Cooperative Device Cloud - Provisioning Embedded Devices in Ubiquitous Environments

vorgelegt von
Dipl.-Inf.
Andreas Kliem
geb. in Berlin

Von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr. Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Dr. h.c. Sahin Albayrak
Gutachter:	Prof. Dr. Odej Kao
	Prof. Dr. Thomas Magedanz
	Prof. Dr. Andreas Polze

Tag der wissenschaftlichen Aussprache: 26.05.2015

Berlin 2015

Acknowledgement

I would like to take the chance to express my gratitude to all the people who helped me making this thesis possible. First and foremost, I would like to thank my advisor Odej Kao for giving me the possibility to enjoy the past four years as a member of his research group, for sharing his knowledge and experience with me, and for supporting me whenever I feared to be lost. Also, I would like to express my appreciation to Thomas Magedanz and Andreas Polze for agreeing to review this work.

This work would not have been possible without the endless patience and support of my family. I would like to thank my mother Marianne, who endured all my mood swings, my father Bernhard, who spend endless hours guiding me through the sometimes rocky road of academia and, last but not least, my sister Christine and all other members of my small but mighty family (especially my cousin Katja who allowed me to be distracted while renovating her house).

I would like to thank all former and current members of the CIT research group for all the discussions and collaboration leading to numerous papers and research results being fundamental for this thesis. I will never forget the plenty evenings we spent together having a bear, an Ouzo or simply watching Big Bang Theory after 12 hours of thinking and coding.

Thanks to Jana Bechstein for proofreading this thesis and helping me with so many administrative issues.

Special thanks to Anette and Jens Ringel and all the staff from Dialysezentrum Potsdam for being patient with me during the final steps of preparing this thesis and giving me the chance to understand how ICT in medicine actually works.

Finally, I would like to say thank you to all my friends who, on the one hand, did not become tired asking me about the progress of my thesis and, on the other hand, were always around when distraction was required.

Abstract

Cloud Computing and the Internet of Things (IoT) are well known principles addressing core challenges of the distributed systems the Internet is founded on. Cloud Computing is already widely adopted in productive environments and has evolved from a "buzzword" to a commonly used technology. In contrast, IoT currently suffers from the lack of foundational design principles that allow overcoming the segmentation into solutions dedicated to particular IoT application domains, such as Automation, E-Health, Smart Homes, or Smart Cities. IoT applications are often tightly coupled to challenges like sensor integration, sensor management, semantics or, the heterogeneity of data and technologies in general. Besides the heterogeneity, the proliferation of IoT related solutions will lead to a continuously increasing amount of the sensors and devices, which are foundational to IoT applications. This leads to the challenge of efficiently managing and provisioning the resources provided by the devices. Looking at the Cloud Computing domain, popular related concepts like on demand provisioning, elasticity or, resource pooling are already available and well investigated.

This work presents the Device Cloud concept, which aims at applying these Cloud Computing concepts to the IoT domain. Sensors and devices will be organized in resource pools and dynamically provisioned to users that can benefit from the resources offered. The Device Cloud will eliminate static bindings between sensors and users and allow accessing any kind of physical IoT devices on demand, similar to the Pay As You Go paradigm popular within the scope of Cloud Computing.

The core question addressed by this thesis is how a dynamic, on-demand provisioning of the devices can be realized independently from the application domain or the technologies a device is based on. A generic, application independent architecture model will be introduced and discussed with respect to the participating actors, their interactions and, security and privacy. Based on an E-Health use case, the generic model will be applied to a specific application domain.

Zusammenfassung

Cloud Computing und das Internet der Dinge (IoT) behandeln grundlegende Prinzipien und adressieren zentrale Herausforderungen der Verteilten Systeme, auf denen das Internet beruht. Cloud Computing hat bereits eine weite Verbreitung in produktiven Umgebungen gefunden und sich von einem "Buzzword" zu einer alltäglich eingesetzten Technologie entwickelt. Im Gegensatz dazu, fehlt es dem IoT an grundlegenden Design Prinzipien, welche es ermöglichen, die bisher voneinander isoliert betrachteten Lösungen für typische IoT Anwendungsdomänen wie Automation, E-Health, Smart Homes oder Smart Cities zu integrieren und in ein einheitliches IoT Konzept einzubetten. Anwendungen des Internet der Dinge gehen oft eng mit Herausforderungen wie Sensor Integration, Sensor Management, Semantik oder der Heterogenität der Daten und Technologien im Allgemeinen einher. Verstärkend zum Problem Heterogenität, wird die fortschreitende Verbreitung des IoT zu einer kontinuierlich ansteigenden Menge an Sensoren und Geräten führen. Dies führt wiederum zu der Herausforderung, die durch die Sensoren und Gerät bereitgestellte Ressourcenmenge effizient zu verwalten und Nutzern zugänglich zu machen (provisionieren). Entsprechende Paradigmen wie das bedarfsgerechte Provisionieren, Elastizität oder das Pooling von Ressourcen sind dabei bereits im Bereich des Cloud Computing verbreitet.

Das zentrale Ziel des durch diese Arbeit präsentierten Device Cloud Konzepts ist es, diese Cloud Computing Paradigmen in die IoT Domäne abzubilden. Sensoren und Geräte werden in Ressourcen-Pools organisiert und den Nutzern dynamisch und bedarfsgerecht zugewiesen. Das Device Cloud Konzept wird die in IoT Applikationen weit verbreiteten statischen Bindungen zwischen Nutzern und Sensoren minimieren und, ähnlich zum Pay As You Go Paradigma des Cloud Computing, den Zugriff auf physische IoT Geräte aller Art nach Bedarf und Anforderungen der Nutzer ermöglichen.

Die für diese Arbeit zentrale Forschungsfrage ist, wie eine dynamische und bedarfsgerechte Zuordnung von Geräten zu Nutzern unabhängig von der Anwendungsdomäne oder den Basistechnologien eines Gerätes realisiert werden kann. Dazu wird ein generisches, von der Anwendungsdomäne unabhängiges, Architekturmodell eingeführt und mit Bezug zu den teilnehmenden Akteuren, deren Interaktionen sowie Fragestellungen der Sicherheit diskutiert. Basierend auf einem Anwendungsfall aus dem E-Health Bereich wird das generische Modell auf eine konkrete Anwendungsdomäne abgebildet.

Contents

1. Introduction

Contents

> "Ubiquitous Computing, often also referred to as Pervasive Computing, is a vision for computer systems to infuse the physical world and human and social environments. It is concerned with making computing more physical, in the sense of developing a wider variety of computer devices can be usefully deployed in more of the physical environment. [...]" - Poslad, 2009

Following the closely related Ubiquitous Computing (UC) [163] and Internet of Things (IoT) [12] visions, connected and embedded devices like smart devices or any general shape of sensing and actuating devices, are finding their way into more and more areas of our everyday lives. The technologies and applications we rely on are heavily influenced by the presence of embedded devices (i.e. embedded systems), regardless whether we are talking about the smart phones most of us use every day, heating control in our homes, computer-aided assistance systems in our cars or, numerous further examples. The dissemination of connected and embedded devices is not only driven by the estimation that the amount of such devices will grow to approximately 15 billion by 2015 and to 200 billion in 2020 [96] or the emergence of IoT related technologies in general as reported by Gartner [28]. Additionally, another important accelerator is given, according to Moore's Law [137], by the increasing features and capabilities embedded devices provide. Arising from the capability to interconnect and to collect data in an ad-hoc and mobile (ubiquitous) fashion, new applications, such as smart grids, traffic congestion, or seamless vital signs monitoring [101], become viable. This results in a mutual reinforcement of the development of both connected and embedded devices and corresponding applications. Well-known application domains with close relation to IoT technologies, for example, are E-Health [161], Transportation, Logistics [167], Automation, and Smart Homes and Cities [36].

Two basic assumptions build the foundation of the Ubiquitous Computing (UC) and Internet of Things (IoT) visions. First, primarily introduced by Weiser's work on UC [163], it is assumed that users are no longer supported by a single monolithic computer system

(e.g. a PC). Rather, a set of embedded devices surrounding them constantly provides the necessary resources and applications to fulfill their everyday needs. The term resources refers to both data and infrastructure (e.g. storage). The second important assumption introduced by the IoT is, that so called things or entities, which usually refer to the embedded devices but can also be applied to users equipped with body area or implanted sensors, are uniquely identifiable and are linked to virtual representations within an information network similar to the Internet. The term virtual representation basically constitutes that a thing is connected to an information network and that it can be accessed through a communication protocol or an interface/service (e.g. a sensor providing data about a user's environment). It becomes obvious that taking advantage of the seamless integration of these things requires a huge amount of technical infrastructure to be developed, deployed and maintained. A definition of IoT putting emphasis on the technical components was given by the Strategic Research Agenda of the Cluster of European Research Projects on the Internet of Things (CERP-IoT) [147]:

> "Internet of Things (IoT) is an integrated part of Future Internet and could be defined as a dynamic global network infrastructure with self configuring capabilities based on standard and interoperable communication protocols where physical and virtual "things" have identities, physical attributes, and virtual personalities and use intelligent interfaces, and are seamlessly integrated into the information network. In the IoT, "things" are expected to become active participants in business, information and social processes where they are enabled to interact and communicate among themselves and with the environment by exchanging data and information "sensed" about the environment, while reacting autonomously to the "real/physical world" events and influencing it by running processes that trigger actions and create services with or without direct human intervention. Interfaces in the form of services facilitate interactions with these "smart things" over the Internet, query and change their state and any information associated with them, taking into account security and privacy issues." - CERP-IoT, 2009

It was emphasized, that the possibility to assign unique identifiers to the things is important, because it allows maintaining state and history, recording the information flow, and keeping track of interactions with other things [147]. This can be recognized as a mandatory requirement if embedded devices become regular participants in an information network, are collaborating with other entities (users and things) and are not only acting as proxies for users into the digital world. Moreover, unique identifiers allow establishing back-end information structures like discovery or repository services required for monitoring purposes or as mediators for self-configuration and optimization capabilities [156]. Apart from the capability of treating things like individuals by assigning globally unique identifiers to them, a variety of supporting technologies like communication and network technologies, data processing technologies, or security and privacy

technologies need to be considered. However, as discussed by Uckelmann et al. [152], one has to keep in mind that these technologies have to be distinguished from the Internet of Things (IoT) and are not to be considered as synonyms when discussed individually. In particular, visions and technologies like Ubiquitous Computing (UC), Embedded Devices, Wireless Sensor Networks (WSNs), or the Internet Protocol (IP), have a strong relation to and partial overlap with IoT, but only considering them together as an overall integrated technology framework covers the whole topic IoT is about. A set of embedded devices connected to the Internet using IP is related or may be a part of but is not yet a full realization of the IoT. A backend-information structure is missing here which permits discovering and provisioning the resources (i.e. data) offered by the embedded devices. Moreover, apart from security and privacy constraints and as usual for most of the regular nodes on the Internet (e.g. servers), accessing the provided resources and services should be possible for other participants and not limited to a small group (e.g. a telecommunication provider).

This results in the assumption that sensors, embedded devices or things in general are resources that provide knowledge (i.e. data) about an entities environment and are able to move in space (i.e. either mobile or stationary things moved by users). The data should be accessible by each other entity that is interested in or may benefit from them. Basically, as shown in Figure 1.1, two options to target this data dissemination problem exist. Both are likely to coexist depending on the application domain. Looking at an example from the E-Health application domain yields a deeper understanding of these two options, which can be characterized as provisioning the data and provisioning the data sources. In E-Health, and in particular in telemedicine applications, sensors are used to monitor the vital signs of a patient. Sensors are usually integrated using gateway devices (e.g. smart phones) and the data collected are forwarded to the clinical information system of the medical facility responsible for the patient's treatment. Nowadays, especially in case of emergencies, several medical facilities can be involved in the treatment process. In order to get a meaningful survey of the patient's condition, each participant needs access to the data collected from the medical sensors. Provisioning the data refers to the concept of Electronic Health Record (EHR) Clouds, where clinical information systems of different medical facilities are linked in order to exchange data on the basis of a-priori defined contracts. This is an important approach, because it allows exchanging data about the history of a patient. However, considering the delay introduced by exchanging real-time data through EHR Clouds, interoperability issues and the static nature of how participants are interlinked (resulting from privacy constraints), provisioning the data sources (i.e. the medical sensors) becomes reasonable. Instead of exchanging the data by linking the clinical information systems, each clinical information system directly accesses the medical sensors a patient is equipped with. Therefore, the medical sensors (i.e. the physical things) must be shared by the participants.

This notion of sharing embedded devices among interested users, is in line with several

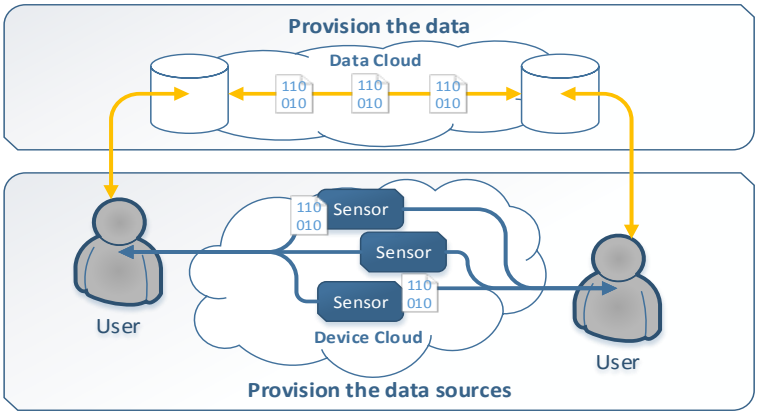


Figure 1.1.: Two basic approaches for data dissemination in IoT applications - Sharing and provisioning the data or the data sources

well established and upcoming paradigms like on-demand resource provisioning and Pay-As-You-Go (PAYG) pricing models known from Cloud Computing [11] or the Sharing Economy [134], which became popular due to car sharing services for instance. However, sharing embedded devices in IoT applications not only targets efficiency and resource utilization, it also enables new applications, like sketched above, by giving users the possibility to share the access to physical resources (i.e. embedded devices) that move in space. Throughout this thesis, I will present a generic architecture approach for sharing embedded devices in various IoT application domains. General issues like interoperability, device integration and abstraction, as well as security and access rights management will be discussed and targeted by the specification of respective backend and middleware components. Proof of concept will be given by applying the generic architecture to a specific use case from the E-Health domain and by a discussion of related usage scenarios.

1.1. Problem Definition

Sharing physical devices like sensors or actuators leads to a paradigm shift in how IoT related applications can be designed. Paradigms like Infrastructure as a Service (IaaS), On-Demand Resource Provisioning or PAYG pricing models became very popular along with the proliferation of Cloud Computing and its applications. The success of Cloud Computing was heavily influenced by these paradigms, because they basically constitute and enable what we understand when referring to the term Cloud: An endless set of resources that we can access and utilize whenever and wherever we want, without having to think about management, availability, utilization or long term contracts.

However, looking at IoT-related applications, a completely different picture of how these applications are designed and work can be observed. Nowadays, IoT applications are often built upon a gateway based approach. This can be briefly described as a single system (e.g. a router or a smart phone) that integrates available sensors, collects data from them and forwards the resulting data streams to application-layer components (e.g. a compute-center hosting data analysis applications). Usually these gateways are designed as closed boxes dedicated to a particular application domain and barely interact with each other (e.g. one gateway integrating Smart Home devices, one gateway integrating E-Health sensors). Due to the variety of communication protocols, devices and sensors are often statically integrated and connected to the gateways and access is limited to a small group of users having control over the gateway. Based on the estimation that the integration of IoT applications, sensors, and smart devices into our everyday lives will proceed further, the amount of resources provided by these sensors and devices will continuously grow. But, in contradiction to Cloud Computing applications, currently no concepts exist for the provisioning of these resources. Therefore, the Device Cloud approach aims at broadening our understanding of the term Cloud. By introducing concepts for the provisioning of sensors and embedded devices on a PAYG basis, the Cloud turns from an endless remote resource to an overall resource surrounding us constantly.

According to the discussion of IoT in the previous section, a lot of technical components, each inducing its own challenges, are required to set up the Device Cloud. Challenges like device integration and abstraction, device deployment and management, resource provisioning, security and in particular access management, or interoperability in general, are well known and partially well investigated regarding certain research domains (e.g. resource management and provisioning in the area of Cloud Computing, or device integration for certain application domains like E-Health or Smart Homes). However, the amount of research effort currently dedicated to the area of IoT, both in terms of general understanding and architecture as well as applications, emphasises that these challenges need to be faced with respect to the requirements of IoT applications. Moreover, new challenges arise out of the adoption and integration of technical solutions originally developed for other application domains. Figure 1.2 gives a brief overview of the challenges to be tackled and how they are related to each other. Throughout the following sections, I will use the general term *device* when referring to any kind of sensor, actuator or embedded system that is an element of the sharing and provisioning process targeted by the Device Cloud.

Based on the ubiquity and mobility of devices and the assumption that gateways dedicated to a particular application domain are not feasible for the Device Cloud, device integration is a twofold challenge. Basically, device integration involves establishing a connection between a device and a gateway that is able to forward and/or (pre)process the resulting data stream (e.g. aggregation). This abstract view introduces two major sources of heterogeneity. First, the huge variety of devices in terms of communication

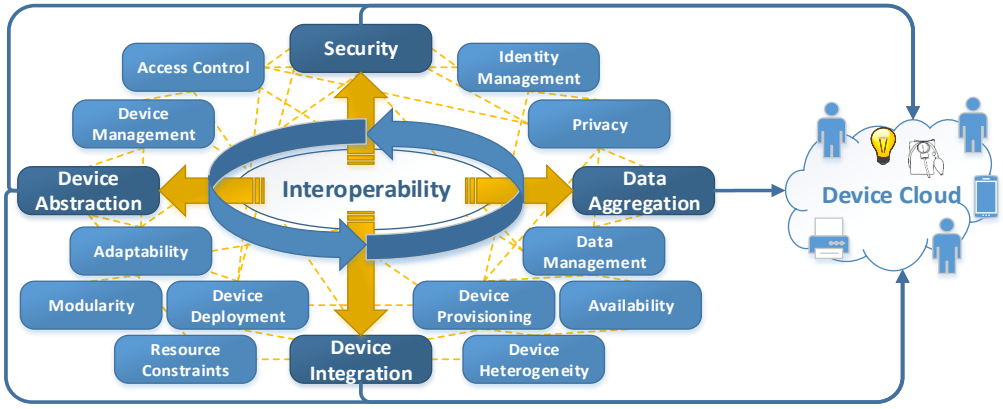


Figure 1.2.: Overview of the main Device Cloud challenges and their relationships

protocols, data formats or, nomenclatures needs to be considered. Second, the gateway devices themselves can differ regarding their specific characteristics (e.g. operating system, offered communication interfaces, available resources) [16] and usually underlie resource constraints. If we assume that devices are provisioned like Cloud resources, integrating them should not be limited to a specific subset of gateway systems. Rather, gateways used within the Device Cloud should be designed as general purpose device integration systems not dedicated to a certain application domain. However, it appears unrealistic to assume, that all software components required for device integration (e.g. discovery modules, protocol and device drivers) can be packaged into one static middleware in a way, that sufficient coverage for even one application domain is given and the middleware will be deployable on the majority of the gateway platforms. Therefore, the Device Cloud requires a modular device integration middleware, which is deployable on a multitude of available gateway platforms and is linked to backend information systems in order to adapt itself to the requirements of the environment (i.e. autonomously reconfigure and reload software modules to handle the discovered devices).

Apart from integration, device abstraction is an important issue [95]. Exposing the capabilities of a device in a vendor-neutral way reduces the risk of a vendor-lock-in and allows for more modularity and openness of the Device Cloud. Replacement of a device should not imply redesign of application components. Introducing a layer of abstraction that provides the capabilities of a certain class of devices in an interface driven way, allows application developers to focus on the application and not on device specific protocol or controller logic. Besides application development, abstraction can also simplify the process of device driver development and increase platform independence. Assuming that

base drivers, responsible for integrating low level protocols such as Bluetooth, ZigBee or, Ethernet, provide their functionalities through well defined interfaces, device specific drivers can be developed on top of them in a platform independent way. This is an important requirement because of the heterogeneity of gateway platforms.

The device integration, abstraction and data aggregation challenges are related to achieving interoperability among the participating entities of the Device Cloud. Standardization is often promoted to be a mitigation to this problem. Although appropriate standards for certain application domains exist (e.g. IEEE 11073 for E-Health or KNX for home automation), one cannot expect that a widespread standardization including all relevant application domains will be achieved in a reasonable time span [29]. New technologies usually introduce new standards. A lot of standards allow for vendor defined extensions in order to keep up with the rapidly evolving market. This often leads to proprietary solutions. Device vendors gain flexibility in handling specific hardware requirements, protecting innovations or optimizing their products towards their design preferences. Additionally, market exclusivity can be achieved, which often forces customers to be dependent on a vendor (i.e. vendor-lock-in). Instead of relying on the assertion of a small set of standards that can cover a comprehensive set of devices, the Device Cloud will focus on a uniform framework and on interoperability enablers like device abstraction and data transformation. Given the modularity of the middleware and the respective backend information systems, a multitude of protocols and data formats can coexist within the Device Cloud.

Besides interoperability and integration issues, major challenges arise from the federation of the Device Cloud. It cannot be assumed that all devices shared among the participants are owned by a single authority. Rather, according to what IoT is about, the Device Cloud will heavily rely on peer to peer collaboration between its participants. Each participant can contribute devices to the Device Cloud, regardless of whether we are talking about a single individual owning less than five devices or a company owning more than thousand devices. Thus, the traditional boundaries between service providers and consumers as observed when referring, for instance, to Cloud Computing, become indistinct. Each participant can take both roles: being a service consumer and provider at the same time. Due to the need of cooperation between participants, security and privacy, especially in terms of identity and access management, are important challenges, too. Sharing devices among participants requires secure granting and withdrawal of access tokens. Having a look at the E-Health domain again, emergency scenarios require reliable mechanisms for taking temporary ownership of a medical device.

In summary, implementing a generic and application domain independent Device Cloud requires a modular architecture tackling the main challenges device integration, device abstraction, interoperability, federation, and security and privacy. It is important that

each challenge is not only considered on its own, but rather put in context to the requirements of the overall approach.

1.2. Contribution

The main contribution of this thesis is given by the specification of a generic Device Cloud architecture, which is not limited to a pre-defined set of devices or application domains. Rather, emphasis will be put on the capability to introduce new devices, protocols, or data formats on demand without having to change parts of the overall architecture. First, participating entities and their properties will be identified, followed by a specification of state models and roles the entities can take within the Device Cloud. Based on the system requirements, necessary technical components as well as their relation to each other will be discussed. Specific implementations of the major technical Device Cloud components, which include a device integration and data aggregation middleware and a backend information system, will be introduced.

In general, the thesis will show the applicability of Cloud Computing paradigms to the IoT domain by enabling the provisioning of (mobile) embedded devices on a peer to peer basis among the participants of the Device Cloud. A modular and adaptive system design will demonstrate that the heterogeneity challenge given by the variety of devices can be tackled without limiting the solution to a certain application domain.

Parts of this thesis have been published in the following publications:

1. Andreas Kliem et al. “Towards self-organization of networked medical devices”. In: *Emerging Technologies Factory Automation (ETFA), 2011 IEEE 16th Conference on*. 2011, pp. 1 –8. DOI: 10.1109/ETFA.2011.6059230

2. Andreas Kliem, Matthias Hovestadt, and Odej Kao. “Security and Communication Architecture for Networked Medical Devices in Mobility-Aware eHealth Environments”. In: *Mobile Services (MS), 2012 IEEE First International Conference on*. 2012, pp. 112–114. DOI: 10.1109/MobServ.2012.15

3. Dimo Ivanov, Andreas Kliem, and Odej Kao. “Transformation Middleware for Heterogeneous Healthcare Data in Mobile E-health Environments”. In: *Proceedings of the 2013 IEEE Second International Conference on Mobile Services*. MS ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 39–46. ISBN: 978-0-7695-5029-9. DOI: 10.1109/MS.2013.15. URL: http://dx.doi.org/10.1109/MS.2013.15

4. Andreas Kliem and Odej Kao. "CoSeMed - cooperative and secure medical device cloud". In: *e-Health Networking, Applications Services (Healthcom), 2013 IEEE 15th International Conference on*. 2013, pp. 260–264. DOI: 10.1109/HealthCom.2013.6720678

5. Andreas Kliem. "CoSeMed: Cooperative and Secure Medical Device Sharing". In: *Cloud Computing Applications for Quality Health Care Delivery*. Ed. by Anastasius Moumtzoglou and Anastasia N. Kastania. Hershey, PA, US: IGI Global, 2014, pp. 201–227. DOI: 10.4018/978-1-4666-6118-9.ch011

6. Andreas Kliem et al. "Self-adaptive Middleware for ubiquitous Medical Device Integration". In: *e-Health Networking, Applications Services (Healthcom), 2014 IEEE 16th International Conference on*. 2014, to appear

7. Andreas Kliem et al. "The Device Driver Engine - Cloud enabled Ubiquitous Device Integration". In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE 10th International Conference on*. 2015, to appear

8. Andreas Kliem and Thomas Renner. "Towards On-Demand Resource Provisioning for IoT Environments". In: *7th Asian Conference on Intelligent Information and Database Systems (ACIIDS), Special Session on Internet of Things, Big Data and Cloud Computing*. 2015, to appear

1.3. Outline of the Thesis

This thesis is structured as follows:

Chapter 2: Background

Chapter 2 introduces basic relevant technologies like Cloud Computing, Mobile Grid Computing, or IoT and Future Internet architectures, and provides a theoretical foundation of the major challenges towards realizing the Device Cloud. Terminologies, definitions, and building blocks, like the OSGi specification, are introduced in order to provide a comprehensive background underlying the Device Cloud architecture.

Chapter 3: Related Work

Chapter 3 provides a review of related research conducted in the areas of Sensor-Cloud integration, IoT architectures, and IoT applications like E-Health or Smart Homes.

Chapter 4: Device Cloud – Overall Concept

Based on a presentation of possible Device Cloud application scenarios, the system requirements are discussed. As a result, entities participating in the Device Cloud and major technical components as well as their properties and relations to each other are specified.

Chapter 5: Device Cloud – Security & Interactions Concept

Chapter 5 will derive a security and interaction model from the defined actors and entities. Emphasis will be put on modelling the state of devices, defining provisioning interactions as well as on authentication and authorization.

Chapter 6: Device Cloud – Architecture

Based on the concepts and the broad definition of technical components presented in Chapter 4 and Chapter 5, a concrete architecture regarding the main building blocks of the Device Cloud is introduced. This includes the device integration and data aggregation middleware and the backend information system.

Chapter 7: E-Health Application Scenario

Chapter 7 will apply the Device Cloud concept to an E-Health application scenario. Based on a set of specific medical devices (e.g. blood pressure sensors), it is demonstrated how the Device Cloud can be deployed and how the constraints of the application domain (e.g. the requirement to align all data streams to a uniform, standardized data format) can be satisfied without changing the generic, application domain independent Device Cloud architecture.

Chapter 8: Conclusion and Future Work

Finally, chapter 8 concludes the thesis and gives an outlook towards future work.

2. Background & Foundations

Contents

Based on the motivation and problem definition, this chapter aims at providing an introduction to concepts that constitute the foundation of the Device Cloud approach. General related technologies and paradigms that deal with connecting devices, data and users over information networks will be discussed within the context of the Device Cloud. Furthermore, specific technologies, like for instance OSGi, required to implement mandatory Device Cloud components and capabilities will be introduced.

2.1. Related Technologies Overview

As already outlined in Chapter 1, the Device Cloud in particular and IoT in general have a strong relationship to and partially overlap with other technologies and paradigms known from the distributed systems and computing domain. In the following, some of these technologies and paradigms, which contribute foundations necessary to set up the Device Cloud approach, will be introduced and put into a contextual relationship.

Cloud Computing

Cloud Computing allows consuming large amounts of resources, like computing, storage, or, software, over the Internet without the need for long term contracts. Users do not have to be aware of the actual physical location or the physical system that provides the resources. Therefore, Cloud Computing is based on virtual infrastructures that allow sharing physical infrastructures among several users, which leads to a high degree of resource utilization and reduces costs. According to the National Institute of Standards and Technology (NIST) [109], one of the essential characteristics of Cloud Computing is the ability to rapidly provision and release configurable computing resources from a shared pool with a minimal management overhead. Characteristics and paradigms like on-demand, shared resource pools or, elasticity, which will be discussed in more detail in Section 2.2, build the foundation of what we understand by the term Cloud and also distinguish it from related concepts such as Grid Computing [57] [21].

