitative requirements.

11.3.2.2   Design Constraints

Design constraints define constraints in the design of an architecture. An example for this is PBL #3, viz.

As one can see, this requirement is indeed a constraint in that it tells the architecture not to include payment transactions in the architecture, something that is tacitly covered in the IoT Context View (see above), but in order to avoid slips during the architecting process, it is often very helpful not only to state what is within the system scope but also what is outside of the system scope. An example for a design constraint at the reference-architecture level is UNI.071, viz. "A system built using the ARM shall provide standardised and semantic communication between services". Here, it is emphasised to standardise interfaces. In other words, non-standardised interfaces do not lie within the scope of the architecture.

PBL #	Requirement type	Description	Rationale	View	Perspective	Functionality group	Functional component
3	Design constraint	Payment transactions are out of scope of the system extension	The current parking-management system relies on the payment system by a third party and this modus operandi is not going to be changed in the new, extended version of the system. (business principle)	None specific	None	None specific	None specific

11.3.2.3 Qualitative Requirements

As discussed above and in Sects. 8.3 and 6.9, qualitative have impacts on more than one view. What is not mentioned in these Sections though, is that qualitative requirements can inform the same architectural design decision. In order to elucidate this point let us look at the three qualitative requirements that inform the decision to store the Parking White List in a Virtual Entity Repository instead of Virtual Entity Resolution FC. These requirements are listed below.

Notice that although all of these requirements are mapped onto the Evolution and Interoperability Perspective, none of them is openly mapped onto the Virtual Resolution Functional Component. This is because perspectives by default do not map on one view nor one FC. So how does one map such qualitative requirements? As discussed in Sect. 6.5, the IoT ARM follows the framework of Rozanski and Woods in that it advocates the choice of tactics in order to successfully map qualitative requirements onto architecture descriptions. Furthermore, as discussed in Sect. 6.9, the IoT ARM also provides guidance in terms of the desing-choice process, viz. what design choices are at hand after a certain tactics has been chosen. One of the design choices spelt out (see Sect. 6.9) is to build the architecture out of models and to couple the blocks loosely. In the context of the PBL architecture this design choice was translated into the decision not to store the Parking White List in the Virtual Entity Resolution FC, but rather to create a new FC, viz. the Virtual Entity Repository. By so doing one decouples, for instance, the evolution of the Virtual Entity Resolution FC from the Parking White List during future PBL version cycles, as long as the interfaces between both are kept up to date. In other words, instead of creating strong ties between resolution and the white list, the coupling is rather loose, and the respective FCs can thus evolve independently of each other.

As discussed above, most of the requirements in Appendix [requirements for concrete architecture] are qualitative in nature. This is mostly due to the fact that the business principles from which these requirements stem are behavioural requirements toward the entirety of the system. An example of this is requirement PBL #1, which stipulates that the system shall be deployable in many countries. Such a requirement has repercussions for many views, for instance information view and deployment view, and it is thus of a qualitative nature.

The Table in Appendix [requirements for concrete architecture] features requirements that are mapped onto perspectives that are not part of the IoT Reference Architecture (see Sect. 8.3). Examples for such requirements are PBL #1 (internationalisation and usability perspective) and PBL #2 (regulation perspective). These requirements are not covered in the IoT Reference Architecture (and thus in the design-choice process) because they are not

PBL #	Requirement type	Description	Rationale	View	Perspective	Functionality group	Functional component
4	Qualitative	The system shall be readily extensible to encompass off-street parking	In a future version of the system, off-street parking might also be offered or the system shall be able to seamlessly cooperate with third-party systems for off-street-parking management. (business principle)	None specific	Evolution and Inter-operability	None specific	None specific
11	Qualitative	The user base for the system shall not be limited to resident parkers and time parkers in future versions of the system.	In the future new business models are envisaged for the resident-parking service. An example is the inclusion of employees of participating companies in the resident-parking pool. By so doing, the customer base for the Control Centre and thus the total revenue that can be generated with the Control Centre can be extended. This will increase the attractiveness of parking services offered to, e.g., municipalities. (business principle)	None specific	Evolution and inter-operability	None specific	None specific

(continued)

PBL #	Requirement type	Description	Rationale	View	Perspective	Functionality group	Functional component
13	Qualitative	Existing system software shall be reused to a maximum.	Decrease the cost and increase the ROI of the enhanced Control Centre by maximum reuse of existing hardware within the system but also outside (compatibility with existing parking-enforcement systems, etc.). (business principle)	None specific	Evolution and Inter-operability	None specific	None specific

important for IoT systems. Rather, we were unable to find IoT-specific aspects and these and other perspectives. Notice that this does not mean that one cannot formulate a design-choice process for these perspectives. Rather, the architect is asked to rely on tactics provided in the literature and to formulate here own design choices. More insight on these and other perspectives and thereto related tactics can be found elsewhere in the literature (Rozanski and Woods 2011).

11.3.2.4 "Other Views"

Information View

The IoT IM details the structure of the information that constitutes a VE and the Service Description of a Service that acts on the VE (see Sect. 7.4.1). In this section we will describe the modelling of these two elements for the PBL use case. Notice that the information view does not cover data formats. These lie within the purview of the deployment view.

Modelling the VE

Following the IoT Information Model (see Sect. 7.4) a VE can have one or more attributes, each having an attribute Name and an attribute Type. In the PBL use case, a VE is an entry in the Parking-White-List database identifying a car that is allowed to use the PBL parking facility. Since we have considered two types of parking cars (the car of a resident-parker and the car of a time-parker), the VE for one car type is slightly different than the other one. The first two attributes (License plate numbers and Parking zone) depicted in Fig. 11.4 are common attributes for both types of VEs, while the third attribute (Parker type) is part of the VE resident-parker. Notice though that we only chose the license plate number as the VE and did not include in the parking zone. This design decision is based on several previous decisions. First, one of the main business principles behind the PBL system is its future extensibility. As discussed in the requirements Section above, there are many requirements that stipulate the evolvability of the PBL system. This, among others, boiled down into loose coupling rather independent FCs. One example for the latter is the choice to introduce a Virtual Entity Repository for the White List in the system architecture. In the information model we drive this modularisation one step further in that we chose a single piece of information, the license-plate number, as the VE. All the other entries in the Parking List are then associated to the VE. By so

doing one can, for instance, include more attributes in future version of the PBL system.

In the following we define each of the three attributes and discuss their decomposition:

- License plate number

  - attributeName: License plate number;
  - attributeType: Car information;
  - Description: It is a common attribute for both VEs. In essence, it is a numerical and alphabetical registration identifier that officially and uniquely identifies the car within an issuing region such as the entire country or entire state;
  - Values: A resident-parker or a time-parker may own and may have parked more than one car on the street. Therefore, it is necessary that this attribute has one or more Values, each one containing the License plate number of a registered car. The example in Fig. 11.4 states the registration of two license plate numbers: "M CJ 1234" and "M JW 5678".

- Parking zone

  - attributeName: Parking zone;
  - attributeType: Parking information;
  - Description: It is a common attribute for both VEs. This attribute identifies the parking zone, where this car is allowed to park;
  - Value: corresponds to the name or the identifier of a parking zone. The example in Fig. 11.4 depicts the registration Zone A.

- Parker type

  - attributeName: Parker type;
  - attributeType: Parking information;
  - Description: It is an exclusive attribute in the VE of a resident-parker. This attribute identifies the time during which, this car is allowed to park. Currently we differentiate between a full time parker (24/7) and a night parker (12/7);
  - Value: it is either 24/7 or 12/7. The example in Fig. 11.4 shows a registered night parker.

For the sake of clarity, we note for the reader that other attributes can be added for each VE as well as other Values and MetaData. These highly depend on the deployment of the PBL facility, which definitely changes e.g., from one city to another.
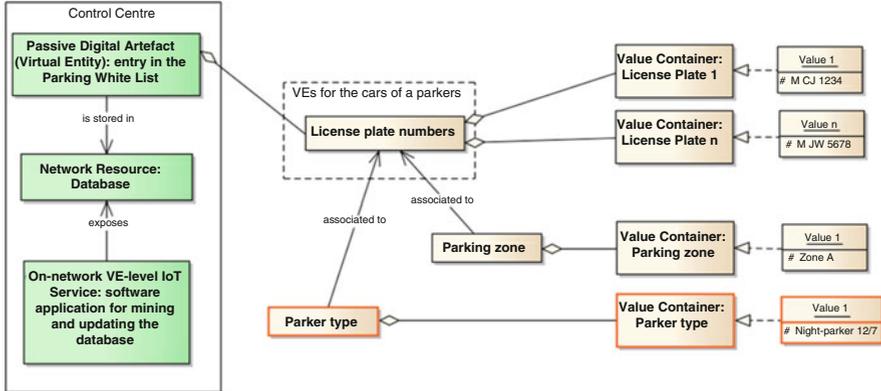
Fig. 11.4 IM of the VE for resident-parker and time-parkers (*Orange edge*: unique to resident-parking)

11.3.2.5 Functional View

Technical Scenarios

Besides the rather static descriptions provided by the Physical Entity View, the IoT Context View, and the business goals (see previous Sections), we have found that the semi-dynamic view of UML use-case diagrams is very helpful in the identifying salient FCs in the functional decomposition and also the interactions and interfaces of said FCs. Below we provide use-case diagrams for all major technical use-case scenarios of the PBL system.

Notice that the system-boundary boxes in the use-case diagrams are not synonymous to the boundary of the PBL system. Rather, they allude to entities in the context view (see Fig. 11.2). All thick-lined boundary boxes are part of the PBL system (see Fig. 11.3).

Purchase (and Change) of Parking Permit

See Fig. 11.5

This diagram summarises how the time-parker interacts with the Control Centre. It has implications for manipulations and thus the interface of the Virtual Entity Repository, but also for the VE Resolution, because in order to extend a permit it first has to be located in the system (Fig. 11.6).
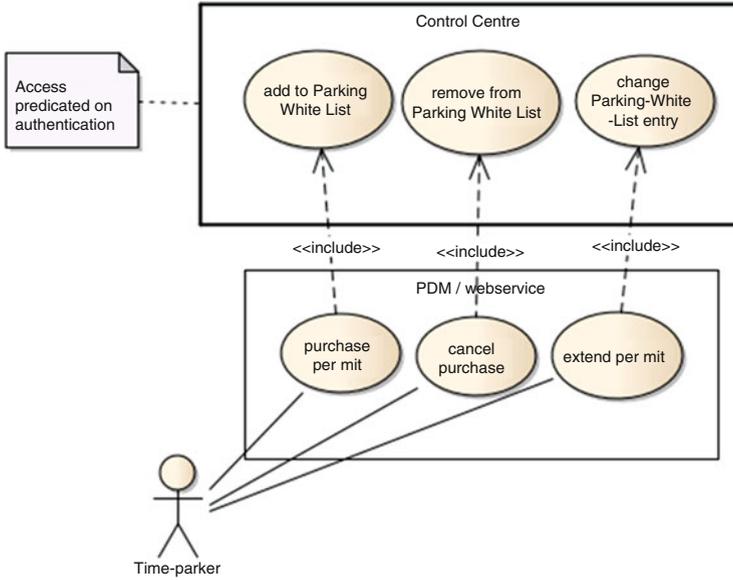
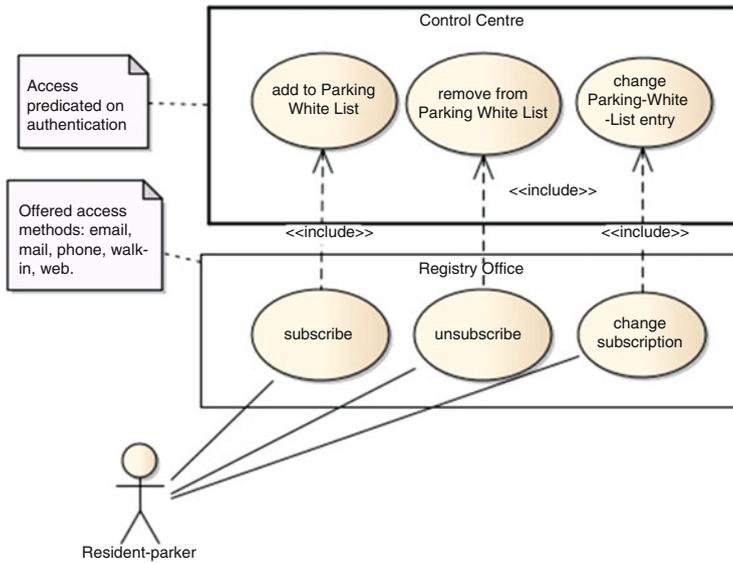**Fig. 11.5** Technical use case – purchase of parking permit by time-parker

Fig. 11.6 Technical scenario – subscribe/unsubscribe/change by resident-parker

(continued)

This technical use case summarises how the resident parker interacts with the Control Centre. Notice that the actions on the primary-service level are not part of the PBL system. It has implications for manipulations and thus the interface of the Virtual Entity Repository, but also for the VE Resolution, because in order to extend a permit it first has to be located in the system.

On-Street Parking

This technical use case summarises the actions triggered when time-parkers and resident parkers actually park their car. Since the time-parking scenario is of an ad-hoc nature (all pertinent actions conducted shortly prior to or during parking), while the resident-parking scenario is of a recurring nature (payment of fees, etc. well in advance to individual parking events), the former incorporates many more use cases then the latter. This technical use case has implications for the interface of the VE FC and the Security FC and the interface the PBL exposes toward the PDM/webserver (Figs. 11.7 and 11.8). This use case has no implications for the architecture.

Parking Enforcement

See (Fig. 11.9)

This use-case diagram summarises the parking-enforcement scenario. Notice that the "get licence plate" use case includes the parked car as an empty system. This technical use case has implications for the interface of the VE FC and the Security FC and the interface the PBL exposes toward the PDM/webserver.
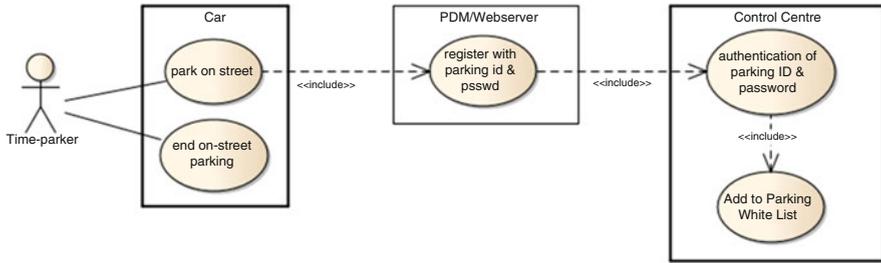
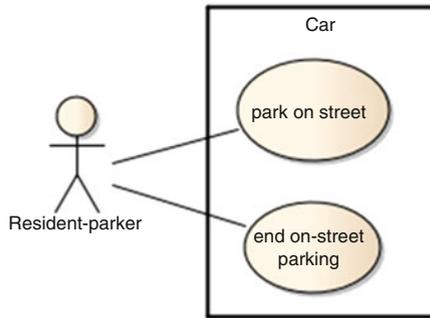**Fig. 11.7** Technical scenario – on-street parking by time-parker

Fig. 11.8 Technical scenario – on-street parking by resident-parker
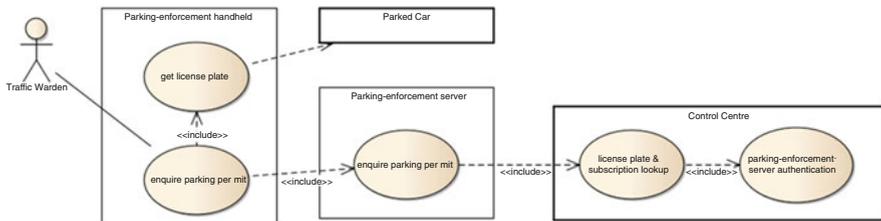

Fig. 11.9 Technical scenario – parking enforcement

Modelling the Service Description

Following the mapping of the IoT DM to Service Description explained in (Martín 2012; Sect. 4.6.3), we model the VE-level IoT software application for mining the Parking-White-List database. Notice that multiple other software applications may act on attributes of VEs and can be modelled as well. Examples of these software applications are updates of attributes; running statistical inference on VEs; applying mathematical operations on VEs; and representing attribute values on graphs and charts. Here we focus on the modelling of the mining software.
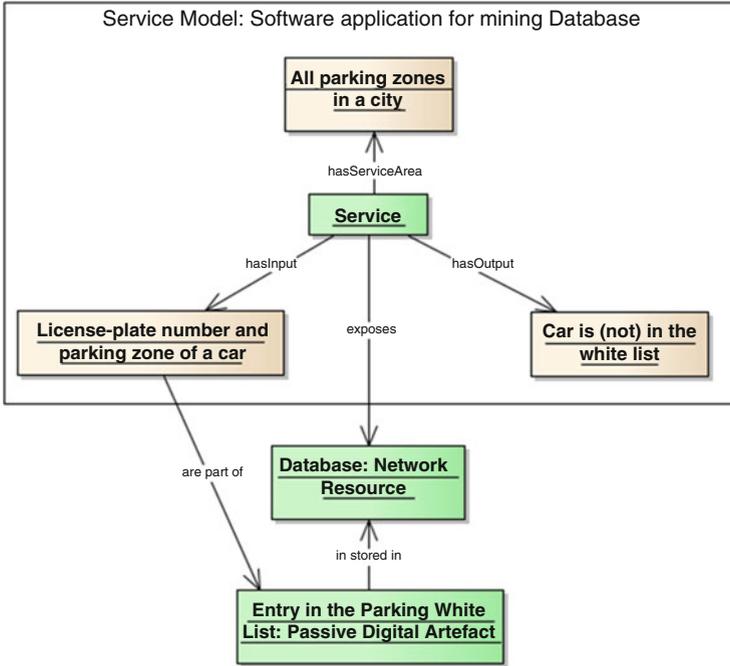
Fig. 11.10 Service description for the PBL system

Figure 11.10 depicts the Service Description of the mining software. In the following, we will highlight the service specifications:

- hasServiceArea: This service runs in the Control Centre that is generally responsible of managing PDMs in a single city. Consequently, all parking zones in a city are affected by this service;
- hasInput: In order for the mining service to verify if a car is allowed to park, the enforcer needs to provide the service with three input data: The current time, the geographical location or the zone of a parked car, and the license plate number of the parked car;
- hasOutput: Having the three aforementioned input data of the parked car, the mining service verifies all the VEs in the Parking-White-List database. After the verification, two results are possible:

  - The given license plate number is matched in one of the VEs: In this case, the service compares the given geographical parking place and parking time with the corresponding attributes of this VE. If the car is not allowed to park at this place and/or at this time, the service decides that it is a violator. Otherwise this car is allowed to park;
  - The given license plate number is not found in the database: In this case, the service decides that the parked car is a violator.

Exposes: As previously explained in the domain modelling of the PBL (see ARM document Sect. 11.2.3), this service exposes the Parking-White-List database as Network Resource.

Chapter 12
ARM Testimonials

**Edward Ho, Tobias Jacobs, Stefan Meissner, Sonja Meyer,
Miguel-Angel Monjas, and Alexander Salinas Segura**

This chapter shows how the IoT ARM is perceived by the IoT community and how the ARM can be placed in relation to existing IoT related standards and research projects. The first sections of this chapter present reverse mappings of existing standards and platforms to the IoT ARM and the last section of this chapter shows a business case evaluation for an example use case in the healthcare domain.

E. Ho (✉)
University of St.Gallen, Dufourstrasse 40a, CH-9000 St.Gallen, Switzerland
e-mail: edward.ho@unisg.ch

T. Jacobs
NEC Laboratories Europe, Software & Services Research Division, NEC Europe Ltd.,
Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
e-mail: tobias.jacobs@neclab.eu

S. Meissner
University of Surrey, Stag Hill, GU2 7XH Guildford, UK
e-mail: s.meissner@surrey.ac.uk

S. Meyer
SAP AG, Dietmar-Hopp-Allee 16, 69190 Walldorf, Germany
e-mail: sonj.meyer@sap.com

Miguel-Angel Monjas
Ericsson Spain, Madrid R&D Center, Technology and Innovation, Vía de los Poblados, 13,
28033 Madrid, Spain
e-mail: Miguel-Angel.Monjas@ericsson.com

A. Salinas Segura
University of Würzburg, Josef-Stangl-Platz 2, 97070 Würzburg, Germany
e-mail: alexander.salinas@uni-wuerzburg.de

12.1 Introduction to Reverse Mapping

In the course of its own project roadmap, our sister project – the *Internet of Things Initiative* (IoT-i) – has targeted three different (but connected) activities directly relating to the IoT-A architecture work as shown below:

1. To review and categorise existing reference models having a connection to the IoT field (or underlying disciplines, as IoT as such is more a technology umbrella). Example of the reference models reviewed by IoT-i are ETSI M2M, IETF Core, EPCglobal, Ucode and NFC to name just a few (IoT-A D1.2);
2. To put online a survey, the goal of which was to capture, people understanding and expectation, as far as reference models are concerned. This exercise was very important because people have generally different understanding about what are reference models, architectures and what they should consist of;
3. Finally, to come back on reference models introduced and summarised in previous versions of this deliverable and to do a reverse mapping exercise towards the IoT Reference Model. The goal of this exercise was to show that the reference model as defined by IoT-A is expressive enough in order to allow a modelling of those (pre- IoT-A) existing IoT reference models using the IoT-A one. In other words, if we would consider that IoT-A does not attempt to define what is an IoT system using sentences and words, but defining models where any IoT system (from the IoT understanding) shall fit, then all those existing reference models would be IoT systems reference models.