Compared to Cloud Computing, the IoT domain suffers from the huge heterogeneity of resources. The resources are far from being organized in a shared pool from which they can be provisioned to the users. However, since the amount of devices and accordingly the amount of resources to be managed and provisioned will continuously grow [96], the demand for pooling these resources and reducing management overhead is well motivated. Therefore, the Device Cloud approach strives to apply the essential Cloud Computing characteristics to the IoT domain and allows for on-demand access to devices.

Cyber Physical Systems

The term Cyber Physical Systems (CPS) stands for the integration of computation with physical processes [97]. A CPS develops if embedded systems interact using a communication infrastructure and therefore lead to environments where computational resources (e.g. software components) and physical resources or processes (e.g. actuators) interact with and affect each other. Typical application domains are automotive systems, avionics, connected medical devices, assisted living or automation [130]. The main design challenge for CPS is the complexity that arises out of the interaction with physical processes that are not predictable. This requires these systems to be built adaptable and robust against unexpected failures and states. Moreover, embedded systems operating in devices we use every day (e.g. TVs or kitchen equipment) are expected to be more reliable and robust than regular computers. Hence, the additional complexity introduced by the Device Cloud vision of pooling, provisioning and sharing embedded systems (i.e. devices) may not reduce the reliability of the shared device itself and therefore needs to be designed robust and adaptable.

Future Internet Architectures

Future Internet Architectures is a generic term for several research projects and initiatives, like FI-PPP [55], FIRE, FI-WARE or, FI-STAR [5] [153], that investigate in the improvement or redesign of the aging IP based infrastructure in order to cope with challenges like ubiquitous network access, mobility, or integrated security, that were not foreseen during the initial development of the Internet [61]. It is assumed, that the increasing amount of users and the demand for future applications require a paradigm shift from machine-centered and packet delivery based infrastructures towards data, content and, user-centered ones [121]. Nowadays, several hundred additional protocols on top of IP exist in order to provide the infrastructure necessary for delivery of the services we use today. This leads to an increasing complexity that makes the management of the Internet infrastructure more and more difficult. Especially the mobility challenge, emerged due to the shift from stationary, PC-based computing to mobile computing, is a driving force for Future Internet Architectures and is related to a major Device Cloud challenge as well. Efficient service delivery requires to hide and cope with the heterogeneity introduced by the different networks (e.g. IP, cellular, sensor networks, wireless) and technical standards common to the domain of mobile computing.

Internet of Everything

Internet of Everything (IoE) is a term created by CISCO [52] that evolved out of the Internet of Things (IoT) [23]. It is assumed, that after the second wave of internet growth, which basically refers to the dissemination of connected mobile devices and lead to IoT, a third wave will also connect people and data to the Internet and lead to IoE. According to CISCO, IoE will integrate people, data, things and processes, that manage the way people, data and things work together. Compared to IoT, which is based on things, IoE is based on people, data, processes and things. By explicitly putting emphasis on the data, IoE brings attention to the importance of security and privacy and protection of personal information. Although the Device Cloud approach is about sharing the devices instead of the data, it has to be ensured, that users that access the shared device pool can establish secure links to the devices they allocate and therefore protect personal information generated by things they do not own.

Machine-to-Machine Communication

Machine-to-Machine (M2M) describes the exchange of information between devices like machines, cars, sensors or, actuators usually performed in an automated manner and without human interaction [165] [24]. Thus, M2M is often referred to as the building

block of IoT, because the virtual representations of things made available by IoT can also be described as the service endpoints to an M2M system [32]. Hence, the application domains of M2M systems or platforms are comparable to the ones mentioned in the context of IoT. Examples are home automation, smart grids, object tracking or, factory automation [146] [112].

M2M has a high relevance to the Device Cloud approach, since it deals with similar challenges like heterogeneity of devices and communication networks, device manageability or scalability in general that altogether lead to the overall problem of device integration. Especially the huge amount of devices to be managed by M2M platforms has recently led to solutions called M2M Clouds or Machine-to-Cloud (M2C). These approaches basically describe the utilization of Cloud services to expose the virtual representation of machines (i.e. things) more efficiently and to enable for large scale management of devices. Especially the term M2M Cloud, as often used by companies to promote their device integration and management platforms, is related to the concept of sensor virtualization discussed in Section 2.2.1. However, some M2C approaches also target the utilization of Cloud resources for offloading of computational tasks, which is related to the integration of sensor networks and Clouds [70].

Mobile Grid and Mobile Cloud Computing

Mobile computing evolved out of the dissemination of small, mobile and wirelessly connected devices like smart phones that offer computing capabilities. The term Mobile Grid covers both, the demand for users with mobile devices to access resources offered by the Grid and the utilization and integration of the resources offered by the mobile devices themselves. Thus, the Mobile Grid can be defined as an extension to the regular Grid [58] providing capabilities to support mobile users and resources in a seamless, transparent, and efficient way [102]. In contrast to the Mobile Grid that explicitly takes the resources offered by the mobile devices into account, Mobile Cloud Computing is focused on the mitigation of performance constraints of mobile devices by offloading data storage and data processing to Cloud infrastructures [42] [71]. Mobile Cloud Computing therefore rather targets accessing Cloud infrastructures with mobile devices than provisioning the resources offered by them. Hence, one could argue that the Device Cloud approach is more related to Mobile Grid Computing. The difference is that not the resources of a mobile device are provisioned, but rather the physical device itself.

Sensor and Actor Networks

A sensor network, often referred to as WSN, is a set of small computational nodes connected through a communication network, that collect and forward information about a

certain environment [127]. WSNs became popular along with applications like weather prediction or monitoring of large environmental areas without existing information acquisition infrastructures or infrastructures damaged because of disasters (e.g. earthquake or forest fire). WSNs can be based on stationary or mobile nodes and can be organized using base stations or in an ad-hoc manner (i.e. Mobile Ad-hoc Network (MANET)), which has a high impact on the underlying access and routing protocols. Due to the huge amount of interacting nodes and the unreliable communication infrastructure, WSNs often rely on protocols that offer self-organization capabilities [143]. Typically, WSNs are based on Mesh-Networks, which means that the data is forwarded from node to node in a multi-hop manner until the destination is reached. According to Poslad [126], the main components of a WSN are sensor and sensor access nodes. Similar to a gateway, the sensor access node connects external entities to the sensor network. An extension to the concept of WSNs is given by Wireless Sensor and Actor Networks (WSANs), which integrate actors in order to perform actions based on the observations made by the sensor nodes [4] [131]. WSANs are more challenging than WSNs, because the presence of actor nodes introduces coordination and communication problems. Events that trigger an actor usually need to be synchronized, coordinated and, delivered in proper ordering and under real-time constraints, which is difficult to achieve in ad-hoc networks.

Apart from the technical details of WSN and WSAN infrastructures, accessing the collected data is a key issue. Similar to M2M Clouds, approaches for Sensor-Cloud integration are discussed to simplify the management of and access to the sensors, to hide the technical details of the WSN infrastructure and to extend the limited capabilities regarding storage and computing resources [67] [7]. Some of these approaches allow providing sensor access to several users based on Cloud services and therefore have a high relevance to the Device Cloud approach. Section 2.2.1 will further discuss these approaches.

Ubiquitous Computing

Ubiquitous Computing [163] [126], sometimes also referred to as Pervasive Computing, acts as a foundation for several of the discussed paradigms (e.g. M2M, IoE, IoT). Ubiquitous computing was already discussed in Chapter 1. Briefly, it targets the ubiquitousness of computer-aided information processing. It is based on the assumption that after the age of mainframes and the age of Personal Computers (PCs), where each user was assigned to a computer, a third age motivated by the proliferation of embedded systems will constitute by assigning each user a set of networked computers embedded into the environment. Due to this decentralized notion of computing, the management and integration of the corresponding nodes becomes much more complex than in centralized, PC based environments. This is one of the challenges the Device Cloud approach

tries to tackle, by applying the Cloud Computing paradigms that allow for easy and homogeneous access to heterogeneous and geographically distributed resources, to the Ubiquitous Computing domain.

2.2. Cloud Computing Fundamentals

Cloud Computing, as already introduced in Section 2.1, is the commercial reality of a concept called Utility Computing [11]. Utility Computing [122] refers to techniques and business models, where a service provider offers computing resources and infrastructure management to its customers, usually on-demand. According to the NIST Cloud definition [109], the model behind Cloud Computing consists of deployment models, service models and five essential characteristics. The deployment model refers to the set of users that may access the Cloud. Private Clouds, for instance, are only provisioned to a single organization while Public clouds like Amazon EC2 [9] are accessible by the general public. Service models refer to the well known paradigms Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and, Software as a Service (SaaS). The essential characteristics, which the Device Cloud aims to apply to the IoT domain can be described as follows:

Cloud characteristics as defined by the NIST

On-demand self-service:
A consumer is enabled to allocate and access computing resources (e.g. server time or network storage) as required without the need for human interaction with each service provider.

Broad network access:
Computing resources are made available over a network and standard mechanisms allow accessing these resources using heterogeneous client devices (e.g. PCs, smart phones).

Resource pooling:
Providers pool and dynamically provision computing resources to multiple consumers, which allows for better utilization. A key driver for resource pooling is virtualization. Pooling implies that the consumer usually has no knowledge about the location of a provided resource.

Rapid elasticity:
The illusion of an endless resource, that can be accessed from anywhere at any time, is created by the Cloud. Resources can be elastically provisioned and released on-demand to achieve a high degree of scalability.

Measured service:
The resource consumption can be monitored in order to provide transparency for

> Cloud service providers and consumers. Some Cloud services may automatically adapt and optimize the resource consumption to fit the current requirements.

An important paradigm or business model not explicitly mentioned but resulting from these characteristics is Pay-As-You-Go (PAYG). Based on monitoring the resource consumption and the ability to elastically release resources if they are not required any more, PAYG enables the consumer to pay only for the resources actually utilized without the need for long term contracts. Another important characteristic, already part of the Utility Computing definition, is the infrastructure management that is implicitly provided to the consumers alongside with the computing resources. These characteristics and paradigms address the major obstacles currently found in the IoT domain: Provisioning resources on-demand through standard mechanisms (i.e. in a homogeneous fashion) and hiding the complex task of infrastructure management.

2.2.1. Sensor Networks & Cloud Integration

Wireless Sensor Networks (WSNs), as introduced in Section 2.1, can consist of several up to thousands of nodes. Usually they underlie serious resource constraints and are often designed towards the specific requirements of the application domain. Sensor access nodes, used to connect external entities to the WSN, hide the complexity of the protocol stack required for (ad-hoc) networking, communication and data aggregation. In contrast to Cloud Computing, consumers usually have to be aware of the actual location, the resource constraints, and the infrastructure management requirements of the sensor nodes. This often limits the set of consumers being able to access the WSN. As a remedy, concepts like Sensor-Cloud integration [168] [7] or Sensor-Virtualization were introduced [6]. These approaches basically aim at overcoming the resource constraints of traditional WSN by integrating Cloud resources and providing access to the physical sensors to multiple users. The latter one is related to the concept of M2M Clouds or M2C by allowing to simplify the management of and access to the sensors using standard Cloud interfaces and applications that hide the actual location and the diversity of the sensors from the consumers (e.g. abstraction from technical details, focus on the semantics, grouping of functional identical sensors as shown in Figure 2.1) [2]. In contrast to M2C solutions, sensor-cloud infrastructures usually follow the public cloud deployment model and do not limit access to a certain company or group of consumers. The term sensor-virtualization is somehow ambiguous with the term Virtual Sensor Network (VSN), but has to be clearly distinguished. VSNs, which will be introduced in the following section, deal with resource sharing, logical grouping and, collaboration of physical sensors nodes, while sensor virtualization is about provisioning the virtual representation (i.e. the Cloud interface) of a physical sensor node. Provisioning or sharing the virtual representation

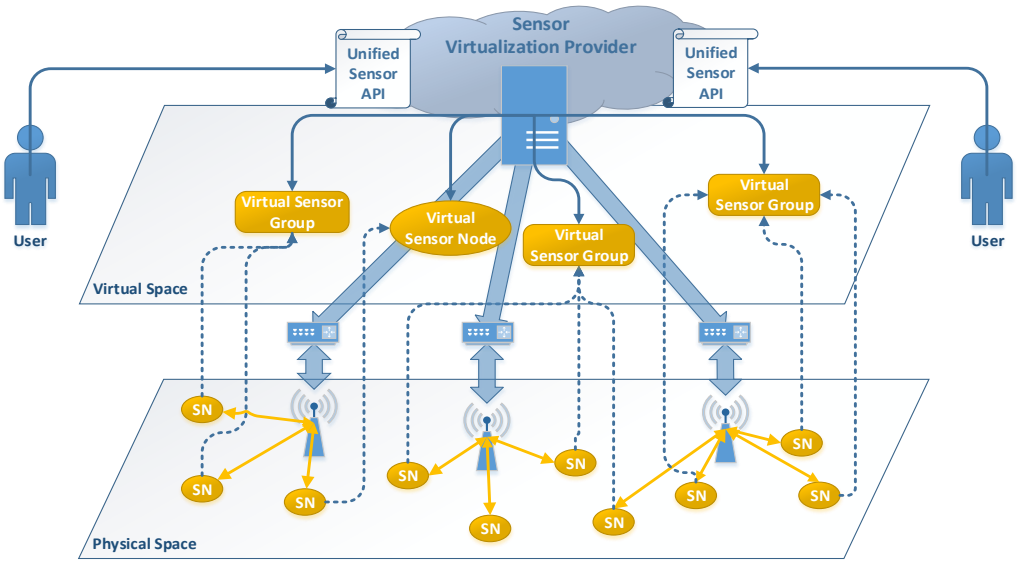


Figure 2.1.: Sensor virtualization as an enabler for unified access to heterogeneous physical resources.

presumes that the consumer trusts in the nodes participating in the data transmission chain between the physical sensor node and the Cloud interface. Section 4.1 discusses the differences between the Device Cloud approach and Sensor Virtualization or Sensor-Cloud infrastructures, respectively.

2.2.2. Cyber-Physical Clouds & Virtual Sensor Networks

Motivated by the increasing capabilities of the nodes participating in WSNs, Cloud characteristics like pooling and allocating a subset of the resources to certain tasks on-demand are applied to sensor networks. Virtual Sensor Networks (VSNs) assume, that the overall set of nodes in a WSN can be virtually grouped into subsets and dedicated to specific applications or tasks [81] [74]. Unlike traditional WSNs, where usually all nodes perform similar tasks, a subset of nodes can be allocated for a given time period to perform specific tasks or react to the current environment. A sensor network deployed in an area recovering from a disaster may, for instance, observe specific events in a certain region. Using VSN, nodes close to that region could be grouped and further investigate the situation by executing specific tasks.

Another related concept for further investigating the possibility of deploying virtual-

ization techniques to sensor nodes is called Cyber-Physical Cloud Computing [38] [87]. By applying lightweight virtualization capabilities to sensor nodes, sensors basically act as servers that move in space and execute virtual sensors. Virtual sensors can migrate between physical ones, which is referred to as cyber-mobility (i.e. moving between sensor hosts). Additionally, virtual devices can move with their current sensor host (if the sensor node is mobile), which is referred to as physical mobility. Similar to regular Cloud Computing, this allows for efficient resource utilization because virtualization allows isolating virtual sensors possibly belonging to different consumers.

2.3. Device Integration, Management & Abstraction

The integration of sensors, actuators or in general (mobile) embedded devices involves several steps like discovery, connection establishment, data transmission, data aggregation and processing as well as storage or visualization. All these steps in turn involve several hardware and software components and usually depend on protocols spanning all layers of the OSI Reference Model [172]. In general, device integration in context of IoT or M2M applications is the challenge of achieving interoperability among a heterogeneous set of interacting systems. More precisely, according to Kindberg et al. [86], two principles need to be considered when developing systems that deal with ubiquitous or pervasive devices: physical integration and the spontaneous interoperability arising from the integration of nomadic physical devices. The following sections will discuss some of the ancillary challenges.

2.3.1. The Standardization Problem

Integrating devices and exchanging information requires some kind of communication, where both the sender and the receiver need to agree on a common language (usually based on a canonical data representation). A standard, which basically can be defined as a set of rules that provide the necessary basis for interaction between two systems [164], can be used as a common language. The standardization problem is a decision problem that deals with choosing the appropriate set of communication standards for an application. Especially regarding the IoT domain, it is difficult to make this decision because a huge variety of competing standards exist. The dissemination of a standard usually depends on the number of authorities and systems using it, which is difficult to estimate in advance [29]. Moreover, standards evolve, sometimes allow for vendor-defined extensions and there are even a lot of entire proprietary solutions. Thus, it is unlikely that the huge set of communication technologies and standards required to achieve interoperability will converge in a way that sufficient coverage regarding device integration can be gained by implementing a set of major standards. Rather, device

integration solutions need to be designed flexibly in terms of adaptability towards new or evolving standards. Modularization and generic data representations, which allow clients to request information in a desired format while hiding the complexity of the heterogeneous landscape of communication technologies, are key factors for sustainable device integration solutions.

The following section gives a subtotal collection of standards and technologies related to the topic of device integration. Since different OSI layers are covered, no classification is given and the standards are only introduced briefly.

List of standards

6LoWPAN:
6LoWPAN is an acronym for IPv6 over Low power Wireless Personal Area Networks. It is based on the IEEE 802.15.4 standard for Wireless Personal Area Networks (WPANs) and was designed in order to enable small, low power devices to use the Internet Protocol. 6LoWPAN can be used for any kind of embedded devices that require wireless internet connectivity at low data rates while having constraints regarding available resources and energy consumption. 6LoWPAN basically defines mechanisms, like header compression, that allow IPv6 packets to be delivered over a IEEE 802.15.4 network. Since 6LoWPAN operates on the network layer, an application layer protocol like CoAP is required to achieve interoperability among communicating devices. [140] [94]

ANT:
ANT is an open access protocol stack for short range, low power wireless sensor networks developed and managed by ANT wireless, a division of Dynastream. ANT was originally dedicated to sports, in particular devices embedded in personal fitness equipment, but it also has applications in health, home automation or the industrial sector. In order to provide interoperability among the devices of different vendors, ANT+ can be added to the base ANT protocol suite. ANT+ basically is a collection of application profiles that describe how certain devices such as heart rate monitors have to be designed in terms of the data exchanged using ANT. [45]

Bluetooth:
Bluetooth is a protocol stack for WPANs developed by the Bluetooth Special Interest Group (SIG) and is also related to the IEEE 802.15 WPAN working group. It allows for connectionless and connection-oriented communication in ad-hoc or piconets. Bluetooth consists of a suite of layered protocols and application profiles, which, similar to ANT+, define the communication behaviour a device has to comply in order to be interoperable with other Bluetooth devices. Bluetooth became popular as a replacement for wired connections especially in the area of entertainment and input devices, but also supports applications in the area of home automation, E-Health or, industry. Since version 4.0, Bluetooth introduced a low energy protocol stack, which allows significantly reducing the energy consumption of Bluetooth devices. Due to the dissemination of personal

health devices, Bluetooth adopted the IEEE 11073 standard for its Health Device Profile (HDP). [22]

CoAP:
As the name suggests, the Constrained Application Protocol (CoAP) is an application level protocol dedicated to resource constrained devices and M2M or IoT applications. It was designed according to RESTful principles [56] in order to be easily mapped to HTTP while keeping focus on low overhead and simplicity. In contrast to web service approaches like DPWS, CoAP uses a binary header format. Similar to DPWS, it can operate on top of 6LoWPAN. Standardization of CoAP is managed by the Internet Engineering Task Force (IETF). [139]

DPWS:
The Device Profile for Web Services (DPWS) is an OASIS standard to provide web services for resource constraint devices. DPWS combines existing WS-* standards like WS-Addressing or WS-Security and defines additional mechanisms required for dynamic discovery and event-based communication. DPWS was originally developed by Microsoft as an alternative to UPnP and therefore is available in current Windows operating systems. Micro implementations that can operate on highly resource constrained devices using 6LoWPAN are available. [114] [98]

EnOcean:
EnOcean is a German company that introduced a correspondent wireless technology covering the OSI layers one to three. The technology is based on energy harvesting and therefore allows for batteryless sensors and actuators like light switches. Common application domains are smart homes, building automation or logistics. In order to achieve interoperability between products of OEM partners and push standardization, the non-profit EnOcean Alliance was founded in 2008. Similar to the Blueetooth application profiles, the EnOcean Alliance governs application level protocols (EnOcean Equipment Profiles) to ensure interoperability. The layer one to three technology was ratified as an international standard in 2012 (ISO/IEC 14543-3-10). [48]

IEEE 1451:
IEEE 1451 is a collection of standards that aims at providing uniform interfaces to interact with sensors or actuators (transducer) independently of the underlying communication technology. The core element of the standard collection is the definition of the Transducer Electronic Data Sheet (TEDS), which contains the information required by a measurement system to interact with a transducer (e.g. ID, calibration and correction data) and therefore allows integrating the transducer in a plug and play manner. The TEDS can be stored within the memory of the transducer or as a separate file downloadable from the internet if legacy transducers without memory need to be integrated. Compared to other device discovery and description technologies like UPnP, TEDS is more suitable for highly resource constrained devices. [76]

ISO/IEEE 11073:
The ISO/IEEE 11073 family of standards is dedicated to interoperable communication of medical devices [75]. 11073 will be further discussed in Chapter 7.

OMA DM:
OMA Device Management is a protocol for device management, sometimes also referred to as Mobile Device Management (MDM), which will be discussed in Section 2.3.3. OMA DM is based on a transport independent request-reply protocol and uses XML based messages. An OMA DM server can manage several clients (i.e. devices), which, for instance, includes fault management, device configuration or software upgrades. OMA DM is governed by the Open Mobile Alliance (OMA). [116]

UPnP:
Universal Plug and Play (UPnP) is a set of IP based application layer protocols that allow for vendor-independent device discovery and interaction. Using the Simple Service Discovery Protocol (SSDP), control points can discover devices and request a XML based description. The description contains general information like vendor or serial number and describes a set of actions and data types for each service offered by a device. UPnP is mostly used in residential networks and entertainment applications. It is governed by the UPnP Forum. [82]

USB:
Universal Serial Bus (USB) is a wired serial communication bus mostly used to connect external devices to PCs. Nowadays USB is the commonly used technology to connect devices like mass storage, printers, web cams or input devices in general. In order to reduce the complexity of device integration, the USB standard defines device classes that allow using generic device drivers with devices from different vendors. [15]

ZigBee:
Similar to 6LoWPAN, ZigBee is a wireless communication technology based on IEEE 802.15.4 with applications in smart homes, automation or, sensor networks in general. Like Bluetooth, ZigBee defines Clusters and Profiles to achieve interoperability among devices from different vendors. A cluster operates according to the client/server principle. The server (e.g. an OnOff cluster in case of a light device) has a set of attributes that can be altered by a client using specific commands. Clusters and devices that implement certain clusters are grouped into profiles dedicated to an application domain (e.g. building automation). [171]

Besides the listed standards and specifications, further ones like FIPA Device Ontology [59], KNX [110], SNMP [144], TR-069 [25] or, Z-Wave [63] exist. However, all these standards just represent a subset of the technologies common for IoT or M2M applications. Thus, the importance of a modular, adaptive and technology independent device integration solution is underlined.

2.3.2. Device Integration

Device integration can be described as the process of identifying and loading appropriate control logic (e.g. a driver) for a device that is connected to a (usually running) system. Connected means, that the device is plugged into the system (e.g. in case of USB)

or appears on a communication network the system is connected to. According to the M2M Alliance [106], basically three major components can be defined when talking about device integration:

M2M System Components

Data End Point (DEP):
The DEP (i.e. sender) refers to a device that needs to be integrated. The DEP collects data from the device or waits for control commands. The DEP usually refers to an embedded microprocessor linked to the device.

Data Integration Point (DIP):
The DIP (i.e. receiver) refers to a system that hosts some kind of monitoring or control application or acts as a data aggregator. It collects data from connected DEPs or sends control commands to them.

Machine Communication Network (MCN):
DEP and DIP exchange information using a MCN both are connected to. The MCN can refer to a bus system (e.g. USB), an IP based network or any other type of communication infrastructure.

Depending on the type of the device the DEP represents (e.g. actuator or sensor), both the DEP and DIP can act as the source or sink of an information flow. In case of M2M, some systems also allow for a direct communication between DEPs. However, in order to establish an information flow between two interacting systems (i.e. DEP and DIP), a sequence of basic steps, as, for instance, described by UPnP, is necessary:

- **Addressing:** Dependent on the MCN, an address needs to be assigned to the DEP. This process is heavily influenced by the MCN technology used. In IP based networks, e.g., DHCP or AutoIP can be used. The address needs to be unique among the participants of the MCN segment DEP and DIP are connected to, since the DIP usually controls multiple DEPs.

- **Discovery:** Once the DEP has a valid address, it can be discovered by the DIP. Both the DEP and the DIP can take an active or passive role. Upon connection to the MCN and address assignment, the DEP can actively announce its presence using broadcast messages or passively wait for search requests issued by the DIP. A lot of discovery protocols such as the Simple Service Discovery Protocol (SSDP) used by UPnP, the Service Discovery Protocol (SDP) used by Bluetooth or, IETF Service Location Protocol (SLP) exist. Some rely on direct communication between the DEP and the DIP, some use a mediator approach, where DEPs announce offered services to a registry or a directory and allow DIPs or consumers to look them up.

- **Description:** Dependent on the expressiveness of the discovery mechanism used, an optional description step can be introduced to gain further information about the DEP, which may be required to load appropriate control logic (i.e. a driver). A driver basically introduces the common language required between DEP and DIP to exchange information.
- **Authentication and Authorization:** Some environments require an authentication and authorization step to ensure that confidence about the identities is given and both DEP and DIP are allowed to exchange information. This step can take place at different layers of the OSI model and can therefore appear at different positions in the sequence of device integration steps.
- **Configuration:** Some protocols require that DEP and DIP enter a configuration step and agree on certain protocol specific properties prior to start exchanging information. The ISO/IEEE 11073 medical device interoperability protocol, for instance, requires that communication partners agree on the data encoding used before exchange of measurements is allowed.
- **Control:** Having configured a communication session, DEP and DIP can start to exchange data. Event-driven or request/reply approaches are common. Usually, the DIP requests data by triggering an action or service offered by the DEP or the DEP pushes data once it is available (e.g. a new measurement was taken).

As already highlighted in the previous section, device integration in the IoT domain is usually subject to a variety of devices and MCNs. Because the capabilities of a single DIP are often limited to a subset of the possible MCN technologies, gateways are introduced. A gateway acts as a bridge between two or more MCNs that may be based on totally different protocols. In doing so, the gateway can act as a prepended DIP or just forward data packets from one MCN to another MCN. Especially the latter case leads to complex scenarios, since the DIP needs to be aware of the protocol used behind the gateway. This often leads to situations where multiple drivers are involved in the integration of a device. Similar to concepts used in operating systems, like the Windows Plug and Play device tree [111], device drivers need to be organized in a stacked fashion to properly represent the topology of integrated devices and reduce the complexity of single device drivers. Therefore, a device integration solution needs to be able to flexibly and transparently manage the composition of device control logic, which is discussed in Section 2.4.2.

2.3.3. Device Management & Abstraction

Device management is the process of administrating or managing devices within an IT infrastructure. With the proliferation of mobile devices and the arising need to centralize

and increase the efficiency of their management, the term Mobile Device Management (MDM) evolved. Especially mobile devices integrated into company networks need to be efficiently provisioned, configured, updated and fault monitored [103]. Standards like the already mentioned OMA DM [116] or TR-069 [25] [129], which is used for remote configuration of devices like modems, routers or, set-top boxes, were introduced to unify MDM. Most MDM protocols provide basic primitives to discover, identify and get and set values on a device's object model, which describes the properties and functions of the device. However, the variety of devices in the IoT domain requires a uniform representation of the object models in order to be manageable by MDM protocols. This problem is intensified, if we not only take into consideration typical MDM devices like smart phones, tablets, routers or modems but also all the sensors and actuators, present in many IoT applications. This leads to the requirement for device abstraction mechanisms.