In this Section we aim at giving some details about this reverse mapping exercise applied to ETSI M2M, EPCglobal and uID. Some of the material in this Section comes directly from the IoT-A D1.5 deliverable (Carrez et al. 2013) (especially the UML figures and concept tables). In order to improve readability, we do not use direct citations, although the work presented in the following Section was performed by the IoT-A project and reported in their deliverable D1.5.

In addition to the standards that we have mentioned above, we also apply the IoT Architectural Reference Model to a concrete architecture, namely the architecture of the MUNICH (MUNICH 2010) project in order to validate the IoT ARM against a real system in contrast to an abstract standard. Furthermore we show a reverse mapping to the information model of the IoT-related research project BUTLER[1].

12.2 Reverse Mapping ETSI M2M

Within the IoT-A D1.5 deliverable, Sect. 3.1.1 discusses the ETSI M2M standard (ETSI TS 102 690). In this section we analyse the ETSI M2M standard. The acronym ETSI stands for *European Telecommunications Standards Institute*

---

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

(ETSI), viz. the standardisation body responsible for this standard. The acronym M2M stands for Machine-to-Machine, which is a pointer to the application field this standard addresses, viz. machine-to-machine communications. Release 1 of this standard was published in October 2011 (ETSI TS 102 690), this discussion within IoT-A also takes the later update (ETSI TS 103 092) into account that was released in May 2012.

The purpose of the ETSI M2M functional architecture is to define a service-capability layer which serves as middleware between applications in the Internet and Devices or gateways residing in local-area networks. The current release is mainly concerned with secure and reliable data transport.

In what follows, we give a more detailed description of a possible reverse mapping of ETSI M2M to the IoT Domain Model and IoT Communication Model as well as their management information model and how it maps to our management model. We also have a brief look at the ETSI M2M security model and how it compares to our threat analysis.

12.2.1 Mapping to the IoT Domain Model

As everything above the ETSI M2M Service Capability Layer is considered an application, there is no explicit concept of a User in ETSI M2M. In particular, Human Users are out of scope, as the standard focuses on machine-to-machine communication. The role of an IoT-A User would typically be taken by ETSI network applications, in some cases also by ETSI gateway applications, because these applications use the information provided by sensing M2M Devices and control the actuation capabilities of Devices.

ETSI M2M defines Sensors and Actuators in a similar way as the IoT Domain Model. However, there is a subtle difference regarding the concept of a Device. While in IoT-A there is a "is-a" relationship between Sensor/Actuator and Device, ETSI M2M defines a Device to be a unit comprising Sensors and Actuators, as well as embedded processing and communication capabilities – so here Sensors and Actuators are part of Devices.

The ETSI M2M defines a Service Capability Layer with standardised interfaces. Since this layer includes similar functionalities to the IoT-A Service level (e.g. registration), it is reasonable to map these functionalities to IoT Services. There are also some differences between ETSI and IoT-A terminology. For example, the ETSI Services are not only exposed towards actors which IoT-A would consider as Users, but also towards (ETSI) applications residing on Devices. Additionally, the concept of IoT Resource (IoT-A) as a native software interface of Devices does not explicitly exist in ETSI M2M – although software components on legacy Devices could be considered as IoT Resources. Instead the term of Resources in ETSI is exclusively used to describe the RESTful interface exposed by the Service Capability Layer.

Table 12.1 Mapping ETSI M2M concepts to the IoT-A Domain Model

ETSI M2M	IoT Domain Model	Comments
Device	Device	Sensors and Actuators are hosted on Devices, they are not special cases of Devices
Sensor	Sensor	The Sensor in ETSI M2M is not a Device
Actuator	Actuator	The Actuator in ETSI M2M is not a Device
Network application	User	In ETSI M2M, there are no Human Users, but only applications that process the data coming from the "Device and Gateway Domain". This concept of an application as a User is reflected in IoT-A
Gateway application	User	In ETSI M2M, there are no Human Users, but only applications that process the data coming from the "Device and Gateway Domain". This concept of an application as a User is reflected in IoT-A
Service	Service	In ETSI M2M, Services are not defined as exposing Resources on Devices, but can interact with the Devices. A Resource concept as in IoT-A does not exist
Resource	Service	Resources in ETSI M2M are defined in analogy to RESTful Service Interfaces

The mapping of ETSI M2M concepts to the IoT Domain Model is shown in Table 12.1.

As the current ETSI M2M release is rather concerned with data transport than with real-world modelling, the (physical, virtual, augmented) entity concept is not defined in (ETSI TS 103 092).

12.2.2 Mapping to the Management FG

Management functionalities are an inherent part of both of IoT-A and ETSI M2M. Both architectures distribute and cluster the management functions into different packages or functional components.

In (ETSI TS 102 690), the following packages are defined for management:

- General Management (GEN): Allows retrieving general information of the M2M Device or gateway, and provides generic mechanism applicable to different specific management functions;
- Configuration Management (CFG): Allows configuration of the device capabilities and features for supporting M2M Services and applications, including activating/deactivating device hardware components or I/Os in the M2M Device or gateway;
- Diagnostic & Monitoring Management (D&M): Allows running specific diagnostic tests on a device and collecting the results or alerts from the M2M Device or gateway. This package is also called Fault and Performance Management;

- Software/Firmware Management (SFW): Allows installation/update/removal of application specific or SCL related software/firmware in M2M Device or gateway;
- Area Network Management (ANW): Allows M2M Gateway-specific configuration and M2M Area Network and Device management through a M2M gateway;
- SCL Management (SCL): Allows remote configuration and retrieval of M2M Device or gateway service capability layer parameters.

In a similar fashion, Sect. 8.2.2 of this document identifies different Functional Components used for management functionalities. These include:

- **Configuration**: Initialising the system configuration. Gathering and storing configurations from FCs and Devices, tracking configuration changes;
- **Fault**: The goal of the Fault FC is to identify, isolate, correct and log faults that occur in the IoT system;
- **Member**: This FC is responsible for the management of the membership and associated information of any relevant entity (FG, FC, VE, IoT Service, Device, Application, and User) to an IoT system;
- **Reporting**: The Reporting FC can be seen as an overlay for the other Management FCs. It distils information provided by them. One of many conceivable reporting goals is to determine the efficiency of the current system;
- **State**: The State FC monitors and predicts state of the IoT system. For a ready diagnostic of the system, as required by Fault FC, the past, current and predicted (future) state of the system are provided.

When mapping these different management components, it becomes obvious once again that the focus of ETSI M2M is narrower in terms of its scope and therefore it is more detailed in the definition of its management capabilities and does not include all of the functionality defined by IoT-A. For instance, there is no equivalent to State FC in terms of its temporal distribution and the related billing capabilities. This aspect is not really central, as it is not contradictory and could be built upon the D&M package. In general however there is a strong overlap, as D&M roughly relates to the Reporting FC, CFG closely resembles Configuration, and both D&M and Fault deal with monitoring functionalities. Error and fault handling as such is handled specifically in the Fault FC, whereas D&M also handles performance management. On the other hand, and in line with the general focus of ETSI-M2M, the different aspects of the Configuration FC are handled in more specific packages in ETSI M2M, such as SCL, ANW, and SFW which each deal with specific functionalities that are subsumed under Configuration in IoT-A. As the different architectures naturally have different levels of abstraction, it is not surprising to not have a 1:1 relationship between the two architectures, but a mapping can be performed easily in both directions.

12.2.3   Mapping to the IoT Communication Model

The ETSI M2M standard defines a Service Capability Layer in order to enable seamless, secure, and reliable end-to-end communication in M2M networks. The ETSI Service Capability Layer can therefore be mapped to the end-to-end layer of the IoT Communication Model (see Sect. 7.6). (ETSI) applications, communicating via the Service Capability Layer, would accordingly be associated to the IoT-A Data Layer (see Sect. 7.6.2), although they do not only exchange data, but also control and management information.

A Network and ID group (see Sect. 7.6.2) is not in the focus of ETSI M2M, and the current bindings to HTTP and CoAP do not assume such a layer. However, in cases where the Service Capability Layer enables a direct connection of mobile Device applications to network applications, an ID layer that describes the Device independently from its network location could assist the Service Capability which provides seamless connectivity. The three communication layers at the bottom of Fig. 7.17 can be considered as identical in ETSI M2M and IoT-A.

From the point of view of ETSI M2M, all actors making use of the Service Capability Layer are applications. The model distinguishes between device applications, gateway applications, and network applications. ETSI M2M also considers so-called legacy Devices; these are Devices that have no own Service Capability Layer and therefore need to be integrated via a gateway application into the M2M system (M2M system is a term implicitly used in ETSI M2M to refer to the overall architecture).

The IoT-A term IoT Device is used more or less in the same way in ETSI M2M, but the concept of an IoT Application does not directly exist in ETSI M2M – mainly because the concept of an application is more broadly defined in ETSI M2M.

12.2.4   Mapping to the Security Model

One of the purposes of the ETSI M2M Service Capability Layer is to address all security requirements of M2M communication. The standard defines a key hierarchy of three levels. The ETSI M2M Root Key is used for mutual authentication between device or gateway nodes and the M2M Service Provider. It is also used for deriving and agreeing on the key of the next layer of the hierarchy – the ETSI M2M Connection Key which is used for every service connection procedure. Finally, the ETSI M2M Application Key is used for securing sessions between specific applications. This largely maps to the IoT-A Key exchange and management functionality in IoT-A with respect to key management is not yet explicitly defined in this document.

Most of the communication security and Service security aspects of the IoT-A security model are implicitly addressed in ETSI M2M – although the terminology of IoT-A is not explicitly used. The ETSI M2M standard describes a range of variants that depend on the security characteristics of the underlying network layers and on the relationships between the M2M service provider and the network operator. For example, if these stakeholders are identical, key provisioning can be significantly simplified. One issue clearly not addressed in ETSI M2M are trust models.

12.2.5 Threat Analysis Mapping

(ETSI TR 103 167) deals with a threat analysis related to the ETSI M2M standard. In a similar way as the risk analysis provided in this document in Section. [Chapter 6 Sect. 6.8] ETSI M2M defines those threats that are most relevant for the standard, and discusses respective countermeasures. Here, the different focus of ETSI M2M in terms of network security becomes obvious again, because most of the threats identified by ETSI M2M deal with keys or message exchange. That means that the scope of IoT-A is broader, as it also includes, for instance, Human Users that do not behave correctly. Consequently, IoT-A refers to a general risk analysis that includes by definition non-malicious behaviour that still imposes a risk on the system. As the scope of IoT-A is broader, not all the risks identified within IoT-A are applicable to ETSI M2M, but the threats of ETSI M2M map well to the risks identified within Sect. 6.8. This is shown in Table 12.2 below.

As we can see in Table 12.2, there is a slight difference between both models regarding the consequence or the cause of a risk, as ETSI M2M has a stronger focus on what actions are actually applied in order to impose a risk on the system, whereas IoT-A focuses more on the consequences of these actions. Nevertheless, there is a good mapping between the two models. The granularity of ETSI M2M is naturally higher, as it focuses on a more narrow class of threats.

12.2.6 Conclusion

If we consider that the aim of the ETSI M2M standard is to provide an M2M architecture with a generic set of capabilities for M2M Services and to provide a framework for developing Services independently of the underlying network, it becomes clear that the scope of IoT-A is much broader, taking the entire Internet of Things domain into account, esp. by explicitly modelling entities and also providing a much more fine-grained set of relationships between the different

Table 12.2 Mapping ETSI M2M threat analysis to the IoT-A risk analysis

ETSI M2M	IoT-A
Threat 1: Discovery of long-term service-layer keys stored in M2M devices or M2M gateways	Attacker gains knowledge of sensitive exchanged data
	Disclosure of identities and cryptographic material
Threat 2: Deletion of long-term service-layer keys stored in M2M devices or M2M gateways	Disruption of a global Service
Threat 3: Replacement of long-term service-layer keys stored in M2M devices or M2M gateways	Disruption of a global Service
Threat 4: Discovery of long-term service-layer keys stored in the SCs of the M2M core	Attacker gains knowledge of sensitive exchanged data
	Disclosure of identities and cryptographic material
Threat 5: Deletion of long-term service-layer keys stored in the SCs of an M2M core	Disruption of a global Service
Threat 6: Discovery of long-term service-layer keys stored in MSBF or MAS	Attacker gains knowledge of sensitive exchanged data
	Disclosure of identities and cryptographic material
Threat 7: Deletion of long-term service-layer keys stored in the MSBF/MAS	Attacker gains knowledge of sensitive exchanged data
	Disclosure of identities and cryptographic material
Threat 8: Discover keys by eavesdropping on communications between entities	Attacker gains knowledge of sensitive exchanged data
	Disclosure of identities and cryptographic material
Threat 9: Modification of data stored in the M2M service capabilities	Alteration of the return value upon service invocation
	Attacker alters leaf-device content so that a user will eventually be redirected to a malicious content
	Attacker alter sensor device so that monitoring of a Physical Entity fails
Threat 10: Provisioning of non-legitimate keys	Disruption of a global Service
Threat 11: Unauthorised or corrupted application and service-layer software in M2M	Attacker impersonates infrastructure Services, compromising IoT functionalities and/or other dependent infrastructure services
Threat 12: Subverting the M2M device/gateway integrity-checking procedures	Alteration of the invocation of a Service
Threat 13: Unauthorised or corrupted software in M2M core	Attacker impersonates infrastructure Services, compromising IoT functionalities and/or other dependent infrastructure services
Threat 14: Subverting the integrity-checking procedures in the M2M core	Alteration of the invocation of a Service
Threat 15: General eavesdropping on M2M service-layer messaging between entities	Attacker gains knowledge of sensitive exchanged data

(continued)

Table 12.2 (continued)

ETSI M2M	IoT-A
Threat 16: Alteration of M2M service-layer messaging between entities	Alteration of the invocation of a Service
Threat 17: Replay of M2M service-layer messaging between entities	Compromised intermediary devices alter traversing data
	Alteration of the invocation of a Service
Threat 18: Breach of privacy due to inter-application communications	User is involved in transactions with a malicious peer
	Attacker gains knowledge of user private parameters
Threat 19: Breach of privacy due to attacks on M2M device/gateway service capabilities	User is involved in transactions with a malicious peer
	Attacker gains knowledge of user private parameters
Threat 20: Discovery of M2M long-term service-layer keys from knowledge of access-network keys	Attacker gains knowledge of sensitive exchanged data
	Disclosure of identities and cryptographic material
Threat 21: Transfer of module containing access-network keys and/or M2M long-term keys to a different terminal/device/gateway	Attacker gains knowledge of sensitive exchanged data
	Disclosure of identities and cryptographic material

kinds of devices, resources and services. While ETSI M2M makes different assumptions, especially in terms of security and communication, the basic concepts are somewhat compatible, at least on an abstract level of discussion. The major difference is that IoT-A is based on the assumption that the IoT Device space can be divided into the two main categories of constrained networks (NTU) and unconstrained networks (NTC), and the security measurements mainly need to address the boundaries between them, whereas ETSI focusses so far on the M2M Service Layer and its interfaces (ETSI TR 103 167) and not on the M2M Area Network Layer, so that IoT-A has a more network centred view of security than ETSI M2M. That being said, the functionalities discussed in Sect. 6.8 largely represent in (ETSI TR 103 167; Sect. 10.2), so that a mapping is feasible on the same abstraction level as the IoT Domain Model can be mapped to the ETSI M2M Service Capability Layer.

12.3   Reverse Mapping EPCglobal

The EPCglobal high-level architecture was introduced briefly in D1.5 deliverable of the IoT-i project (Haller 2012) Figure 12.1 gives a simplified view of the EPCglobal system architecture, taken from [EPCglobal]. It is worth noting that the tag itself is not represented as this figure ends up (at the bottom) with the air interface.
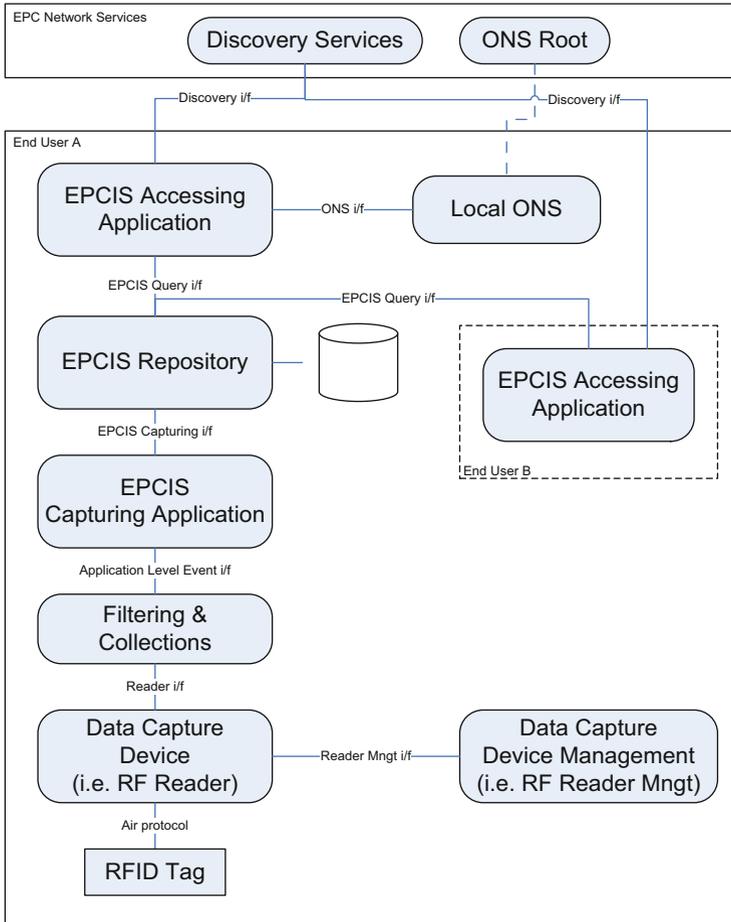
Fig. 12.1 EPCglobal system architecture (simplified)

12.3.1 Mapping to the Domain Model

In the EPCglobal architecture, the unique identifier associated with a physical object is the *Electronic Product Code* (EPC). It is defined by the EPCglobal Tag Data Standard, which defines its structure and encoding rules. Uniqueness of encoding structure (in order to avoid name collisions) is ensured by the use of a central Registration Authority.

The EPC Network Services in Fig. 12.1 are under the responsibility of the EPCglobal central authority and they are responsible for respectively providing discovery service to EPCglobal parties (end-users). The *Object Naming Service* (ONS) root management is also under the responsibility if the central authority

since it is the one allocating the EPC blocks. Local ONS are under the responsibility of the EPC manager (one per registered end-user).

After getting the address of an *Electronic Product Code Information Service* (EPCIS) responsible for the EPC of interest, an EPCIS Accessing Application will use the EPCIS query interface (i/f) to query additional information about an EPC (like class level/instance level or transactional data about a particular EPC). EPCIS query interface uses both push and pull mode, which means that it can be also used to receive notifications of observations concerning a particular EPC.

EPCIS Repository is the functional block, located at the "end-user A" side, deals with storage of information (of any nature) it wants to share with other parties (e.g. end-user B) of the EPC Global network. Of course all interfaces have to be implemented following the EPCglobal standards, however a certain level of freedom is left to "end-users" as for how those block shall be implemented.

The ONS block is a simple look-up Service that will map an EPC to the address of a designated EPCIS Service by which information about the EPC can be found.

The Filtering & Collection functional block is responsible for collecting raw tag data following policies defined by the EPCIS Capturing Application box. Example of such policy is: gathering all EPC of a certain class that have been read on a certain date, location and time interval.

The EPCIS Capturing Application supervises the operation at the lower level of the model and provides business context by coordinating with other components involved in a given business process. Again, a lot of freedom is left to the end-user for implementing this box, as far as the Application Level Event (ALE i/f) and capturing i/f are implemented according to the EPCglobal standards.

To finish up with the lower level, the Data Capture Device box (Tag Reader) is the one observing events relating to RFID Tags. The corresponding Reader i/f provides those events to the Filtering & Collection box.

The purpose of the reverse mapping is to check if the EPCglobal architecture is compatible with the IoT Reference Model.

The EPCglobal architecture illustrated in Fig. 12.1 is not exactly an EPCglobal domain model (as we understand IoT Domain Model in IoT-A), but rather a high-level diagram of a concrete architecture. Because the two models are not exactly similar in nature (i.e. IoT Domain Model is clearly at the "concept" level while the EPCglobal is a high level system architecture description) the reverse mapping of the EPCglobal architecture towards the IoT Domain Model is not a straightforward or simple process.