In computer science, the abstraction principle [125] is known as a recommendation to programmes to avoid duplicity in their program code. In Java, for instance, an abstract class can be used as a skeleton to implement logic that is required by all inheriting classes. An interface can be used to introduce abstract definitions for different implementations which share the same meaning. Basically, abstraction means to introduce different layers of complexity, where each layer is hiding the complexity of the layer below. A prominent example is the OSI model [172]. The transport layer, for instance, abstracts away the technical details and complexity of the physical, data link, and, network layer and allows application layer protocols to easily exchange data. In terms of device abstraction, especially the interface concept is of notable importance because it allows defining common interfaces, sometimes also called categories or profiles, for devices of the same class with different implementations (e.g. temperature sensors from different vendors).

Device abstraction is a crucial requirement for device integration solutions because it allows decoupling devices from applications and enhances the portability [115]. Application developers are enabled to focus on the logic, without having to deal with device type or vendor specific details of the I/O protocol. Devices of the same type can be easily exchanged and the risk of a vendor-lock-in is reduced. According to Vaughan et al. [155], three basic levels of abstraction can be defined:

Levels of Abstraction

Character Device Model:
Originated from Multics and UNIX, this refers to the most basic level of abstraction, where each device can be treated as a data file. The model defines access to the devices by providing five basic operations (open, close, read, write, ioctl).

Interface(Driver) Model:
On top of the basic operations offered by the Character Device Model, the Interface

Model provides device independence by refining the basic operations with knowledge about the structure and data format of a data stream. Thus, device type related operations can be offered by the interface and application developers are no longer required to take care of device specific protocol or control logic.

Client-Server Model:
The third level of abstraction targets the decoupling of the application (i.e. client) and the server that manages the actual device control. Besides platform independence (the client only has to be aware of the protocol offered by the server), this abstraction simplifies concurrent access to devices because session and transaction management can be realized by the server and there is no need for synchronization between multiple applications accessing the same device. Other challenges like access control are also easier to tackle introducing this level of abstraction.

The Character Device Model can be used to simplify the development of driver services by providing an abstraction for different communication protocols (e.g. Bluetooth or ZigBee). Similar to the concept of composing drivers introduced in Section 2.3.2, device drivers can use them without knowing about the implementation details of a certain protocol, just using the basic operations to operate on the data stream. A possible approach to achieve this layer of abstraction is the OSGi IO Connector Service Specification (IOCS) [119], which is based on the J2ME I/O packages. However, the Character Device Model still requires the application to ship detailed knowledge about the structure and format of data as well as higher level functions offered by a device. Therefore, uniform access to, for instance, temperature sensors from different vendors at least requires the Interface Model to be applied. As mentioned, interfaces (i.e. device categories) allow uniformly describing devices with different control syntax and implementation but the same control semantics.

Providing the Client/Server Model is a challenge allied to providing device management capabilities through defined management protocols as mentioned above (e.g. OMA DM or SNMP). Device properties and operations shall be accessible through well defined interfaces, while offering features like session management or access control. Because a variety of device management standards and protocols exist, it is difficult to choose one standard appropriate for all application domains. Instead, multiple standards and protocols should be supported without having to adopt the underlying abstraction models or the data structures used to maintain the properties and operations of a device, which again leads to the requirement of a uniform representation of the device object models. Section 2.4.2 will introduce the OSGi DMT Admin Service Specification (DmtAS) [119], which aims at providing a solution for simultaneously supporting different management and device access protocols by introducing a generic and uniform representation for device object models.

2.3.4. Interoperability

In general, interoperability can be defined as:

> "The ability of two or more systems or components to exchange information and to use the information that has been exchanged." - IEEE, 1990 [72]

Achieving interoperability is one of the major challenges found in IoT application domains that need to integrate a huge variety of heterogeneous devices and systems (e.g. smart homes [124]). Although heterogeneity is often considered an obstacle for interoperation of distributed systems, it can also be recognized as a feature, because heterogeneity allows efficiently designing a systems towards the specific needs of the target environment [157]. Therefore, it is likely that the landscape of devices and systems in the IoT domain will remain heterogeneous, which leads to the need of interoperability enablers like middleware solutions or gateways. Different levels of interoperability have been defined. A substantial classification, additionally covering the topic composability, is given by the Levels of Conceptual Interoperability Model (LCIM) [150] [160]:

Levels of Conceptual Interoperability Model

Level 0 - No Interoperability:
A stand-alone system with no interoperability.

Level 1 - Technical Interoperability:
A communication infrastructure, that allows two or more systems to exchange data, exists.

Level 2 - Syntactic Interoperability:
A common data format is unambiguously defined between the systems. This, for instance, includes how much bytes are required for a certain data type or which sequence of data types is exchanged (i.e. the structure of data).

Level 3 - Semantic Interoperability:
The interacting systems share the meaning of the data (e.g. a received number is a temperature measurement expressed in Celsius).

Level 4 - Pragmatic Interoperability:
Pragmatic interoperability means that two interacting systems are aware of how the exchanged data is used (i.e. the context). Pragmatic interoperability deals with the problem that exchanged messages result in their intended effect (i.e. the actual effect does not differ from the intended one) [13]. In other words, the system knows how to combine and use data in order to achieve the desired effect. This means, that, given a certain context, pragmatic interoperable systems need to know which operations are allowed to be triggered and which grouping of data types is required. If, for instance, two E-Health systems know that *M* stands for a male patient, they have reached semantic interoperability. If these systems know, that they have to

include a *M* as the sex identifier in order to request data about all male patients, they have reached pragmatic interoperability.

Level 5 - Dynamic Interoperability:
The behaviour of interacting systems becomes predictable. Dynamic interoperable systems unambiguously define the effect of an exchanged message (i.e. it is defined whether a certain message results in a change of state).

Level 6 - Conceptual Interoperability:
The highest level of interoperability in the LCIM requires a well documented conceptual model, which defines data, processes and constraints of the interacting systems. Processes refers to the behaviour and how the data changes. Constraints refers to the assumptions that constrain the values of the data and the behaviour of the processes.

Because the device integration solution cannot predict the behaviour of the application actually using the devices, pragmatic interoperability is the highest level to be reached. This level partially aligns to the Interface Model abstraction level defined in Section 2.3.3.

Related to the semantic interoperability level, terminology management is an important issue. A terminology (sometimes also referred to as nomenclature) defines a set of terms and their corresponding meanings. Terms can be used to annotate exchanged data and enhance the semantic interoperability between interacting systems. Basically known from linguistics, terminology management is the process of organizing, extending and keeping a terminology unambiguous. Regarding the IoT domain, this is of notable importance because the amount of available devices and features continuously grows. Prominent examples for terminologies and terminology management can be found in the E-Health domain (e.g. HL7, SNOMED-CT, IEEE 11073) and will be discussed in Chapter 7.

2.4. OSGi

OSGi is a specification of a platform-independent (Java based) framework for modularized development of applications and services based on the principle of component-based software engineering [68]. A component, sometimes also referred to as a module, is a package of software that encapsulates related functions and data and provides these as services to other components using well defined interfaces. A component provides the implementation of a certain set of system processes defined by an interface, which is accessible by other components. According to the modularity of a system, an important capability of components is their exchangeability. OSGi allows updating or replacing a

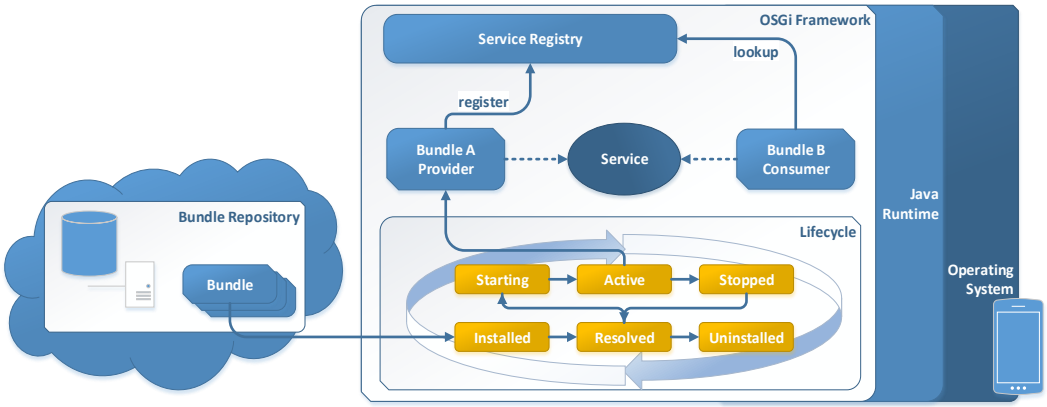


Figure 2.2.: Loosely coupled interaction of dynamically deployed OSGi bundles through services.

component at runtime without breaking the system. Thus, OSGi provides a dynamic service execution environment that is able to deploy and wire services in shape of components at runtime.

OSGi is governed by the OSGi Alliance, formerly known as the Open Service Gateway initiative. As the name suggests, it was originally dedicated to gateway platforms used for device integration (e.g. routers). Its ability to dynamically wire services at runtime based on the requirements of the environment (i.e. the devices to be integrated), makes it a good candidate for device integration solutions. However, due to its extensive service delivery and modularization capabilities, OSGi today is also common in enterprise environments (e.g. as the foundation of the Eclipse platform). OSGi consists of a core specification [120], which defines mandatory core APIs, and a compendium specification [119], which defines additional services dedicated to certain use cases. Proprietary as well as open-source implementations are available.

2.4.1. Core Specification

The foundation of each OSGi deployment is the framework, which provides an runtime environment for downloadable applications called bundles. As shown in Figure 2.2, a device hosting an OSGi framework can download, install and remove bundles on demand, while the framework manages the installation, wiring with other bundles and resolution of dependencies. Bundles refer to components and can be envisioned as OSGi applications shipped by developers. Bundles basically are Java Archive (JAR) files with additional OSGi meta data. The OSGi framework isolates bundles from each other,

by introducing a specific hierarchy of class loaders. This allows mitigating effects that occur when different versions of the same libraries are required, sometimes also referred to as JAR hell. Each bundle is required to specify its dependencies to other bundles (i.e. imports) and which of its classes it will offer in return (i.e. exports). This is done using OSGi meta data files contained in the bundle. Bundles typically provide implementations of interfaces (i.e. services) to other bundles within the OSGi framework. In order to properly manage the deployment of bundles and their interaction with other bundles, OSGi defines several layers:

- **Module Layer:** The Module Layer adds a modularization system to the standard Java API. It defines the rules for sharing Java packages between bundles or hiding them, respectively. This is based on the specific class loaders already mentioned and the import and export definitions of bundles.
- **Life Cycle Layer:** The Life Cycle Layer defines an API for starting, stopping, installing, updating or, uninstalling bundles. Since changes in the life cycle of a bundle can happen at runtime, an event API is provided that allows managing and controlling the life cycle operations of the OSGi framework.
- **Service Layer:** The Service Layer allows for loosely coupled applications by introducing the service- provider and consumer pattern. It enables bundles to register, search for and bind to services only by specifying the corresponding interface. Mechanisms like dynamic service tracking allow coping with life cycle issues of bundles (e.g. updating or replacement of a service providing bundle).
- **Security Layer:** The Security Layer integrates Java 2 security into the OSGi framework and adds capabilities for secure packaging and signing of bundles and applying permissions to them (e.g. permissions to access the network). This is important because bundles can be downloaded and deployed from possibly untrusted remote sources.

2.4.2. Compendium Specification

The OSGi compendium specification adds a set of higher level specifications to the core of OSGi that can simplify the development in case of certain requirements. Examples are:

- **Declarative Services:** provides mechanisms that allow for simpler and automated service wiring.
- **Event Admin Service Specification:** provides a publish/subscribe based eventing mechanism.

- **Configuration Admin Service Specification:** allows configuring deployed bundles and alter the configuration at runtime.
- **IO Connector Service Specification:** integrates the Java 2 Micro Edition (J2ME) IO packages.
- **Repository Service Specification:** allows managing access to external repositories that provide bundles.

Two of the compendium specification are of notable importance regarding device integration and will be discussed in more detail throughout the following sections.

OSGi Device Access Specification

The set of available devices in an IoT application is constantly changing due to mobile or wireless devices being plugged (connected) or unplugged. The OSGi Device Access Specification (DAS) describes basic mechanisms regarding the dynamic integration of devices into a service platform without limiting the concepts to any specific type of device or network technology. In particular, it targets the attachment of services to detected devices and the handling of life-cycle issues that arise when devices are plugged or unplugged at runtime. Moreover, it defines mechanisms to dynamically load and attach services from external repositories if new devices appear (i.e. bundles containing device drivers). The following core entities are defined by the DAS to model the dynamic device integration process:

- **Device:** A Device Service represents any kind of device. It can refer to a single physical device, a network (if the physical device is a gateway or any kind of networking interface), or to a refinement of an already existing device. Refinement means that several Device Services represent a physical device at different levels of abstraction. A Device Service must belong to a Device Category.
- **Device Category:** The Device Category refers to the Interface Model abstraction level introduced in Section 2.3.3. The Device Category specifies the interface needed to communicate with a device (i.e. the interface the Device Service exposes) and properties that can be accessed to monitor the state of the device (e.g. battery level). Additionally, properties required for matching Devices and Drivers are defined.
- **Driver:** Driver Services provide the actual control logic and can attach to Device Services to either refine an existing device or to integrate a new device (if, for instance, the Device Service represents a network). In most cases Driver Services will expose new Device Services that can be accessed by applications or can be

refined again by other Driver Services. The attachment is based on the Device Category the Device and the Driver Service belong to.

- **Device Manager:** The Device Manager is a singleton that manages the attachment process. It waits for Device Services being registered with the framework and searches for suitable Driver Services. It uses the Driver Locator and Selector to perform the matching and attachment processes. It allows only one Driver service to attach to a Device Service.
- **Driver Locator:** The Driver Locator provides the bridge to external driver bundle repositories.
- **Driver Selector:** The Driver Selector provides refinements to the matching algorithm that attaches Driver to Device Services.

Thus, the core concept of the DAS are Device and Driver Services that are attached to each other by the Device Manager based on Device Categories. Device Categories introduce abstraction, since devices from different vendors belonging to the same class usually expose the same interface as defined by the category. It is important to mention that DAS only defines the concept of Device Categories, but does not describe any specific category or terminology on its own.

Upon registration of a Device Service, the Device Manager can search for suitable Driver Services based on the Device Category. The main responsibility of the Driver Service is to provide additional control logic that allows, based on the Device Category the Driver is attached to, further refining the device, exposing additional operations or, performing discovery (e.g. if the Category the Driver was attached to belongs to a network). The process of attaching additional logic results in new Device Services registered by the Driver Service, which are again related to a Category and therefore can be subject to another round of attaching Device Drivers. This results in a flexible composition of device integration logic that allows reflecting the actual topology in a modular way (e.g. a Bluetooth driver refined by a Bluetooth discovery driver refined by the drivers that represent the actual Bluetooth devices). This approach also allows for complex scenarios as often found in the automation or smart home domain when bridges are used. An Ethernet or CAN bridge can be easily mapped and CAN based devices can be integrated, although they follow a different networking protocol using different addressing schemes.

The initiation of the attachment iteration is given by so called base drivers, which register the first Device Services present. Base drivers usually link to driver of the operating system and represent the hardware interfaces the host of the OSGi framework provides. The process how base drivers are deployed is not further specified by the DAS and thus, will be discussed in Chapter 6.

OSGi Dmt Admin Specification

As introduced in Section 2.3.3, Mobile Device Management (MDM) is an important feature for applications communicating with a variable set of heterogeneous devices. Therefore, a lot of standards like OMA DM, TR-069 or SNMP exist. According to Section 2.3.1, it is difficult to make a commitment for one protocol because this could exclude certain classes of devices and may not be sustainable in terms of further development of protocols and devices. Therefore, multiple protocols should be supported. This requires a uniform representation of the device object models, which can be translated to a certain protocol based on the requirements of the environment. Moreover, not only management protocols but additionally application layer protocols can benefit from a uniform representation because in conjunction with Device Categories (i.e. the Interface Model abstraction) uniform access to the capabilities of heterogeneous devices can be gained. Therefore, the OSGi DMT Admin Service Specification (DmtAS) aims at providing a data structure for such an uniform representation of the device object models. DMT is an acronym for device management tree, which means that similar to the Windows Device Management [111], a tree based data structure is used.

Based on a root device node, which usually refers to the host device (i.e. the device that executes the OSGi framework), each integrated device is represented by a child node. Leaf nodes can be used to represent the properties or operations offered by a device. Applications can interact with devices by accessing the nodes of the DMT. The DmtAS manages the access by specifying mechanisms used for sessions, transactions and access control. Because the structure of the DMT is complex and not each client application should have to be aware about it, it is not preferred to work directly on the DMT. Instead, a plugin mechanism is introduced. Plugins can take responsibility for a sub-tree of the DMT by managing access and providing interfaces to be used by the clients. These plugins decide where and when nodes have to be manipulated and provide a DMT view of the OSGi services represented by the nodes they are responsible for (e.g. a Device Service). Therefore, plugins act as a proxy between the clients and the OSGi services represented by the DMT. However, the preferred way for applications to interact with the DMT is using protocol adapters. A protocol adapter, for example, stands for a device management protocol and translates the uniform DMT representation into the representation used by the protocol. Protocol adapters additionally introduce the Client/Server Model abstraction discussed in Section 2.3.3.

The DmtAS does neither define any specific structure nor the layout or granularity of the information stored in the tree. As mentioned, the knowledge about the structure has to be introduced using a plugin, which decides where to add or modify nodes. The structure has to be defined carefully, since it has an impact on the possibilities to search for information or devices. Basically, the structure should reflect the actual topology

of the devices, which means that a Bluetooth based personal health device is added as a child node of the device representing the Bluetooth network. This simplifies the management of the tree in case the Bluetooth network device disappears. However, if another ZigBee based personal health device becomes available and an application would search for all personal health devices, the whole tree would have to be traversed. A possible structure of the DMT is further discussed in Section 6.2.2.

2.5. Security

Securtiy and privacy are core requirements for a lot of IoT application domains (e.g. E-Health, Smart Homes). Security can be basically classified as communication and computer security [33]. Communication security covers attacks against communication links while computer security deals with detection of compromised nodes and recovery from malicious states (e.g. a compromised Data Integration Point (DIP)). The following principles act as a common classification for communication security [107]:

Communication Security Principles

Authentication:
Authentication refers to the ability of a communication partner to proof its identity. This principle is usually covered by using shared secrets. An attacker basically tries to make the recipient believe that the message comes from an authentic source.

Availability:
Availability refers to the ability to communicate with a desired receiver. Especially in case of emergencies, availability becomes a principle of secure communication. A typical attack is denial of service.

Confidentiality:
Confidentiality means that nobody but the intended receiver of a message shall be able to access and interpret the carried information. Eavesdropping, which can be achieved by intercepting the communication link or accessing stored data, is a typical attack on confidentiality. Encryption can be used to cover confidentiality.

Integrity:
Integrity means that messages transmitted are not modified or manipulated by an attacker. Even if confidentiality is given, the integrity principle can be violated by manipulating the data blindly. Signatures or checksums can be used to ensure integrity.

Communication security, common threats and possible attacks have been well investigated over the past years for related domains such as WSNs [123] [159] [90]. However,

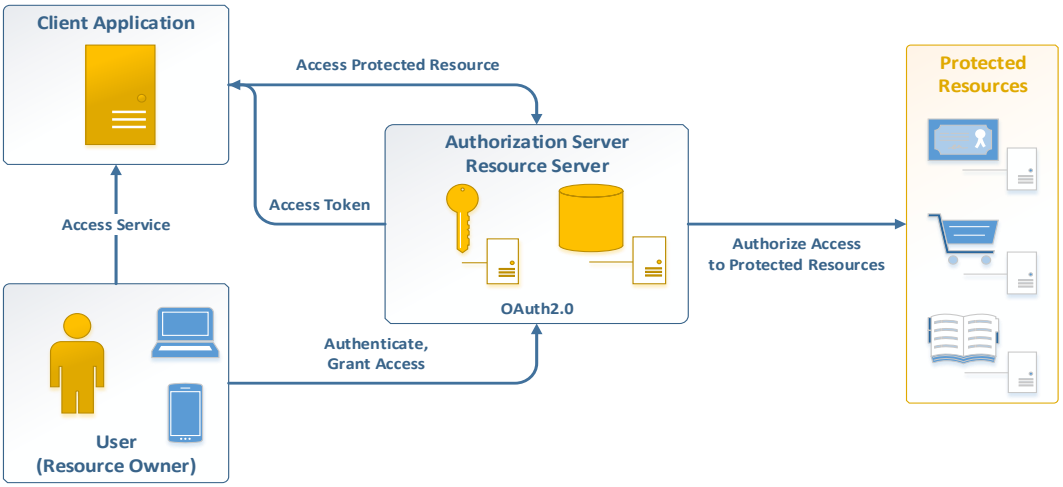


Figure 2.3.: OAuth2.0 based authorization separating the client from the resource owner role.

IoT applications typically face a major heterogeneity especially regarding facilitated application layer protocols, Machine Communication Networks (MCNs) and, devices. Most of these technologies bring their own securtiy and concepts, which makes it difficult to find an overall concept and achieve end-to-end security. A lot of research is currently conducted in the area of IoT security [108] [162] [8]. Challenges basically arise out of the highly distributed nature of IoT environments that connect hundreds of (mobile) resource constrained devices using possibly vulnerable wireless links. Especially privacy is a major issue because all these connected devices collect data about the environment of the users or about the users themselves. Other issues are protocol and network security, identity management, trust and governance, fault tolerance or, cryptography for resource constrained devices [133].

2.5.1. OAuth2.0 & OpenID Connect

The authentication principle is crucial when a multitude of entities collaborate. Providing trust about the identity of a communication partner is difficult to achieve, in particular if a lot of heterogeneous devices and users communicate with each other like in the IoT. The most common authentication protocol used within distributed systems is Kerberos [113]. Another upcoming approach, popular especially in the area of providing access to web based APIs, Apps or mobile devices, is OAuth2.0 in conjunction with OpenID Connect.

OAuth2.0 [66] is an authorization protocol that follows a particular interesting approach with regard to the peer to peer resource sharing vision of the Device Cloud. When using Cloud services, we often face situations where we grant access to a service to resources hosted by another service (e.g. a social network service that integrates with an e-mail service). This means the resource requesting client (i.e. the third party operating the Cloud service) differs from the resource owner (i.e. the user). Thus, the resource owner usually has to share its credentials with the third party, which introduces serious obstacles (access cannot be revoked without revoking access of all third parties, third party usually stores the user's credentials, validity of access is difficult to manage). As shown in Figure 2.3, OAuth separates the client from the resource owner and issues different credentials to the client than the resource owner uses. Therefore, similar to the Ticket Granting Server used in Kerberos, an authorization server is required. Often, the authorization server and the resource server (i.e. the server providing the resources owned by the resource owner) are operated by the same authority.

OAuth is an authorization protocol and does not provide authentication. This has lead to several proprietary extensions. However, since version 2.0 OpenID Connect [141] introduces an authentication layer on top of OAuth2.0. It enables the authorization server to issue and transmit so called ID Tokens that contain information about the identity of a user and whether the user is authenticated. The backend and methods used by the authorization server to authenticate users (e.g. the user directory) are not specified.

3. Related Work

Contents

The Internet was basically developed to allow for communication between computers that are situated at different sites. As reported by Atzori et al. [14], this host-to-host based communication paradigm is not likely to be suitable for hundreds of things that will be connected to the Internet. Instead, it has to be considered that the Internet is primarily used as an information dissemination system, where accessing data and not a specific host becomes more and more important. Thus, IoT is an interdisciplinary research domain that requires to tackle challenges in the areas of identification, tracking, distributed intelligence, communication protocols, security and privacy, or information management based on integrated approaches. Therefore, this chapter will introduce related research on general purpose IoT architectures, middleware solutions and IoT applications.

3.1. IoT Architectures

One of the core components of IoT architectures are middleware solutions because the complexity of the technical layers is hidden from the application developer, which leads to more robust and reusable code. A Service Oriented Architecture (SOA) based middleware approach for intelligent networked embedded systems was presented by the European research project HYDRA [47]. The resulting LinkSmart middleware [92] focuses on the semantic representation of devices and the possibility to uniformly access each device using semantic web services. A mechanism called HYDRA-enabling that allows wrapping APIs of devices with web service extensions is provided (i.e. WSDL files extended by semantic descriptions). HYDRA both supports to integrate out-of-the-box HYDRA-enabled devices (i.e. the semantic web service is hosted by the device) or native devices using proxies. The implementation of the mapping from the web service to the actual device specific API has to be provided by the service developer because HYDRA

supports different communication technologies (e.g. ZigBee or Bluetooth). A possible foundation for proxies as well as HYDRA-enabled devices is introduced by the Contiki platform [43], which is an operating system dedicated to run on resource constrained devices in IoT applications and WSNs. Contiki incorporates support for protocols like 6LoWPAN or CoAP (see Section 2.3.1). Additionally, it was designed for the dynamic loading and replacement of services and custom program code, which is an important feature for networks of hundreds of resource constrained nodes that cannot be physically collected and reprogrammed. Prasad et al. [128] presented a hybrid approach between the HYDRA and the ASPIRE middleware, which additionally allows incorporating the development and deployment of RFID based applications. In general, a lot of research projects conducted in the last years or still ongoing investigate in challenges related to IoT middleware solutions. Further examples are BRIDGE, EBBITS or SENSEI, most of which are organized within the CERP-IoT research cluster [147].

An approach linked to the exploitation of services and web technologies was presented by Guinard et al. [65]. A refinement of the IoT, called the Web of Things (WoT), that aims at integrating devices not only into the Internet (i.e. the network layer), but into the Web (i.e. the application layer), is discussed. This is achieved by embedding web servers or, similar to HYDRA, using proxies, which are called Smart Gateways in this case. Furthermore, it is proposed to rely on the REST architectural style and use HTTP as the application level protocol. Because HTTP has some drawbacks regarding the event-driven communication pattern, which is common for sensor push interaction (i.e. sent data as soon as it becomes available), the usage of real-time Web technologies (e.g. HTTP server push [44]) is discussed. The feasibility of the approach is shown by integrating the Web of Things architecture in a Smart Metering and a general fashion WSN platform. Kovatsch et al. [93] extended the WoT approach by introducing the Thin Server architecture. They argue that changes in the device or the application also require to change the gateways since they carry pieces of the overall application logic. It is proposed to move all application logic to dedicated application servers, while thin servers embedded into the devices only act as wrappers around the sensors and actors offered by the devices. Similar to WoT, a REST based pattern is used by the devices to export their basic functionalities to the application servers.

In [51], Evangelos et al. first introduce three perspectives (RFID, Smart Object, Social) to the IoT and its architectures and present, similar to Prasad et al. [128], a hybrid approach that is able to handle both RFIDs and smart objects (i.e. devices). It is argued that smart objects, in contrast to passive RFID tags, allow deploying and executing parts of the application logic used within IoT applications and therefore introduce additional requirements to the architecture. A superset of objects, consisting of objects supporting primitive functionalities (i.e. RFIDs) and objects supporting complex functionalities (i.e. smart objects), is defined. Subsequently, a middleware consisting of three technical core layers is described. The Objects Abstraction layer is based on ontologies and

allows translating between services and device specific APIs. A Service Management layer provides basic management operations like device discovery, mapping of services to devices or, status monitoring. Finally, the Service Composition layer allows for the composition of services made available by the Management layer using workflow and service description languages like BPEL or WSDL.

The approaches discussed so far can be classified as things- or object-centric [142] [91], because emphasis is put on the integration of devices with enhanced capabilities in terms of carried application logic and their representation as accessible services. Another class of architectures, as discussed by Gubbi et al. [64], are internet-centric approaches, that focus on internet services while the objects deliver the data. Therefore, the authors introduced a platform based on public and private Clouds that offers storage, which enables sensing providers to offer their data and application developers to access and analyze the data using provided analytic tools. An IoT API is provided, that allows transparently accessing sensed data regardless whether they were stored intermediary in a database or gathered on demand from the sensors. The overall approach is related to the concept of Sensor-Cloud integration, which is further discussed in Section 3.3.

An overall problem, addressed by the IoT-A project [17], is the lack of a reference architecture that introduces an overall understanding of the entities and their relations within different IoT application domains. Often, devices and applications are designed with respect to a dedicated use case or application, which limits interoperability and leads to island solutions, which is often referred to as the Intranet of Things [173]. The IoT-A project tries to mitigate this obstacle by defining an IoT Reference Architecture Model (ARM) applicable to different application domains. Each specific architecture is derived and inherits from the ARM with the possibility for own choices regarding design issues like security or real-time requirements. The presence of the basic reference model ensures interoperability. IoT-A describes a set of reference models (e.g. domain model, information model, communication model), an application independent reference architecture according to ISO/IEC/IEEE 42010 [77] and guidelines that assist in deriving a specific architecture from the reference architecture model.

Semantics & Knowledge Management

The importance of semantics and knowledge management is underlined due to its presence as a major paradigm of the IoT as discussed by Atzori et al. [14]: internet-oriented (middleware), things-oriented (sensors) and, semantic-oriented(knowledge). Semantic-oriented IoT is based on the assumption that the number of things involved will increase rapidly. Thus, technologies related to representation of information, storage, search or information organization in general are required. Therefore, semantic technologies like ontologies, mediation of semantically heterogeneous data, reasoning, or, context awareness

will become increasingly important. Kjær [88] presented an analysis of context-aware middleware characteristics and application types and benchmarked some middleware systems available (mostly as outcomes of research projects like Aura, CARISMA, CORTEX). Toma et al. [151] discussed the main challenges for the development of semantic technologies with regard to the IoT. They concluded that the major steps required to unleash the potential of IoT applications using semantics are:

- Providing modelling and language support in order to properly describe IoT objects.
- To allow for reasoning over data generated by the objects.
- Implement semantic execution environments and architectures that pay respect to IoT requirements.
- Provide a scalable storing and communication infrastructure.

In [85], Katasonov et al. present a semantic middleware for the IoT used as a basis for the UBIWARE research project, which connects an agent-based approach with semantic technologies. Each resource to be connected is, according to general IoT vision of providing virtual representations for physical entities, represented by an autonomous software agent. The connector between the agent and the resource (e.g. a sensor) is an adapter, which allows for semantic mediation in terms of data structuring or conversion of semantic representations. Semantic technologies are used to describe the services delivered by resources and to specify the expected behaviour of a resource, which refers to the level of dynamic interoperability discussed in Section 2.3.4.

3.2. IoT Applications

As already introduced in Chapter 1, IoT applications are found in a variety of domains. Examples are Smart Homes [39], Smart Cities [83], Smart Grids [84], Building Automation, E-Health [80] or, due to their origin from RFID based systems, in logistics or object tracking [99]. A detailed discussion of application domains in general and applications in particular is given by Atzori et al. [14] or the CERP-IoT research cluster [147].

Each of these applications basically employs similar architectural patterns, while differing regarding specific technologies used. This underlines that IoT architectures can lead to a convergence of these application domains. An example is presented by Li et al. [100]. The described Smart Community evolves from several smart homes connected to each other, which can be imagined as an ancestor to the Smart City. A Home Domain, referring to the private network of the smart home, a Community Domain referring to the network of home gateways shielding each Home Domain and, a Service Domain referring to a trusted third party offering service dispatching and emergency services are

defined. It is assumed that a wireless ad-hoc network exists between the home gateways (i.e. within the Community Domain). The authors investigate in security and reliability of the networking (e.g. routing) used within the Community Domain and discuss the detection of home gateways with inaccurate behaviour (e.g. incorrect messages, failures while forwarding messages). The authors also discuss the trade-off between security, privacy, and access control, which is of notable importance when dealing with E-Health applications and emergency scenarios. Attribute based encryption schemes are proposed to allow for fine-grained access control to privacy-constrained data dependent on the current situation (i.e. normal vs. emergency).

Schreiber et al. [138] presented the PerLa language and middleware which aims at bridging the gap between the various pervasive information system application domains by providing a general fashion architecture that is able to adapt to the requirements of different application domains without redesign or recoding of components. Basically, the requirements are considered as the set of necessary sensors and their corresponding data resources (i.e. what is sensed). It is argued, that pervasive information systems usually have to deal with two activities: data gathering (i.e. sensor integration) and data processing. Regarding the first activity, the middleware introduces the concept of Function Proxy Components (FPC). FPCs interact with physical devices and provide abstraction to higher layers by masking the functional communication details and mapping them to a uniform interaction pattern. FPCs can be created on-demand by a factory using an XML Device Descriptor. It is assumed that devices to be integrated can provide such a Descriptor, which results in the ability of the platform to adapt to new types of devices. Regarding data processing a SQL-like query language is proposed. Based on the FPCs, functional details like communication paradigms or protocols are masked and all sensors are managed as a database. A Query Analyser is used to perform user queries based on the instantiation of query executers that are linked to the FPCs.

Based on the smart home application domain, Cheng et al. [35] showed the applicability of the OSGi Device Access Specification (DAS) as introduced in Section 2.4.2. The approach aims at overcoming the protocol heterogeneity by integrating multiple common smart home protocols into a uniform service-oriented platform. Several base drivers were developed to cover Bluetooth, ZigBee and Tmote based protocol stacks. Additionally, bundles that allow integrating DPWS and UPnP based devices were provided. Transcoding services were used to hide the protocol heterogeneity from the application layer. Discovery of devices is conducted automatically by the developed base driver, which register appropriate services with the OSGi service registry upon detection of a new device.

3.3. Sensor – Cloud Integration

Sensor-Cloud infrastructures, as discussed in Section 2.2.1, are a hot topic [7] because they apply the Cloud based service models to sensor networks. Sensor network operators can offer sensing capabilities as a service to Cloud users and benefit from a scalable pool of resources provided by the Cloud (e.g. for resource intensive data analysis jobs). Cloud users can access sensors using well defined interfaces without having to care about the technical details of the underlying sensor network infrastructure. Moreover, as discussed by Yuriyama et al. [168], introducing virtual representations of a physical sensor allows sharing and provisioning them among multiple users and introduces the possibility to group sensors. Grouping can simplify the process of accessing and controlling semantically equal sensors. A virtual sensor group consists of one or more virtual sensors and is created by clients to access sensors. Virtual sensors are created if a physical sensor is registered, which can be done by any participant of the Sensor-Cloud infrastructure. Templates are used to specify physical sensors, which means that users registering physical sensors have to choose a compliant template. Since a virtual sensor can be part of multiple virtual sensor groups, the virtual sensor additionally has to avoid inconsistent commands.

A related approach, called Cloud-assisted remote sensing (CARS), was described by Abdelwahab et al. [1]. CARS focuses on the underlying Cloud ecosystem and the application of the resource sharing and PAYG pricing models. A cross domain provisioning of sensor resources is described, which means that for instance a security camera deployed in a mall can also be used for data analysis jobs (e.g. market studies, customer behaviour). Therefore, a multi layer architecture with three service models, according to IaaS, PaaS and SaaS is defined. The Sensing and Actuating Infrastructure as a Service (SAIaaS) model offers access to physical sensors using their virtual representations (similar to the virtual sensor concept). It requires that the physical sensor network resources can serve multiple sensing tasks concurrently. Clients are not allowed to make changes to the physical instances. The Sensing and Actuating Platform as a Service (SAPaaS) model adds a set of APIs to the SAIaaS layer in order to allow for the development of sensing and actuating applications without having to worry about the physical infrastructure or details of the sensor network specific APIs. The Sensing Data and Analytics as a Service (SDAaaS) model provides access to the sensed data and other information like the context in which the data was collected or its accuracy. It is assumed, that most applications will rely on this model, because the only requirement is access to the information and there is no need to change the underlying physical sensors or their virtual representations.

Integrating sensor infrastructures into Cloud environments leads to a huge amount of sensors continuously producing data. This data needs to be properly processed and

analyzed in order to gain benefit from Sensor-Cloud infrastructures. The resulting Big Data problem was discussed by Zaslavsky et al. [170]. The core challenge again is related to the topic of semantics and knowledge management. A lot of sensors produce semantically equal data while using heterogeneous interfaces and data formats, which requires to introduce mediation layers. Moreover, sensed data is often annotated with context information, which has implications on the resources required to analyze them.

4. Device Cloud – Overall Concept

Contents

This chapter is intended to provide an overall understanding of the Device Cloud and its fundamental concepts. Based on a review of possible application scenarios, the system requirements will be discussed. As a result, the actors and their virtual representations, which are required to formulate the Device Cloud concepts, will be presented.

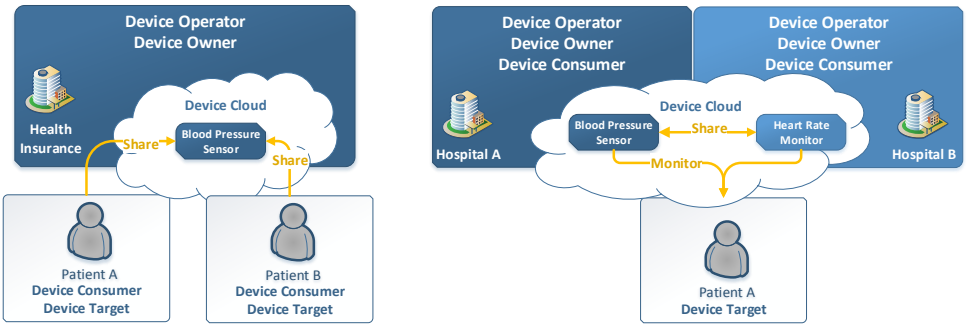
4.1. Principles of Sharing

From the perspective of a user device provisioning can be envisioned as a sharing process. Along with the term Sharing Economy, sometimes also referred to as Shareconomy, sharing enabled by internet platforms became very popular [134]. Rising with services like sharing files or other digital contents, the trend evolved towards sharing physical things in recent years [26]. Examples are car sharing, apartment sharing or sharing of other ordinary things like tools. The major concept behind the Device Cloud is also sharing the devices. It is related to, but has to be differentiated from, infrastructures like sensor virtualization solutions, which are about sharing the virtual representation (i.e. access to the data). Sharing is basically motivated by the ability to increase utilization

and therefore save money. Things owned are often only used for a short period of time. Sharing these things can enable the owner to increase the utilization and earn money and allow the user to save money because borrowing is usually significantly cheaper than buying the thing.

From a technical point of view, the Device Cloud deals with provisioning access to exclusive and non-exclusive resources. Usually, most devices are exclusive resources because there is an exclusive communication link between the device and the system (e.g. a gateway) integrating it. However, some devices may allow multiple systems to communicate with them. These device are non-exclusive resources. Again, we have to distinguish from approaches like sensor virtualization discussed in Section 2.2.1. If the integrating system allows multiple users to access the device through a service interface, then we talk about sharing the virtual representation, not the device itself. For some use cases sharing the virtual representation fits well. Sharing only the virtual representations means that we keep the bindings between the device and the integrating system and just provision access to the integrating system or some other higher level service that exposes the device as an interface. Each sensing device would become a non-exclusive resource in this case. However, there are serious constraints with this approach, especially regarding security and privacy:

- **Security:** Integrating systems are more vulnerable to attacks because they provide an operating system and allow executing custom code. Some environments may not permit communicating or further integrating with such systems. If, for instance, a patient is monitored by medical devices and these devices are integrated by the patient's smart phone, a hospital may not allow communicating with the devices through the patient's private smart phone.
- **Privacy:** Provisioning access only to the integrating system relies on trusting the entity having control over the integrating system and each node participating in the transmission chain between the device and the integrating system. Moreover, as the virtual representation would be shared, which basically means the data is shared and not the device, legal issues could be faced.
- **Device Mobility:** Since some devices move in space, it cannot be guaranteed that the corresponding integrating system is always in range or moves with the device. For instance, a patient may be carried to a hospital in an emergency and his smart phone may not be taken along with him, but the medical sensors monitoring him may still operate.
- **Real Time:** Accessing the device through its virtual representation may involve several systems (e.g. if the representation is exposed using a backend server system). The resulting delay and loss of control over the participating nodes may not be suitable for all application scenarios.



(a) Two Consumers requesting access from one Device Owner

(b) Two Consumers sharing on a peer to peer basis - Device Target differs Consumer

Figure 4.1.: E-Health use cases that illustrate different principles of sharing devices and the different roles the participating entities can hold.

Thus, the Device Cloud is about sharing the devices (i.e. the data sources) and not their virtual representations (i.e. the data). It will be assumed that the integrating systems always belong to an entity and are not shared. However, the concept of sharing the virtual representations can be introduced to the Device Cloud as an additional feature with minimal overhead. This is briefly discussed in Section 5.2.4.

Following the concept of mutual exclusion known from concurrent programming [41], the Device Cloud needs to provide locking mechanisms in order to synchronize access to the devices. Similar to read-locks and write-locks one could distinguish between sensing and actuating devices by introducing functional device classes like Exclusive Sensing Device, Non-Exclusive Sensing Device, Exclusive Actuating Device and so on. However, according to the concept of device categories discussed in Section 2.4.2, a more generic classification based on the device categories can be defined. This results in three functional device classes to be considered:

Functional Device Classes

Exclusive Transducer Device:
The cardinality between the device and the integrating system is 1:1. Only one integrating system can consume the device based on its category. An exclusive lock is applied.

Non-Exclusive Transducer Device:
The cardinality between the device and integrating systems is 1:N. Based on the category, multiple integrating systems can consume the device simultaneously. A shared lock is applied.

Composite Transducer Device:
A generalization of Non-Exclusive Transducer Devices. Composite devices can offer multiple categories (e.g. due to multiple sensing devices embedded into one physical device), which can be grouped to sets of one or more categories. Each set either refers to an Exclusive Transducer Device or a Non-Exclusive Transducer Device.

In general, the process of device provisioning or sharing can be defined as an owning entity granting a device access lock to a consuming entity. However, since we deal with physical things that can move in space, sharing can have multiple shapes, as shown in Figure 4.1. For the purpose of an overall sharing definition, we have to consider at least three more roles than the already given owner and consumer. The following general roles are defined to describe the device sharing process between participating entities:

General Device Cloud Roles

Device Owner:
An entity that owns a device holds this role for the particular device.

Device Consumer:
An entity requesting access for the pupose of reading data from the device or controlling it holds the consumer role.

Device Integrator:
The entity that integrates the device (i.e. has established a communication link to the device).

Device Target:
In most cases this role is similar to the entity that refers to the thing, the being or the environment sensed by the device.

Device Operator:
If an entity exists that operates or manages the device provisioning in charge of the Device Owner, it holds this role.

The ordinary case just involves the Device Consumer that requests access from the Device Owner, while the Device Target is the same as, belongs to, or is visible to the Device Consumer without violating any legal regulations. The Device Integrator role is also held by the Device Consumer. More complex scenarios involve a Device Target or a Device Integrator that does not belong to the Device Consumer and are likely to include a Device Operator. In general, all scenarios require the Device Consumer, upon requesting a certain Device, to establish a relation between the Device and the Device Target. Otherwise the recorded data would miss necessary contextual information and could be useless for most of the use cases. The following application scenarios will illustrate the relationships between the roles involved in the sharing process.

4.1.1. Application Scenarios

Based on the assumptions, discussed in Chapter 1, that the amount of devices surrounding is constantly growing and that each device can be considered as a resource providing data about the user's environment, the need for on-demand provisioning these resources is well motivated from a technical point of view. Having a look at what the term device provisioning can be referred to from the user's perspective and what kind of applications this technical capability will make possible, allows understanding the framework conditions to be considered during concept and architecture design.

E-Health - Sharing Medical Sensors

E-Health is a promising application domain for the Device Cloud because often several medical devices are used to monitor the condition of a patient and several Care Delivery Operators (CDOs) are involved in the patient's treatment process (e.g. a home doctor and a hospital). If we assume that a patient is already equipped with medical sensors, for instance, due to participation in a telemedicine program, and is being transferred to a hospital due to an emergency, the hospital's physicians can benefit from directly accessing the medical sensors. This use case includes Device Consumer A and B, where A refers to the organization being responsible for the telemedicine program (e.g. the patient's home doctor or a health insurance) and B refers to the hospital. Furthermore, A usually takes the role of the Device Owner and Device Operator, too. The Device Target is the patient. Upon discovery of the medical sensors, B will request access to the sensors from A. It is presumed that each sensor (i.e. device) has a globally unique identifier and that a backend information system exists that allows B to identify the specific type of each medical sensor as well as the corresponding Device Owner A. Upon request, A has to decide whether B is allowed to access the medical sensors monitoring the given Device Target (i.e. the patient). The E-Health scenario will be discussed in more detail in Chapter 7.

Smart Home - Sharing in Living Communities

When referring to the term Smart Home, we often think of applications like automated heating or light control, securing our environment with cameras or motion detectors, monitoring the state of our kitchen devices or managing our entertainment devices, for instance. Sharing can be used to improve many of these applications. In most of the Smart Homes several individuals coexist, regardless whether we are talking about families, visitors or apartment sharing communities, where each individual brings devices that are used by the whole community. Two simple sharing use cases are found when

thinking of visitors or entertainment devices. Each visitor usually has its own preferences regarding room temperature and light control. For the period of his stay the visitor could be allowed to access the respective sensors and actuators in his guest room and use his own smart phone and already stored preferences to control their behaviour. The second simple use case is given by all the screen devices available in the Smart Home. Each resident can temporarily take control over a screen and use it to display preferred contents (e.g. a movie or video game streamed by his smart phone). More complex scenarios evolve out of the interaction of multiple Smart Homes or a Smart Home and devices in the public domain, which the term Smart City refers to.

Smart Cities - Sharing in the Public Domain

The term Smart City has a high relevance to the term IoT [169]. The foundation is given by ICT enabled infrastructures that provide a continuous flow of information helping the city and its citizens to optimize and improve the quality of life [18]. Nowadays, it is often stated, that the Smart City is about collecting the data from the sensors, managing the data using Cloud infrastructures and delivering services on top of the data to improve and optimize the citizens' life [136]. However, the Device Cloud approach can broaden this view by allowing citizens to directly interact with the sensors deployed in the public domain. Following the light control example given for a Smart Home, a citizen could take control over the lanterns placed in front of his home when he leaves or an incident is detected by the security system of his Smart Home. In general, sharing in the context of a Smart City is about provisioning access to devices that are placed in public areas and are available to multiple independent individuals (i.e. Device Consumers). The role of the Device Owner can be taken by any public authority, a company offering services or even a single individual offering sensing capabilities to other citizens. The Device Target is likely to refer to a thing or an environment in a public area, but can, in case of interacting Smart Homes, also refer to beings or to private things.

Sharing devices deployed in public areas can be illustrated by two simple use cases. The first use case involves devices deployed by public authorities, which can be found, for instance, in museums or art galleries. Installations exhibited in museums these days often include audio, video and light systems. Basically, the visitor is guided through the museum by an audio guide, and each installation follows a fixed-schedule arrangement of the audio, video and light show. Instead, the Device Cloud approach could allow every visitor to directly interact with audio, video and light devices and therefore interactively participate in the installation, or just replay interesting parts without having to wait for the next turn in the schedule. The second use case concerns devices offered by companies like treadmills or bicycle ergometers found in a fitness center. A customer is enabled to link each device he is using to his smart phone, which can allow for loading personalized

training plans or analysing the data after the training. Moreover, from a technical point of view, the customer allocates the resource treadmill on demand and, similar to a Cloud Computing provider, the fitness center could offer Pay-As-You-Go (PAYG) pricing models.

4.2. Device Cloud Concept

Based on the fundamental definitions and principles of sharing given in Section 4.1, this section will describe the participants (i.e. actors) of the Device Cloud, how they interact and how the basic roles *Device Owner, Device Consumer, Device Integrator, Device Target and Device Operator* can be mapped to them. In order to provide a holistic model of actors and interactions, the major technical components required to describe the interactions are introduced. As a result, a reference list containing all major Device Cloud actors and technical components will be given and a consistent naming convention used throughout the rest of the thesis will be established.

Figure 4.2 gives an overview of the main interactions necessary to enable the device provisioning. The interactions result in the forming of a **Federated Device Pool**. Similar to the notion of the Cloud, this can be transparently accessed by Consumers in order to request and use device resources. Related to the concept of Cloud Federation [34], the Federated Device Pool can be envisioned as the sum of all devices offered by the different participants of the Device Cloud. According to the IoT definition given in Chapter 1, it is assumed that each device in the pool has a globally unique identifier. In order to properly manage the devices and the provisioning process, each interaction between the participants needs to be tagged with such an ID. The ID has to be assigned to a device by the Device Vendor, which usually is the first participant announcing a device to the Device Cloud, since the lifecycle of a device starts with its deployment.

The deployment of a device is usually initiated by a **Device Vendor** selling a device to a customer. Devices can be sold to any actor that can hold the *Device Owner* role (e.g. the Consumer). Selling a device presumes that the customer is able to integrate and use it. One of the major requirements given in Chapter 1 was, that the Device Cloud is not limited to a pre-defined set of devices or application domains. Moreover, due to the standardization problem discussed is Section 2.3.1, vendors are not forced to comply to any certain standard or pre-defined data format. Hence, if a vendor releases a new device type, a mechanism to automatically announce and integrate the device control logic is necessary. Therefore, the Device Cloud introduces the concept of a device knowledge base, which is called the **Device Directory**. The Device Directory can be envisioned as a directory service managing devices instead of users like LDAP-based services [158] do. Besides general device related information like descriptions of the device type or the

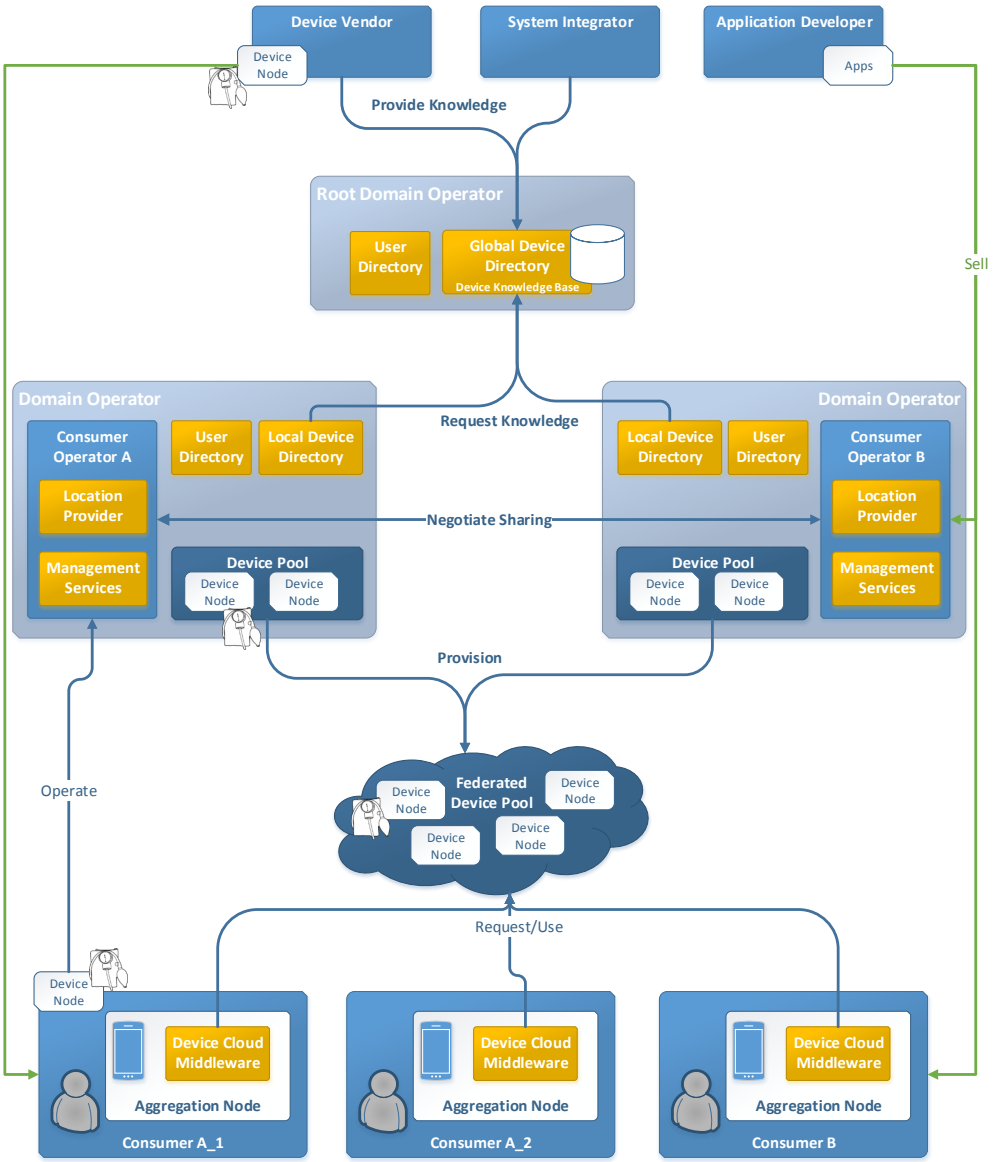


Figure 4.2.: Overview of the actors, their relations to each other and the major technical components building the foundation of the Device Cloud.

device capabilities, the directory maintains a record for each concrete device instance, which basically includes the ID mentioned above. Furthermore, a set of software modules, that allow integrating and handling a device is stored (e.g. device drivers, discovery modules). A set of dependencies allows the knowledge base to express which software module is applicable for which device. Thus, if the Device Vendor sells a new device, it has to register the device by announcing its ID and the corresponding device type to the directory. If the vendor releases a new device type, it has to provide the necessary description as well as the software modules required to handle the device (at least a device driver). However, some application scenarios may require using devices that are not registered by their vendors or that do not provide an interoperable data format. In this case **System Integrators** can participate in the Device Cloud, register and describe the device in charge of the vendor or provide additional software modules (e.g. data transformation modules or custom device drivers). In general, besides managing the device knowledge base, the Device Directory acts as a marketplace for software modules, which can be used by the Device Cloud participants to offer or consume these modules based on their requirements.

Since device management in terms of the Device Cloud not only includes managing the device knowledge base, but must also offer services for accounting, user management or device provisioning, it is not feasible to rely just on one single authority that provides all these services globally. This is underlined by the increasing amount of devices that needs to be managed. Therefore, the Device Cloud distinguishes between managing the device knowledge base and offering services related to device provisioning by introducing two **Operator** participants. An Operator provides the backend information system required to set up core functionalities of the Device Cloud. Multiple Operators can coexist, whereas, according to the separation of concerns, two kinds of Operators have to be distinguished. A **Domain Operator** is a trusted authority that operates a Device Directory (i.e. device knowledge base) and a User Directory. The aim of the User Directory is to provide Identity and Access Management (IAM) capabilities that allow authenticating each principal known within the domain of the Operator and authorize access to the resources offered by the Device Directory. Principal can refer to each other participant of the Device Cloud (e.g. Device Vendor, Consumer, Consumer Operator). The Device Directory served by the Domain Operator basically represents the pool of devices that can be provisioned (i.e. that become part of the Federated Device Pool). Within the overall Device Cloud infrastructure, one Root Domain Operator hosting the Global Device Directory exists. The Root Domain Operator does usually not participate in the device provisioning process. Instead, it just operates the global device knowledge base used by Device Vendors or System Integrators to announce knowledge. Regular Domain Operators serve a Local Device Directory, that acts as a partial mirror to the global instance, while only devices the Operator is responsible for or the Operator has requested from other Operators are represented. Maintaining a local mirror is necessary

because additional meta data required to describe the current state of a device within the provisioning process must be maintained. The global instance is used to synchronize general, stateless information about the devices between the Operators and allows each Operator to determine, which Operator is responsible for which device. Moreover, since it cannot be assumed that a relationship between each Operator and Vendor exists, the global Device Directory simplifies the process of knowledge dissemination and ensures that each Operator can gain access to all records of the device knowledge base.