So in the following we use the EPCglobal system architecture in order to extract the EPCglobal concepts and then build an EPCglobal domain model taking a basis the generic IoT DM (meaning we try linking the EPCglobal concepts using the IoT

Table 12.3 Mapping EPCglobal concepts to the IoT Domain Model

EPCglobal concept	IoT ref. model concept	Comments
Entity	Physical entity	Is the Physical object been tracked by the EPCglobal system
End-user	User	The user managing and using the EPCIS, and reading the EPC
Partner user	User	The user willing to access EPC information for their own business
Physical entity	Physical entity (special case of)	Corresponds to physical objects like parcels, objects etc.. . .
Location	Physical entity (special case of)	Places, room, lift,. . .
RFID tag	Tag	The physical tag embedding the EPC
Tag reader	Device/Sensor	
Reader interface	Service	
EPC manager	Service	Is granted a portion of the naming space and assigns EPC to products
EPCIS accessing application	User	Located at end-user side that is willing to access EPC related information
EPCIS service	Service	Service that encompasses interfaces for data exchange (through the EPCIS Query Interface e.g.) and specification of Data (EPCIS data standard)
EPCIS query interface	Service	Interface exposed by the EPCIS and accessed by the EPCIS Accessing Application
EPCIS capture interface	Service	
EPCIS repository	Service/Resource	Exposes the EPCIS Query Interface. Stores info about EPCs events. . .The actual functionality of storing (e.g. in a data base) could/should be modelled as a Resource whereas the component that exposes the interface would be a Service. Of course that could be implemented tightly coupled as one software component
EPC record	Virtual entity	Consists of all info related to EPC (stored in EPC Data Base)
EPC data base	Network resource	
EPCIS capturing application	Service	Exposes the EPCIS capture interface
Filtering & collection	Service/Resource	Exposes the filtering and collection interface. Collects tag reads over time intervals constrained by events definition by the EPCIS Capturing Application. Filtering functionality may be modelled as a Resource, whereas exposing the interface as a Service

DM relationships, and try mapping the EPCglobal concepts to the IoT-A concepts) Nevertheless, we still executed to reverse mapping by building an EPCglobal domain model taking a basis the generic IoT DM (meaning we linked the
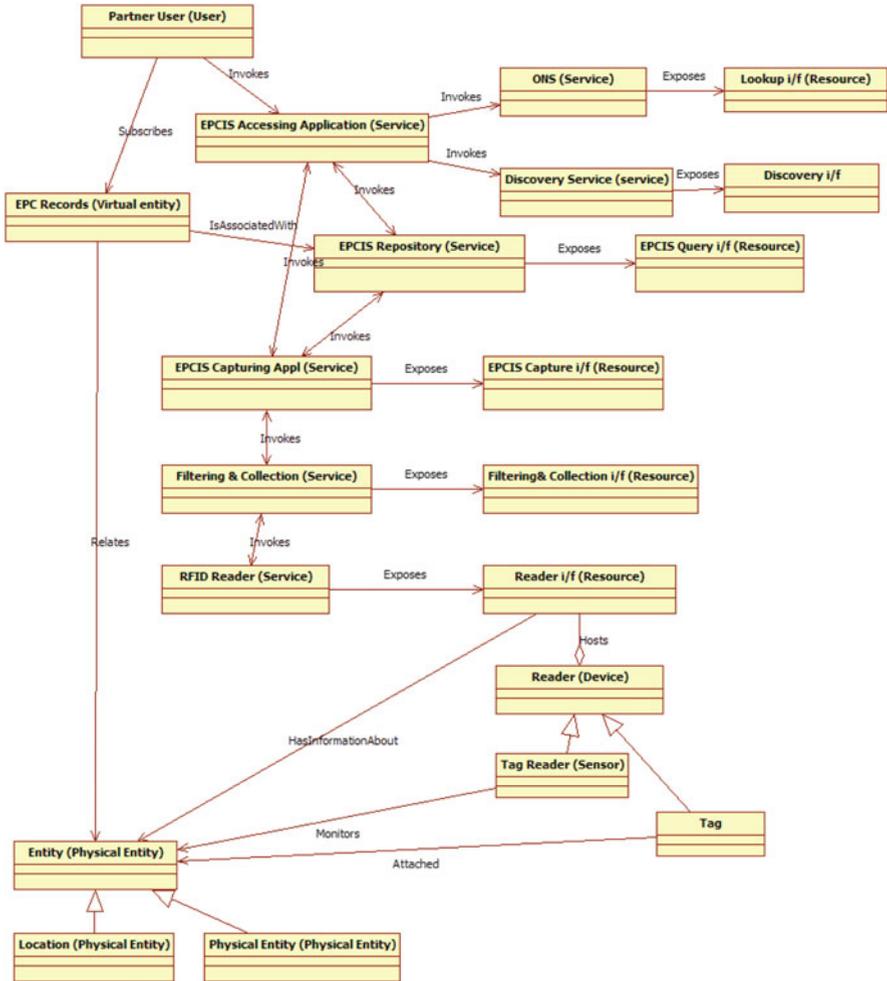
Fig. 12.2 EPCglobal domain model fit into the IoT Domain Model

EPCglobal concepts using the IoT DM relationships, and mapped the EPCglobal concepts to the IoT-A concepts).

First we identified a list of concepts that can be extracted from the EPCglobal system architecture and mapped them to the corresponding IoT DM concepts. This mapping is illustrated in Table 12.3.

Then according to the IoT Domain Model, the kind of concepts it handles and how those concepts are connected through relationships, the following (see Fig. 12.2) and consistent UML EPCglobal domain model could be extracted. As it fits the IoT Domain Model framework it can be argued that EPCglobal fits the IoT

Domain Model and that EPCglobal is truly an IoT system from the IoT-A definition point of view.

However, during this reverse mapping exercise IoT-A raised few comments:

1. Difficulties to model interfaces in general, as interfaces are not part of the IoT Domain Model in IoT-A. But it can also be argued that "interface" is purely a software concept which makes great sense in an architecture but making no sense at the concept level (i.e. in a domain model). Again this can be due to the fact that they (IoT-A) tried to fit somehow a system architecture into a domain model;
2. EPCglobal does not emphasise the need for Augmented Entities. They are therefore not part of the model;
3. Difficulties to model that a User can be responsible for managing a Tag (therefore End-user has not been included in the model);
4. There is a need for introducing end-users formally in the model with roles. It must be possible to express the fact that end-users with management role can associate information to a tag for instance.
5. It should be possible to express the fact that User can discover Services, that Services can discover Resources, that Resources can discover Resources (to be discussed which combinations make most sense);
6. Some links between IoT Domain Model and IoT Information Model should be explicitly described within the IoT Domain Model, like "*-description publishing"
7. Discovery and publishing are important concepts in IoT they should be very visible in the IoT Domain model as said in 7/ and 8/
8. We don't show here the reverse mapping to the IoT Information model, but it was pretty clear that the IoT Information Model is a meta-model that cannot really be used to model the class structure of the EPCglobal data handled at the different levels in the architecture (e.g. at Tag level, reader level, Filtering & Collection etc.. . .), in particular the IoT Information Model does not consider events (and EPCglobal is intensively using the notion of event). We reckon that most likely this is not the role of the IoT Information Model to model in a fine-grained way the class structure of a software system, especially when the class structure is clearly not IoT-specific.

12.3.2 Mapping to Information Model

As far as information is concerned, the main input in the EPCglobal reference architecture is the description of the EPC Information Service and the description of data the end user can share through the EPCIS interface.

EPCIS data within a so-called EPCIS record can be divided in several categories as follows[2] (see also Table 12.4):

---

[2] Excerpt from the EPCglobal Architecture document.

Table 12.4 Mapping of the EPCglobal information model to the IoT Information Model

EPCglobal concept	IoT ref. model concept	Comments
RFID tag	Device/Tag	Virtual entity representing RFID tag associated with the Physical Entity
EPC	Virtual entity	Electronic product code. It is encoded on the RFID tag
EPCIS event	Value	Might be just a wrapping of IPCIS data in the form of an event...
EPCIS data	Value	Is the data associated with the Physical Object and therefore contained in the EPCIS Virtual Entity
EPC record	Virtual entity	Consists of all info related to Physical Object identified by EPC (stored in EPCIS Data Base), i.e. IPCIS Data
EPCIS static data	Value	Contains class level Data and Instance level Data
EPCIS transactional data	Value	Relates to observations (instances, quantity within a class)

- **Static Data**: class level and instance level data, which do not change over time during the physical object life span

    - **Class Level Data**: there remain identical to any object which is an instance of that class
    - **Instance-Level Data**: the data may vary within objects instance of a class. Typical examples are lot number, expiry date, number within a lot, S/N etc...

- **Transactional Data**: which changes and grows over the physical object life span, possibly created by more than one actor along a supply chain for instance:

    - **Instance observation**: it records events concerning the Physical Objects and often relates to dimensions like time, location, other EPC, and business process steps
    - **Quantity observation**: records events concerned with measuring the quantity of objects within a particular class. Five dimensions: time, location, object class, quantity, business step.
    - **Business Transaction Observation**: records association between one or more EPC and a business transaction. Four dimensions time, one or more EPCs, business process step, business transaction id.

12.3.3 Security Model

As explained in the EPCglobal Architecture Framework document [EPCglobal], the EPCglobal Architecture Framework allows for many different authentication technologies across the different interfaces. It is however recommended in the EPCglobal architecture document, that the X.509 certificate-based method should

be used by end-users when accessing the EPCIS interface for example. Typical case occurs when the EPCIS Accessing Application of an accessing end-user (referred as Partner user in the architecture framework) is willing to access the EPCIS service of the primary end-user (the one owning the EPCIS data for instance). If used the X.509 certificates are expected to comply with the X.509 Certificate Profile [Cert1.0] which provide minimum level of security.

At the network level some network standards within EPCglobal rely on *Transport Layer Security* (TSL), some others EPCglobal standards rely on HTTPS (HTTP over TLS) for the purpose of Data protection.

At higher level both EPCIS Capturing I/f and EPCIS Query i/f standards are allowing authentication of client's identity so that companies (owners of the data) can decide very precisely whether access to that data can be granted or denied. For the query interface, Applicability Statement 2 (AS2) is used for communication with external partners. This RFC (4130) specifies how to securely transport data over Internet and allows in particular for mutual authentication, data confidentiality and integrity and non-repudiation. Those security qualities are required in the ARM. AS2 uses x.509 certificate as defined above.

The high level interface (AuthX) used for Authentication in the ARM Security Model does authorise for the use of X.509 certificates.

12.4   Reverse Mapping Ucode

The *Ubiquitous ID* (uID) architecture is an architecture proposed by Prof. Sakamura (from the University of Tokyo) (Koshizuka, Sakamura 2010) to implement the concept of Ubiquitous Computing (ubicomp). Ubiquitous computing is a paradigm coined initially by Mark Weiser in the late 80s (see Weiser 1991). It touches many aspects of computing, like OS's, displays, intelligent user interfaces, wireless communication and networking. In the vision of ubicomp, the computer as we use to know it today, has mostly (if not totally) disappeared. It has become invisible and ambient. While IoT as such is not ubicomp (for instance intelligent user interfaces are not clearly part of IoT field) it can be argued that IoT offers means for implementing partly the ubicomp concept, spreading intelligence among objects of extremely different natures, enabling for cooperation between objects and humans and creating awareness about the surrounding (Context awareness) in a fully connected environment.

The intelligent features or Services implemented through this paradigm can be enabled only if information about the objects, places, Devices, etc. is available to those Services. We therefore talk about "intelligent" "smart" or more specifically "context-aware" Services. This only works if those objects, places, Devices of interest can be uniquely identified at any point in time. The uID architecture relies on an identification technique called ucode (ubiquitous code) which can be considered as the cornerstone of the uID architecture. The ucode model is a descriptive technique that establishes relationships between Physical and Virtual Entities through relationships between ucodes.

Fig. 12.3 Ubiquitous ID architecture
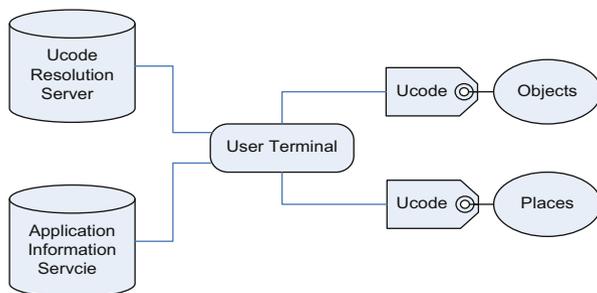
The basic principles of the uID architecture consist of uniquely identifying entities of interest with ucodes, maintaining databases that contain information about the entities, ensuring data and privacy protection and opening this platform through open APIs.

In order to enable those principles fundamental technologies and mechanisms such as ucode structure, ucode tag, ucode readers and terminals, ucode relational databases managing the entities information and ucode information servers are used. These different components are detailed in the following subsections. The simplified architecture shown in Fig. 12.3 is taken from (Koshizuka, Sakamura 2010).

12.4.1 uCode Model

In the ucode model [UID Architecture], unique identifiers are assigned to:

- **Objects**: tangible objects of the real world (industrial product, piece of art, everyday objects,..) as well as intangible ones like pieces of digital media or source code;
- **Spaces**: monuments, streets, etc.
- **Concepts**: relationships between objects and spaces of the real world, which are also named "entities". Those relationships are used to define complex context information, and are defined using a description framework called ucode Relation (ucR) model. Simple context information relates to objects and places directly.

It is worth noting that the uniquely assigned code does not contain any information about the entity. Relevant information about the tagged entity is stored in an application Information Service which can be located by resolving the ucode. A distinction is made between *physical* ucode which are by definition stored in a Tag attached to the entity, and logical ucode which are not stored in any Tag and are mainly used for identifying intangible objects as described above (including relationships between ucodes).

The main idea behind allocating ucode to relationships between entities comes from the Resource Description Format used to model knowledge. RDF knowledge

is made of triple <**subject**, **relation**, **object**> where each constituent of the triple is made of a URI. In the case of ucode relationship each ucR unit is a triple of ucodes. Information associated with the two entities and the relation can therefore be found querying the ucode resolution server.

In addition resulting from this establishment of relations between entities, are graphs (ucR graph) where single "subject" ucode gets linked to many "object" ucode via various "relations". Objects which are not ucodes are called "atoms". A subject ucode pointing via a relation towards a URL is a typical example of such rules involving atoms.

12.4.2   ucode Resolution Server

The resolution of ucode is achieved in the ucode Resolution Server The simplified resolution consists of taking the ucode read by the reader, searching for ucR units that correspond to that ucode and returning to the mobile terminal the addresses of content associated with the ucode via the relational database introduced earlier (similar to a triple store).

The Ubiquitous ID architecture can be simply described as follows (Fig. 12.3):

From the descriptions of the various entities of the architecture (see Fig. 12.3) above, the following table of concepts could be derived (Table 12.5):

In turn, the reverse mapping produced the following UML (see Fig. 12.4) below:

The uID architecture uses the uCR to describe complex context information via relationships between real-world entities (Koshizuka, Sakamura 2010). So-called uCR units consist of a triple of ucodes: subject ucode, relation ucode, and object ucode. The object ucode can be replaced by simple literals, hence it becomes possible to express attributes of a real-world entity as a uCR unit, e.g., <ucode X, "hasBrandName", "GoldenTea">.

It is not feasible to try to map the uCR model directly to IoT Information Model. The IoT Information Model provides a vocabulary for describing IoT systems and it does not, explicitly prescribe how information should be represented. The uCR, on the other hand, can be used to represent relations between any kinds of objects identified with ucodes much in the same way as RDF is used to represent resources identified with URIs. Therefore, the relation between IoT-A information model and the uCR model is actually complementary by nature and the uCR should be seen as a an alternative way (for XML, RDF, binary etc.) to represent IoT Information Model concepts.

12.4.3   Conclusion

To conclude, when mapped to the generic IoT-A the uID provides implementations for only a small subset of the functionalities defined in IoT ARM. First, the ucode provides a globally unique way identify physical (and virtual) objects. These

Table 12.5 Mapping of uID concepts to the IoT reference model

uID concept	IoT reference model concept	Comments
Tangible object	Physical entity	
Intangible object	Digital artefact	If the intangible object is a representation of a tangible object, then it is also a Virtual Entity
Location	Physical entity	Location is not modelled explicitly in the IoT Domain Model. However, a specific (possibly tagged) place can be regarded as a Physical Entity
uCR model	Relates to information model	uCR can be used for representing IoT-A Information Model instances
ucode	No direct relation	The ucode can be used as an globally unique identifier for any instance of the IoT-A RM concept
ucode resolution gateway	Network-based resource	Provides a ucode resolution over HTTP
ucode signature server	Network-based resource	Prevents ucode counterfeiting by verifying and generating signatures
ucode management server	Network-based resource	Manages the allocated ucode space
ucode issue gateway	Network-based resource	Provides a HTTP interface for obtaining ucode issued by ucode management server
ucode entry update gateway	Network-based resource	Provides a HTTP interface for updating ucode resolution entry
Top level domain server	Network-based resource	Hierarchical component of the ucode resolution server
Second level domain server	Network-based resource	Hierarchical component of the ucode resolution server
ucode resolution server	Network-based resource / Service	The resource would be exposed through a resolution service
Application information service	Service	Provides infrastructure and application services
ucode tag	Tag	
User terminal	(Device)	Is a device that reads ucodes and provides services based on the ucode to a user. A user terminal that is just used to run an application or display some information is not in the scope of the IoT Domain Model. However, a user terminal containing a reader is in the terms of the IoT Domain Model a Device with an embedded Sensor Device
Reader	Device/Sensor	

ucodes can be used as identifiers for any instance of the IoT ARM concept. Second, the uID provides a way to resolve the address of the information service hosting data about the object identified with a ucode. This functionality is basically a subset of the functionality defined for the IoT-A resolution infrastructure. Third, the uID provides methods (i.e. the ucode Relational Model) for representing relations
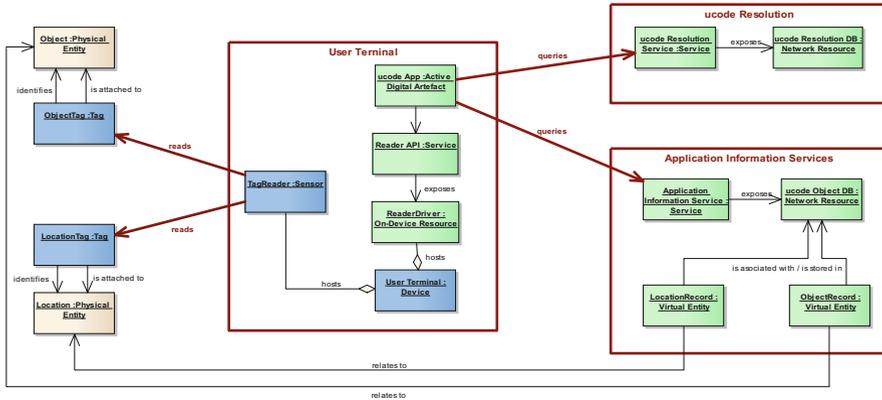
Fig. 12.4 uID architecture fit into the IoT Domain Model

between ucodes. This functionality can be used for representing IoT Information Model concepts.

12.5 Reverse Mapping BUTLER Information Model

12.5.1 Introduction

BUTLER's mission is to provide context-aware services within an IoT environment[3]. What is really striking about that is that there does not seem to be an explicit and widely accepted definition of what context-awareness really means. Although an intuitive definition of what context means can be found easily ("the conditions and circumstances that are relevant to an event, fact, etc.") (Dey 2001), a more formal definition is needed. If we look at other projects within the FP7 umbrella, we can see what FI-WARE[4] provides – a data/context management section. Although not straightforward, it states that "Context [..] is represented through context elements" and that these elements "are typically created containing the value of attributes characterising a given entity at a given moment". Therefore, we can state that the context "characterises a given entity at a given moment". That definition gives rise to a discussion of what an entity is and the extent to which characterisation must take place.

---

[3] http://www.iot-butler.eu/

[4] FI-WARE (http://www.fi-ware.eu/) is a European FP7 Research Project aiming to foster the emerging Future Internet by creating an open architecture and a reference implementation of a novel service infrastructure, building upon generic and reusable building blocks developed in earlier research projects.

12.5.2 *Reverse Mapping of IoT Domain Model*

With regard to the first issue, the IoT-A Domain Model (IoT-A DM) fits perfectly within such a definition, as it introduces the concept of Virtual Entity. These Virtual Entities are the main concept handled by the IoT-A model since they represent the entities in the real world that designers of IoT applications consider relevant. The remaining main concepts introduced by the IoT-A DM (Resources and Services) are naturally associated to the Virtual Entities. A straightforward conclusion is that context should also be "associated" to Virtual Entities. However, the IoT-A DM does not explicitly consider the context. This is irrelevant; we will return to this issue later.

The second issue (what the context means for a given entity) is actually related to the deployment and implementation of a given IoT scenario and not to the definition of the model. However, it is worth mentioning that the components of the context (the "context elements" mentioned by FI-WARE) are totally dependent on the needs and requirements of the consumers of the functionality exposed by entities (again, in a specific scenario). In BUTLER, we introduce a model in which, given a Virtual Entity, it is possible for the consumer of the functionality (usually an application developer) to define at any time the relevant context for this entity. This type of context declaration operation defines the context elements and the data sources these context elements will depend on. It is also important to acknowledge that a given entity context relies not only on the information that devices can gather about it but also (and sometimes mainly) on dynamic data sources that are not actually "device-originated".

BUTLER has taken the IoT-A DM as its main inspiration. However, it has been simplified somewhat to increase the readability and clarity of the model. For instance, the Augmented Entity is not considered; Digital Artefacts – and therefore a non-Human User – have been removed as well; Network Resources have been also dismissed and therefore, the BUTLER model contemplates only On-Device Resources. UML has been used to illustrate the model graphically similarly to the way the IoT-A DM does. We will highlight the main differences and additions we have considered.