The process of provisioning a device involves several steps and management services like accounting or decision making. In order to hide this complexity from the end users (i.e. Consumers), the second Operator type, called **Consumer Operator**, is introduced. A Consumer Operator can act under one or more Domain Operators. Accordingly, a Domain Operator can serve multiple Consumer Operators. However, for the sake of simplicity, it is assumed in this thesis that the Consumer Operator is bound to one Domain Operator. From the perspective of the Domain Operator, the Consumer Operator is a client accessing the resources of the Device Directory. Similar to an Internet Service Provider (ISP), the Consumer Operator provides its customers access to the Device Cloud infrastructure (i.e. associated Consumers). Consumer Operators provide the Management Services required to interact with the Device Cloud and provision devices (e.g. decision policies, device locking algorithms or modules for accounting). The Consumer Operator refers to the *Device Operator* role.

The actual provisioning of a device is triggered by the participant **Consumer** that refers to the role *Device Consumer*. A Consumer can be a single individual using Smart Home or E-Health services at home or a company that participates in the Device Cloud to optimize the management of their device resources among their employees. From a technical point of view, each Consumer is represented by one or more instances of the **Device Cloud Middleware** which can be deployed, for instance, on smart phones, routers, or regular PCs. The Device Cloud Middleware offers device integration, device abstraction and data aggregation capabilities and can be defined as a modular execution environment for software modules provided by the Device Directory. Device provisioning is triggered upon discovery of a device that is of interest for the corresponding Consumer and basically involves the following steps:

1. **Consumer** – discover and identify
   a) discover device and extract device ID
   b) request device type description and details from the associated Operator using the device ID
2. **Operator** – synchronize local device knowledge base
   a) check whether the device is already known, if not request details from Global Device Directory using the device ID
   b) update Local Device Directory

   c) return results to Consumer

3. **Consumer** – request device access
   a) check local device allocation policy if device is of interest
   b) if device is of interest, try to allocate it from Operator (i.e. request a device lock)
4. **Operator** – provision device
   a) if device does not belong to own Device Pool, determine responsible Operator and try to allocate device
   b) if device can be provisioned to Consumer (i.e. a lock can be granted to the requesting Consumer), create or modify an existing lock
   c) return device access token (i.e. lock) to Consumer
5. **Consumer** – integrate device
   a) if device allocation has succeeded, load required software modules from the Operator's device knowledge base
   b) deploy software modules to Device Cloud Middleware and integrate device

In summary, the Device Cloud concept is based on a modular Device Cloud Middleware that is able to deploy software modules provided by the distributed device knowledge base at runtime, in order to adapt itself to the requirements of the environment. The device knowledge base and other backend information services (i.e. the Device Cloud infrastructure) are managed by a set of cooperating Operators that virtually combine their local Device Pools to an overall pool (i.e. Federated Device Pool) containing all devices available. Apart from the general concept design, an entity may take several roles and act as multiple participants. The Consumer Operator and the Domain Operator could be hosted by the same legal entity. A Consumer Operator could operate an own set of devices and therefore additionally take the Consumer role.

4.2.1. List of Actors & Components

Following reference lists will summarize all major actors and components of the Device Cloud. In order to properly distinguish actors and the roles they can take, roles, as defined in Section 4.1, are typed in italic letters.

Device Cloud Actors

Device Vendor:
The Device Vendor manufactures and sells the devices provisioned within the Device Cloud. The vendor provides device descriptions and software modules to the global device knowledge base operated by the Root Domain Operator and is responsible for registering each sold device with the Global Device Directory.

Possible Roles: Device Owner

System Integrator:
The System Integrator provides custom knowledge (i.e. software modules) for devices and device types already registered with the Global Device Directory (e.g. data transformation modules). Each participant acting as the *Device Consumer* role can utilize these modules.

Possible Roles: None

Application Developer:
The Application Developer is an optional actor that can offer applications based on the Device Cloud infrastructure. This actor is only mentioned for the sake of completeness and will not be further discussed.

Possible Roles: None

Domain Operator:
The Domain Operator serves a Device Directory and a User Directory offering IAM services for all clients known within the context of the Domain Operator. Basically, the Domain Operator protects access to the resources offered by its Device Directory. Multiple domains are supported by the Device Cloud to simplify IAM processes. Similar to the concept of Kerberos realms, cross-domain interactions are possible (see Chapter 5).

Possible Roles: None

Consumer Operator:
The Consumer Operator provides the backend information systems required to manage the devices, the consumers, the provisioning process, and, other services like accounting. The Operator can have a set of customers using its services to access the Device Cloud (i.e. a set of Consumers). Operators can either manage the devices in charge of their customers, own and provision a set of devices themselves, or follow a hybrid approach (i.e. also hold the *Device Owner* role). Operators can also act as *Device Consumers* if they request devices from the pool (i.e. from other Operators) in charge of their customers.

Possible Roles: Device Operator, Device Owner, Device Consumer

Consumer:
The Consumer uses one or more instances of the Device Cloud Middleware to interact with Consumer Operators and allocate devices. If the Consumer owns devices (i.e. holds the *Device Owner* role) and wishes to offer them to other Consumers, a Consumer Operator has to be mandated to act as the *Device Operator* for this particular set of devices.

Possible Roles: Device Owner, Device Target, Device Consumer, Device Integrator

Device Cloud Node Types

Device Node:
Device Nodes are the physical devices shared by the Consumers (i.e. devices in the pool). These nodes sense the environment, are highly resource constrained and usually do not allow for custom components.

Composite Device Node:
A composite Device Node can have several Device Nodes embedded into one physical node (e.g. a display with an embedded web cam). It can, but does not have to be possible to provision the embedded nodes independently.

Aggregation Node:
Aggregation Nodes host the Device Cloud Middleware used by the Consumers to share, integrate and collect data from Device Nodes (e.g. smart phones, routers, PCs). Aggregation nodes are not shared and therefore do not become part of the Device Pool.

Composite Aggregation Node:
Similar to composite Device Nodes, composite Aggregation Nodes can have several embedded Device Nodes, which, in contrast to the Aggregation Node itself, can become part of the Device Pool (e.g. a smart phone with a camera device).

Backend Node:
Backend Nodes (i.e. dedicated servers) are used by Operators to provide the backend information system. These nodes host components like the Device Directory or the Management Services.

Device Cloud Components

Device Directory:
The Device Directory hosts the device knowledge base and acts as a directory service for devices. Besides device type information, corresponding configurations or dependent software modules, the Directory knows every device in the pool by maintaining their respective IDs.

- *Global Device Directory:* The Global Device Directory is used to synchronize the local instances hosted by each Domain Operator. It maintains all stateless information and software modules available in the Device Cloud and allows determining which Operator is responsible for which device.
- *Local Device Directory:* Local Directories are a partial mirror of the global instance reflecting all devices a Domain Operator is responsible for. In contrast to the global instance, Local Device Directories usually also maintain stateful or private information like the provisioning state of a device or security

credentials required to connect to a device.

User Directory:
Similar to LDAP based directory services, the User Directory maintains all users known to a Domain Operator (i.e. other actors like Consumers or Consumer Operators). It allows for authentication and authorization. Usually the User Directory will be implemented as an interface or wrapper around an existing directory service of the Operator.

Location Provider:
Each Consumer Operator may optionally act as a Location Provider. Location is an important property for provisioning decisions. Usually the location of a device can be easily determined by an Aggregation Node with a GPS sensor (e.g. a smart phone). However, some scenarios may require a different approach to determine the location and the distance between devices and Consumers (e.g. inside buildings, where the shortest euclidean distance is not always equal to the shortest distance between a Consumer and a device) [149].

Management Services:
Management Service is a generic term for services a Consumer Operator has to provide in order to implement the provisioning process (e.g. decision policies, accounting, device locking algorithms, device access negotiation between Operators). In order to enhance modularity, the overall concept separates these services from the directories.

Device Cloud Middleware:
The Device Cloud Middleware is deployed on Aggregation Nodes and can be envisioned as an execution environment for software modules provided by the Device Directory. The Device Cloud Middleware is the Consumer's interface to its corresponding Operator. Upon discovery of a device, the device access tokens and necessary integration knowledge is requested from the Operator and deployed at runtime in order to properly integrate and handle the device. The entity having control over a middleware instance that currently integrates a certain device (e.g. a user with a smart phone), acts as the *Device Integrator* for this particular device.

4.3. System Requirement Analysis

The following section will give a brief overview of the main functional and non-functional requirements of the Device Cloud. Many of the IoT architecture approaches presented in Section 3.1 already discuss challenges and requirements like the demand for Quality

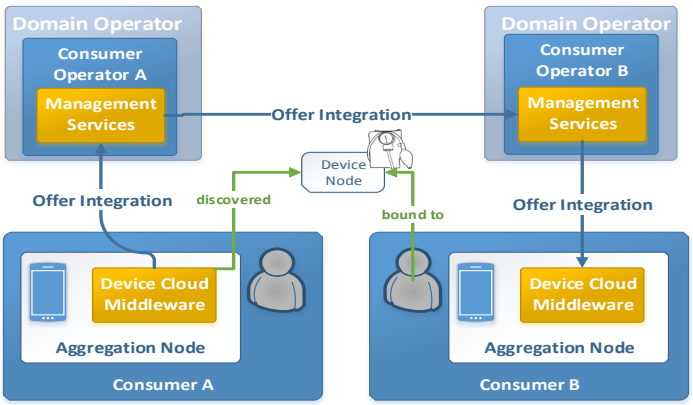


Figure 4.3.: Integration offer triggered by discovery of a device already bound to a Consumer.

of Service (QoS) or Service Level Agreements (SLAs). However, since a comprehensive discussion of these requirements would go beyond the scope of this thesis, emphasis will be put on the topics device pooling and sharing.

4.3.1. Functional Requirements

On-demand Device Access:

Each Consumer shall have the possibility to request access to each device in the Federated Device Pool on demand. The *Device Owner* or the *Device Operator* managing the device in charge of the owner shall have the possibility to evaluate and accept or decline each request. No need for legal contracts or direct relationships between the *Device Consumer* and the *Device Owner* shall exist. Pay-As-You-Go (PAYG) accounting policies shall be supported.

Reliable Granting and Withdrawal:

The Device Cloud shall ensure that no Consumer (i.e. *Device Integrator*) is able to integrate a device without holding a valid device lock. The locking decision made by the *Device Owner* or *Device Operator* must respect the definition of functional device classes given in Section 4.1 (e.g. exclusive or non-exclusive devices). The lease granted to a *Device Consumer* can be constrained by different levels of service quality. In case of an emergency, for instance, the *Device Operator* can decide to force a disconnect and grant the device to another *Device Consumer*. The Device Cloud needs to ensure that the withdrawal, either scheduled or unscheduled, is

reliable (i.e. it is ensured that the device can be provisioned to another Consumer and there is no further connection to the previous Consumer).

Integration Offer and Integration Request:
Besides requesting devices that are of interest, Consumers shall have the possibility to offer the integration to the *Device Owner* or the *Device Consumer* as a service. This can be useful if a device is out of range of the currently leasing *Device Consumer* and becomes visible to another Consumer, that offers to integrate the device in charge as shown in Figure 4.3 (i.e. the roles *Device Consumer* and *Device Integrator* are taken by different entities).

Multi-Operator Environment:
A Consumer shall have the possibility to interact with different Consumer Operators (i.e. the Consumer is a customer of different Operators). Each Consumer Operator is permitted to request and offer each device in the Federated Device Pool to its customers. This allows Operators to provide services crossing different application domains without owning the necessary devices.

Consumer based Device Sharing:
Each Consumer shall be able to share private devices with other consumers by facilitating the device management services offered by its corresponding Consumer Operator(s). This also includes the definition of accounting policies, if supported by the chosen Operator.

Device Access Management:
A *Device Owner* shall be able to define device access policies for its devices. The *Device Owner* can define which other entities or groups of entities have the permission to act as a *Device Consumer* for its devices.

4.3.2. Non-functional Requirements

Plug and Play and Automated Deployment:
Device integration shall take place with minimal deployment and management overhead. Given the presence of a required communication technology, each Device Cloud Middleware instance shall be able to integrate each device as well as search for and deploy the required software modules autonomously and on demand.

Technology and Protocol Agnostic:
The Device Cloud infrastructure shall be designed in a technology and protocol independent manner. Based on modularity features, knowledge required to integrate and handle newly developed devices shall be injectable at runtime without

the requirement to manually change or adapt core components of the infrastructure. This is important due to the decreased time to market of new technologies, protocols and devices. There must be a mechanism to introduce new knowledge to the Device Cloud infrastructure.

Adaptability and Openness:
The infrastructure, in particular the Device Cloud Middleware, shall be able to autonomously adapt itself to the requirements of the current environment. This means that the Device Cloud Middleware acts as a general device integrator, not statically related to any pre-defined set of devices, application domains or vendors. The Middleware is able to load and unload integration knowledge on demand and thus optimize its resource utilization. Well defined interfaces allow different actors (e.g. system integrators, application developers) to participate in the Device Cloud and provide integration knowledge. The global device knowledge base (i.e. Global Device Directory) is accessible by each participant and provides knowledge in an open and common format.

Platform Independence:
The Device Cloud Middleware has to ensure platform independence. Each software module provided by the knowledge base needs to be compliant with the Device Cloud Middleware specifications and is therefore executable on each instance, regardless of the underlying platform. Restrictions are only given by the availability of a certain communication technology, which can differ between middleware instances, and by issues arising from mutual exclusion of software modules or their dependencies.

Unique Identifiers:
A unique identifier needs to be assigned to each actor and especially to the devices. This is required due to the synchronization between the Device Directory instances and due to the federated device pooling. Additionally, unique IDs allow identifying the initiator of a certain action (e.g. a device integration request) or, the owner of a device or a certain record in the device knowledge base. They are fundamental for security features like code signing or authentication. A restriction to this requirement is given for Consumers, which just have to be unambiguous within the domain of their corresponding operator. Otherwise, the identity management would not be scalable.

Peer to Peer Collaboration:
Because of the federated shape of the device pool and the possibly huge amount of participating actors that can hold the *Device Owner* role, a peer to peer style of interaction is required. This aligns with the basic motivation of the Device Cloud, which constitutes on Consumers that request direct access to devices owned by

other participants. Operators can act as proxies for their corresponding Consumers, if devices from different local pools are requested. The Global Device Directory acts as a registry for devices, but negotiation of device access and provisioning always happens on a peer to peer basis between the involved participants.

Security and Privacy:
The Device Cloud Infrastructure shall provide necessary security and privacy features. This can include code signing mechanisms to ensure the integrity of software modules, mechanisms to ensure integrity and confidentiality for exchanged data, authentication and authorization mechanisms or, anonymization techniques, if, for instance, the *Device Consumer* role differs from the *Device Integrator* role. Consumers shall have the possibility to define different levels of confidentiality or integrity when searching for appropriate devices or software modules, since different application domains have different security and privacy constraints (e.g. entertainment versus E-Health).

4.4. Entity Model

Entities are virtual representations of the actors and roles and usually refer to records stored in the Device or the User Directory. The following sections discuss the entities required to model the actors and their relationships as introduced in Section 4.2. Starting with general properties and abstract definitions common to several entities, each specific entity required will be defined.

4.4.1. General Properties & Entities

Table 4.1 lists general properties common to all entities. The properties *EntityOwner* and *EntityVersion* are important in terms of access to entities and synchronization between the Device Directories. Each request to a Device Directory must be issued within an authenticated session, which means the requesting principal was authenticated and has furnished proof about its identity. Details about an authenticated session and authorization are discussed in Section 5.1.

Along with the *EntityOwner* and *EntityVersion* properties, the *EntityDomain* is important for the purpose of synchronization between the Device Directories (note that User Directories usually do not synchronize among each other). Synchronization is important for knowledge dissemination within the Device Cloud. Moreover, a synchronization protocol, as discussed in Section 5.2.2, ensures that different participants always have a consistent view on the entities through their corresponding Local Device Directories. This is important because each interaction in general and the device provisioning in

Table 4.1.: Properties common to all entities

Property	Description
EntityType	The type of the entity (e.g. device instance, device type, platform module).
EntityID	The identifier of the entity. Concatenated with the *EntityType*, an entity must be unambiguously identifiable among all entities of a Domain Operator. Some entities are even required to be unique among the whole Device Cloud (e.g. entities representing devices or Operators).
EntityDomain	Refers to the domain (i.e. Domain Operator) the entity originates from. EntityOwner, EntityOperator and EntityID may be only valid within the domain.
EntityVersion	The version of the entity. The version is a comparable property based on the format used by the Apache Maven project. It allows verifying whether a local copy of an entity managed by another Device Directory is up to date.
EntityOwner	The owner of the entity. Needs to be a principal entity managed by a User Directory.
EntityOperator	Refers to an optional entity that manages the entity in charge of the owner (e.g. a Consumer Operator as described in Section 4.2). Similar to the EntityOwner, the EntityOperator needs to be a principal entity.
PrivateEntity	A flag denoting whether the entity is private or not. A private entity is not publicly accessible and not synchronized with other Device Directories.
PermissionSet	An optional property allowing to specify read or write access to properties of an entity more precisely.

particular are based on creating, accessing or modifying entities protected by the Device Directory. The *EntityVersion* property is usually automatically updated by the Device Directory in order to avoid undefined behaviour during synchronization. The *EntityDomain* property allows identifying the original source of an entity. If a Device Directory granted write access for an entity managed by another Device Directory, the *EntityDomain* property allows propagating the changes back. Hence, if a local copy of an entity is created by a Device Directory due to synchronization, the *EntityOwner*, *EntityOperator*, *EntityDomain*, *EntityVersion* properties remain unchanged. This ensures that the synchronization protocol is always able to identify the responsible originator and the current version and will not accidentally override global information. In general, only the *EntityOwner* or a mandated *EntityOperator* of an entity are able to modify it, while only the *EntityOwner* can grant respective access permissions.

However, some Operators may also want to achieve custom behavior for certain software modules or device configurations or even add software modules that are private to their organization. In this case, entities having the *PrivateEntity* flag set can be added to a Device Directory. In order to avoid unregulated knowledge dissemination, entities added to entities corresponding to another *EntityOwner* remain private by default (e.g. private configuration entries).

Since devices used within the Device Cloud are not restricted to any certain type, technology, or standard, and each Consumer or Operator may have different requirements regarding their deployment, it is difficult to define a uniform format of descriptions or configurations. Therefore, two abstract entities are defined, that allow attaching user defined knowledge to each entity. Additionally, abstract entities to describe the mobility and movement as well as the requirement to authenticate certain entities are defined:

Abstract Device Cloud Entities

Attachment Entity:
An attachment refers to any kind of resource usually stored on the file system (e.g. a picture or a binary). Attachment Entities can have several attachments. An ID, unique within the context of the entity, is assigned to each attachment. Typical use cases are software modules that correspond to an OSGi bundle (i.e. a jar file) or certificates used for the purpose of asymmetric cryptography.

Configurable Entity:
A configurable entity holds a set of configuration entries expressed as Java properties (e.g. key-value pairs). Each entry again is backed by the global entity definition and can therefore be defined as private. This means, that each local copy of an entity can have two virtual sets of configuration entries. One global set managed by the *EntityOwner* and one local set managed by the entity controlling the local copy (usually an Operator). In most cases, the global set is managed by the Global

Device Directory.

Location Tagged Entity:
An entity whose location can be monitored. This can be an important parameter when having to decide about competing device access requests or requests to devices that are already provisioned. Because a lot of devices are mobile, this entity defines a mobility property:

- *MobileEntity*: Devices or Aggregators can either be mobile or stationary. Stationary does not mean that the entity cannot be moved manually.

Principal Entity:
A Principal Entity is an entity that can be authenticated by the User Directory using its *EntityID* and some kind of security credentials (e.g. certificate or password). Examples are Consumer Operators or Consumers. The Device Cloud requires that a public-private key pair exists for each Principal Entity. Thus, Principal Entities are managed by the Domain Operator's User Directory. Except from all other entities, Principle Entities usually do not have the *EntityOwner* property set because often the principle itself is the owner and setting this property in each case can lead to infinite loops.

4.4.2. Device Directory Entities

Several entities are defined to model the physical devices of the Device Cloud. Basically, a physical device is represented by a Device Instance that corresponds to a certain Device Type, complies to a certain Device Class and requires a set of software modules to be integrated and handled. The relation between instance, type and class can be exemplary envisioned as a vendor that sells a blood pressure monitor (class) which has a product number or model code (type) and a serial number (instance) that identifies the concrete physical instance.

Device Directory Entities

Device Instance:

Inherits from: Attachment Entity, Configurable Entity, Location Tagged Entity
EntityOwner: Principal Entity being the *Device Owner*
EntityOperator: Consumer Operator mandated by the *Device Owner*

The Device Instance represents a concrete physical device that is part of the Device Pool and can be provisioned to Consumers. It is described by a serial

number and a Device Type it corresponds to. As a configurable and attachment entity, it can hold information that only applies to the concrete physical instance it represents (e.g. security credentials such as a Bluetooth pin required to connect to the device). Following additional properties are defined:

- *DeviceInstanceState*: The current state of the represented device (e.g. provisioned or idle). This property has the following sub-entries:
  1. *CategoryStates* – According to the *CategorySets* property of the linked Device Type entity, this is a set of tuples. Each tuple {ID, state} represents the state of the corresponding *CategorySet* entry. The *CategoryStates* property is basically introduced to model composite devices. The entries of *CategoryStates* remain disabled until the *RootState* has transitioned to a state that allows the device to be provisioned.
  2. *RootState* – The *RootState* is considered as long as the device is not in a provisionable state.
- *PublicKey*: The public key of the Device Instance. Keys are important during the provisioning process and allow proving if a Device Lock is valid.
- *PrivateKey*: The private key of the Device Instance which is only accessible by the EntityOwner of the Device Instance.
- *CategorySets*: This property can be used to overwrite the corresponding Device Type property, in case a *Device Owner* wants the locking policy to differ from the global one defined by the Device Type.

Device Type:

Inherits from: Attachment Entity, Configurable Entity
EntityOwner: Principal Entity usually referring to the Device Vendor
EntityOperator: Usually the Root Domain Operator

The Device Type refers to a type of devices, which means a set of devices from a certain vendor that complies to the same specifications (i.e. have the same model or product number). It stores information that are applicable to all Device Instances of that type (e.g. vendor name, description). Each Device Type corresponds to at least one Device Category and maintains a set of dependencies on Platform Modules. According to the initial definition of functional device classes, the Device Type entity defines the following property:

- *CategorySets*: Contains a set of triples composed of an identifier, an integer value and a set of Device Category identifiers, which have to match the Device Categories linked to the Device Type. The integer value defines the amount of locks that can exist simultaneously for the set (i.e. group) of categories (1 stands for exclusive, >1 stands for non-exclusive, 0 stands for disabled and, -1 stands for an unlimited amount).

Device Category:

Inherits from: Attachment Entity, Configurable Entity
EntityOwner: Principal Entity usually referring to the Device Vendor or a standardization authority
EntityOperator: Usually the Root Domain Operator

As already discussed in Section 2.4.2, the Device Category is the core enabler for abstraction and interoperability. Each device compliant to a certain category agrees on the interface defined by the category. Thus, the Device Category specifies how to communicate with a device of the corresponding class in a vendor and technology independent way. The Device Category consists of an interface, specifying the available methods to interact with the device, a set of OSGi related service properties including, for instance, model name or serial number and a set of match values, which are used to attach devices and corresponding drivers as introduced in Section 2.4.2. Device Categories can be subject to a hierarchy of inheritance (e.g. a dimmable light device refines a light device), which is discussed in Section 6.2.2. The following property is defined:

- *PlatformModuleSet*: Besides analysing the Platform Modules linked to a Device Type, this set allows searching for Platform Modules that belong to this category. If this category can be the result of a discovery process, this property must be defined.

Platform Module:

Inherits from: Attachment Entity, Configurable Entity
EntityOwner: Principal referring to the developer, usually the Device Vendor
EntityOperator: Usually the Root Domain Operator

Platform Modules refer to the software modules provided by the Device Directory and executed by the Device Cloud Middleware in order to integrate or handle devices. Examples are device drivers, discovery modules, data transformation or utilization modules or, core modules of the Device Cloud Middleware itself. Each Platform Module must have at least one attachment, that refers to the OSGi bundle deployable to the Device Cloud Middleware. Each bundle must comply to the interfaces defined by the Device Cloud Middleware as described in Section 6.2. Four additional properties are defined:

- *Dependencies*: A set of dependencies on other Platform Modules (given the identifier and a version range).
- *Exclusions*: An optional set of exclusions (e.g. two discovery modules using the same port).
- *InputFormat*: A set of input data formats accepted by the Platform Module. This is primarily related to aggregation layer modules such as data trans-

formation modules. Device integration layer modules (e.g. discovery, device driver) are likely to have the Raw (in case of base drivers) or the Device Category format set.
- *OutputFormat*: Similar to InputFormat.

Device Lock:

Inherits from: Attachment Entity, Configurable Entity
EntityOwner: Usually equal to EntityOperator
EntityOperator: Consumer Operator Entity mandated by the *Device Owner*

The Device Lock represents stateful information about the provisioning of a device. A valid lock means, that the Device Instance it corresponds to is currently provisioned to a Consumer or another Consumer Operator (which acts as a *Device Consumer* in this case). With respect to the functional device classes defined in Section 4.1, multiple Device Locks can exist for one Device Instance. Each Device Lock has a mandatory access token attachment, which can be used by other entities involved in the provisioning to prove the validity of the lock. This is discussed in detail in Section 5.1. Six additional properties are defined:

- *Validity*: Usually a time span denoting how long the lock is valid.
- *LockingEntity*: A Principal entity the lock was granted to.
- *Aggregator*: A flag denoting if and which Aggregator currently integrates the corresponding device.
- *OperatorLock*: Indicates whether the lock was granted to an Consumer Operator or to a Consumer.
- *CategorySet*: Defines the group of Device Categories the lock belongs to (according to the *CategorySets* property of the Device Type).
- *Revoked*: A flag denoting whether the lock was temporarily revoked (e.g. in case of an emergency integration by another Consumer).

Aggregator Instance:

Inherits from: Device Instance Entity
EntityOwner: Principal Entity being the *Device Owner*
EntityOperator: Consumer Operator mandated by the *Device Owner*

The Aggregator Instance is a specialization of the Device Instance. Although Aggregators are not shared or provisioned, they need to be represented by the Device Directory in order to have a consistent model of device integration as discussed in Section 6.2.2. The Root Domain Operator must reserve an appropriate range of IDs for Aggregators, because the overall ID range is shared with the Device Instances. Aggregator Instances can but do not have to maintain the *DeviceInstanceState* property. This is only required if the Aggregator is a

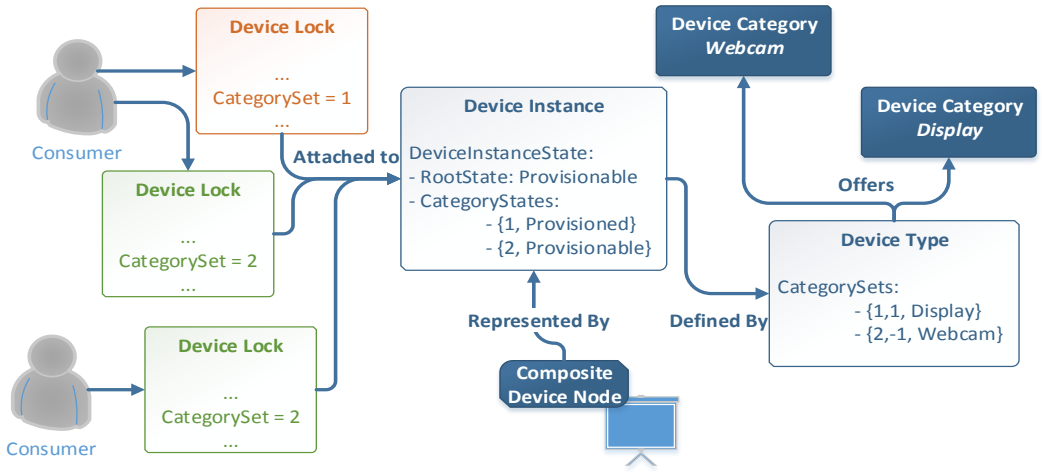


Figure 4.4.: Relationship of functional device classes, category groups and device locks.

> Composite Aggregation Node and is willing to add its embedded devices to the Device Cloud. The *CategorySets* property of the corresponding device type does not allow for provisioning of embedded devices by default. Thus, this feature has to be explicitly activated by overriding the global *CategorySets* property using the respective Device Instance property.