The relationship between Users, Physical and Virtual Entities is almost identical to the ones suggested by the IoT-A DM. Besides the simplification already mentioned, it is worth noting that the BUTLER Domain Model introduces additional relationships that are not expressed by the IoT-A DM. For instance, there can be additional relationships between *Users* and *Physical Entities*. The most obvious is the "ownership" (or at least entitlement to the management of the Physical Entity). The relevance of this relationship relies on the access permissions it derives (that is, the owner of a house will have the "right" to get information about his home and to adjust the desired temperature, while a stranger will not, at least not until the owner gives him the right to). On the other hand, the BUTLER Domain Model introduces the BUTLER terminology and therefore, instead of talking about Devices, we
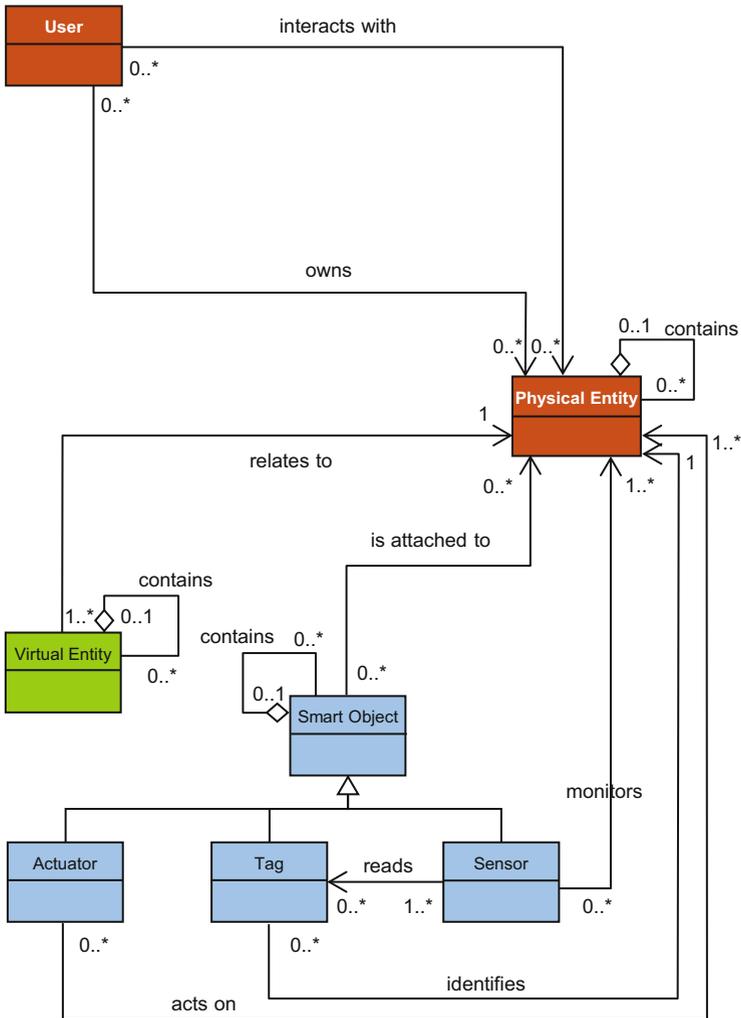
Fig. 12.5 Relationships between users, physical and virtual entities

introduce the Smart Object concept, which is equivalent to the Device concept within the IoT-A DM (Fig. 12.5).

As with the IoT-A DM, Resources are introduced to bridge the gap between the *Virtual Entities* and the *Smart Objects*, enabling the monitoring and manipulation of *Physical Entities* from the digital world. *Resources* are the software components that actually provide information about, or enable the actuation on *Physical Entities*. BUTLER simplifies the management of Resources, focusing on *On-Device Resources* (those deployed locally on the *Smart Object* attached to the *Physical Entity*; these types of Resources are typically sensor *Resources* that

Fig. 12.6 Introduction of the resource concept and its relationship to devices and physical entities

provide sensing data, or actuator *Resources*). It is possible to model other Resources deployed externally to *Smart Objects* that run somewhere in the network as generic *Network Resources*. These Resources can process data, for example, taking sensor information as input and generating aggregated or higher-level information as output (for instance, a dynamic data source providing dynamic weather forecasts or energy consumption estimates). Also, *Network Resources* can be storage *Resources* storing information coming from *On-Device Resources* and thus provide information about *Physical Entities* (i.e. location and state-tracking information (history), static data, such as product type information, and many other properties). Other external data sources, even Human Users, can also update the information in a storage *Resource* (Fig. 12.6).

The primary relationship between *Physical Entities* and their digital counterparts, the *Virtual Entities* on one hand, and the *Smart Objects* and the

Resources they host on the other, is achieved by means of associations. Therefore, *Users* are enabled to act on or to know about *Physical Entities* by means of the associations between *Virtual Entities* and *Resources*. For each *Virtual Entity* there can be associations with different *Resources* that may provide different functionalities, such as retrieving information or enabling the execution of actuation tasks on the Virtual Entities. When a *User* wishes to acquire information about or to actuate on a given *Physical Entity*, she would perform a discovery process determining which *Resources* associated to the *Virtual Entities* representing the *Physical Entities* enable actuation or data access. Next, the User would pick up the *Resources* that match her requirements and invoke them. However, it is unlikely that the User would directly invoke *Resources*. She would do it instead through a *Service* or application that accesses Resources to perform its business logic.

Finally, it is necessary to acknowledge that both *Smart Objects* and *Users* can be modelled as a *Physical Entity*. The same may happen with *Smart Mobiles* (the client device used by users in the BUTLER terminology) (Fig. 12.7).

Here we can see a main divergence from the IoT-A DM, since that model introduces an explicit relationship between Services and Virtual Entities. Although the nature of the relationship is not explicit in the model, the IoT-A Information Model offers additional information about what such a relationship looks like: the Virtual Entity attributes are used to associate Services to Virtual Entities. We prefer a model in which the context is made explicit in the BUTLER Domain Model (and not disguised as the Virtual Entity attributes).

As described in the initial section of this chapter when describing the FI-WARE data/context, the context elements are associated to the entities the system handles. BUTLER proposes to associate *Contexts* to *Virtual Entities*. Therefore, it will be possible to handle the context of the *Physical Entities* represented by the *Virtual Entities* the *Context* is associated to. On the other hand, several *Contexts* can be associated to a given *Virtual Entity* just to reflect the fact that different "consumers" will have a different need or view of the context associated to a given entity (Fig. 12.8).

On the other hand, the attributes of *Contexts* in BUTLER will be mostly created from data obtained from *Resources*. Each attribute will be the result of an operation executed over data elements from one or several Resources. Such Resources may or may not be associated to the *Virtual Entity* the *Context* is associated to (Fig. 12.9).[5]

Finally, Services will be entitled to use not only low-level *Resources* when they need to know about the status of *Physical Entities*, but also richer *Contexts* (Fig. 12.10).

---

[5] For instance, the context associated to a house can include the outdoor temperature. This temperature value can be exposed through a Resource associated to the Weather Service, which in turn has also been modeled as a Virtual Entity. Although the Resource exposing the temperature is not associated to the Virtual Entity representing the given house, an element of its context relies on this "external" Resource.

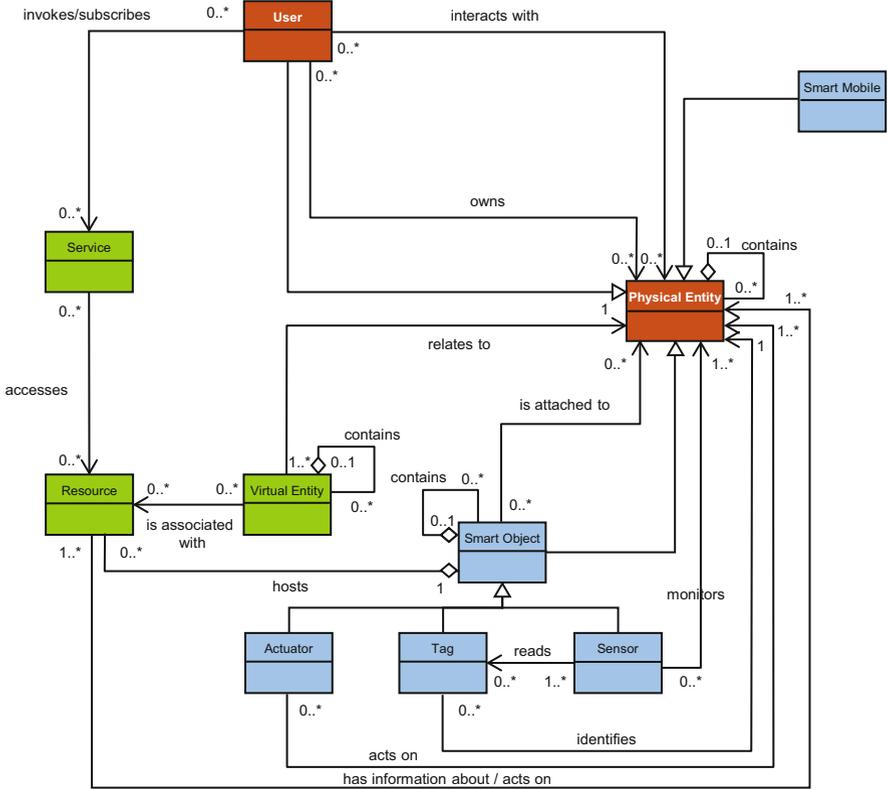**Fig. 12.7** Users to service/resource relationship

12.6 Reverse Mapping MUNICH Platform

The goal of reverse mapping an existing system towards the IoT Reference Model is to show that an existing system that has been designed without applying the IoT ARM can be redesigned according to the IoT ARM. By doing so the IoT ARM shows its potential for being a reference model for any kind of IoT systems.

12.6.1 Use Case Description

The use case is about counting "stomach towels" which are used inside the abdomen during surgery of a human. After the operation it needs to be assured that no towels are retained in the abdominal cavity of the patient's body. Therefore, each towel is fitted with a 13.56 MHz RFID tag which enables tracing the towels
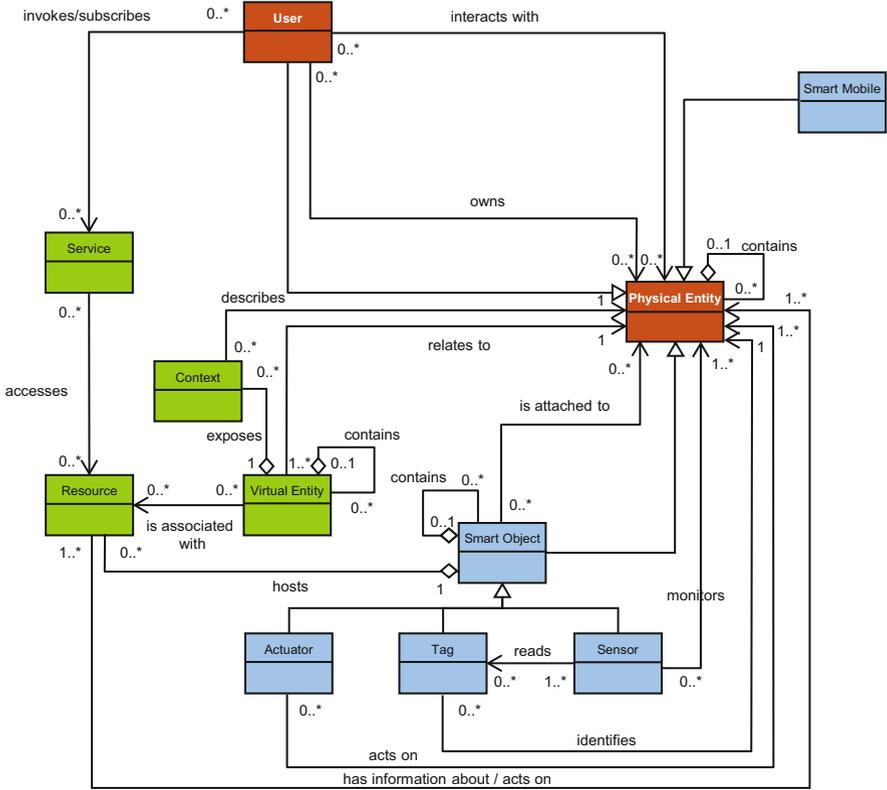
Fig. 12.8 Context to virtual entity relationship

before, during, and after the surgery. The RFID-tagged towels may be tracked by three antennas from different positions in the operating theatre:

- Mayo stand (instrument table): towel is unused;
- Operation table: towel is in use;
- Used towel container: towel is used

Each towel will be used in a specific order: First a batch of "unused" stomach towels resides on the instrument table. Towels that are put into the abdominal cavity are declared as "in use". Finally, towels that are not needed anymore after the surgery are put into the towel container where their status is set to "used".

Every time an RFID reader recognises a tagged towel appearing or disappearing in its range an event is generated and stored in an event-log database hosted in the cloud.
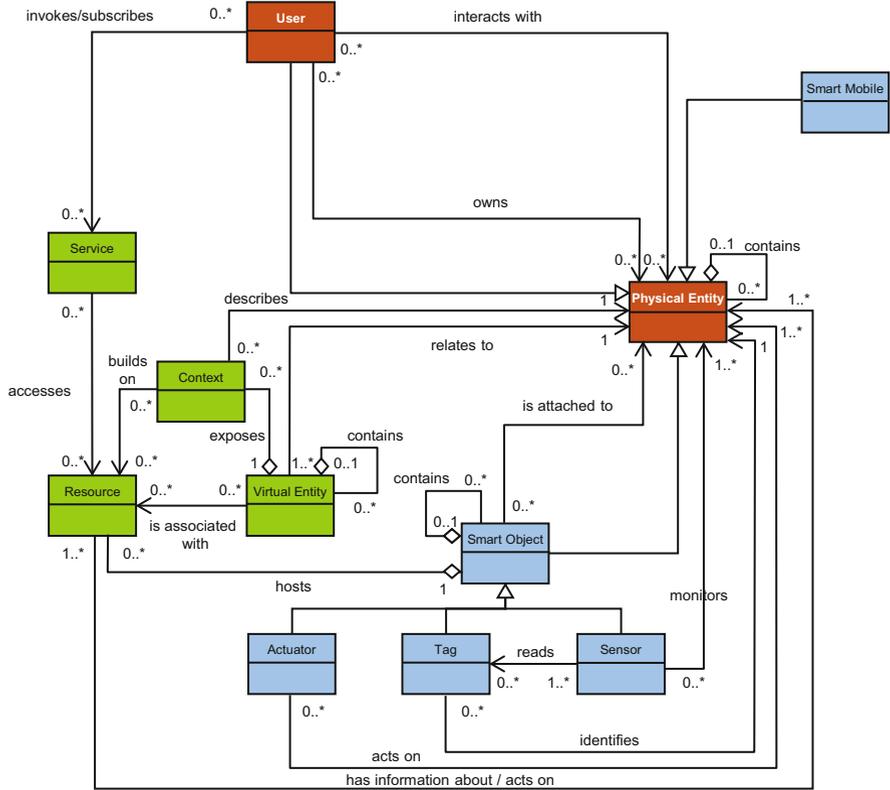
Fig. 12.9 Context to resource relationship

12.6.2 Use Case Objective

It must be assured that no towel**s** are left inside the patient's abdomen when the operation has finished. In more technical terms it means that after finishing the operation all the towels that were "in use" must be in state "used" meaning in the used towel container.

12.6.3 Current System Architecture

So far the use case has been designed to run with a certain type of RFID-readers only that are connected via USB-cable to a laptop computer that is hosting the application. The MUNICH-platform (MUNICH 2010) depicted in Fig. 12.11 provides a cloud storage system indicated as 'Open Nebula Core' that stores the events captured every time the 'Object Inventory Service' notice a change in the
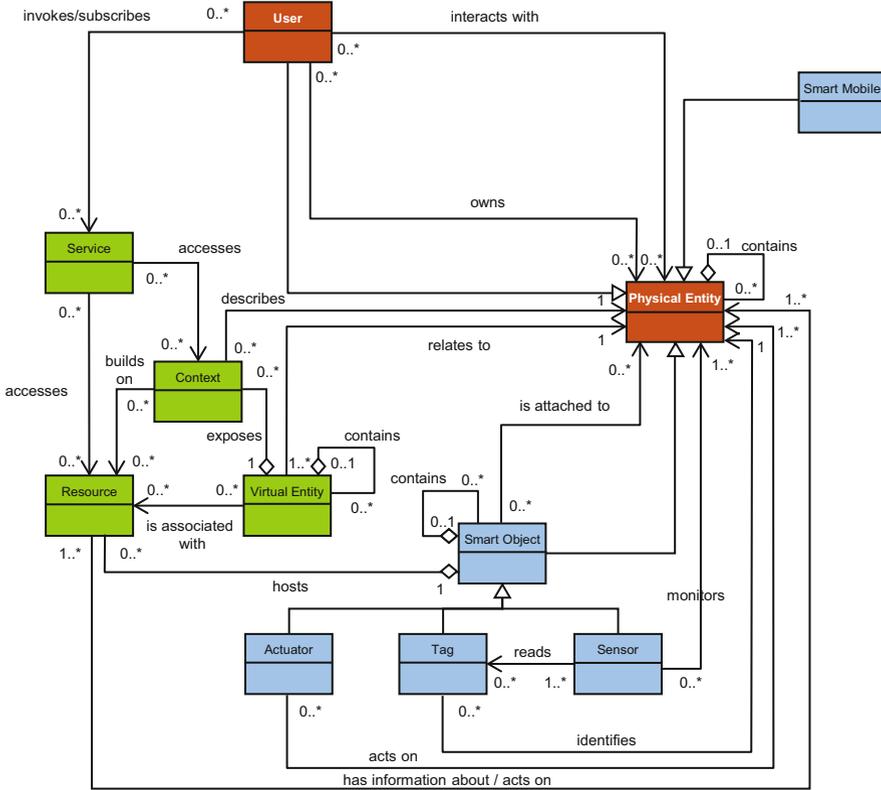
Fig. 12.10 Service to context relationship

number of towels in their respective range by invoking the 'Event Service'. The application that monitors the status of the towels during the operation invokes methods provided by the 'Operation Theatre Service'. The API to store and retrieve information from and to the cloud storage system is technology-specific. If an architect decides at a later point in time to change from Open Nebula to another technology the system needs to be adapted to the changes in the API.

12.6.4 Enhancement by Using IoT Reference Architectural Model

Making the use case demonstrator IoT-A conform means making the system more evolvable and future-proof. By using RFID reader services a technology agnostic layer is introduced that is not so much dependent on today's lifecycle of technologies. With the current solution the software needs to be updated when a
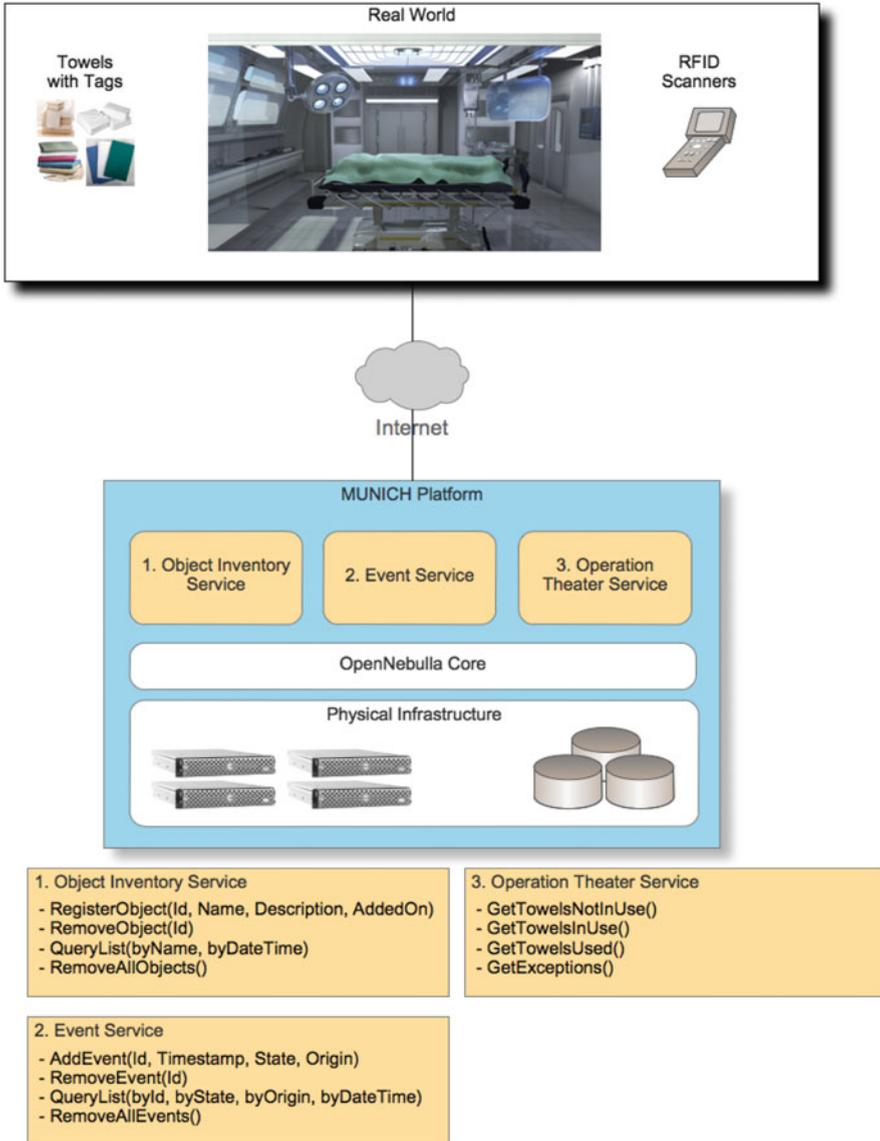
Fig. 12.11 Current architecture of MUNICH platform (MUNICH 2010)

new type of RFID reader needs to replace a current one. Also extending the use case with another RFID reader or another type of sensor will be much easier once IoT-A is applied. Thus the IoT ARM contributes towards scalability in this use case too. The restriction in evolvability applies to the cloud storage component too since the current system is designed to be used with certain cloud storage software. It is not

easy to substitute the component in case the software is discontinued or no longer appropriate. In case the services are modelled according to technology agnostic IoT-A specifications the system will be more future proof. In order to make the use case IoT-A-compliant, the following architectural process will be undertaken.

1. Specification of Business Process Model;
2. Specification of Domain Model;
3. Specification of Information Model;
4. Specification of Functional View;
5. Specification of Services and Interactions between components.

12.6.5   Specification of IoT Business Process Model

The use case has been formalised as IoT Business Process Model by a domain expert in Fig. 12.12. The modelling notation used is described in (Meyer et al. 2011). The operation scenario is a sub-process of the overall Emergency operation process that may include the arrival of the patient via ambulance and the availability of data record for the patient in the hospital's data base. The towels being used during the surgery are associated to the patient identified in the database record. This way it is possible to verify which towels have been used for which patient. The towels are the entities of interest (depicted by the box with the cow icon) in this scenario. The RFID reading processes are running in parallel on all three positions in the operating theatre that are equipped with the RFID readers. The used towel container is denoted as waste bin in Fig. 12.12. Each RFID reader sub-process sends events to the Event History database upon detection of tagged towels. The 'Monitor towel process' analyses the events that have arrived in the database, determines the current state for each towel, and calculates the number of towels that are currently inside the body of the patient.

12.6.6   Specification of IoT Domain Model

Based on the Business Process Model presented before, a domain model can be derived that identifies the Physical and Virtual Entities, the IoT Services, the Devices, Resources, and the users that are involved in the use case. The Human User is the doctor or other medical staff who is responsible to monitor the towels in the operation theatre. The actual monitoring of the towels by comparing the used towels with the ones currently in use is done by software implementing the 'Monitor towel process' as depicted in Fig. 12.12. The User checks only that no towels are still in use when the operation is about to end. The software 'Operation Theatre Application' is modelled as Active Digital Artefact. Each towel is a Physical Entity that has one RFID tag attached so that the number of towels corresponds to the number of tags. Each physical towel has a digital counterpart
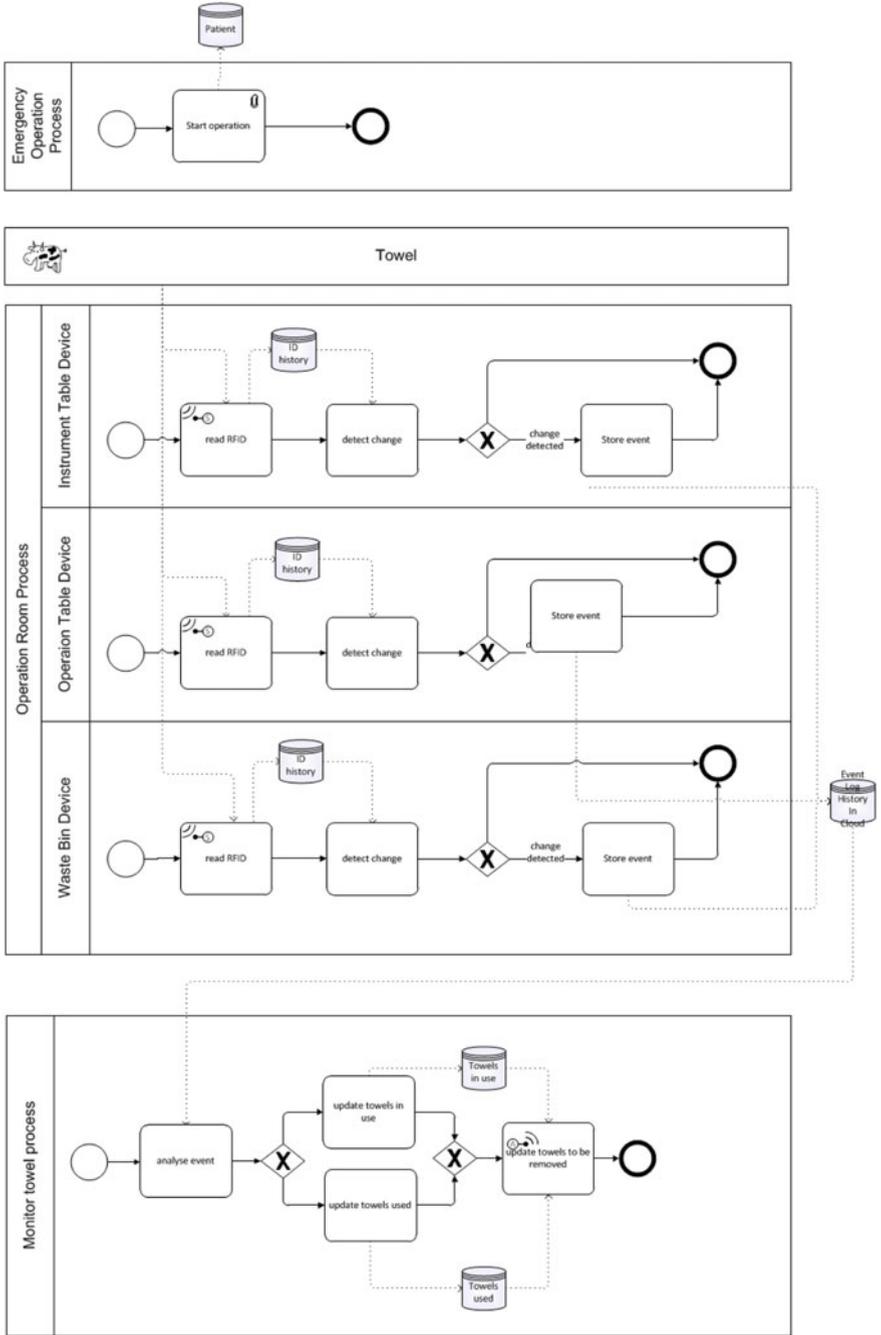
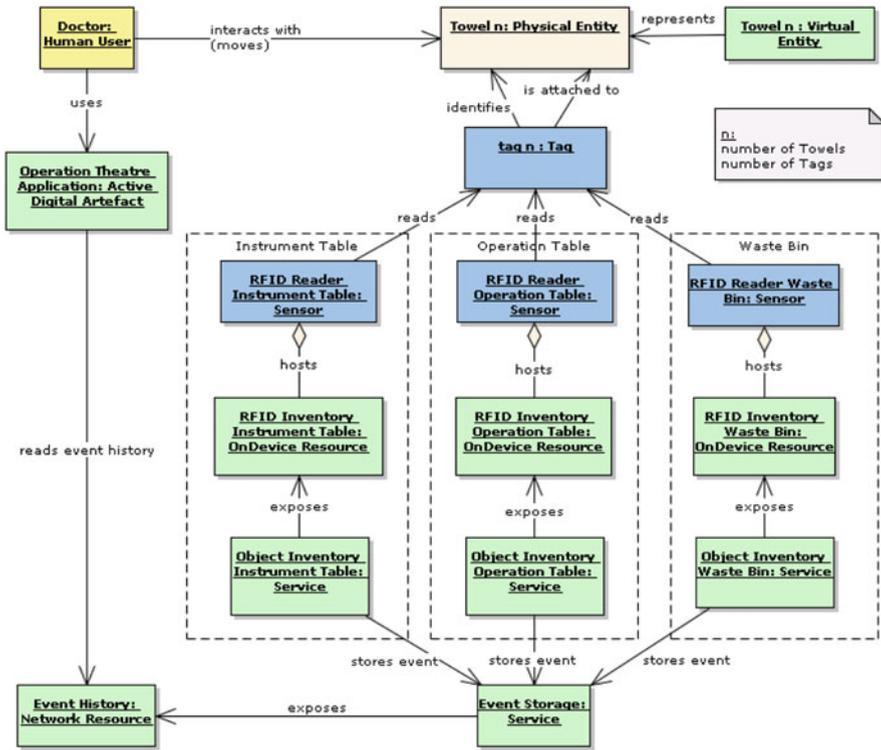**Fig. 12.12** IoT business process model of MUNICH use case

Fig. 12.13 Domain model of MUNICH platform

modelled as Virtual Entity. There are three RFID readers deployed in the scenario at different significant locations of the operation theatre (Instrument Table, Operation Table, and Waste Bin) that are modelled as Sensor Devices. Each of the Sensors hosts an OnDevice Resource that is exposed by an 'Object Inventory Service' as depicted in Fig. 12.11. These services store events by invoking the 'Event Storage Service' that exposes the Network Resource 'Event History'. This Resource is also exposed to the 'Operation Theatre Application' by the Event History (Fig. 12.13).

12.6.7 Specification of Functional View

The realisation of the use case according to the IoT ARM a Functional View is tailored to the use case needs to be specified. The Functional View for the MUNICH platform is depicted in Fig. 12.14. No IoT Service Resolution is required, because all needed services are already known to the system at design time. A VE Resolution FC is included in the FV. This FC is able to resolve particular towels to the IoT

Service they are currently associated with. The 'VE & IoT Service Monitoring' FC is used to update the current state of towels whenever these VEs change their position in the operating theatre. Whenever VEs change their positions their associations between the VEs and the IoT Services reading the RFID tags change too. No Service Organisation functions are required in this use case since the binding of services is static and can therefore be hardwired. To accommodate IoT Business Process Management functionality that is required in the MUNICH platform the respective FG is included in the FV. The process model diagram depicted in Fig. 12.12 was created by the 'Process Modelling' FC and this model is executed by the 'Process Execution' FC. The Functional View of the MUNICH platform includes IoT Services for the RFID readers and for Event Storage Resources. The Application in the FV is the use case as described the beginning of this Section. The Devices are the RFID readers and Tags used in the operational theatre which communicate to the IoT Services by 'End To End Communication' and 'Network Communication' FCs. The entire FV is depicted in Fig. 12.14.

12.6.8   Specification of IoT Information Model

The IoT Information Model specified for this use case also addresses relationships between entities that are not depicted in the IoT Domain Model before. Some more entities appear in the IoT Business Process Model shown before in Fig. 12.12. For instance it is depicted that an 'Operation' is held for a 'Patient' and thus the 'PatientIdentifier' (valid in the clinic) is assigned to an 'Operation'. Operations are processes with a defined status at any point in time: 'before', 'in', and 'after Operation'. There is also an unknown status in case the status cannot be obtained. The towels are represented as VEs with domain attributes that are essential for the use case. The towel's identifier stored into a RFID tag is one of the attributes as well as the current state of a towel that can be one of 'unused', 'in use', and 'used'. Again there is an 'unknown' state specified in case the state cannot be obtained by the system. The aforementioned designated locations of the operating theatre are reflected in the Information Model as attributes of the VE 'Towel'. For simplification the allowed values for this attribute {InstrumentTable; OperationTable; WasteBin; unknown} are not visualised as ValueContainer. With the aforementioned attribute values the OperationTheatreApplication is able to relate the current location of the towels (retrieved through the RFID readers) to the respective state of the towel: {instrument table = 'unused'; operation table = 'in use'; waste bin = 'used'} (Fig. 12.15).
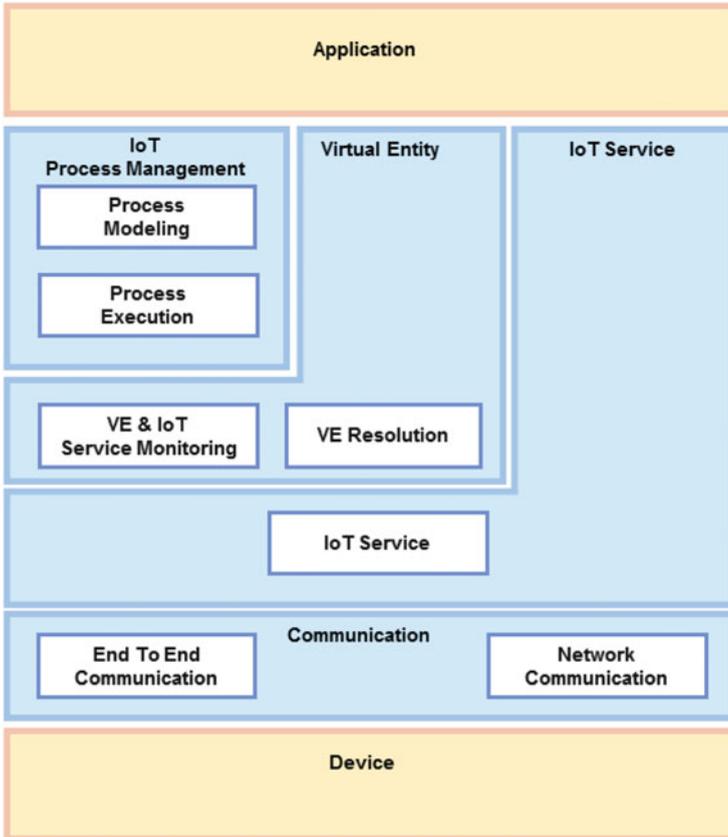
Fig. 12.14 Functional view of the MUNICH platform

12.6.9 Specification of IoT Services and Interactions

In the following an example description is given for one of the three 'Object Inventory Services' specified in the IoT Domain Model before (Fig. 12.16).

The sensing service 'ObjectInventoryServiceOPtable' exposes the Resource RFIDInventoryOperationTable hosted on the Sensor that observes the area OperationTable during the operation. The duration is determined by the Op123Schedule. The output of the service is described in a domain specific operation-ontology by the class ListOfRFID that defines a list of identifiers the RFID reader has detected. The service can be invoked by accessing the service endpoint objInventoryOPtableRestSE that provides a RESTful web service on the endpoint host optablehost. An HTTP GET method call on port 4355 on the root path '/' of this host will return the list of identifiers the RFID sensor has read.
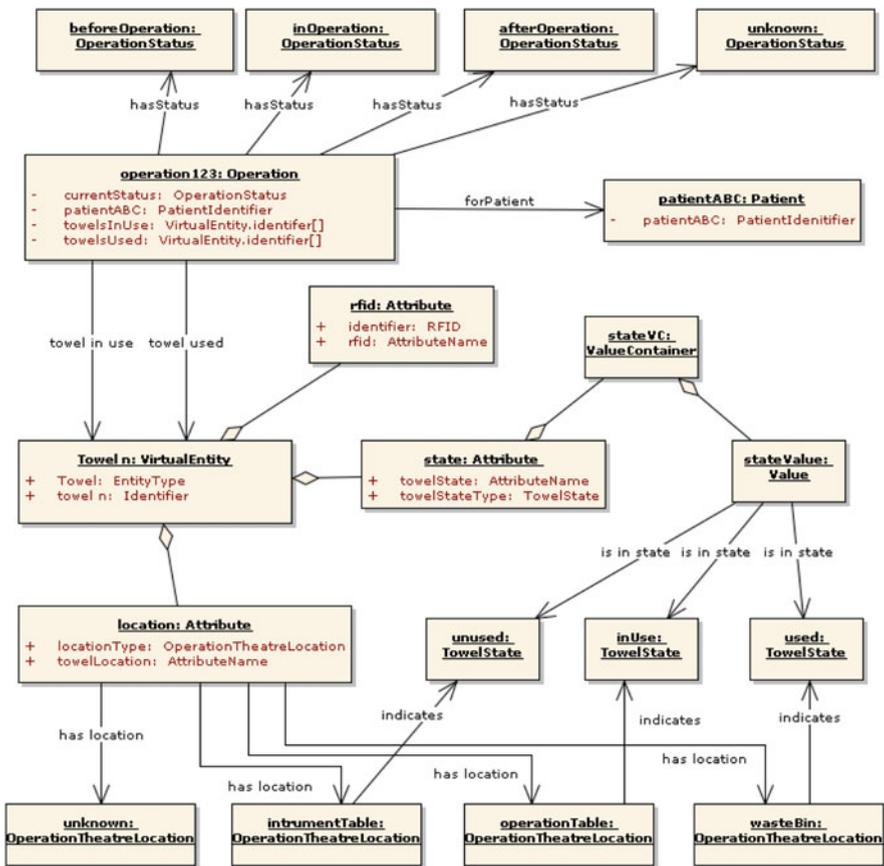
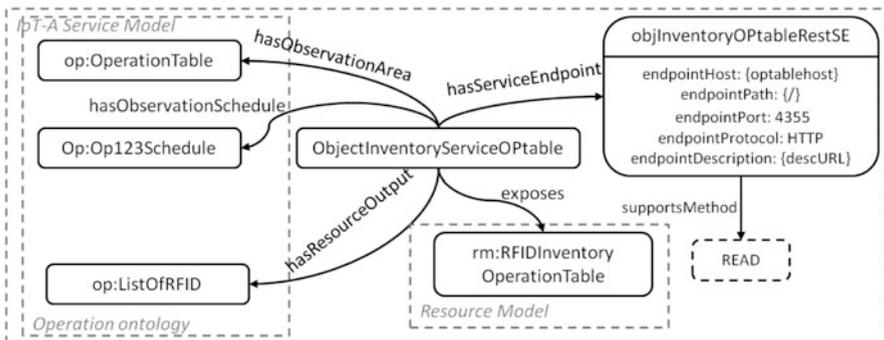**Fig. 12.15** Information model of MUNICH platform

Fig. 12.16 Service description MUNICH platform

The use case is driven by events using asynchronous communication. Events are sent to the Event History network resource every time an RFID reader recognises a change in the number of RFID-tags in its observation area by using IoT Service storeEvent(event). The Event History resource provides another IoT Service that allows the subscription to notifications about the change in the status of towels, e.g. from unused to in use.

The structure of an Event data type is given as follows:

- Origin: {RFID reader instrument table; RFID reader operation table; RFID reader waste bin}
- Type: {RFID tag gone; RFID tag added; unrecognised tag}
- Time stamp

The sequence diagram below illustrates the interactions between Physical Entities and Functional Components of the architecture. The doctor takes a new towel out of the box on the instrument table and uses it in the patient's abdomen located on the operation table. The system detects the move of the towel from the instrument to the operation table by the disappearance of the respective RFID tag that is attached to the towel together with the appearance of the same RFID tag on the operation table. The Event Storage Service evaluates these single events towel disappeared on instrument table and towel appeared on operation table to a complex event towel in use (Fig. 12.17).

12.6.10 MUNICH Platform Conclusion

The previous Sections have shown that an existing system can be reverse engineered by applying the IoT ARM. Beginning from an existing system the modelling of the IoT Domain Model and Information Model has been demonstrated. With the help of these models the respective IoT Service Descriptions have been derived and the interactions between the Resources have been specified. The exercise did not include all the steps of the process to derive a concrete architecture based on the IoT ARM. There was neither a requirements analysis nor a security risk analysis undertaken. The purpose of this exercise is to demonstrate the usage of the models in first place. Since the functionality of the system has not changed a comprehensive requirements analysis has been skipped. Also the security risks are seen as manageable since the operating theatre is a well-secured and closed environment anyways. Only the event related service makes connections to external environments, but that was the case for the original system already and therefore no changes in security risks are expected. Particular platforms and solutions to implement the use case are not recommended here; technologies that would be suggested in this document might be outdated by the time of reading this document and therefore obsolete.
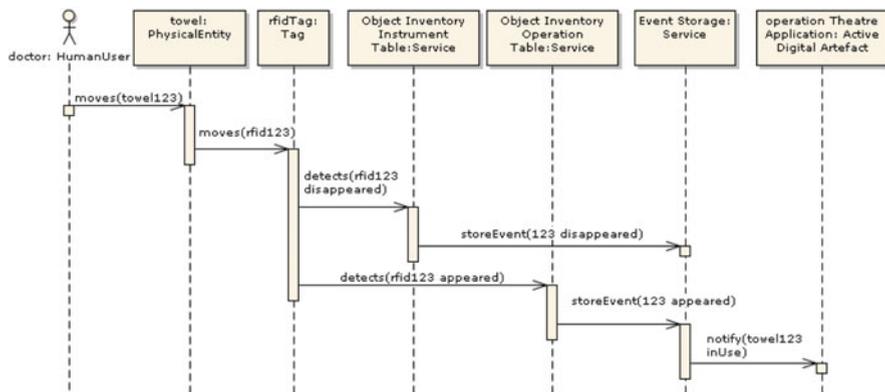
Fig. 12.17 Interactions MUNICH platform

12.7   Conclusions About Reverse Mapping

In this Section we have provided a reverse mapping of the IoT Architectural Reference Model with several standards from the field of IoT as well as a concrete architecture in order to provide an architectural validation, namely whether it is possible to map existing standards to the IoT ARM. If this was not possible, then the validity of the ARM itself would be questionable.

As we have seen in the detailed discussion of the different standards, whether a mapping is possible or not largely depends on the level of detail that we apply to the mapping. Especially for the Domain Model this becomes clear when we pick up the concept of a "Service": All the standards we looked at provide services in one way or the other, so that at a superficial glance a mapping is trivial. However, when we take the exact definition of that term in the different standards, we realize that there is not always a 1:1 correspondence between the standards. For instance, in ETSI M2M a service is not defined as "exposing resources on devices, but can interact with the devices." A resource concept as in IoT-A does not exist, so that compared to the definition of services and resources in the ARM, the distinction between a resource and the service as it is made in IoT-A does not exist in ETSI M2M. From a high-level perspective, though, the Domain Model usually maps rather well to the different standards. Also, the Communication Model and security aspects are rather compatible between the standards and the ARM. The latter is not surprising, as security aspects in the world of IoT are commonly derived from a well-established body of security research with fixed and clear terminology, quite unlike the Internet of Things domain. Also, it must be noted that the scope of IoT-A is broader than the scope of any of the individual standards. This is not surprising, as IoT-A aims to provide a Reference Architecture for all different kinds of specific architectures and use cases, and therefore must be broader by definition. Different parts of the IoT ARM are therefore only partially or not covered at all by different standards. For

instance, EPCglobal is highly RFID centric and therefore neglects certain aspects such as the IoT Communication Model, however the mapping to the IoT Domain Model and also to the Security and Information Model works reasonably well at the appropriate level of abstraction.

While the mapping of the different standards can be regarded as successful, when being performed at the appropriate level of detail, the real litmus test is the mapping of a concrete architecture to the IoT ARM. We have provided such a mapping for the MUNICH platform and have provided detailed information about the Domain Model, the Information Model, a process modelling based on the BPMN extensions developed in IoT-A (Meyer et al. 2013) and have discussed the service modelling in detail. Of course, we cannot generalize this successful exercise to any existing concrete architecture, but it still demonstrates nicely, how the IoT ARM can be applied to a concrete architecture. We are confident that other architectures from the domain of IoT map equally well to the IoT ARM.