The Device Lock is the representative of the core capability of the Device Cloud. It allows provisioning devices to *Device Consumers*, which can refer to Consumers or Consumer Operators (i.e. Intra-Operator provisioning). The Consumer Operator managing the device can grant the lock to its own Consumers or to other Consumer Operators. In case the lock is granted to another Operator, the device will usually be further provisioned to Customers of that Operator. It is important to note, that there is a one to one relation between a Device Lock and an Aggregator. In case of devices offering non-exclusive locks, this results in the requirement to create multiple locks for each requesting Consumer. Accordingly, in case of Intra-Operator provisioning, the consuming Operator has to request one lock for each Consumer the device is provisioned to. Figure 4.4 depicts the relationship between the functional device classes and the *CategorySets* and *DeviceInstanceState* properties. In case of exclusive transducer devices, *CategorySets* will contain exactly one entry with a lock-amount of 1. In case of non-exclusive transducer devices, *CategorySets* will contain one entry with a lock-amount of -1 or >1. Composite transducer devices are modelled using multiple *CategorySets* entries.

A Consumer Operator acting as the *Device Consumer* for a particular device is not

allowed further provisioning the device to other Operators. This restriction is made due to the requirement of withdrawing a valid device lock (see Section 4.3). Otherwise, the Operator holding the *Device Operator* role would have no knowledge about the consuming Operator and a withdrawal request would be subject to a cascaded operation. Moreover, because the provisioning negotiation is based on peer to peer interaction between Operators, the managing Operator would not be involved in the provisioning decision and policies could be violated (e.g. a device could be granted to an Operator blacklisted by the managing Operator).

Permissions constraining the provisioning process can be added by *Device Owners* as configuration entries to the Device Instance Entity. If the *Device Owner* wants to prevent the *Device Operator* from modifying these permissions, the *EntityOperator* property of the configuration entry can be left blank. Permissions applicable to the interaction between Operators or constraining the provisioning of a whole class of devices (i.e. Device Category), which usually spans devices of several *Device Owners*, can be managed by each Consumer Operator. The actual mechanisms to implement such global permission sets or to implement features like role based access control (i.e. grouping of Consumers) are out of scope of this work.

Besides permissions, information regarding the *Device Target* can be added to the configuration of a Device Instance. The *Device Target* role is not explicitly modelled by an entity, because of the huge variety of possible targets. As defined in Section 4.1, a target may belong to a human being, a physical thing or, even an area or an environment. However, Operators may require Consumers to specify a valid target upon request to a device and include this information in their provisioning decisions. A Consumer Operator could, for instance, apply a policy that each request for a medical device contains a valid patient identifier. In general, if no target is specified, the *Device Consumer* requesting access will be treated as the target. In case of a non-exclusive or composite device and support for multiple *Device Targets*, the information can be additionally added to the Device Lock and overwrite the corresponding Device Instance entry. Whether the *Device Target* entry is kept private or is accessible by all Device Cloud participants depends on the privacy requirements of the application domain.

The Platform Modules available can be basically classified into core modules, device integration modules and aggregation modules:

General Platform Module Classification

Core Platform Module:
Core modules provide the core functionality of the Device Cloud Middleware and belong to the Device Cloud infrastructure itself. These modules are usually not added by vendors or system integrators. However, they are stored in the Device Directory for the purpose of simplifying and customizing the Device Cloud Middle-

ware deployment (i.e. bootstrapping) process, which is described in Section 6.2.

Device Integration Platform Module:
Device integration modules refer to discovery or device driver modules. These modules are usually tightly coupled to device types based on the Device Category.

Aggregation Platform Module:
The third class refers to modules that perform manipulation on the data received from a device and are executed by a separate part of the middleware. Examples include data transformation modules or aggregation modules. Processing of a data stream often involves several aggregation, manipulation and transformation steps. Therefore, Platform Modules of this class define input and output data formats (e.g. ISO/IEEE 11073 or HL7 for medical applications) to simplify the process of module composition. A Consumer profile is used to specify the set of modules required for a certain device or application as well as their composition. This has to be done manually and is described in Section 6.1.2. However, the specified input and output formats allow automatically injecting transformation modules, if necessary. The input and output format properties are usually ignored in case of modules belonging to the other classes.

4.4.3. User Directory Entities

Entities maintained by the User Directory are basically introduced to model authentication, access control and accounting. Hence, these entities inherit from the Principal Entity. The identity of a User Directory entity is only valid within the scope of the corresponding domain (i.e. Domain Operator). An entity spanning multiple domains, needs to be registered with the Root Domain. An example are Operators that can negotiate a device provisioning between domains. Note that User Directory entities need to have a representation in the Device Directory. Otherwise setting the *EntityOwner* and *EntityOperator* properties would be impossible.

User Directory Entities

Aggregator:

Inherits from: Attachment Entity, Configurable Entity, Principal Entity, Location Tagged Entity

The Aggregator entity is used to authenticate an Aggregator within the context of a domain.

Consumer:

Inherits from: Attachment Entity, Configurable Entity, Principal Entity, Location Tagged Entity

The Consumer entity is used to authenticate the Consumers within the context of a domain and link additional information such as the Consumer Profile.

Consumer Profile:

Inherits from: Attachment Entity, Configurable Entity

The Consumer profile describes the set of devices required by the Consumer and the set of aggregation layer modules required by the Consumer's applications to process the data recorded from the devices. It consists of entries that specify a particular Device Category as an input and attach a set of Platform Module compositions. Device integration modules like device drivers are not part of the profile because they are deployed automatically. Entries in the Consumer Profile are usually added manually or automatically by applications that require certain inputs. A physician, for example, could add an entry to the profile of his patient that specifies to integrate a blood pressure sensor and apply a transformation to the HL7 data format in order to be interpretable by its clinical information system. However, this assumes that the physician can gain access to the Consumer Profile resource.

Operator:

Inherits from: Attachment Entity, Configurable Entity, Principal Entity

The Operator entity is used to authenticate an Operator and store contact information. The following properties are defined:

- *ProtocolURI*: Defines the URI of the communication protocols used to interact with the Operator.
- *DomainOperator*: Flag indicating whether the entity refers to Domain or a Consumer Operator.

Vendor:

Inherits from: Attachment Entity, Configurable Entity, Principal Entity

The Vendor entity is representing knowledge contributing entities being able to be authenticated by the IAM service of a Domain Operator. Examples are Device Vendors or System Integrators.

Linking the Consumer Profile to the User Directory (i.e. the Domain Operator) ensures

that only one profile exists within one domain, even if multiple Consumer Operators serve a single Consumer. Thus, synchronization is simplified. The disadvantage of this approach is, that the Domain Operator has to maintain knowledge that goes beyond authentication and protecting access to resources. If a Consumer participates in several domains and hence has several Consumer Profiles, the synchronization between them is not defined. The *PermissionSet* property is used to define which Consumer Operators can gain access to a Consumer's Profile.

Operator Entities and Vendor Entities are usually managed by the Root Domain Operator because they interact with the root domain. In contrast to Consumer and Aggregator Entities, which are only valid within the context of a certain domain, Operator and Vendor Entities are valid within the context of the whole Device Cloud. Managing Consumer and Aggregator Entities is partitioned and carried out by Domain Operators because a central, overall IAM instance would not be scalable and the amount of Operator and Vendor entities is far below the amount of Consumers and Aggregators.

4.4.4. Management Service Entities

Entities maintained by the Management Services of a Consumer Operator are used to model, monitor and supervise the device provisioning process.

Management Service Entities

Aggregator Agent:

Inherits from: Attachment Entity, Configurable Entity
Maintained by: Aggregator
EntityOwner: Principal referring to an Aggregator
EntityOperator: Consumer or Consumer Operator Entity

The Aggregator Agent is the virtual representation of an Aggregator. It is used by the Device Cloud Middleware to interact with its Domain Operator and a set of Consumer Operators.

- *Bound*: Indicates whether the Aggregator is statically bound to one Consumer or can serve multiple Consumers.

Consumer Agent:

Inherits from: Attachment Entity, Configurable Entity
Maintained by: Consumer Operator Management Services
EntityOwner: Consumer Entity
EntityOperator: Consumer Operator Entity

The Consumer Agent Entity is the virtual representation of a Consumer and its Consumer Profile in scope of a Consumer Operator. Thus, several Consumer Agent entities corresponding with the same Consumer Entity can exist. Synchronization is achieved through transactional access to the singleton Consumer Profile Entity provided by the Domain Operator.

Operator Agent:

Inherits from: Attachment Entity, Configurable Entity
Maintained by: Consumer Operator Management Services
EntityOwner: Consumer Operator Entity
EntityOperator: Consumer Operator Entity

Similar to the Consumer Agent, the Operator Agent is the virtual representation of a Consumer Operator. An Operator Agent is used in case the provisioning and the resulting integration has to be supervised by the *Device Owner* or *Device Operator*, which is discussed in Section 5.1.

The Consumer Agent is involved in each device provisioning process, regardless whether the Aggregator is performing a device integration request or a device integration offer. Operator Agents are only used in case the provisioning requires supervision (e.g. a forced withdrawal due to an emergency is possible). Details about the behaviour of Consumer Agents, Operator Agents and Aggregators during the device integration process are discussed in Section 5.1 and Section 5.2.

The Management Services of a Consumer Operator may internally maintain a representative of an Aggregator Agent for management purposes. It could be used to define the bootstrap configuration (i.e. which core modules of the middleware are to be deployed) and to monitor which devices and which Consumers are currently linked to an Aggregator. However, it is not defined how a Consumer Operator may implement such measures. The *EntityOperator* property of an Aggregator Agent defines, which Principal legally owns and therefore operates the Aggregator. If the *EntityOperator* refers to a Consumer (i.e. the Aggregator is owned by an end-user), the *Bound* property must be true.

Aggregator Classification

Bound Aggregator:
A bound Aggregator statically serves one Consumer. The corresponding Consumer Profile is linked to the Aggregator Agent by connecting to the Consumer Agent hosted by the Consumer Operator. However, if the Aggregator device is owned by the consumer (i.e. a private device) and the Consumer interacts with several Con-

sumer Operators, the Aggregator Agent has to interact with multiple Consumer Agents. Thus, the Consumer has to define a priority ordering, which allows the Aggregator Agent to chose a Consumer Agent upon discovery of a device. Aggregator agents may provide a rule engine or a negotiation mechanisms that allows choosing a Consumer Operator based on the conditions (e.g. price) or QoS offered, but this is out of scope of this thesis. Bound Aggregators can be used to establish a relation between a device and the *Device Target* because they are usually physically operated by a Consumer.

Unbound Aggregator:
An unbound Aggregator is always owned by a Consumer Operator and can serve multiple Consumers. Similar to the bound Aggregator, this results in multiple Consumer Agents being linked to the Aggregator Agent. Hence, a priority flag has to define the order in which each Consumer Agent is notified about the presence of the device. The process of determining the priority depends on the application scenario and is not further specified here. A simple approach would be to use FIFO ordering.

The *EntityID* property of the Agents is equal to the *EntityID* property of the corresponding Principal Entity. For example, an Aggregator Agent will have the same *EntityID* as the corresponding Aggregator Entity stored in the User Directory.

5. Device Cloud – Security & Interactions Concept

Contents

Based on the Entity Model, this chapter will discuss the security requirements arising out of the Domain based entity distribution and introduce a security model allowing for authentic interactions between the Device Cloud actors. An interactions model will depict communication protocols, message flows and states required to accomplish the provisioning of devices.

5.1. Security Model

Apart from the general issues discussed in Section 2.5, the Device Cloud concept leads to five major security and privacy requirements:

1. Interacting participants need to be able to proof their identity. The source of a triggered interaction needs to be authentic and unambiguous in order to ensure the validity of the data maintained by the Device Directories.
2. Device access tokens need to be authentic. Only an actor holding the *Device Owner* role or a delegate (e.g. the *Device Operator*) shall be able to create a valid access token.

3. Private knowledge about devices (e.g. security credentials like Bluetooth PINs) needs to be kept confidential. Neither a *Device Consumer* nor an intermediary Operator shall be able to access or modify private knowledge.

4. As discussed in Section 4.3, the *Device Owner* or the *Device Operator* need to be able to withdraw a valid device access token in order to re-provision the device in case of emergency.

5. Integrity and confidentiality of Consumer related device data (i.e. data generated by a device currently provisioned to a Consumer) need to be ensured if required by the Consumer.

5.1.1. Trusted Platform

Solution approaches to these requirements are based on the foundational assumption that the Device Cloud Middleware hosted on the Aggregators as well as the Domain Operators are trusted participants and offer a trusted platform. In case of Domain Operators, which perform operations similar to those of an authentication server in Kerberos, trust is a necessary precondition. In case of the Device Cloud Middleware, trust can be achieved by mechanisms described in the context of OSGi security. Basically, the middleware consists of a set of core Platform Modules as introduced in Section 4.4.2. These core modules provide the execution environment for all other Platform Modules and are managed and delivered by the Root Domain Operator. It is assumed that all core modules are signed by the authority operating the Root Domain. Thus, the Device Cloud Middleware is able to authenticate the signer and to ensure, that a core module has not been modified. Regular Platform Modules need to be signed, too. Either the Root Domain Operator signs a Platform Module in charge of the originator (i.e. delegation model) or the modules are signed by the participating Device Vendors and System Integrators themselves. The first approach requires the Root Domain Operator to manually check and release each module, while it simplifies the permission management to be employed by the Device Cloud Middleware. The second approach allows for more fine grained permission sets, based on the actual originators of Platform Modules. However, Consumer Operators are required to manually define the permission sets for their belonging Aggregators, which may not be feasible due to the huge amount of Device Vendors and System Integrators possibly participating in the Device Cloud. As a remedy, a hybrid approach using certification chains and trusted Device Vendors could be used. In general, it has to be ensured that only properly signed and trusted core modules are deployed and that the Device Cloud Middleware is able to authenticate a Platform Module and apply permissions based on its origin.

Regardless of the signing approach used, OSGi provides the Conditional Permission Admin Service [120] to manage the permission of a bundle (i.e. core or Platform Module). Conditions are used to decide whether a certain set of permissions is applicable to a bundle, which can include the signer of a bundle or user-defined conditions, like the type of a Platform Module (e.g. device drivers may require different permissions than data transformation modules). A permission, for instance, covers the ability to access file or network resources or to manage other services within the middleware. Thus, constraining Platform Modules to a minimal set of necessary permissions (e.g. a device driver cannot open network sockets to hosts outside of the local subnet or data transformation modules cannot access the network at all) further mitigates the possibility of potential malicious bundles performing unintended operations.

5.1.2. Authentication and Authorization

Based on the Identity and Access Management (IAM) capabilities offered by the User Directories, the Device Cloud performs authentication and authorization of its Principal Entities. The User Directory of a domain can be envisioned as a Kerberos Realm. It is important to mention that this does not imply that the Kerberos protocol is used for authentication within the Device Cloud. However, similar to Kerberos a trust relationship between the domains (or at least the root and its child domains) needs to exist because provisioning interactions can span domains.

As mentioned, Principal Entities either have a globally valid identity (Operators) or an identity only valid in the context of a domain (Consumer, Aggregator). Principals with globally valid identities employ interactions that cross domains and therefore need to be authenticated not only within their domain. Thus, these entities are managed by the User Directory of the Root Domain Operator. The Root Domain needs to be trusted by other domains to allow Operators to be authenticated by regular domains and access their Device Directory or communicate with other Consumer Operators. This is required in case of Intra-Operator provisioning (i.e. a device is provisioned to another Operator). Staying with the Kerberos terminology, the User Directory stands for the Authentication Server (AS) and the Device Directory for the Ticket Granting Server (TGS).

An even more appropriate terminology, fitting the general entity properties defined in Table 4.1, is given by the OAuth2.0 authorization protocol (see Section 2.5.1). OAuth2.0 defines the roles resource owner, resource server, client and authorization server. A client willing to access a resource protected by the resource server needs to present a valid access token. Only the resource owner can issue the token. Therefore, the authorization server has to authenticate the resource owner and issue, on behalf of the resource owner and respecting its policy, an access token to the client. As shown in Figure 5.1, this model can be easily mapped to the Device Cloud and the process of provisioning. In simplified

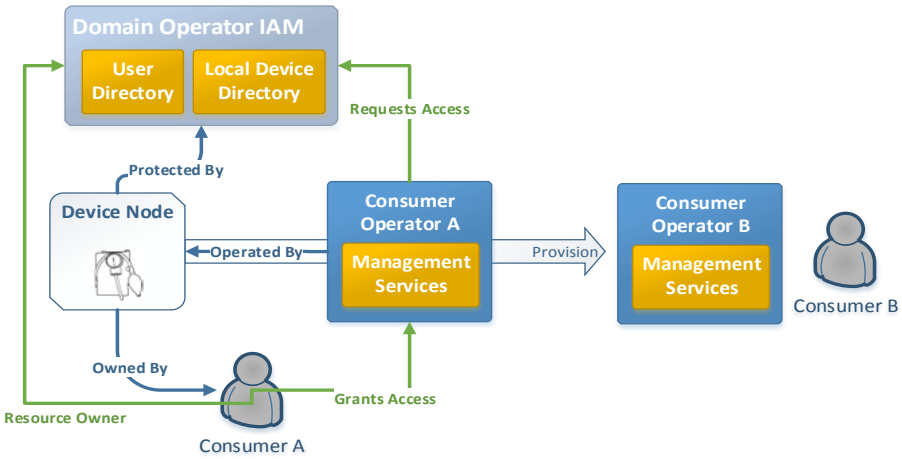


Figure 5.1.: Consumer Operators require permission of the Device Owner to provision devices.

terms, provisioning a device requires that the Consumer Operator, being responsible for provisioning and granting access locks, gets access to a Device Instance entity and modifies its state. As mentioned, all device related entities are maintained by the Device Directory and each entity has an *EntityOwner* property. Hence, the Device Directory is a resource server that protects access to device related entities based on the *EntityOwner* property. Before being able to modify the state, the Consumer Operator (i.e. the client) needs to be authorized by the *EntityOwner* to access a Device Instance. Therefore, the *EntityOwner* has to instruct the User Directory (authorization server) to issue an access token for the Consumer Operator. Issuing an access token can be twofold: either the *EntityOwner* grants a temporally limited access token like defined by OAuth2.0 or unlimited access is granted by utilizing the *EntityOperator* property. It is assumed, that both the *EntityOwner* and the Consumer Operator have been authenticated in advance.

Authentication means to employ proper mechanisms to prove the identity a principal claims to have. A principal must be characterized through well-defined properties in order to allow for an unambiguous identification [46]. Most authentication protocols are based on knowledge like shared secrets (e.g. passwords). The Device Cloud security concept is based on asymmetric encryption (i.e. a public and a private key for each Principal entity). This technique can be used along with several authentication concepts (e.g. Challenge-Response methods) and specific protocols (OpenID Connect) and additionally allows ensuring integrity (i.e. using signatures) and confidentiality (i.e.

using encryption). The public key is stored by the trusted Domain Operator which is responsible for the Principal while the private key remains private to the Principal.

In order to obtain a key pair, a Principal Entity is required to register with a Domain Operator. It is assumed that a Root Domain exists and that each Principal knowns the authentic public key of the corresponding Operator. The registration of Operators and Vendors is not further specified because in contrast to Consumers or Aggregators, this process is conducted rarely, usually involves legal contracts and does not need to be automated. However, an Operator or Vendor registration must result in the creation of the respective Principal Entity, which is, together with the generated public key, maintained by the Root Domain Operator. The following behaviour is defined for the remaining Principal Entities:

Principal Registration

Aggregator:
An Aggregator device becomes a Device Cloud Aggregator entity by deploying the Device Cloud Middleware to it. The bootstrap process, described in Section 6.2.1, is basically initiated by instantiating a bootstrap core module to the Aggregator device (i.e. the Aggregator Agent). Before loading the Core Platform Modules, the Aggregator Agent has to register itself with the Domain Operator. An appropriate request (within an encrypted session - e.g. using TLS) to the User Directory has to contain the public key generated by the Aggregator Agent and the ID corresponding to the *EntityOperator*. The User Directory assigns an ID, creates the Aggregator entity and sets the *EntityOperator* property (the ID needs to be obtained from the Root Domain because it belongs to the same range as the Device Instance ID – see Section 4.4.2). As a result, the ID is returned and the Aggregator Agent can start communicating with other participants (usually the first step will be to load the core modules from the Device Directory). Additionally, the User Directory takes care of registering the Aggregator Instance with the Device Directory. Registration of an Aggregator Agent presumes that the principal corresponding to the *EntityOperator* is known by the User Directory. Note that in case of Aggregator entities, the *EntityOperator* property usually refers to the owner of the respective Aggregation Node device.

Consumer:
Similar to the Aggregator, the Consumer has to register with the User Directory (e.g. using a web based registration form). How and where the Consumer stores its private key is not defined here, because Consumers can utilize several Aggregators and even the Consumer Agents can concurrently exist at several locations. Moreover, Consumer Agents are hosted by Consumer Operators and hence are not appropriate for managing the private key.

Section 5.1.6 will discuss further details of the principal authentication. Once a principal has been authenticated, its permission to access an entity is modelled by the general

properties of the entity itself. Thus, the Device Directory establishes a dynamic access control matrix with $M_t : S_t \times O_t \rightarrow 2^R$. Following access modes $r \in R$ are defined:

- *Read-Only*: read access to an entity
- *Read-Write*: read and write access to an entity
- *Append*: permission to append objects (i.e. other entities) to an entity
- *Control*: permission to grant and revoke permissions for an entity
- *Create*: permission to create an entity of a particular type
- *Delete*: permission to delete an entity

The objects $o \in O_t$ refer to the entities (and each corresponding property) maintained by the Directory at time t. The subjects $s \in S_t$ refer to the principals known to the User Directory. Unless specified otherwise by the *PermissionSet* property, the global access policy according to the definition of general entity properties (see Section 4.4.1) is:

Global Entity Access Policy

Given $o \in O_t$ and $s \in S_t$, while s refers to an entity:

If $s = o.EntityOwner$:

- *Read-Write* – except for the *o.EntityType* and *o.EntityID*, which are immutable after initial assignment (Read-Only in this case)
- *Append*
- *Control* – i.e. setting *o.EntityOwner*, *o.EntityOperator* and, *o.PermissionSet*
- *Delete*

If $s = o.EntityOperator$:

- *Read-Only* – for all general properties except the *o.EntityVersion*
- *Read-Write* – for the *o.EntityVersion* and all other properties not belonging to the group of general ones
- *Append* – (e.g. a Device Lock attached to a Device Instance)

If $s \neq o.EntityOwner \wedge s \neq o.EntityOperator$:

- *Read-Only* – if *o.PrivateEntity* is false
- *Append* – the *PrivateEntity* property of the attached entity must be true

Regarding the *Create* permission, the following policy is applied based on the entity type:

- *Device Instance, Device Type, Device Category, Aggregator Instance* – s = Domain Operator or Vendor
- *Device Lock* – s = Consumer Operator
- *Platform Module* – s = Domain Operator, Consumer Operator or, Vendor
- *Aggregator, Consumer, Operator, Vendor* – s = Domain Operator
- *Consumer Profile* – s = Consumer or Consumer Operator

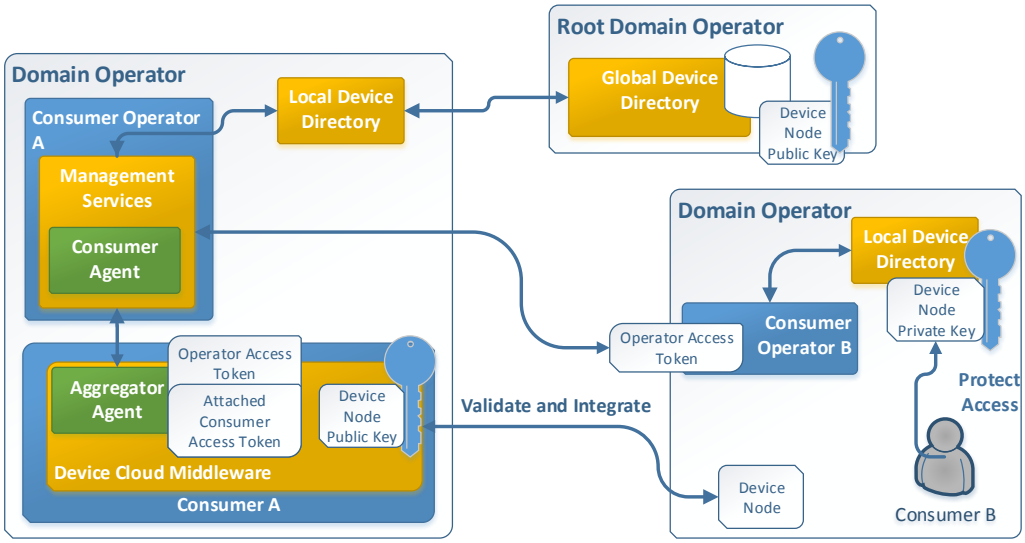


Figure 5.2.: Access token used to validate the Consumer's permission to integrate a device.

The following entity types define additional permissions:

- *o.EntityType = Device Instance:*
  - *o.PublicKey:* – *Read-Only* for all s ≠ *o.EntityOwner*
  - *o.PrivateKey:* – *Read-Write* if s = *o.EntityOwner* and *Read-Only* if s = *o.EntityOperator*, no access otherwise
- *o.EntityType = Device Lock:*
  - *o.Aggregator:* – *Read-Write* if *o.OperatorLock* is true and s = *o.LockingEntity*

Device Cloud implementations may further refine this policy. An E-Health use case may for instance require a more restrictive policy. As already mentioned, the authentication and authorization protocol (e.g. OAuth2) may introduce access tokens that take precedence over the policy specified by the Device Directory.

5.1.3. Device Access Token

As shown in Figure 5.2, device access tokens are used to validate that a Consumer is allowed to integrate a device. In other words, using the device access token, the Device

Table 5.1.: Device access token properties

Property	Description
DeviceID	The EntityID of the Device Instance the token belongs to.
LockID	The EntityID of the Device Lock the token belongs to.
Issuer	The EntityID of the Operator issuing the token.
Validity	Refers to the validity property of the Device Lock entity.
CategorySet	The set of Device Categories this lock is valid for (e.g. if a device offers multiple categories).
OperatorManaged	A flag denoting whether the device integration has to be supervised by the issuing Operator. This property is important for withdrawal of access tokens and distribution of private knowledge.
OperatorToken	Indicates whether this token was issued to a Consumer Operator or a Consumer (i.e. Operator Token or Consumer Token). If OperatorToken is true, the properties PublicKey and EmbeddedToken become mandatory.
PublicKey	Mandatory in case of an OperatorToken. Refers to the public key of the Consumer Operator this token was granted to.
AttachedToken	Mandatory in case of an OperatorToken. Used by an intermediary Consumer Operator to further re-provision the device to its Consumers (i.e. further specify the validity). The attached token is added by the intermediary Operator and therefore not part of the signature.

Cloud Middleware is able to determine that a device was provisioned to the *Device Consumer* by the *Device Owner* or an entity authorized to do so (e.g the *Device Operator* mandated by the *Device Owner*). Therefore, additionally to Principal Entities, there is a private and a public key for each Device Instance. The public key is stored by the Global Device Directory (i.e. Root Domain) and is thus accessible by all Device Cloud Participants. The private key is stored by the Domain Operator the device corresponds to and is only accessible by the *Device Owner*. The *Device Owner* can authorize a *Device Operator* to access the private key. Thus, it is ensured, that a device access token signed with the private key of the corresponding Device Instance was issued by the responsible *Device Operator* or *Device Owner*.

Table 5.1 lists the properties of an access token. It has to be distinguished between access tokens issued to a Consumer Operator and access tokens directly issued to a Consumer. If an intermediary Consumer Operator is involved in the provisioning of a device, the validity of the token usually needs to be modified after creation of the initial token. Otherwise, re-provisioning the device to several of its customers would not be possible for the intermediary Operator. Hence, the capability of attaching another token to an existing one is given. However, the Device Cloud Middleware still needs to be able to verify the authenticity of the token (i.e. only the *Device Owner* or *Device Operator*

can be the issuer). Therefore, in case an Operator token is issued, the possibility to include the public key of the Consumer Operator that further provisions the device to its Consumers, is given. The public key can be used by the Device Cloud Middleware to verify the attached token. The validity property of the embedded token must not violate the original validity. Only one token can be attached. Otherwise, the constraint, that an Operator having requested an access token from another Operator cannot further provision the device to other Operators, could be violated (see Section 4.4.2).