12.8 Business Case Evaluation Example

12.8.1 Introduction

In the healthcare use case, to show the real-world value of the ARM, we focus on an IoT system that has already been implemented. In combination with the reverse mapping (see Sect. 12.6), we show that not only can the IoT ARM describe existing IoT systems (and by extension, help realise such systems), but that these systems also bring value. We evaluated the operating efficiency and profitability of such an IoT system.

This use case was implemented and carried out by several companies and universities in the framework for the Initiative for Cloud Computing in Health Care (henceforth referred to as the "MUNICH platform"). The MUNICH platform addresses two main problems: debris left in the human body after surgery and time-consuming process steps with no added value ("non-productive time"). A third auxiliary problem is the on-going integration of software and solutions from third party providers, which the IoT-A ARM would address.

Regarding the debris problem, in spite of safety checks already implemented, debris (tools, towels, consumables) is still left in the body during surgical procedures in 1:10,000 cases (Kranzfelder et al. 2012). In these cases, 70 % of the debris comes from surgical towels and 30 % from other surgical equipment (Kranzfelder et al. 2012). The consequences for the patient are a 40 % morbidity rate with a 5 % mortality rate (Kranzfelder et al. 2012). Regarding non-productive time, this refers to steps such as documenting and registering towels before the operation, subsequent counting of towels during the operation, and searching for towels when something is amiss; none of these steps add value, but instead address a problem created by the process itself.
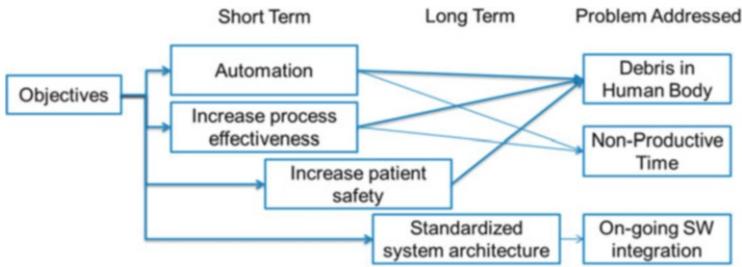
Fig. 12.18 Objectives of the healthcare use case and the problems addressed

Accordingly, a solution that addresses the tracking of surgical towels would mitigate these problems significantly. We can therefore map the MUNICH platform's objectives and solutions as shown in Fig. 12.18.

Real-time monitoring and location of all towels reduces the risk of debris in the human body because manual error-prone counting and searching is avoided (MUWS 2013). Therefore, the automation reduces manual errors. The process improvement increases the transparency of the process and reduces the risk of documentation errors that can also lead to debris in the human body. Experts estimate that a 100 % failure protection is possible with this solution (Kranzfelder et al. 2012). Addressing the debris problem meets short-term objectives of automation and improved process effectiveness, and in the mid-term, increases patient safety.

For the non-productive time problem, automation and the resultant process improvement remove the error-prone steps of documenting and registering towels before the operation, subsequent counting of towels during the operation, and searching for towels when something is amiss.

For the long-term problem of integrating new software developments from the hospital and their third party solution providers, the IoT-A ARM provides a standardised reference architecture. This would simplify the complexity of the architecture and make integration of new components into the system easier.

12.8.2   Cost and Benefit Models

The inputs for our analysis consisted of a cost model and a benefit model. The cost model factored in non-recurring costs (NRC) such as the RFID antenna and readers. The main cost driver is the hardware investment for the RFID antennas, which amounts to €49,500 – 58 % of the total non-recurring cost (€85,600). Beyond this initial investment, the cost model also factored in recurring costs (RC), such as the RFID-tagged towels, the software and system licensing fee, staff training, and the maintenance costs. The main cost driver of the recurring cost group is the operating fees of the system provider. This cost element has the most significant impact on the

cost model and accounts for 98 % of the yearly RC of €1,034,000. A price change in the service fee has a dramatic impact on the total cost structure over time. Therefore, this price change will be part of a specific sensitivity analysis.

The total cost (NRC+RC) development over a 6 year period was subsequently computed and input into a combined cost-benefit model (see 0 Cost-benefit analysis).

The benefit model is composed of three benefits; the calculated yearly benefits are in brackets: RFID-supported surgery (€815,000), cost savings from prevention of surgical errors (€370,000), and RFID-supported surgery preparation (€104,000). The "RFID-supported surgery" model provided the highest benefit, accounting for 63 % of total benefits. Non-tangible benefits not directly linked to a monetary outcome include an increase in surgical scheduling each year due to reduced preparation time, and hospital reputation improvements due to improved safety.

The total benefit over a 6 year period was subsequently computed and input into a combined cost-benefit model (see 0 Cost-benefit analysis).

12.8.3   Cost-Benefit Analysis

Figure 12.19 presents the yearly and cumulative cash flows. The cost-benefit analysis demonstrates a positive investment result. The discount factor is assumed at 8 % and the net present value is €805,000. The payback period is less than 1 year. Within Germany, according to healthcare experts, this would meet the requirement of a 1 year payback period for new investments in a German hospital.

12.8.4   Sensitivity Analysis

With the sensitivity analysis, we can investigate the impact of changing the major calculation variables. The following impacts shown in Table 12.6 will be discussed:

The results of the sensitivities are always evaluated with respect to the final effect on the discounted cumulative cash flow. The sensitivity analysis will be summarized with a best/worst case scenario.

12.8.4.1   Sensitivity Analysis for the Cost Model

The cost model sensitivity analysis investigates the impacts on the cost model if a parameter is changed. Reducing the critical risk factors (CRF) by 10 % leads to an increase in the total cash flow from €805,000 to €1,187,000, which is an increase in the net present value of 47 %. On the other hand, increasing the CRF by +10 % or +20 % due to higher NRC and RC lowers the net present value to €423,000 or €41,000 respectively.
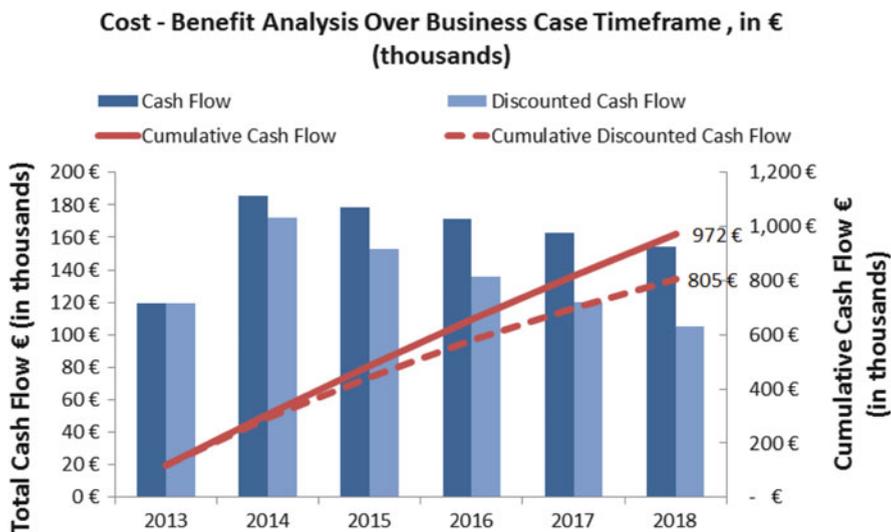
Fig. 12.19 Cost-benefit analysis over the business case timeframe (healthcare case)

Table 12.6 Models and parameters varied in the sensitivity analysis for the healthcare case

	Model element changed		
	Cost model	Benefit model	(c) General calculation assumptions
Change in variables	Critical risk factors: Software risk = SR Hardware risk = HR Personnel risk = PR Maintenance risk = MR System service fee (SFS)	Benefit variation factor (BSF)	Discount Rate (DF) Frequency of surgeries (TAoS)

The main cost driver for recurring costs (RC) is the system service fee. An increase of 10 % in the service fee per surgery from €20 to €22 reduces the net present value by two thirds to €270,000. The profitability limit is reached by increasing the fee to €23/surgery. The cost model sensitivity analysis is depicted in Fig. 12.20.
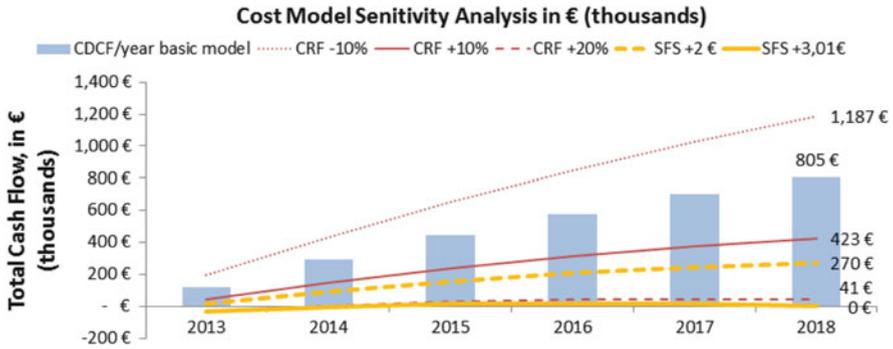
Fig. 12.20 Cost model sensitivity analysis (healthcare case)

12.8.4.2 Sensitivity Analysis Regarding Benefit Model Robustness

This analysis aims to investigate the robustness of the benefit model and the impact on the cost-benefit results. To demonstrate the development of the model, three different scenarios are simulated: (1) Benefits increase by 10 % (2) Benefits decrease by 10 % and (3) Benefits decrease by 15 %. The results of these simulations are summarized in Fig. 12.21. The net present value is exactly 0 when the benefits are reduced by 12.4 %.

Notably, the net present value is very sensitive to changes in the benefit model; there are large benefit differences between (1) and (2).

12.8.4.3 Sensitivity Analysis for the Assumptions in the General Calculation

After the sensitivity analysis of costs and benefits variations, the analysis is extended to variations of the general calculation assumptions that affect both models. Two parameters are used to simulate the results. The first is the change in the discount rate (DF) to reflect different risk perceptions and interest rate influences. The second parameter concerns the frequency of the surgeries per year (TAoS), which is a basic quantity variable (see Fig. 12.22).

A variation in the discount rate of ± 2 % leads to an increase/decrease in the net present value of ± 4 %. If a 12 % discount rate is assumed, the net present value falls to €741,000 (-8 %).

If the hospital performs 25 % fewer surgeries per year, the net present value decreases to €524,000 (-35 %). In contrast, if the number of surgeries per year increases by 25 %, the net present value rises by 35 % (€1,087,000). The net present value is zero if the hospital performs 71.5 % fewer surgeries per year.
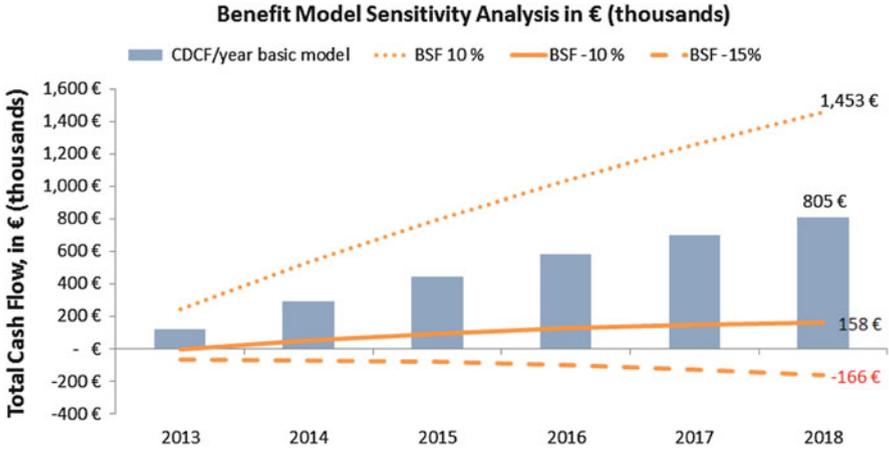
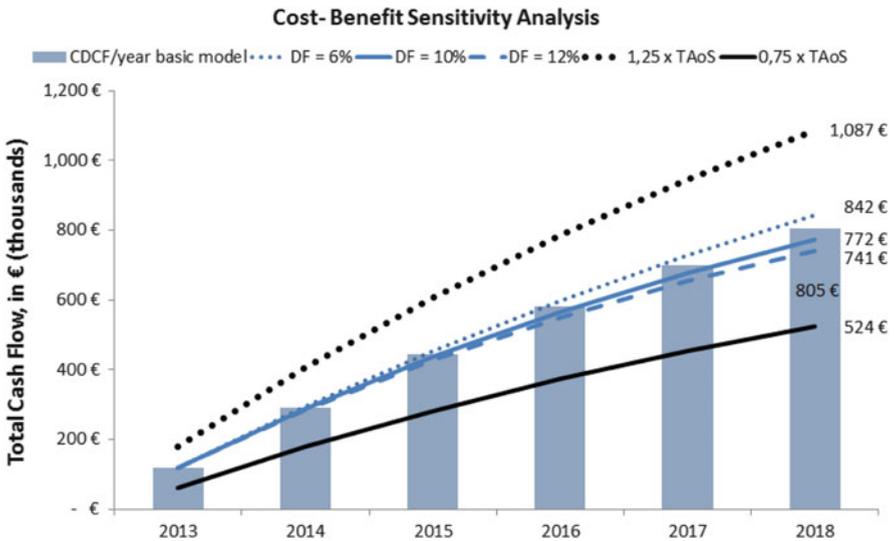**Fig. 12.21** Benefit model sensitivity analysis (healthcare case)

Fig. 12.22 Cost-benefit sensitivity analysis (healthcare case)

12.8.4.4   Best/Worst Case Scenario

By combining cost and benefit variation in the sensitivity analysis, best and worst case scenarios can be elaborated. For example, if the system service cost is reduced by €1/surgery (= −5 %) and the hospital performs 25 % more surgeries annually, then the net present value rises significantly to €1,421,000 (+77 %). The best case scenario is based on the assumption that the service provider can lower the cost of the service fee due to cheaper maintenance costs, additional development support
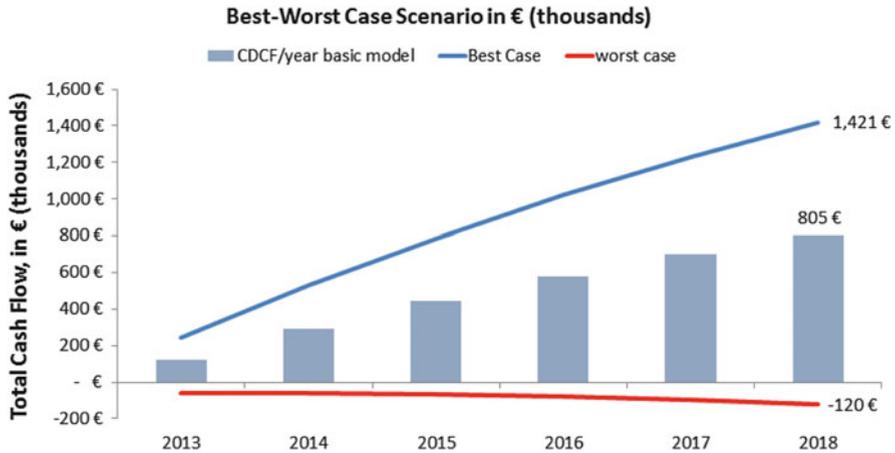
Best-Worst Case Scenario in € (thousands)

Fig. 12.23 Best and worst case scenario (healthcare case)

from using the IoT ARM, and from economies of scale effects. As a result of using the system and thereby reducing the errors, it is assumed that the hospital gains a better reputation and is more efficient, and accordingly, the number of surgeries per year rises.

In a worst case scenario it is assumed that the benefits are lowered by 5 %, the system service fee is €2/surgery more expensive (+10 %), and the number of the surgeries is reduced by 25 %. In this worst case scenario, the net present value is completely destroyed and always negative (see Fig. 12.23).

We observe that the economic feasibility of the case depends to a high degree on the system service fee of the service provider. The feasibility is also sensitive to fluctuations in the benefits. Further investigation about the reliability of the cost estimates is necessary. This information can be gained from the pilot deployments of the system with RFID-equipped towels. A test case is currently running in Munich at the university hospital "Rechts der Isar". When the pilot case is finished, a more reliable assessment of cost and benefits will be possible. The service provider would then also have better information for the calculation of the cost of the service fee.

Chapter 13
Summary and Outlook

Alessandro Bassi

Five years ago, when we organised the very successful meeting "Beyond RFID: the Internet of Things",[1] we were pioneering a new space. At that time, a global network of interconnected objects was just a very fancy and rather fuzzy concept. Today, this topic is clearly mainstream.

Cisco[2] and Ericsson[3] have published white papers clearly showing the relevance and the importance of IoT-related technologies for their strategic offering; to mention just a few numbers, for whatever they are worth, apart from the famous forecast of 50 billion interconnected devices by 2020,[4] Cisco foresees a related market value of $14.4 trillion.[5] These companies are just the tip of the iceberg: McKinsey, for instance, recently published a report[6] estimating the IoT impact on the global GDP as between $2.7 and $6.2 trillion annually by 2025, an impact which is beyond that of big data. In 2012, Gartner Research identified the IoT as one of the top ten technology trends for the years to come.[7]

As members of the IoT-A consortium, we are rather proud to be at the forefront of this wave. Back in 2009, we clearly identified the main technological showstoppers for the development of a global IoT vision. The problems we faced back then were threefold.

As early developments were clearly not coherent, and showed little if any possibility of integration in bigger systems, scalability capabilities or an ability to

---

[1] http://www.smart-systems-integration.org/public/internet-of-things

[2] http://share.cisco.com/internet-of-things.html

[3] http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf

[4] http://share.cisco.com/internet-of-things.html

[5] http://iotevent.eu/cisco-sees-14-trillion-opportunity-in-iot/

[6] http://www.mckinsey.com/insights/business_technology/disruptive_technologies

[7] http://www.gartner.com/newsroom/id/2209615

A. Bassi (✉)
Alessandro Bassi Consulting, 3, Avenue de Cannes, 09160 Juan Les Pins, France
e-mail: alessandro@bassiconsulting.eu

adopt strong security policies, the need was very evident: to define a common IoT ground where services, applications and products could have solid roots. This common ground is established by making different technologies talk to each other, allowing existing and new IoT-related developments to belong to the same space.

Furthermore, it is important to define a way of moving from ground-breaking ideas to real products and services.

Finally, we needed to show not only the research community but the whole community of innovators that IoT technologies can actually be used to implement their projects, providing innovation managers and architects with the necessary tools to do so.

The IoT-A project tackled the first point by upgrading existing technological developments. Communication protocols related to constrained devices were studied and extensions were proposed, tested and promoted in the appropriate standardisation fora.

As far as defining ways of moving from ground-breaking ideas to real products and services is concerned, in many formal and informal meetings we see reactions such as "what the heck?" when we explain what IoT can do for a business domain. There is a clear need for education and information that is missing at the moment. Innovation directors may have a very clear vision of what they want to achieve, but there is no way for them to understand the complexity of their challenges, to select the best architectural design patterns that can solve their issues, and to decide which technologies to use to implement a solution in practice. We see this as a vertical challenge: from a vision to a product, designing the right set of models, architectures and tools. This point was addressed by the development of the Architectural Reference Model (ARM), which includes all necessary models and design patterns for developing a real product.

Finally, after tackling both of these dimensions, as already stated, we needed to show not only the research community but the whole community of innovators that IoT technologies can actually be used to implement their projects, giving innovation managers and architects the necessary tools to do so. A silver bullet in a drawer does not solve any issues; we needed to "go out" and reach the widest possible range of most diversified audiences in order to make our work worthwhile. This book clearly addresses this third aspect of communication.

We are also aware, however, that all dimensions need further work. From the very beginning, we intended the ARM to be an iterative effort. The set of models and architectural choices will evolve, and the project partners are seeking suitable ways of making the concepts long-lasting, well beyond the project's lifetime. What is important is that the ARM and all related developments cannot be "locked" in the sole ownership of one single organisation or group – every instrument that can promote and develop the architectural development of the IoT further must be able to use the ARM work done in IoT-A as a base.

The horizontal integration between different technologies will also require updates. As different IoT-related technologies evolve, there will be a need to develop different interfaces at any level, from device level to services. Further

investigation and efforts will be required: in particular, considering possibly revo-lutionary developments such as quantum technologies. Within IoT-A, we tried to provide guidelines for developments in some areas, such as protocols; however, as long-term forecasts are very often off-target, only time will tell exactly which areas will need closer attention.

Last, but certainly not least, a considerable amount of work must be done to develop sustainable security and privacy policies. Even before the IoT, RFID technologies were subject to a very negative "big brother" image. These considerations are very topical today, with the disclosure of the US government's PRISM program, and any technology for interconnected objects may be rejected on the basis that it violates basic privacy principles. Therefore, governance schemes that on one hand are privacy-friendly, and on the other hand secure, must be agreed upon and implemented, along with widespread education on the societal benefits of IoT.

Appendix A

Terminology

This appendix aims at defining the terminology introduced in this book. Please, always refer to the online version of the IoT-A terminology webpage at http://www. iot-a.eu/public/terminology. Note also, words written in *italic* in the Definition column own an entry in the table providing their specific definition in IoT context.

Term	Definition	Source
Active Digital Artefact	*Active Digital Artefacts* are running software applications, agents or *Services* that may access other *Services* or *Resources*.	Internal
Active Digital Entity	Any type of active code or software program, usually acting according to a *Business Logic*. Obsolete: the term to be used is *Active Digital Artefact*	Internal
Actuator	Special *Device* that executes a change in the physical state of one or more *Physical Entities*.	Internal
Address	An address is used for locating and accessing – "talking to" – a *Device*, a *Resource*, or a *Service*. In some cases, the ID and the Address can be the same, but conceptually they are different.	Internal
Application Software	"Software that provides an application *service* to the *user*. It is specific to an application in the multimedia and/or hypermedia domain and is composed of programs and data".	ETSI Technical report ETR 173, Terminal Equipment (TE); Functional model for multimedia applications. Available online: http://www.etsi.org/deliver/etsi_etr/100_199/173/01_60/etr_88173e01p.pdf
Architectural Reference Model	The IoT-A architectural reference model follows the definition of the IoT reference model and combines it with the related IoT reference architecture. Furthermore, it describes the methodology with which the reference model and the reference architecture are derived, including the use of internal and external stakeholder requirements.	Internal
Architecture	"The fundamental organization of a *system* embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution".	IEEE (1471–2000) IEEE recommended practice for architectural description of software-intensive systems
Architecture Vision	"A high-level, aspirational *view* of the target *architecture*."	The Open Group (2008) TOGAF version 9 – a manual, 9th ed., First Impression. s.l. Van Haren Publishing, Zaltbommel

Aspiration	"*Stakeholder* Aspirations are statements that express the expectations and desires of the various *stakeholders* for the *services* that the final [*system*] implementation will provide."	E-FRAME project, available online at: http://www.frame-online.net/top-menu/the-architecture-2/faqs/stake holder-aspiration.html
Association	An association establishes the relation between a *service* and *resource* on the one hand and a *Physical Entity* on the other hand.	Internal
Augmented Entity	The composition of a *Physical Entity* and its associated *Virtual Entity*.	Internal
AutoID and Mobility Technologies	"Automatic Identification and Mobility (AIM) technologies are a diverse family of technologies that share the common purpose of identifying, tracking, recording, storing and communicating essential business, personal, or product data. In most cases, AIM technologies serve as the front end of enterprise software *systems*, providing fast and accurate collection and entry of data. AIM technologies include a wide range of solutions, each with different data capacities, form factors, capabilities, and "best practice" uses. AIM technologies also include mobile computing devices that facilitate the collection, manipulation, or communication of data from data carriers as well as through operator entry of data via voice, touch screens or keypads. Each member of the AIM technology family has its own specific benefits and limitations – meaning there is no "best" technology. Rather, applications may be best served by one or more AIM technologies. Multiple AIM technologies are often used in combination to provide enterprise-wide solutions to business issues.	Association for Automatic identification and mobility, online at: http://www.aimglobal.org/

Term	Definition	Source
	Most AIM technologies are defined by international and national technical standards. International, national or industry application standards also exist to define the use of AIM technologies."	
Business Logic	Goal or behaviour of a system involving *Things*. *Business logic* serves a particular business purpose. *Business logic* can also define the behaviour of a single or multiple Physical Entities, or a complete business process.	Internal
Communication Model	The communication model aims at defining the main communication paradigms for connecting elements, as, in the IoT-A case, defined in the domain model. This model provides a set of communication rules to build interoperable stacks, together with insights about the main interactions among the elements of the domain model.	Internal
Concrete Architecture	A concrete architecture is an instantiation of the ARM applied to a practical use case.	Internal
Controller	Anything that has the capability to affect a *Physical Entity*, like changing its state or moving it.	Internal
Device	Technical physical component (hardware) with communication capabilities to other ITC systems. A *device* can be either attached to or embedded inside a *Physical Entity*, or monitor a *Physical Entity* in its vicinity.	Internal
Digital Artefact	*Virtual Entities* are *Digital Artefacts* that can be classified as either active or passive.	Internal
Digital Entity	Any computational or data element of an ITC-based system. Obsolete: the new term to be used is *Digital Artefact*.	Internal

Discovery	Discovery is a *service* to find unknown *resources/services* based on a rough specification of the desired result. It may be utilized by a human or another *service*. Credentials for authorization are considered when executing the discovery.	Internal
Domain Model	"A domain model describes objects belonging to a particular area of interest. The domain model also defines attributes of those objects, such as name and identifier. The domain model defines relationships between objects such as "instruments produce data sets". Besides describing a domain, domain models also help to facilitate correlative use and exchange of data between domains".	Consultative Committee (2006) The consultative committee for space data systems. Information 50 Architecture reference model. CCSDS_312.0-G-0. http://cwe.ccsds. org/sea/docs/SEA-IA/Draft%20Documents/IA%20Reference%20Model/ccsds_rasim_20060308.pdf. Accessed 30 Apr 2013
Energy-harvesting Technologies	"*Energy-harvesting* (also known as power harvesting or energy scavenging) is the process by which energy is derived from external sources (e.g., solar power, thermal energy, wind energy, salinity gradients, and kinetic energy), captured, and stored. Frequently, this term is applied when speaking about small, wireless autonomous *devices*, like those used in wearable electronics and wireless *sensor networks*. Traditionally, electrical power has been generated in large, centralized plants powered by fossil fuels, nuclear fission or flowing water. Large-scale ambient energy, such as sun, wind and tides, is widely available but technologies do not exist to capture it with great efficiency. Energy harvesters currently do not produce sufficient energy to perform mechanical work, but instead provide very small amount of power for powering low-energy electronics. While the input fuel to large scale generation costs money (oil, coal, etc.), the "fuel" for energy harvesters is naturally present and	Wikipedia Contributors (2013) Energy harvesting. Wikipedia, the free encyclopedia. http://en.wikipedia. org/wiki/Energy_harvesting. Accessed 22 July 2013

Term	Definition	Source
	is therefore considered free. For example, temperature gradients exist from the operation of a combustion engine and in urban areas, there is also a large amount of electromagnetic energy in the environment because of radio and television broadcasting".	
Entity of Interest (EoI)	Any physical object as well as the attributes that describe it and its state that is relevant from a *user* or application *perspective*. The term is obsolete in the IoT-A reference model: the term *Physical Entity* should be used instead	Internal
Gateway	A Gateway is a forwarding element, enabling various local networks to be connected. Gateways can be implemented in *Device* that provides protocol translation between peripheral trunks of the IoT that are provided with lower parts of the communication stacks. For efficiency purposes, *gateways* can act at different layers, depending on which is the lowest layer in a common protocol implementation. *Gateways* can also provide support for security, scalability, service discovery, geo-localisation, billing, etc.	Internal
Global Storage	*Storage* that contains global information about many *entities of interest*. Access to the *global storage* is available over the *Internet*.	Internal
Human	A *Human* that either physically interacts with *Physical Entities* or records information about them, or both.	Internal
Identifier (ID)	Artificially generated or natural feature used to disambiguate things from each other. There can be several *IDs* for the same *PhysicalEntity*. This set of *IDs* is an attribute of a *PhysicalEntity*.	Internal
Identity	Properties of an entity that makes it definable and recognizable.	Internal

Information Model	"An *Information Model* is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. The advantage of using an Information Model is that it can provide sharable, stable, and organized structure of information requirements for the domain context. The *Information Model* is an abstract representation of entities which can be real objects such as devices in a network or logical such as the entities used in a billing system. Typically, the *Information Model* provides formalism to the description of a specific domain without constraining how that description is mapped to an actual implementation. Thus, different mappings can be derived from the same *Information Model*. Such mappings are called data models."	Autonomic Internet (AutoI) Project Information Model, Deliverable D3.1. Available online at: http://ist-autoi.eu/
Infrastructure Services	Specific services that are essential for any IoT implementation to work properly. Such services provide support for essential features of the IoT.	Internal
Interface	"Named set of operations that characterize the behaviour of an entity."	Open GeoSpatial Consortium (2002) The open GIS abstract specification topic 12: the openGIS service architecture. http://portal.opengeospatial.org/files/?artifact_id1221. Accessed 14 June 2011
Internet	"The *Internet* is a global *system* of interconnected computer networks that use the standard *Internet* protocol suite (TCP/IP) to serve billions of *users* worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks of local to global scope that are linked by a broad array of electronic and optical networking technologies. The *Internet* carries a vast array of information *resources* and *services*, most notably the	Wikipedia Contributors (2013) Internet Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Internet. Accessed 22 July 2013

Term	Definition	Source
	inter-linked hypertext documents of the World Wide Web (WWW) and the infrastructure to support electronic mail.	
	Most traditional communications media, such as telephone and television *services*, are reshaped or redefined using the technologies of the *Internet*, giving rise to *services* such as Voice over *Internet* Protocol (VoIP) and IPTV. Newspaper publishing has been reshaped into Web sites, blogging, and web feeds. The *Internet* has enabled or accelerated the creation of new forms of *human* interactions through instant messaging, *Internet* forums, and social networking sites.	
	The *Internet* has no centralized governance in either technological implementation or policies for access and usage; each constituent network sets its own standards. Only the overreaching definitions of the two principal name spaces in the *Internet*, the *Internet*-protocol address space and the domain-name *system*, are directed by a maintainer organization, the *Internet* Corporation for Assigned Names and Numbers (ICANN). The technical underpinning and standardization of the core protocols (IPv4 and IPv6) is an activity of the *Internet* Engineering Task Force (IETF), a non-profit organization of loosely affiliated international participants that anyone may associate with by contributing technical expertise."	
Internet of Things (IoT)	The global network connecting any smart object.	Internal
Interoperability	"The ability to share information and services. The ability of two or more systems or components to exchange and use information. The ability of systems to provide and receive services from other systems and to use the	The Open Group (2008) TOGAF version 9 – a manual, 9th ed., First Impression. s.l. Van Haren Publishing, Zaltbommel

services so interchanged to enable them to operate effectively together."

Term	Description		Reference
IoT Service	Software component enabling interaction with *resources* through a well-defined interface, often via the Internet. Can be orchestrated together with non-IoT services (e.g., enterprise services).	Internal	
Leaf Device	A node placed in outer part of routing area of a network. It is referred to as leaf with reference to the routing tree, where links are the branches and the outer nodes of the network are the leaves of the tree.	Internal	
Local Storage	Special type of *Resource* that contains information about one or only a few *Entities* in the vicinity of a *device*.	Internal	
Location Technologies	All technologies whose primary purpose is to establish and communicate the location of a *device* e.g. GPS, RTLS, etc.	Internal	
Look-up	In contrast to *Discovery*, *Look-up* is a *Service* that addresses exiting known *Resources* using a key or *Identifier*.	Internal	
M2M (also referred to as machine to machine)	"The automatic communications between *devices* without *human* intervention. It often refers to a *system of* remote *sensors* that is continuously transmitting data to a central *system*. Agricultural weather sensing *systems*, automatic meter reading and *RFID* tags are examples."		Computer Dictionary Definition, available online at: http://www.yourdictionary.com/m2m. Accessed 22 July 2013
Microcontroller	"A *microcontroller* is a small computer on a single integrated circuit containing a processor core, memory, and programmable input/output peripherals. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a typically small amount of RAM. *Microcontrollers* are designed for embedded applications, in contrast to the microprocessors used in personal computers or other general purpose applications.		Wikipedia Contributors (2013) Internet Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Microcontroller. Accessed 22 July 2013

Term	Definition	Source
	Microcontrollers are used in automatically controlled products and *devices*, such as automobile engine control *systems*, implantable medical *devices*, remote controls, office machines, appliances, power tools, and toys. By reducing the size and cost compared to a design that uses a separate microprocessor, memory, and input/output *devices*, *microcontrollers* make it economical to digitally control even more *devices* and processes. Mixed signal *microcontrollers* are common, integrating analog components needed to control non-digital electronic *systems*".	
Network-based resource	*Resource* hosted somewhere in the network, e.g., in the cloud.	Internal
Next-Generation Networks (NGN)	"Packet-based network able to provide telecommunication *services* and able to make use of multiple broadband, QoS-enabled transport technologies and in which *service*-related functions are independent from underlying transport-related technologies"	ETSI Technical report ETR 102 477 Corporate telecommunication Networks (CN); Mobility for enterprise communication, online at: http://www.etsi.org/deliver/etsi_tr/102400_102499/102477/01.01.01_60/tr_102477v010101p.pdf
Observer	Anything that has the capability to monitor a *Physical Entity*, like its state or location.	Internal
On-device Resource	*Resource* hosted inside a *Device* and enabling access to the *Device* and thus to the related *Physical Entity*.	Internal
Passive Digital Artefact	*Passive Digital Artefact Digital Artefacts* are passive software elements such as data-base entries or other digital representations of the *Physical Entity*.	Internal
Passive Digital Entities	A digital representation of something stored in an IT-based system. Obsolete: the term to be used is *Passive Digital Artefact*.	Internal
Perspective (also referred to as architectural perspective)	"Architectural perspective is a collection of activities, checklists, tactics and guidelines to guide the process of ensuring that a *system* exhibits a particular set of	Rozanski, N (2013) The context viewpoint http://www.viewpoints-and-perspectives.info/home/viewpoints/context/. Accessed 08 July 2013

	closely related quality properties that require consideration across a number of the *system*'s architectural *views*."	Internal
Physical Entity	A Physical Entity is a discrete, identifiable part of the physical environment that is of interest to the user for the completion of her goal. Physical Entities can be almost any object or environment; from humans or animals to cars; from store or logistics chain items to computers; from electronic appliances to closed or open environments.	Internal
Reference Architecture	A reference architecture is an architectural design pattern that indicates how an abstract set of mechanisms and relationships realises a predetermined set of requirements. It captures the essence of the architecture of a collection of systems. The main purpose of a reference architecture is to provide guidance for the development of architectures. One or more reference architectures may be derived from a common reference model, to address different purposes/usages to which the Reference Model may be targeted.	Internal
Reference Model	"A reference model is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details. A reference model may be used as a basis for education and explaining standards to non-specialists."	OASIS-RM. Reference model for service oriented architecture 1.0 http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

Term	Definition	Source
Requirement	"A quantitative statement of business need that must be met by a particular *architecture* or work package."	Open Group (2009) TOGAF 9
Resolution	*Service* by which a given ID is associated with a set of *Addresses* of information and interaction *Services*. Information *Services* allow querying, changing and adding information about the thing in question, while interaction services enable direct interaction with the thing by accessing the *Resources* of the associated *Devices*. Resolution is based on a priori knowledge.	Internal
Resource	Heterogeneous, generally system-specific, software components that store or process data or information about one or more *Physical Entities*, or that provide access to measurements and actuations in the case of *Sensors* and *Actuators* respectively.	Internal
RFID	"The use of electromagnetic or inductive coupling in the radio frequency portion of the spectrum to communicate to or from a tag through a variety of modulation and encoding schemes to uniquely read the *identity* of an RF Tag."	ISO/IEC 2382–1 Information technology – Vocabulary – Part 1: Fundamental terms, online at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=7229
Sensor	Special *Device* that measures physical characteristics of one or more *Physical Entities*.	Internal
Service	Platform-independent computational entity that can be used in a platform-independent way.	Internal
Stakeholder (also referred to as system stakeholder)	"An individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system."	IEEE (1471–2000) IEEE recommended practice for architectural description of software-intensive systems
Storage	Special type of *Resource* that stores information coming from *Resources* and provides information about *Entities*. They may also include *Services* to process the information stored by the *Resource*. As *Storages* are *Resources*, they can be deployed either on-device or in the network.	Internal

System	"A collection of components organized to accomplish a specific function or set of functions."	IEEE (1471–2000) IEEE recommended practice for architectural description of software-intensive systems
Tag	Label or other physical object used to identify the *Physical Entity* to which it is attached.	Internal
Thing	Generally speaking, any *physical object* in combination with its *digital representation*. In other words, it denotes the same concept as an *Augmented Entity*.	Internal
User	A *Human* or some *Active Digital Artefact* that is interested in interacting with a particular physical object.	Internal
View	"The representation of a related set of concerns. A *view* is what is seen from a *viewpoint*. An architecture *view* may be represented by a model to demonstrate to stakeholders their areas of interest in the architecture. A *view* does not have to be visual or graphical in nature".	Open Group (2009) TOGAF 9
Viewpoint	"A definition of the perspective from which a *view* is taken. It is a specification of the conventions for constructing and using a *view* (often by means of an appropriate schema or template). A *view* is what you see; a *viewpoint* is where you are looking from - the vantage point or perspective that determines what you see".	Open Group (2009) TOGAF 9
Virtual Entity	Computational or data element representing a *Physical Entity*. Virtual Entities can be either Active or Passive *Digital Artefacts*.	Internal
Wireless communication technologies	"Wireless communication is the transfer of information over a distance without the use of enhanced electrical conductors or "wires". The distances involved may be short (a few meters as in television remote control) or long (thousands or millions of kilometres for radio communications). When the context is clear, the term is often shortened to "wireless". Wireless communication is generally considered to be a branch of telecommunications."	Wikipedia Contributors (2013) Internet Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Wireless. Accessed 22 July 2013

Term	Definition	Source
Wireless Sensors and Actuators Network	"Wireless *Sensor* and *Actuator* Networks (WS&ANs) are networks of nodes that sense and, potentially, control their environment. They communicate the information through wireless links enabling interaction between people or computers and the surrounding environment."	OECD (2009) Smart sensor networks: technologies and applications for green growth. Available online at: http://www.oecd.org/dataoecd/39/62/44379113.pdf
Wireline communication technologies	"A term associated with a network or terminal that uses metallic wire conductors (and/or optical fibres) for telecommunications."	Setzer-Messtechnik (2010) Setzer-Messtechnik glossary. Available online at: http://www.Setzer-Messtechnik,2010.at/grundlagen/rf-glossary.php?lang=en

Appendices B–E

Appendix B

Requirements (available at http://www.iot-a.eu/public/requirements)

Appendix C

Use cases, sequence charts and interfaces (available at http://www.iot-a.eu/public/public-documents/d1.5/view)

Appendix D

Process and Methodology (available at http://www.iot-a.eu/public/public-documents/d1.5/view)

Appendix E

Requirements for the Concrete Architecture (available at http://www.iot-a.eu/public/public-documents/d1.5/view)

References

Activiti BPM Platform (2012) Activiti

IEEE Architecture Working Group (2000) IEEE Std 1471–2000, Recommended practice for architectural description of software-intensive systems

Ashton K (2009) That 'Internet of Things' thing. RFID J, http://www.rfidjournal.com/articles/view?4986

Association for Automatic identification and mobility, online at: http://www.aimglobal.org/

Barnaghi P, Meissner S, Presser M, Moessner K (2009) Sense and sens'ability: semantic data modelling for sensor networks. In: ICT Mobile Summit 2009, 10 June 2009–12 June 2009, Santander, Spain

Bauer M, De S, Longo S (2013) "WP4 white paper on resolution infrastructure interface binding". www.iot-a.eu/public/public-document/WhitePaperWP4/view

Bell-La Padula (1976) Secure computer system: unified exposition and multics interpretation. MITRE report ESD-TR-75-306. http://csrc.nist.gov/publications/history/bell76.pdf

Ben Saied Y, Olivereau A (2012) HIP Tiny Exchange (TEX): a distributed key exchange scheme for HIP-based internet of things. 3rd International Conference on Communications and Networking (ComNet), Hammamet/Tunisia

Ben Saied Y, Olivereau A (2012) D-HIP: a distributed key exchange scheme for HIP-based internet of things. First IEEE WoWMoM workshop on the internet of things: smart objects and services, IoT-SoS, San Francisco, CA, USA

Boehm BW (1988) A spiral model of software development and enhancement. Computer 21(5):61–72

Brucker AD et al (2012) SecureBPMN: modeling and enforcing access control requirements in business processes. ACM, New York, pp 123–126

Bui N (ed) (2011) Project deliverable D1.1 – SOTA report on existing integration frameworks/architectures for WSN, RFID and other emerging IoT related Technologies. http://www.iot-a.eu/public/public-documents/project-deliverables/1/1/110304_D1_1_Final.pdf/at_download/file. Accessed 09 June 2011

Business Process Model And Notation (BPMN) (2011) OMG specification. Object Management Group

BUTLER (2011) Ubiquitous, secure internet-of-things with location and contex-awareness. Project homepage, http://www.iot-butler.eu/. Accessed 11 July 2013

Carrez F (ed), Bauer M, Boussard M, Bui N, Jardak C, De Loof J, Magerkurth C, Meissner S, Nettsträter A, Olivereau A, Serbanati A, Stefa J, Thoma M, Walewski JW (2013) Final architectural reference model for the IoT v3.0. Internet of things – Architecture – Project Deliverable D1.5

Computer Dictionary Definition, online at: http://www.yourdictionary.com/computer/m2-m

PISA Consortium (2003) Handbook of privacy and privacy-enhancing technologies: the case of intelligent software agents. http://www.cbpweb.nl/downloads_technologies/pisa_handboek.pdf

Consorzio FR (2011) Internet of things architecture – Project Deliverable D1.1

Consultative Committee (2006) The consultative committee for space data systems. Information Architecture reference model. CCSDS_312.0-G-0. http://cwe.ccsds.org/sea/docs/SEA-IA/Draft%20Documents/IA%20Reference%20Model/ccsds_rasim_20060308.pdf. Accessed 30 Apr 2013

NGSI Context Management Specification (2010) Open mobile alliance. http://www.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20101207-C/OMA-TS-NGSI_Context_Management-V1_0-20100803-C.pdf. Accessed 11 June 2012

DARPA (1970) DoD networking model. http://www.freesoft.org/CIE/Course/Section1/5.htm

De S (2012) Concepts and solutions for entity-based discovery of IoT resources and managing their dynamic associations. EC FP7 IoT-A Deliverable 4.3

de las Heras R (ed) (2011) Project deliverable D4.1 – Concepts and solutions for identification and lookup of IoT resources, December 2011. Available at: http://www.iot-a.eu/public/public-documents/documents-1/