Besides issuing the token, a Device Lock entity is created. Device Lock entities are subject to immediate synchronization between Device Directories. In case an Operator Token is issued between different domains, the issuing Operator has to propagate the corresponding Device Lock to the respective domain. In return, the intermediary Operator, that requested the access token, has to propagate changes back to the origin (i.e. the domain of the issuing Operator). If the intermediary Operator re-provisions the device, it must modify the *Aggregator* property of the Device Lock. This is required for the provisioning interactions described in Section 5.2. As discussed, the Domain Operator of the intermediary Consumer Operator ensures, based on the access policy of the Device Directory that only the desired intermediary Operator is able to modify this property.

5.1.4. Device Access Withdrawal

Reliable withdrawal of an access token requires that the issuing Consumer Operator is able to instruct the Device Cloud Middleware to release the device, even if the token is still valid. The challenge is, that the issuing Operator usually does not has control over the Device Cloud Middleware if the intermediary Consumer Operator scenario is considered. Using the intermediary Consumer Operator as a proxy to issue the withdrawal request, requires its consent and can introduce delay, which might be inappropriate for emergency cases like given in the E-Health domain. Therefore, device provisionings classified to allow forced withdrawals require that the Operator managing the device supervises the integration. Additionally, since it is assumed that the Device Cloud Middleware is a trusted platform and all its core modules are signed by the Root Domain, the behaviour, in contrast to the Management Service implementation of the Consumer Operator, is predictable. The requirement to supervise a device integration must be acknowledged by the Consumer (or its Consumer Operator) before the provisioning is conducted. Besides withdrawing tokens, supervisioning a device integration also allows fulfilling requirement no. 3 mentioned above (confidentiality of private data required for the integration - e.g. a Bluetooth PIN).

The *OperatorManaged* property of the access token indicates that integrating the device must be supervised. If the Aggregator Agent receives such a token, it must establish an

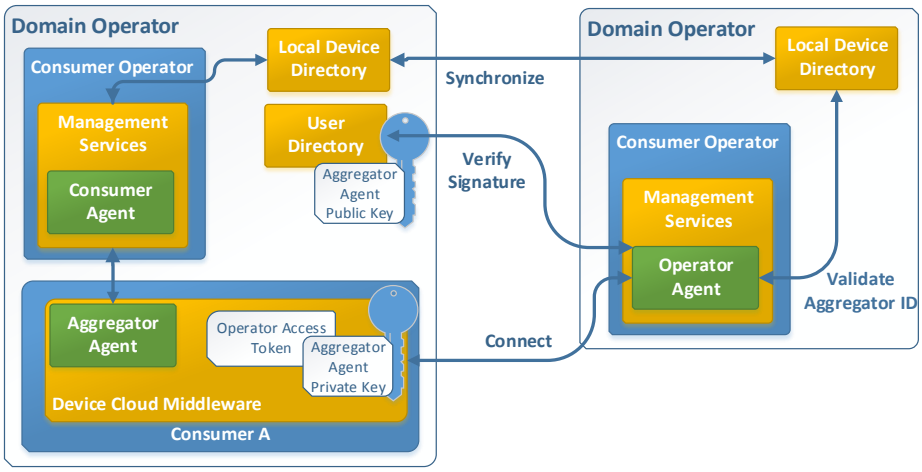


Figure 5.3.: Cross-domain Aggregator Agent authentication conducted by Operator Agent.

encrypted session to the issuing Consumer Operator (i.e. the respective Operator Agent) using the *Issuer* property of the token, as shown in Figure 5.3. This operation can cross domains. However, the Operator Agent needs to prove that the Aggregator Agent is authentic and was mandated by its Operator to integrate the device. After having authenticated (e.g. using the protocol discussed in Section 5.1.6), the Aggregator Agent resubmits the access token to the Operator Agent in order to verify that it is permitted to integrate the device. Based on the token, the Operator Agent is able to identify the corresponding Device Lock and can check, whether the declared *Aggregator* property matches the requesting one. As mentioned in the previous section, the Device Lock is immediately synchronized and an intermediary Operator is required to modify the *Aggregator* property if it re-provisions the device to one of its Consumers.

After the authenticity and the permission of the Aggregator Agent to integrate the device has been proved by the Operator Agent, the Aggregator is allowed to request private data required for device integration. Based on an application specific withdrawal policy, the Aggregator Agent has to keep the connection alive in order to allow for immediate withdrawals or poll the Operator Agent using a pre-defined interval. Once the token becomes invalid, the private data received must be deleted immediately.

5.1.5. Confidentiality of Consumer Data

Employing appropriate mechanisms to ensure confidentiality and integrity of Consumer data as introduced by requirement no. 5, presumes understanding the possible data flows as well as the placement of the data sinks (i.e. applications). The Device Cloud defines two types of applications: Aggregation Platform Modules which are hosted by the Device Cloud Middleware and applications that are hosted externally. The latter class can refer to any generic type of data sink (e.g. a database or a Cloud service). Regardless of the application type, data streams will never be routed through the Device Cloud backend infrastructure. Thus, only the Device Cloud Middleware is involved in the processing of Consumer data flows.

In case of external applications, data flows have to be routed towards sinks outside of the Device Cloud infrastructure. The actual mechanisms used for this purpose (e.g. publish-subscribe systems) are not defined. However, in general, applications willing to receive data from devices integrated by the Device Cloud have to register with the Consumer's profile, which basically means that the Consumer authenticates with its Domain Operator and adds appropriate entries to its profile as shown in Figure 5.4. Such an entry has to contain an output module, which belongs to the class of Aggregation Platform Modules. The output module links the data streams to the external application. Hence the output module is responsible to conduct proper integrity and confidentiality measures. Since only the Consumer has knowledge about the supported security protocols of the application, the Consumer has to define these measures by choosing a fitting output module and providing required configuration parameters to the module.

In case of internal applications, it has to be considered that, as discussed above, the Device Cloud Middleware is assumed to be a trusted platform. However, due to the possibility of adding private Platform Modules to a Local Directory, malicious code could be injected. Since this code is not properly signed, the Device Cloud Middleware has to be configured to accept and execute such modules. Moreover, the permission system can be used to restrict the permissions of such a module as much as possible. In general, using signatures and encryption within the Device Cloud Middleware would introduce major obstacles regarding efficiency and resource consumption while providing minor benefits. As the middleware is deployed on one single Java Virtual Machine, each Aggregation Platform Module would have to decrypt, sign and encrypt the data again, after it has been modified. Therefore, encryption and signatures within the middleware would not offer enhanced security. If isolation of Consumer data is required by an application scenario, a Device Cloud Middleware instance would have to be statically allocated to a single Consumer. Isolation techniques like sandboxes or virtualization, as introduced in Section 2.2.1, could be used to deploy several middleware instances on one Aggregator device.

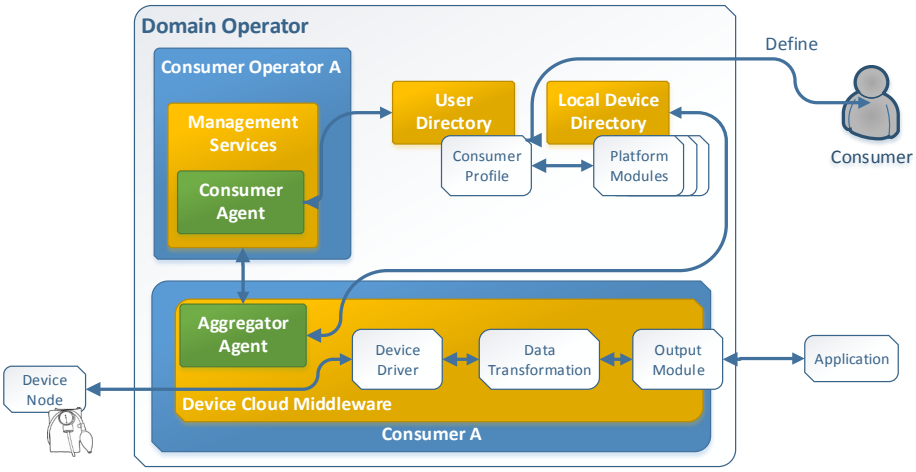


Figure 5.4.: Application integration through a data flow defined by the Consumer Profile.

5.1.6. Discussion

In order to evaluate the feasibility of the domain based public key infrastructure in terms of authentication, the following principals (according to Section 4.4.3 and Section 5.1.2) are defined with respect to the communication links possible (according to Section 5.2):

- **R:** the principal referring to the Root Domain Operator
  - $R \rightarrow D$: Synchronization with a Local Domain (i.e. Local Device Directory)
- **D:** a principal referring to a Domain Operator (public key is known to/maintained by **R**)
  - $D \rightarrow R$: Synchronization with the Root Domain (i.e. Global Device Directory), accessing the IAM services of the Root Domain
  - $D \rightarrow D$: Synchronization with another Local Domain (i.e. Local Device Directory)
- **B:** a principal referring to a Consumer Operator (public key is known to/maintained by **R**)
  - $B \rightarrow B$: Intra-Operator provisioning, i.e. requesting access to a device managed by another Consumer Operator
  - $B \rightarrow D$: Accessing the Device Directory, accessing the IAM services of the Local Domain
  - $B \rightarrow R$: Accessing the IAM services of the Root Domain

- **A:** a principal maintained by **D** (i.e. Aggregators and Consumers whose public key is known to/maintained by **D**)
  - $A \rightarrow B$: Accessing the Consumer Operator services
  - $A \rightarrow D$: Accessing the IAM services of the Local Domain, accessing the Consumer Profile (in case A refers to a Consumer)

As already discussed, it is encouraged that an implementation of the Device Cloud relies on a mature protocol for authentication and authorization (e.g. Open-ID Connect). However, in order to prove the general applicability of the domain and public key based approach, the following authentication protocol is defined on the basis of Kerberos. A correct authentication protocol should ensure that the participating principals are convinced to interact with each other and not with an attacker. The Burrows-Abadi-Needham (BAN) Logic [27] will be used to conduct a basic protocol analysis.

Authentication Protocol

(1) $A \rightarrow D : \{\{A, B\}_{K_A^{-1}}\}^{K_D}$
Request is signed with A's private key and encrypted with D's public key

(2) $D \rightarrow R : \{\{A, B\}_{K_D^{-1}}\}^{K_R}$
D checks authenticity and forwards request

(3) $R \rightarrow D : \{\{T_R, K_{AB}, B\}_{K_R^{-1}}\}^{K_D}, \{\{T_R, K_{AB}, A, Domain_A\}_{K_R^{-1}}\}^{K_B}$
R returns the generated session key and a ticket to D

(4) $D \rightarrow A : \{\{T_R, K_{AB}, B\}_{K_D^{-1}}\}^{K_A}, \{\{T_R, K_{AB}, A, Domain_A\}_{K_R^{-1}}\}^{K_B}$
D returns the session key and ticket to A

(5) $A \rightarrow B : \{\{T_R, K_{AB}, A, Domain_A\}_{K_R^{-1}}\}^{K_B}, \{T_A, A\}^{K_{AB}}$
A presents the ticket to B and tries to open a session

(6) $B \rightarrow A : \{T_A + 1\}^{K_{AB}}$
B verifies the ticket – i.e. issued by R

The shown protocol covers the most complex scenario, where A has to establish a secure communication link with B. Similar protocol(steps) can be defined for the other possible communication links. In case of, for instance $B \rightarrow B$, the steps 2 and 3 can be omitted.

Since A and B can belong to different domains (e.g. an Intra-Operator provisioning crossing domains), the protocol needs to ensure, that B can determine whether A is authentic or not, even if A is not known within the domain of B. However, it is important to mention that the protocol does not provide any knowledge regarding the authorization of A in this case. Communication between A and B crossing domains is only given by an Operator supervised provisioning (i.e. an Aggregator connects to a Consumer Operator) (see Section 5.1.4 for further details about the authorization in this case).

In order to apply BAN rules to the protocol, it has to be transformed to an idealized representation. This, for instance, includes, that all plaintext messages are removed and that all keys transferred within messages are treated as shared keys (e.g. a session key shared between A and B). It has to be noted, that the BAN logic is restricted to statements regarding mutual beliefs of the participants and does not provide statements about the confidentiality of messages.

BAN – Idealized Authentication Protocol

$$
\begin{aligned}
&(1) \quad A \rightarrow D : \{B\}^{K_A^{-1}} \\
&(2) \quad D \rightarrow R : \{B\}^{K_D^{-1}} \\
&(3) \quad R \rightarrow D : \{\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_R^{-1}}\}^{K_D}, \{\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_R^{-1}}\}^{K_B} \\
&(4) \quad D \rightarrow A : \{\{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_D^{-1}}\}^{K_A}, \{\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_R^{-1}}\}^{K_B} \\
&(5) \quad A \rightarrow B : \{\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_R^{-1}}\}^{K_B}, \{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_{AB}} \\
&(6) \quad B \rightarrow A : \{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_{AB}}
\end{aligned}
$$

Based on the idealized protocol, a set of assumptions is required. The assumptions describe the beliefs the participating principals possess before the protocol is initiated (e.g. K_A is the public key of A). Accordingly, the security characteristics to be ensured after completion of the protocol have to be defined as a set of assertions (e.g. K_{AB} is a secret session key shared between A and B). Using a set of rules offered by the BAN logic, the assertions have to be deducted from the assumptions and the protocol steps. The following assumptions and assertions are defined:

BAN – Authentication Protocol Assumptions and Assertions

Assumption 1 - Public Keys:

Assumption A targets the beliefs of the principals regarding the distribution of public keys. The first statement, for instance, declares: R beliefs that K_D is the public key of D (i.e. R knows the public key of D).

$$
\begin{aligned}
&R \mid\equiv \overset{K_D}{\mapsto} D \quad D \mid\equiv \overset{K_R}{\mapsto} R \quad B \mid\equiv \overset{K_R}{\mapsto} R \quad A \mid\equiv \overset{K_D}{\mapsto} D \\
&R \mid\equiv \overset{K_B}{\mapsto} B \quad D \mid\equiv \overset{K_A}{\mapsto} A
\end{aligned}
$$

Assumption 2 - Key Generation and Trust:

R is convinced in its capability to generate shared keys used by A and B. D and B trust in R's capability.

$$
R \mid\equiv A \overset{K}{\leftrightarrow} B \quad D \mid\equiv R \mid\Rightarrow A \overset{K}{\leftrightarrow} B \quad B \mid\equiv R \mid\Rightarrow A \overset{K}{\leftrightarrow} B
$$

Moreover, A trusts D, which means that A beliefs that D has the authority to prove the validity of a key generated by R.

$$A \mid\equiv D \mid\Rightarrow A \overset{K}{\leftrightarrow} B$$

Assumption 3 - Freshness:
D and B are convinced of the freshness of time stamp T_R. A and B are additionally convinced in the freshness of T_D and T_A respectively.

$$D \mid\equiv \sharp(T_R) \quad B \mid\equiv \sharp(T_R) \quad A \mid\equiv \sharp(T_D) \quad B \mid\equiv \sharp(T_A)$$

Assertions:
Both A and B are convinced to use a shared and secret session key. Moreover, A and B are convinced that the respective partner is convinced that K_{AB} is a shared key.

$$\begin{aligned}
(1): &\quad A \mid\equiv A \overset{K_{AB}}{\leftrightarrow} B \\
(2): &\quad B \mid\equiv A \overset{K_{AB}}{\leftrightarrow} B \\
(3): &\quad A \mid\equiv B \mid\equiv A \overset{K_{AB}}{\leftrightarrow} B \\
(4): &\quad B \mid\equiv A \mid\equiv A \overset{K_{AB}}{\leftrightarrow} B
\end{aligned}$$

In order to show assertion one ($A \mid\equiv A \overset{K_{AB}}{\leftrightarrow} B$), the BAN rules can be applied to message 4 ($\{\{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_D^{-1}}\}^{K_A}$) from the idealized protocol.

$$(1): \quad \frac{A \mid\equiv \overset{K_A}{\mapsto} A, A \triangleleft \{\{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_D^{-1}}\}^{K_A}}{A \triangleleft \{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_D^{-1}}}$$

Since A knows its own private key, it can see (decrypt) the message. Subsequently, the message meaning rule and assumption one can be applied to deduct that A beliefs that the message was sent by D.

$$(2): \quad \frac{A \mid\equiv \overset{K_D}{\mapsto} D, A \triangleleft \{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_D^{-1}}}{A \mid\equiv D \mid\sim \{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

Afterwards, the freshness and the nonce verification rule can be applied to deduct that A is convinced of the freshness of the message and furthermore is convinced that D beliefs in the message.

$$\begin{aligned}
(3): &\quad \frac{A \mid\equiv \sharp(T_D)}{A \mid\equiv \sharp\{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}} \\
(4): &\quad \frac{A \mid\equiv \sharp\{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}, A \mid\equiv D \mid\sim \{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}}{A \mid\equiv D \mid\equiv \{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}}
\end{aligned}$$

Using the belief rule, it can be deducted, that A believes, that also D believes that K_{AB} is a fresh and secret key shared between A and B.

$$(5): \quad \frac{A|\equiv D|\equiv\{T_D, A \overset{K_{AB}}{\leftrightarrow} B\}}{A|\equiv D|\equiv A \overset{K_{AB}}{\leftrightarrow} B}$$

Finally, the jurisdiction rule and assumption two can be applied to deduct the initial assertion one.

$$(6): \quad \frac{A|\equiv D|\Rightarrow A \overset{K}{\leftrightarrow} B, A|\equiv D|\equiv A \overset{K_{AB}}{\leftrightarrow} B}{A|\equiv A \overset{K_{AB}}{\leftrightarrow} B}$$

Assertion two can be shown similarly using message five of the idealized protocol:

$$(1): \quad \frac{B|\equiv \overset{K_B}{\mapsto} B, B\triangleleft\{\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_R^{-1}}\}^{K_B}}{B\triangleleft\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_R^{-1}}}$$

$$(2): \quad \frac{B|\equiv \overset{K_R}{\mapsto} R, B\triangleleft\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_R^{-1}}}{B|\equiv R|\sim\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(3): \quad \frac{B|\equiv\sharp(T_R)}{B|\equiv\sharp\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(4): \quad \frac{B|\equiv\sharp\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}, B|\equiv R|\sim\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}}{B|\equiv R|\equiv\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(5): \quad \frac{B|\equiv R|\equiv\{T_R, A \overset{K_{AB}}{\leftrightarrow} B\}}{B|\equiv R|\equiv A \overset{K_{AB}}{\leftrightarrow} B}$$

$$(6): \quad \frac{B|\equiv R|\Rightarrow A \overset{K}{\leftrightarrow} B, B|\equiv R|\equiv A \overset{K_{AB}}{\leftrightarrow} B}{B|\equiv A \overset{K_{AB}}{\leftrightarrow} B}$$

Since B now believes in K_{AB}, it can decrypt part two of message five. Hence, the message meaning and the nonce verification rule can be used to deduct assertion four ($B \mid\equiv A \mid\equiv A \overset{K_{AB}}{\leftrightarrow} B$).

$$(1): \quad \frac{B|\equiv A \overset{K_{AB}}{\leftrightarrow} B, B\triangleleft\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_{AB}}}{B|\equiv A|\sim\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(2): \quad \frac{B|\equiv\sharp(T_A)}{B|\equiv\sharp\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(3): \quad \frac{B|\equiv\sharp\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}, B|\equiv A|\sim\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}{B|\equiv A|\equiv\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(4): \quad \frac{B|\equiv A|\equiv\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}{B|\equiv A|\equiv A \overset{K_{AB}}{\leftrightarrow} B}$$

Finally, assertion three can be deducted using message 6. Note, that since A generated T_A, it is implied that A beliefs that T_A is fresh.

$$(1): \quad \frac{A|\equiv A \overset{K_{AB}}{\leftrightarrow} B, A \triangleleft \{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}^{K_{AB}}}{A|\equiv B|\sim \{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(2): \quad \frac{A|\equiv \sharp\{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}, A|\equiv B|\sim \{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}{A|\equiv B|\equiv \{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}$$

$$(3): \quad \frac{A|\equiv B|\equiv \{T_A, A \overset{K_{AB}}{\leftrightarrow} B\}}{A|\equiv B|\equiv A \overset{K_{AB}}{\leftrightarrow} B}$$

Open Issues

The security model discussed covers principals and components belonging to the Device Cloud infrastructure. Providing an overall security model would require to adapt the protocols used by the devices (i.e. the protocols used to establish a communication link between a device and an Aggregator), which is not feasible. Hence, the Device Cloud has to rely on the security measures offered by the devices themselves. The individual measures conducted by each protocol are subject to a huge heterogeneity and sometimes do not even exist. Therefore, the Device Cloud cannot ensure a higher level of confidentiality or integrity as provided by the protocol the integrated device is based on.

A related issue is given by missing mechanisms to ensure the authenticity of devices and the generated data. If an attacker is able to guess a valid Device Instance ID, the corresponding device could be simulated, which can result in malicious data inferred to the Device Cloud. This issue can be mitigated by relying on devices with appropriate security measures in case of application domains with high security requirements. Additionally, the Device Cloud could monitor, if a Device Instance is discovered while already being integrated and active at the same time (in case of non-exclusive devices this policy requires that a device is discovered twice by the same Aggregator). The issue of data authenticity for mobile sensors is further discussed by Gilbert et al. [62].

Another issue is given by the Aggregator device hosting the Device Cloud Middleware. Although the Middleware itself is assumed to be trusted, the Aggregator device is not. The individual controlling the device could try to dump the memory and thus gain access to private Device Instance data (see Section 5.1.4) or the private key of the Aggregator principal. Using the private key, a compromised Consumer Operator could, for instance, make another Consumer Operator believe that he is integrating a device and thus directly gain access to private Device instance data. Additionally, the individual controlling the device could modify the time. An accurate time is not only important for authentication protocols, it is also required to determine the validity of device access

tokens. As a remedy, the Device Cloud Middleware could rely on an external source (e.g. a NTP server) and avoid to determine the time using system calls.

Besides private data required to establish a session to a device, the data generated by a device could be accessed. In order to mitigate arising privacy issues, the Device Cloud Middleware needs to properly clean up data generated due to a device session. This not only includes data cached by Platform Modules, but additionally targets data cached by the device itself (i.e. an internal history). Thus, Device Driver Platform Modules need to ensure that an internal device cache is cleaned up, if possible. This approach is not feasible in case a device needs to maintain data internally (e.g. Holter ECGs) or a session is disconnected unexpectedly. As a solution, the Device Cloud could enforce drivers to reset the internal cache if the *Device Target* (i.e. a patient in case of E-Health applications) changes.

5.2. Interaction Model

The following sections will discuss the interactions between the different entities that are required to model the device deployment and provisioning processes. Basically, an interaction is either triggered by adding a new device to the Device Cloud or by an Aggregator discovering a device. The latter either leads to a device integration request or to a device integration offer. An integration request is triggered, when a Consumer, that is interested in the device, is bound to an Aggregator. A device integration offer is triggered if no such Consumer is bound. Before describing the interactions, the device state model as well as general communication links and protocols offered by the entities will be illustrated.

5.2.1. Device State Model

Provisioning of a device is based on the current state of the device. Figure 5.5 shows the possible states and transitions.

Device Instance States

Manufactured:
A virtual state indicating that a new Device Instance was manufactured by a vendor.

Announced:
The vendor has announced the Device Instance to the Device Cloud by adding it to the knowledge base of the Global Device Directory. This includes assigning a unique identifier and attaching the Device Instance to the corresponding Device

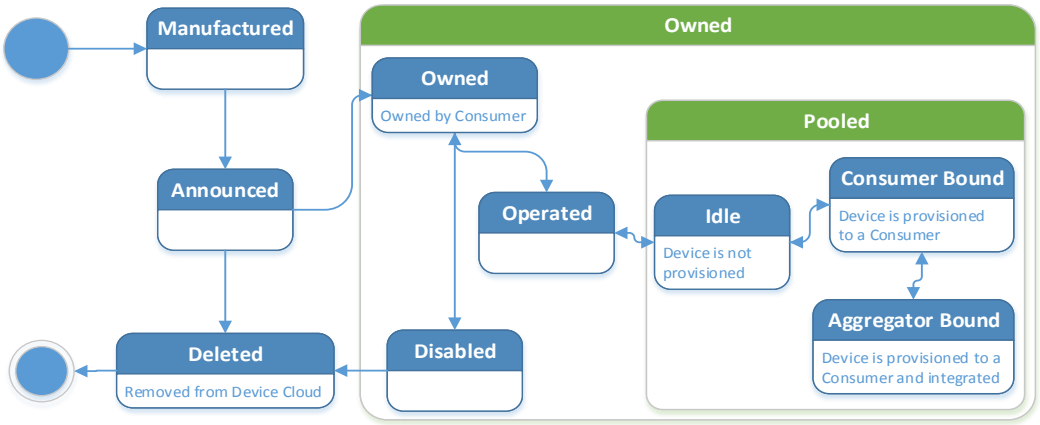


Figure 5.5.: Device Instance state machine diagram.

Type. Until this state, the *EntityOwner* of the Device Instance refers to the Vendor.

Owned:
Ensures: EntityOwner is defined
Afterwards, the device is usually sold to a Consumer or an Operator running a set of own devices. This refers to the Device Deployment Interaction discussed in Section 5.2.3. Transitioning to the Owned state means that a valid *Device Owner* was defined (i.e. the *EntityOwner* property was set).

Operated:
Ensures: EntityOperator is defined
Before the Device Instance can be added to the Federated Device Pool and provisioned, the *Device Operator* needs to be elected (i.e. the *EntityOperator* property is set), which results in a transition to the Operated state.

Idle:
Ensures: Device was added to the device pool – i.e. ready to be provisioned
While the Device Instance stays in the Operated state, it can only be used by the *Device Owner*. In order to be added to the device pool and provisioned, the *Device Owner* has to notify the *Device Operator*, which leads to a transition to one of the Pooled states. The Pooled state contains three sub-states. As long as the Device Instance is not being provisioned to Consumers, it remains in the Idle state.

Consumer Bound:
Ensures: Device Lock was created
If a Consumer successfully requested access to the device, the corresponding Device Instance transitions to the Consumer Bound state. This means that a Device Lock was created, whose *Aggregator* property was not defined yet. Since devices can

be mobile and the lock can span an unspecified period of time, multiple Aggregators can be involved in the integration process of a device bound to a Consumer. Therefore, locking and integration are separated from each other.

Aggregator Bound:
Ensures: Aggregator property of the Device Lock is defined
If the device is integrated by an Aggregator, the Device Instance transitions to the Aggregator Bound state and the Aggregator property of the Device Lock is set. This means that the Device Instance is finally provisioned and integrated and all mandatory roles defined in Section 4.1 are assigned (*Device Owner, Device Integrator, Device Operator, Device Consumer*). As mentioned in Section 4.4.2, definition of the *Device Target* role remains optional.

Disabled:
If a device has to be removed from the Device Cloud (e.g. due to being broken), it can transition to the Disabled state. This results in all attached roles, except the *Device Owner*, being removed from the Device Instance. Additionally, the device is removed from the device pool. This state can also be used to define a new owner.

Deleted:
The Disabled state allows transitioning to the Deleted state, which means that the device and its corresponding Device Instance entity are completely removed from the Device Cloud.

As already discussed in Section 5.1, provisioning a device is based on accessing and modifying its state (i.e. the *DeviceInstanceState* property of the Device Instance). According to the general access policy, the property can be modified by the *EntityOwner* and, in case it is defined, the *EntityOperator*. If several copies of the Device Instance were propagated to Local Device Directories, inconsistencies may occur. Disseminating state updates immediately among all local copies of a Device Instance would introduce unnecessary additional overhead, because only the home domain of the Device Instance (i.e. the domain equal to the *EntityDomain* property) needs to inspect the state and is allowed to make device provisioning decisions. Thus, it is claimed, that the domain referring to the *EntityDomain* property of the Device Instance must have consistent knowledge about the *DeviceInstanceState* of the Device Instance. This is straightforward in case of exclusive devices, as defined in Section 4.1. However, in case of non-exclusive or composite devices, multiple Device Locks can exist. The following rules apply to the entries of the *DeviceInstanceState* property, if the Device Instance is pooled:

The *CategoryState* properties are only activated, if the *RootState* property of the Device Instance has transitioned to the Idle state (see Section 4.4.2).