De S, Barnaghi P, Bauer M, Meissner S (2011) Service modelling for the Internet of Things. Computer Science and Information Systems (FedCSIS), Federated Conference on IEEE

De S, Elsaleh T, Barnaghi P, Meissner S (2012) An internet of things platform for real-world and digital objects. Scalable Comput: Pract Ex 13(1):45–57, West University of Timisoara

Dey AK (2001) Understanding and using context. Pers Ubiquit Comput 5(1):4–7

Digital Payment Techno (2013) Digital payment pechnologies, "Pay-by-Licence Plate". http://www.digitalpaytech.com/products/operational-modes/pay-by-plate.aspx. Accessed 12 Apr 2013

Ebios (2010) Agence Nationale de la Sécurité des Systèmes d'Information (ANSSI). Ebios 2010 – expression of needs and identification of security objectives. Technical report. 2010

EC FP7 IoT-A Project Deliverable D1.5. Available online at: http://www.iot-a.eu/public/public-documents/d1.5/view

EC FP7 IoT-A project deliverable D2.6 – events representation and processing, http://www.iot-a.eu/public/public-documents/documents-1

E-FRAME project, available online at: http://www.frame-online.net/top-menu/the-architecture-2/faqs/stakeholder-aspiration.html

EPC Information Services (EPCIS) Version 1.0.13 Specification

EPC Tag Data Standard (TDS), current version v.16 (2011 September 9), online at : http://www.gs1.org/gsmp/kc/epcglobal/tds

Erder M, Pureur P (2003) QFD in the architecture process. IT Professional 5(6):44–52

Eschenauer L, Gligor VD (2002) A key-management scheme for distributed sensor networks. Proceedings of the 9th ACM conference on Computer and communications security. Washington, DC, pp 41–47

ETSI Corporate telecommunication Networks (CN); Mobility for enterprise communication, online at: http://www.etsi.org/deliver/etsi_tr/102400_102499/102477/01.01.01_60/tr_102477v010101p.pdf

ETSI Technical report ETR 173, Terminal Equipment (TE); Functional model for multimedia applications. Available online: http://www.etsi.org/deliver/etsi_etr/100_199/173/01_60/etr_173e01p.pdf

Fiedler M (ed), Bui N, De Loof J, Haller S, Hinkelmann M, Ho E, Magerkurth C, Mättig B, Martin Romero G, Savry O, Serbanati A, Zeybek E (2012) Internet of things – architecture – project deliverable D7.2 – exact definition use case 1 and use case 2

Flextronics (2005) Software systems FCAPS White Paper http://marco.uminho.pt/~dias/MIECOM/GR/Projs/P2/fcaps-wp.pdf

Fowler M (2003) UML distilled: a brief guide to the standard object modeling language, 3rd edn. Addison-Wesley Professional, Boston

Furness A (2009) Ontology for identification. CASAGRAS Final Report, Annex C. http://www.grifs-project.eu/data/File/Casagras_Final%20Report.pdf. Accessed 18 Apr 2012

Gambetta D (2000) Can we trust trust? Trust: making and breaking cooperative relations, electronic edition. Department of Sociology, University of Oxford, pp 213–237

Gamma E et al (1994) Design patterns: elements of reusable object-oriented software. 1. s.l. : Addison-Wesley Professional

Ganeriwal S, Srivastava MB (2004) Reputation-based framework for high integrity sensor networks. 2nd ACM Workshop on Security of Ad Hoc and Sensor Networks, Washington, DC, pp 66–77

Genetec (2013) Parking enforcement and management.http://www.genetec.com/Solutions/Pages/parking-enforcement-and-inventory.aspx. Accessed 12 Apr 2013

Georgetown University (2013) Disaster recovery http://continuity.georgetown.edu/dr/. Accessed 12 Apr 2013

Greenfield N (2009) FCAPS management for smart grid – high-level summary. AEP IT Security Engineering. http://osgug.ucaiug.org/UtiliComm/Shared%20Documents/AMI-NET/FCAPS%20Management%20for%20the%20Smart%20Grid.pdf

Gruschka N, Gessner D (2012) EC FP7 IoT-A Project Deliverable D4.2. Available online at: http://www.iot-a.eu/public/public-documents/d4.2/view

Haller S (2010) The things in the internet of things. Tokyo: s.n

Heer T, Garcia-Morchon O, Hummen R, Keoh SL, Kumar SS, Wehrle K (2011) Security challenges in the IP-based internet of things. Wireless Pers Commun 61(3):527–542. ISSN 0929–6212 (doi: 10.1007/s11277-011-0385-5)

Houyou AM et al (2012) D2.3 – Plug & work support mechanisms. IoT@Work

Hull E, Jackson K, Dick J (2011) Requirements engineering, 3rd edn. Springer Publishing Company, New York

Hyttinen P (ed), Azzabi R, Bauer M, Christophe B, Saied YB, Boudguiga A, De S, Gessner D, Hyttinen P, Kiljander J, Longo S, Olivereau A, Serbanati A, Stefa J (2013) Internet of things – Architecture – Project deliverable D4.4 – Final Design and Implementation Report

IBM (2012) Circular versus Archive transactional logging. http://www-01.ibm.com/support/docview.wss?uid=swg21087828. Accessed 12 Apr 2013

IEEE (1471–2000) IEEE recommended practice for architectural description of software-intensive systems

IETF (1998) IETF RFC 2401 security architecture for the internet protocol. http://www.ietf.org/rfc/rfc2401.txt 1998

IETF (2008) IETF RFC 5246 The Transport Layer Security (TLS) protocol http://tools.ietf.org/html/rfc5246

IETF (2011) IETF RFC 6101 The Secure Sockets Layer (SSL) protocol version 3.0. http://tools.ietf.org/html/rfc6101

Information architecture reference model. Online at: http://cwe.ccsds.org/sea/docs/SEA-IA/Draft%20Documents/IA%20Reference%20Model/ccsds_rasim_20060308.pdf

Information Model, Deliverable D3.1, Autonomic Internet (AutoI) Project. Online at: http://ist-autoi.eu/autoi/d/AutoI_Deliverable_D3.1_-_Information_Model.pdf

Information technology – vocabulary – part 1: fundamental terms. Online at: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=7229

International Organization for Standardization (2009) Selection and use of the ISO 9000 family of standards http://www.iso.org/iso/home/store/publications_and_e-products/publication_item.htm?pid=PUB100208. Accessed 23 May 2013

IoT-I Deliverable D1.5 Final white paper defining a reference model for IoT, Stephan Haller (ed). http://www.iot-i.eu/public/public-deliverables/

IoT-A FP7 Project (2012) Terminology – IOT-A: internet of things architecture. http://www.iot-a.eu/public/terminology. Accessed 12 Apr 2013

IoT-A FP7 Project (2013) Requirements – IoT-A: internet of things architecture. http://www.iot-a.eu/public/requirements. Accessed 21 June 2013

Island Group (2012) PRESTO 1000 pay & display monitoring software http://www.islandgroup.co.uk/ParkingSolutionsPaD.aspx. Accessed 12 Apr 2013

ISO (1994) Information technology – open system interconnection – basic reference model: the basic model. http://www.ecma-international.org/activities/Communications/TG11/s020269e.pdf. Accessed 21 June 2013

ITU-T (1997) M.3400 TMN management functions

Karlof C, Wagner D (2003) Secure routing in wireless sensor networks: attacks and countermeasures. Ad Hoc Netw 1(2–3):293–315

Koshizuka N, Sakamura K (2010) Ubiquitous ID: standards for ubiquitous computing and the internet of things. IEEE Pervasive Comput 9(4):98–101

Kozel T (2010) BPMN mobilisation. Proceedings of the European conference of systems: World Scientific and Engineering Academy and Society (WSEAS), Puerto De La Cruz, Tenerife

Kranzfelder M, Zywitza D, Jell T, Schneider A, Gillen S, Friess H, Feussner H (2012) Real-time monitoring for detection of retained surgical sponges and team motion in the surgical operation room using RFID technology: a preclincial evaluation. J Surg Res 175(5):191–198

Kruchten PB (1995) The 4+1 view model of architecture. IEEE Software 12(6):42–50

Lefort L (2005) Ontology for quantity kinds and units: units and quantities definitions. W3 Semantic Sensor Network Incubator Activity

Levis P, Gay D (2009) Tiny OS programming. 1. s.l. Cambridge University Press, Cambridge, UK

Lu B, Pooch UW (2005) A lightweight authentication protocol for mobile Ad Hoc Networks. Int J Inf Technol 11(2):119–135

MacKenzie CM, Laskey K, McCabe F, Brown P, Metz R (2006) Reference model for service oriented architecture 1.0. Available online at: https://www.oasis-open.org/committees/download.php/19679/

Machine-to-Machine Communications (M2M) Threat analysis and counter-measures to M2M service layer. Available at http://www.etsi.org

Magerkurth C (ed), Bauer M, Boussard M, Bui N, Carrez F, Giacomin P, Ho E, Jardak C, De Loof J, Magerkurth C, Meissner S, Nettsträter A, Olivereau A, Serbanati A, Thoma M, Walewski J W (2012) Internet of things – architecture – project deliverable D1.4 – Converged architectural reference model for the IoT v2.0

Magerkurth C (ed), Salinas Segura A, Vicari N, Boussard M, Meyer S (2013) Internet of things – architecture – project deliverable D6.3 – final requirements list

Magerkurth C, Sperner K, Meyer S, Strohbach M (2011) Towards context-aware retail environments: an infrastructure perspective. Mobile Interaction in Retail Environments (MIRE 2011), Stockholm, Sweden

Martín G (ed) (2012) Resource description specification. IoT-A deliverable D2.1

Mathur CN, Subbalakshmi KP (2007) Security issues in cognitive radio networks. In: Cognitive networks: towards self-aware networks. Wiley, Chichester, pp 284–293

Menezes AJ, Vanstone SA, Van Oorschot PC (1996) Handbook of applied cryptography. CRC Press, Boca Raton

Meyer S (2012) Concepts for modeling IoT-Aware processes. EC FP7 IoT-A Deliverable 2.2.

Meyer S, Sperner K, Magerkurth C, Pasquier J (2011) Towards modeling real-world aware business processes. Proceedings of the Second International Workshop on Web of Things, p 8. ACM

Meyer S, Ruppen A, Magerkurth C (2013) Internet of things-aware process modeling: integrating IoT devices as business process resources. In: Proceedings of 25th International Conference on Advanced Information Systems Engineering (CAISE '13), Valencia

Microsoft Corporation (2003) Threat modeling. http://msdn.microsoft.com/en-us/library/ff648644.aspx. Accessed 13 May 2013

Microsoft Corporation (2013) Transactional replication. http://msdn.microsoft.com/en-us/library/ms151176.aspx. Accessed 12 Apr 2013

MUNICH Platform Workshop (MUWS) (2013) Klinikum rechts der Isar

Miller J, Mukerji J (eds) (2003) MDA guide version 1.0.1. Framingham, Massachusetts

Muller G (2008) A Reference Architecture Primer. Available at: http://www.gaudisite.nl/info/ReferenceArchitecturePrimer.info.html

Multi-National Initiative for Cloud Computing in Health Care (MUNICH) (2010) http://munichplatform.eu/. Accessed 08 July 2013

OASIS-RM. Reference model for service oriented architecture 1.0 http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf

OCTAVE® Information Security Risk Evaluation

OECD (2009) Smart sensor networks: technologies and applications for green growth. December, online at: http://www.oecd.org/dataoecd/39/62/44379113.pdf

Open GeoSpatial Consortium (2002) The open GIS abstract specification topic 12: the openGIS service architecture. http://portal.opengeospatial.org/files/?artifact_id=1221. Accessed 14 June 2011

Open GeoSpatial portal, the OpenGIS abstract specification topic 12: the OpenGIS Service architecture. Online at: http://portal.opengeospatial.org/files/?artifact_id=1221

Open Group (2009) TOGAF 9

OSGi Alliance (2012) OSGi alliance specifications. http://www.osgi.org/Specifications/HomePage. Accessed 29 May 2013

Pastor A, Alcatel-Lucent Bell Labs France (eds) (2011) Internet of things – architecture – project deliverable D6.2 – updated requirements list

Peris-Lopez P, Hernandez-Castro JC, Estevez-Tapiador JM, Ribagorda A (2007) Solving the simultaneous scanning problem anonymously: clumping proofs for RFID Tags. security, privacy and trust in pervasive and ubiquitous computing, SECPerU 2007. Third International Workshop, pp 55–60

Perrig A, Szewczyk R, Tygar JD, Wen V, Culler DE (2002) SPINS: security protocols for sensor networks. http://www.cs.berkeley.edu/~tygar/papers/SPINS/SPINS_wine-journal.pdf

Perrig A, Stankovic J, Wagner D (2004) Security in wireless sensor networks. Communications of the ACM – Wireless Sensor Networks, 47(6), http://dl.acm.org/citation.cfm?id=990707

Phares B (2007) Demonstration of the electrochemical fatigue sensor system at the Transportation Technology Center Facility. Calgary, Alberta, Canada: s.n.

PIA (2013) The privacy impact assessment handbook. http://www.ico.gov.uk/for_organisations/data_protection/topic_guides/privacy_impact_assessment.aspx. Accessed 29 May 2013

Pohl K (2010) Requirements engineering: fundamentals, principles, and techniques, 1st edn. Springer Publishing Company, New York

Polastre J, Szewczyk R, Culler D (2005) Telos. Enabling ultra-low power wireless research. IEEE Press, Piscataway

Pras A (1995) Network management architectures. PhD Thesis, University of Twente

Raymond K (1995) Reference Model of Open Distributed Processing (RM-ODP): introduction. http://dl.dropbox.com/u/40619198/rmodpwiki/files/Tutorials/ODP_Tutorial-icodp95.pdf. Accessed 08 May 2013

Reliable Software Inc (2012) Designing for failure. http://www.reliablesoftware.com/DasBlog/PermaLink,guid,33102321-b3e5-48d4-8de6-62175b9ad09e.aspx. Accessed 12 Apr 2013

Rescorla E (1999) Diffie-Hellman key agreement method. Network Working Group. http://tools.ietf.org/html/rfc2631

Römer K, Mattern F, Dübendorfer T, Senn J (2002) Infrastructure for virtual counterparts of real world objects. Technical Report, ETH Zürich

Rossi M (ed) (2012) IoT-A D3.3 deliverable: protocol suite http://www.iot-a.eu/public/public-documents

Rossi M (ed) (2013) IoT-A D3.6 deliverable: IoT protocol suite definition

Rowley J (2007) The wisdom hierarchy: representations of the DIKW hierarchy. J Inf Sci 33 (2):163–180

Royce WW (1970) Managing the development of large software systems. Proceedings IEEE WESTCON, Los Angeles (August 1970) Reprinted in ıt Proceedings of the ninth international conference on software engineering, March 1987, pp 328–338

Rozanski N (2013) The context viewpoint http://www.viewpoints-and-perspectives.info/home/viewpoints/context/. Accessed 08 July 2013

Rozanski N, Wodds E (2005–2013) Applying viewpoints and views to software architecture. http://www.viewpoints-and-perspectives.info/vpandp/wp-content/themes/secondedition/doc/VPandV_WhitePaper.pdf. Accessed 08 May 2013

Rozanski N, Woods E (2011) Software systems architecture – working with stakeholders using viewpoints and perspectives. Addison Wesley, Boston

SAP Research, S. AGA.P. Internet of Services: about USDL

Scheer A, Cameron I (1992) Architecture of integrated information systems: foundations of enterprise modelling. Springer, Berlin/Heidelberg

Sclater N (2007) Mechanisms and mechanical devices sourcebook, 4th edn. McGraw-Hill, New York, 25

Serbanati A, Medaglia CM, Ceipidor UB (2011) Building blocks of the internet of things: state of the art and beyond. In: Dr. Turcu C (ed) Deploying RFID – challenges, solutions, and open issues ISBN: 978-953-307-380-4, InTech, online at: http://www.intechopen.com/books/deploying-rfid-challenges-solutions-and-open-issues/building-blocks-of-the-internet-of-things-state-of-the-art-and-beyond

Setzer-Messtechnik (2010) Setzer-Messtechnik glossary. July, online at: http://www.Setzer-Messtechnik,2010.at/grundlagen/rf-glossary.php?lang=en

Shames P, Yamada T (2004) Reference architecture for space data systems. s.l. : DSpace at Jet Propulsion Laboratory. http://trs-new.jpl.nasa.gov/dspace-oai/request

Signavio GmbH (2012) Signavio core components

Silver B (2009) BPMN method and style. Cody-Cassidy Press, Dallas

SPARQL (2008) W3C. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/. Accessed 07 Nov 2012

Sperner K, Meyer S, Magerkurth C (2011) Introducing entity-based concepts to business pro-cess modeling. Business Process Model and Notation, pp 166–171

Tamblyn S, Hinkel H, Saley D (2007) NASA CEV reference GN&C architecture

The Consultative Comm (2006) The Consultative Committee for space data systems. Information Architecture Reference Model. CCSDS_312.0-G-0. http://cwe.ccsds.org/sea/docs/SEAIA/Draft%20Documents/IA%20Reference%20Model/ccsds_rasim_20060308.pdf. Accessed 15 June 2011

The Open Group (2008) TOGAF version 9 – a manual, 9th edn, First Impression. s.l. Van Haren Publishing, Zaltbommel

UID Center Specifications, online at: http://www.uidcenter.org/spec#UID-00010

Usländer T (ed) (2007) Reference model for the orchestra architecture (RM-OA) V2. Open Geospatial Consortium Inc., OGC 07-024

Verbauwhede I, Schaumont P (2007) Design methods for security and trust. http://cecs.uci.edu/~papers/date07/PAPERS/2007/DATE07/PDFFILES/05.1.1_2.PDF. Accessed 11 May 2013

Vicente-Chicote C, Moros B, Álvarez JAT (2007) REMM-Studio: an integrated model-driven environment for requirements specification, validation and formatting. J Object Technol 6(9):437–454

Voelksen G (ed) (2013) EC FP7 IoT-A project deliverable D2.6 – events representation and processing, http://www.iot-a.eu/public/public-documents/documents-1

Volere (2013) Atlantic Systems Guild Ltd. Volere Requirements Resources. 1995–2003 http://www.volere.co.uk/. Accessed 04 June 2013

Wang Yang, Kobsa Alfred (2008) Privacy-enhancing technologies. CMU School of Computer Science

Wang Q, Jäntti R, Ali Y (2012) On network management for the internet of things. 8th Swedish National Computer Networking Workshop (SNCNW). http://users.tkk.fi/wangq1/SNCNW_OnNetworkManagement.pdf

Weber RH, Weber R (2010) Internet of things: legal perspectives. ZIK no 49. Springer, Berlin

Weiser M (1991) The computer of the 21st century. http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html. Accessed 11 July 2013

Whittington P (2010) A car parked in the Gatwick North Terminal Flightpath long stay car park http://www.geograph.org.uk/photo/2350033. Accessed 13 Apr 2013

Energy Harvesting page on Wikipedia (2010) Online at: http://en.wikipedia.org/wiki/Energy_harvesting

Internet page on Wikipedia (2010) Online at: http://en.wikipedia.org/wiki/Internet

Microcontroller page at Wikipedia (2010) Online at: http://en.wikipedia.org/wiki/Microcontroller

Wireless page on Wikipedia (2010) Online at: http://en.wikipedia.org/wiki/Wireless

Wikipedia Contributors (2012a) FCAPS. Wikipedia, the free encyclopedia https://en.wikipedia.org/wiki/Fcaps. Accessed 01 Oct 2012

Wikipedia Contributors (2012b) State machine replication. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/State_machine_replication. Accessed 12 Apr 2013

Wikipedia Contributors (2012c) Virtual synchrony. Wikipedia, the free encyclopedia http://en.wikipedia.org/wiki/Virtual_synchrony. Accessed 12 Apr 2012

Wikipedia Contributors (2012d) Eventual consistency. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Eventual_consistency. Accessed 12 Apr 2012

Wikipedia Contributors (2013a) 4+1 architectural view model. Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=4%2B1_architectural_view_model&oldid=548663664. Accessed 11 Apr 2013

Wikipedia Contributors (2013b) View model. Wikipedia, the free encyclopedia http://en.wikipedia.org/w/index.php?title=View_model&oldid=543215912. Accessed 12 Apr 2013

Wikipedia Contributors (2013c) Zachman framework. Wikipedia, the free encyclopedia http://en.wikipedia.org/w/index.php?title=Zachman_Framework&oldid=541396757. Accessed 10 May 2013

Wikipedia Contributors (2013d) Pay and display. Wikipedia, the free encyclopedia http://en.wikipedia.org/w/index.php?title=Pay_and_display&oldid=547120864. Accessed 20 May 2013

Wikipedia Contributors (2013e) Virtual synchrony. Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Virtual_synchrony. Accessed 12 Apr 2012

Wikipedia Contributors (2013f) Eventual consistency. Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Eventual_consistency. Accessed 12 Apr 2012

Wikipedia Contributors (2013g) Pay and display. Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Pay_and_display&oldid=547120864. Accessed 20 May 2013

Woods E, Nick R (2008) The system context architectural viewpoint. Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on IEEE. 2009

Woods E, Rozanski N (2005) 5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05), Pittsburgh, pp 25–35

Wu NC, Nystrom MA, Lin TR, Yu HC (2006) Challenges to global RFID adoption. Technovation 26(12):1317–1323

Zachman JA (1987) A framework for information systems architecture. IBM Syst J, 26(3). IBM Publication G321-5298

Zocher W (2013) RFID in OP. https://www.youtube.com/watch?v=8PK-jpds2qk&feature=youtu.be. Accessed 25 May 2013