Algorithm 5.2.1 DeviceInstanceState

Require: $RootState = Idle$
  $DT \leftarrow DeviceInstance.DeviceType$
  $Locks \leftarrow \{x | x \in DeviceInstance.locks \wedge x.active = true\}$
  for all $cs \in DeviceInstanceState.CategoryStates$ **do**
    $id \leftarrow CategoryStates.CategoryGroupID$
    $activeCategoryLocks \leftarrow |\{l | l \in Locks \wedge l.CategorySet = id\}|$
    if $DT.getGategorySet(id).possibleLocks > activeCategoryLocks$ **then**
      $cs.state \leftarrow Idle$
    else if $DT.getGategorySet(id).possibleLocks = activeCategoryLocks) \wedge \exists l\{l \in activeCategoryLocks \wedge l.Aggregator = undefined\}$ **then**
      $cs.state \leftarrow ConsumerBound$
    else
      $cs.state \leftarrow AggregatorBound$
    end if
  end for

5.2.2. Communication Protocols

Figure 5.6 depicts the major communication protocols used between the entities. Communication only takes place among Principal Entities. Before entering the operational state of a communication protocol, the Auth Protocol must have been completed in order to ensure that a principal is authentic and allowed to access the resources offered by a protocol. In order to reduce complexity, the links between the participants are only shown conceptionally. Possible links established between the principals are described below:

Device Cloud Communication Protocols

Auth Protocol:

Offered by: Domain Operator – User Directory
Used by: Aggregator Agent, Consumer, Consumer Operator, Domain Operator

The Auth Protocol must be completed prior to any other protocol. It aims at authenticating a Principal Entity and providing a token, giving proof about the resources the entity is allowed to access. Each of the subsequent protocols is based on sessions and requires that a valid token is presented before entering the operational state. Based on the actual technology used to implement the protocol, a principal opening a session to a particular protocol endpoint, can either interact with its User Directory to obtain a token or generate a custom, protocol defined token and sign it using its private key. The latter case would require the protocol offering principal to validate the token by connecting to the User Directory. The

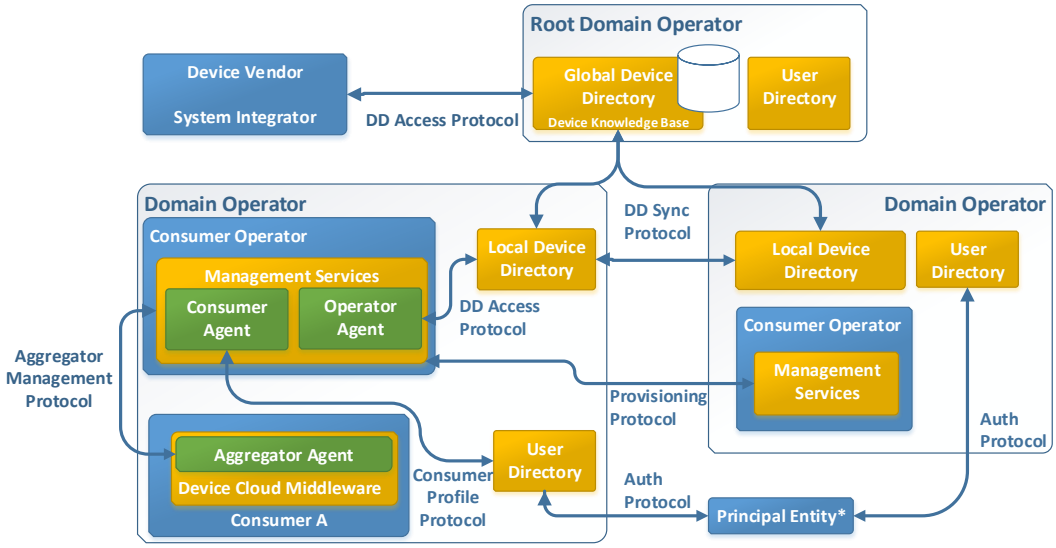


Figure 5.6.: Overview of the major communication protocols used by the entities.

first case, for instance, could be implemented based on OAuth2.0 and OpenID Connect.

DD Access Protocol:

Offered by: Domain Operator – Device Directory
Used by: Aggregator Agent, Consumer, Consumer Operator, Domain Operator

The Device Directory Access Protocol is used to add, modify or remove information stored in the device knowledge base (e.g. add or modify Device Types, Device Categories, etc.). In order to enter the operational state, a principal must have been authenticated by the User Directory of the Domain Operator (i.e. the Auth Protocol). The authorization to access resources offered by the Device Directory is modelled by the general entity properties *EntityOwner*, *EntityOperator* and, *PermissionSet* as well as the general access policy defined in Section 4.4.1.

DD Sync Protocol:

Offered by: Domain Operator – Device Directory
Used by: Domain Operator – Device Directory

The Device Directory Synchronization Protocol is used to synchronize the knowledge bases of the Device Cloud domains (i.e. Device Directories). This protocol is only applicable between Device Directories (i.e. only Domain Operator principals can invoke it).

Provisioning Protocol:

Offered by: Consumer Operator – Management Services
Used by: Consumer, Consumer Operator

The Provisioning Protocol is used to negotiate access to a device. It involves an entity acting as a *Device Operator* and an entity acting as a *Device Consumer*, while the latter one can refer, according to the concept, to a regular Consumer or to a Consumer Operator. It is basically used to decide about integration requests and integration offers, which means it is used to access or modify stateful information about a device. The protocol will either lead to a Device Lock being created or to a reject of the integration request.

Consumer Profile Protocol:

Offered by: Domain Operator – User Directory
Used by: Consumer, Consumer Operator – Consumer Agent

The Consumer Profile Protocol is an extension of the Auth Protocol because it is used to access the Consumer Profile maintained by the User Directory. This protocol can be invoked by Consumers and Consumer Operators that are allowed to access the protocol of a particular Consumer.

Aggregator Management Protocol:

Offered by: Consumer Agent, Operator Agent, Aggregator Agent – in case of incorporated Aggregator management capabilities
Used by: Aggregator Agent, Operator Agent – in case of incorporated Aggregator management capabilities

The Aggregator Management Protocol is used by the Aggregator Agent to access the Consumer or Operator Agents of a Consumer Operator. This protocol is used to trigger integration requests or integration offers. Additionally, an implementation can embed Aggregator management capabilities.

Synchronization Protocol

The DD Sync Protocol is fundamental to the provisioning capabilities of the Device Cloud. It ensures that each participant can access knowledge required to integrate and handle devices. Sometimes also referred to as replication[1], the synchronization protocol ensures that the data of the device knowledge base is properly disseminated among the Domains (i.e. the Device Directories). The consistency model used by the Device Directories is, similar to DNS, Eventual Consistency [148]. This means that a relatively high degree of inconsistency is tolerated, and it is only ensured, that updates will be propagated to all replicas, but not immediately. Moreover, there is only a small group of processes performing updates on entities and the design of the Device Cloud nearly eliminates write-write conflicts. The following types of entity-replicas are defined:

Entity Replica Types

- **Master Copy:** The *EntityDomain* property is equal to the Domain of the Device Directory. Thus, only one master copy can exist.
- **Local Copy:** The *EntityDomain* property is not equal to the Domain of the Device Directory.

According to Section 5.1.2, Read-Write access is very restrictive. In most cases, only the principals referring to the *EntityOwner* and *EntityOperator* property are allowed to modify the entity. Moreover, these principals usually only interact with the Device Directory of the Domain they belong to. Hence, only one replica (i.e. the Master Copy) will be written. Thus, no write-write conflicts arise because the Device Directory can employ concurrency control mechanisms (e.g. locks).

However, in general it is allowed to write Master and Local Copies. The following policy is applied to the synchronization protocol:

Synchronization Protocol Policy

- Local Copies are only created upon request.
- Updates of the Master Copy are only propagated upon request.
  - A callback mechanism similar to the Andrew File System [69] can, but does not have to be used to propagate the changes (if not, Device Directories have to poll for changes if required).
  - The *EntityVersion* property can be used to check if a Local Copy is up-to-date.

[1] Although replication and synchronization are often used as synonyms, replication means that there are two or more copies of all data, while synchronization is about keeping two or more copies up-to-date without requiring that each copy contains all data

  - It must be ensured that no concurrent access to the Master Copy is possible (e.g. using locks).
- Only the Master Copy can be deleted. Deletion of a local copy means that no further synchronization with the corresponding peer takes place.
- Before a Local Copy can be written, the Synchronization Protocol has to obtain a lock from the Master Copy. The modification must be propagated to the Master Copy immediately.
- Entities marked as private will never be propagated.

This policy ensures a consistent view on the state of a device if an Intra-Operator provisioning, crossing domains, is conducted. A Device Lock created will be immediately synchronized to the domain of the consuming Operator. If the consuming Operator further provisions the device to a Consumer and defines an Aggregator, it modifies the *Aggregator* property. Due to the policy, the local Device Lock copy is immediately synchronized with the master copy maintained by the domain the device belongs to. Since a one to one relation between Device Locks and Aggregators exists (see Section 4.4.2), only one such Local Copy of a Device Lock can exist. Thus, the overall performance is not significantly reduced by acquiring remote locks. If multiple Device Locks exist (e.g. in case of a non-exclusive device) and the corresponding Local Copies are modified concurrently, the state of the corresponding Device Instance is kept consistent using Algorithm 5.2.1.

Nevertheless, the synchronization protocol in particular and all other protocols that span multiple participants and lead to modification or creation of entities (e.g. Provisioning Protocol), must provide means for safe transactions. A Read-Write operation on a Local Copy of a Device Lock, for instance, can be based on primary two-phase locking (2PL) [148]. The transaction scheduler resides on the machine of the Device Directory that is accessed. Because a Local Copy is written, the scheduler has to acquire a lock from the lock manager associated with the Device Directory maintaining the Master Copy.

5.2.3. Provisioning Interactions & Algorithms

Based on the communication protocols, the following sections will describe the core interactions necessary to provision devices to Consumers. Four core interactions are defined:

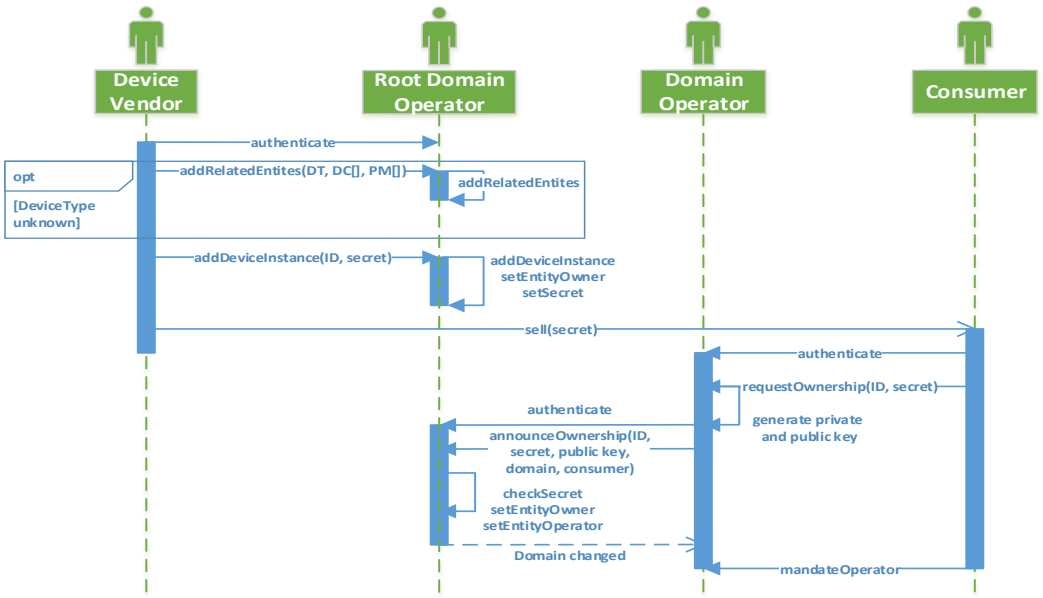


Figure 5.7.: Simplified overview of the Device Deployment interaction.

Device Cloud Interactions

Device Deployment:
Deploy (i.e. announce) a device to the Device Cloud – results in the creation of appropriate Device Directory entities.

Device Identification:
Dissemination of device related knowledge among the Domains (i.e. Device Directories).

Integration Request:
Request access to a device of interest.

Integration Offer:
Offer integration of a device that was discovered but is not of interest.

Device Deployment

The Device Deployment interaction, summarized in Table 5.2 is used to announce a device to the Device Cloud and define its *Device Owner* and *Device Operator*. Basically,

Table 5.2.: Device Deployment interaction summary

Trigger:	Device manufactured by Vendor
Result:	Device Instance created and *EntityOwner*, *EntityDomain*, and, *EntityOperator* defined
Involved:	Consumer, Domain Operator, Device Vendor
Protocols:	Auth Protocol, DD Access Protocol

it can be divided into two steps: device announcement and definition of the device owner and operator.

Device Vendor → Root Domain Operator
The announcement of a device results in the creation of a Device Instance entity and optionally includes the creation of a Device Type entity, a set of Device Category entities and a set of Platform Module entities. This is the case, if the device corresponds to a new class or model of devices. In each case, the vendor initially becomes the *EntityOwner* of the created entities. Besides assigning the unique ID to the Device Instance, a secret has to be attached in order to allow another entity to take the *Device Owner* role and become the *EntityOwner*. The secret can be added as an configuration entry or an attachment to the Device Instance. By default, a principal being able to present the secret is granted the permission to modify the properties *EntityOwner* and *EntityDomain*. As a result of the registration by the Device Vendor, the Device Instance has transitioned to the state Announced.

Device Vendor → Consumer
The device is sold and the secret is revealed to the Consumer.

Consumer → Domain Operator
After a Consumer has purchased the corresponding device, it can instruct its Domain Operator to take ownership. Therefore, the Domain Operator presents the secret, sets *EntityDomain* and the *EntityOwner* properties, creates the public and private keys and attaches the public key. This results in a degradation of the initial Master Copy to a Local Copy, while the new Master Copy is maintained by the Consumer's domain. Additionally, the Device Instance has transitioned to the state Owned.

In order to allow the device to be provisioned, the Consumer has to mandate a Consumer Operator, by setting the *EntityOperator* property. The corresponding Consumer Operator has gained the permission to access and modify required properties of the Device Instance, create Device Locks and is thus able to provision the device. As a result, the Device instance has transitioned to the Operated state.

In summary, the Device Deployment interaction, as illustrated in Figure 5.7, includes announcing a device and creating required entities within the Global Device Directory, defining the *EntityOwner*, *EntityDomain*, and *EntityOperator* properties, and attaching a public-private key pair to the created Device Instance. Exceptions can occur if any of

the participating entities was not properly authenticated or the presented secret is not valid (i.e. the Consumer is not allowed to take ownership). Additionally, the Consumer has to mandate a Consumer Operator belonging to the same domain.

Device Identification

Table 5.3.: Device Identification interaction summary

Trigger:	Any kind of request containing an unknown Device Instance *EntityID*, usually discovery of an unknown device.
Result:	Knowledge about the device was replicated to the domain of the requesting entity and the Device Instance is known.
Involved:	Aggregator Agent, Consumer Operator, Domain Operator
Protocols:	Auth Protocol, DD Access Protocol, DD Sync Protocol

The Device Identification interaction, as summarized in Table 5.3, is a simple interaction that covers the dissemination of publicly available device related knowledge between the different Device Cloud participants, based on the DD Sync Protocol (usually Local and Global Device Directories). It presumes that there is a Device Instance corresponding to the device, not being in the states manufactured or deleted. No state transitions are triggered by this interaction. The interaction is triggered when an Aggregator discovers a device.

Aggregator Agent → Domain Operator
As discussed, the Device Cloud concept assumes that the discovery process can extract the *EntityID* of the Device Instance corresponding to the device. Given the ID, the Aggregator is able to request knowledge about the device from its corresponding Domain Operator. Using the DD Access Protocol, the Local Device Directory is queried for the Device Instance, the Device Type, the Device Category and possibly the corresponding Platform Module instances.

Domain Operator → Root Domain Operator
If the Local Device Directory does not know the given *EntityID*, the Global Device Directory is queried. This results in the creation of a local copy of the Device Instance and related entries. Thus, device related knowledge always originates from and is propagated by the Global Device Directory. The *EntityVersion* property is used to determine if an already existing local copy is up to date.

Domain Operator → Domain Operator (optional)
The *EntityDomain* property of the queried Device Instance is utilized to determine the Domain Operator responsible for the device. This optional step is necessary if the Device Instance belongs to a different domain. Hence, the master copy could have been updated

without propagating the changes back to the Root Domain. The responsible Domain Operator can be queried to check if the Device Instance replicated from the Root Domain is up to date. Moreover, the Consumer Operator being the *EntityOperator* can be determined.

If the discovered device is of interest, the Aggregator Agent can decide to request access. Exceptions can occur if a participating entity failed to authenticate. Note, that some knowledge included in the Entities may not be publicly available and is excluded by the Device Directory (usually a local one) before transmitting the results. This for instance refers to attachments or configuration entries that are marked as private (e.g. a Bluetooth PIN required to communicate with the device).

Integration Request

Table 5.4.: Integration Request interaction summary

Trigger:	Discovery of a device.
Result:	Provisioning of the device – Device Instance switched to Aggregator Bound state and Device Lock was created.
Involved:	Aggregator Agent, Consumer Agent, Consumer Operator, Domain Operator, Operator Agent(optional)
Protocols:	All defined protocols

The Integration Request interaction covers the core device provisioning capability of the Device Cloud. It is triggered by an Aggregator that has discovered a device. Before an Integration Request can be triggered, the Device Identification interaction has to be completed successfully. Therefore, it is assumed that the Aggregator has requested all relevant device related knowledge from its corresponding Local Device Directory (i.e. Domain Operator). This includes the Device Instance entity, the corresponding Device Type entity, the set of corresponding Device Category entities and optionally, the set of required Platform Module entities. Since the transmission of Platform Modules usually requires more bandwidth, Platform Modules may be requested delayed after the successful completion of the Device Integration interaction. The interaction consists of three steps: checking if a Consumer bound to the Aggregator is interested in the device, identifying the Operator responsible for the device, and requesting access to the device.

Depending on the characteristics of the request (e.g. Intra-Operator, cross-domain), several Device Cloud participants can be involved. The description given below, will cover the most complex case involving an Intra-Operator provisioning crossing domains. The following participants are defined:

- *Aggregator Agent:* The Aggregator having discovered the device and issuing the request.
- *Consumer Agent:* The Consumer Agent linked to the Aggregator.
- *Consuming Domain Operator:* The Domain Operator managing the domain the request originated from (i.e. being responsible for the Aggregator and Consumer that issued the request).
- *Owning Domain Operator:* The Domain Operator managing the device which is subject of the request.
- *Consuming Consumer Operator:* The Consumer Operator managing the Consumer Agent (i.e. the source of the request).
- *Owning Consumer Operator:* The Consumer Operator managing the device.
- *Operator Agent:* The Operator Agent representing the Owning Consumer Operator.

After discovering a device, the Aggregator Agent has to check if a Consumer linked to it is interested in the device. Therefore, a Consumer Agent has to be elected according to the definition of bound and unbound Aggregators given in Section 4.4.4. If no Consumer is linked to the Aggregator, an Integration Offer, as discussed in the next section, can be issued instead of the Integration Request.

Aggregator Agent → Consumer Agent
The Aggregator notifies the elected Consumer Agent that a device was discovered using its representing Device Instance, Device Type and Device Category entities.

Consumer Agent → Consuming Domain Operator (optional)
The Consumer Agent may optionally request the Consumer Profile from the User Directory if multiple Consumer Agents corresponding to the same Consumer entity exist and synchronization is required.

Consumer Agent → Consuming Consumer Operator
If the Consumer Profile was evaluated and the Consumer Agent decides, that the device is of interest, it triggers an access request. The request contains the Device Instance ID and additional parameters like the desired usage time or which capabilities of the device are requested (i.e. *CategorySets*). The request is issued to the Management Services of the Consuming Consumer Operator using the Provisioning Protocol.

Consuming Consumer Operator → Consuming Domain Operator (optional)
Using the *EntityOperator* and *EntityDomain* properties, the Owning Consumer Operator has to be identified. This step can become unnecessary in case of protocol optimization (e.g. caching of the previously issued Device Identification request). The following steps are only necessary in case of Intra-Operator and cross-domain provisioning.

Consuming Consumer Operator → Owning Consumer Operator (optional)
The Consuming Consumer Operator requests a device provisioning from the Owning Consumer Operator. From the perspective of the Owning Consumer Operator, it acts like a regular Consumer requesting a device access token.

Owning Consumer Operator → Owning Domain Operator (optional)
The Device Directory of the Owning Domain Operator is queried in order to check whether the device is currently provisionable given the request parameters (e.g. desired lock time).

If true, the involved Consumer Operators may enter a negotiation phase to determine the conditions of the provisioning (e.g. price, lock time, Operator supervised, forced withdrawal allowed). If the Owning Consumer Operator decides to provision the device, a Device Lock entity is created and the *DeviceInstanceState* property is updated according to the definitions given in Section 5.2.1. The Device Lock entity is immediately replicated to the Consuming Domain Operator.

Owning Consumer Operator → Consuming Consumer Operator (optional)
The Owning Consumer Operator creates a device access token as described in Section 5.1.3 and returns it to the Consuming Consumer Operator, which now holds a valid lock for the device.

Consuming Consumer Operator → Consuming Domain Operator (optional)
The device can be re-provisioned to the initially requesting Consumer Agent. Therefore, a new token has to be attached to the initial one and the Device Lock entity has to be updated by defining the *Aggregator* property. As discussed in Section 5.2.2, the Device Lock is immediately synchronized with its master copy maintained by the Owning Domain Operator.

Consuming Consumer Operator → Consumer Agent, Aggregator Agent
The device access token is returned to the Consumer Agent. The Consumer Agent will usually update the Consumer Profile (mark the corresponding device entry as integrated) and forward the token to the Aggregator Agent, which has to validate it.

Aggregator Agent → Operator Agent (optional)
If the token defines that the provisioning has to be supervised by the Owning Consumer Operator, the Aggregator Agent establishes a session to the corresponding Operator Agent as described in Section 5.1.4.

Aggregator Agent → Consumer Agent
The Consumer Agent is notified about the successful completion of the device integration. If an exception occurs, the whole operation has to be rolled back.

Aggregator Agent → Consumer Agent
Upon release of the device connection, the Aggregator Agent must notify the Consumer Agent (regardless whether the connection was released due to an expired token, an interruption of the connection or on demand of the Consumer himself). In order to simplify protocol design with respect to transactions, this notification can also be used if an error during the initial integration happened.

Figure 5.8 gives a simplified overview of the interaction. The provisioning decision, made by the Owning Consumer Operator, basically depends on the *DeviceInstanceState* property and the request parameters (i.e. which category was requested in case of non-exclusive or composite devices). The following policy is applied:

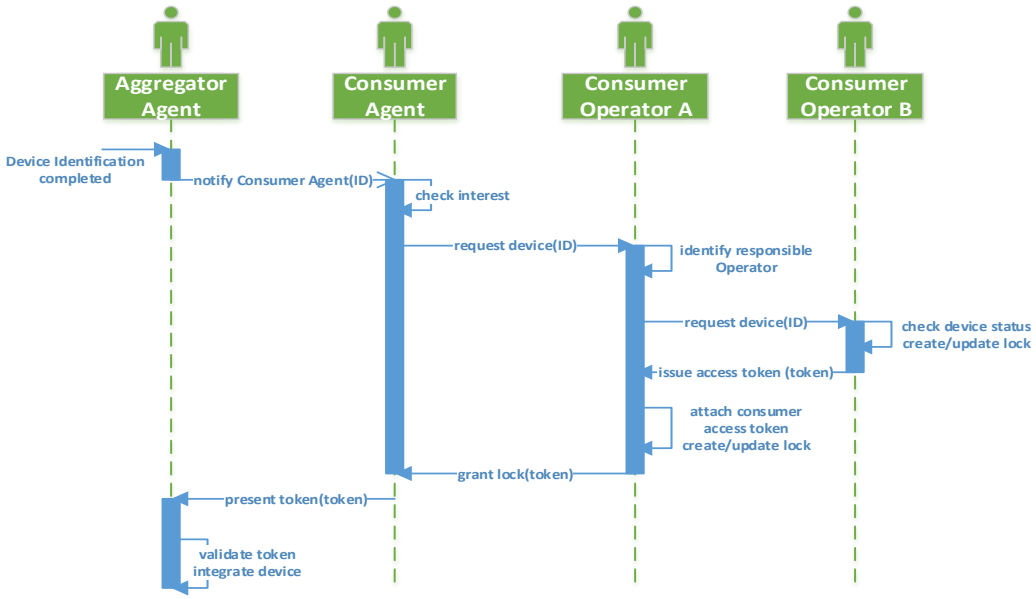


Figure 5.8.: Simplified overview of the Integration Request interaction.

Integration Request evaluation based on the Device Instance State

Manufactured, Announced, Owned, Disabled, Deleted:
The device cannot be provisioned.

Operated:
The device can be provisioned if the requesting Consumer refers to the *EntityOwner* of the Device Instance.

Idle:
The device can be provisioned.

Consumer Bound:
The device can be provisioned if a Device Lock exists, that has no Aggregator defined and belongs to the requesting Consumer (i.e. the *EntityOwner* of the Consumer Agent and the *LockingEntity* property of the Device Lock are equal). The Consumer Operator will modify the *Aggregator* property of the Device Lock in this case.

Aggregator Bound:
The current Device Lock can be extended or renewed if a Device Lock exists that belongs to the requesting Consumer. The *Aggregator* and *LockingEntity* properties of the Device Lock must match the requesting Aggregator Agent and Consumer Agent.

A Device Instance being in the Consumer Bound state can additionally lead to an Integration Offer issued by the Operator, which is discussed in the next section. Other exceptional and special cases, like releasing a Device Lock in an emergency case as introduced in Section 4.3, are discussed in Section 5.2.3. Exceptions besides authentication or authorization failures can occur if the device is not provisionable given the request parameters, if the token could not be validated or, if a required Operator supervision failed (e.g. the Operator Agent failed to validate the Aggregator Agent).

Integration Offer

Table 5.5.: Integration Offer interaction summary

Trigger:	Discovery of a device.
Result:	Provisioning of the device – Device Instance switched to Aggregator Bound state and Device Lock was created.
Involved:	Aggregator Agent, Consumer Agent, Consumer Operator, Domain Operator, Operator Agent(optional)
Protocols:	All defined protocols.

As the name suggests, the Integration Offer interaction, summarized in Table 5.5, can be envisioned as a reverse Integration Request. It is either triggered by an Aggregator that has discovered a device, but is not linked to a Consumer Agent interested in the device, or due to a rejected Integration Request. An Integration Offer can result from an Integration Request, if the owning Consumer Operator cannot provision a device because the permitted limit of simultaneously active locks is reached and the Operator is not willing to release an active lock. However, one of the active locks may be only bound to a Consumer (i.e. the Aggregator property is not set), which could lead to an Integration Request by the Operator. This Request is similar to an Integration Offer triggered by the initially requesting Aggregator (i.e. the Aggregator just offers its device integration capabilities).

The Integration Offer is illustrated using a scenario that involves three Consumer Operators: The Operator the integration was initially offered to, the Operator holding the *Device Operator* role and an Operator holding the *Device Consumer* role. For the sake of simplicity, it is assumed that all Consumer Operators belong to the same domain and that no Operator supervision is required. If different domains are involved, the scenario would involve further Device Directory synchronization steps.The following participants are defined:

- *Aggregator Agent:* The Aggregator having discovered the device and offering the integration.
- *Offering Consumer Agent:* The Consumer Agent representing the Consumer Operator the integration is initially offered to (i.e. the Consumer Operator managing the Aggregator device).
- *Offering Consumer Operator:* The Consumer Operator the integration was initially offered to.
- *Owning Consumer Operator:* The Consumer Operator equal to the *EntityOperator* property of the Device Instance.
- *Consuming Consumer Operator:* The Consumer Operator currently holding an active lock for the device.
- *Consuming Consumer Agent:* The Consumer Agent corresponding to the *LockingEntity* property of the active device lock.

Before triggering the offer, the Aggregator Agent has to determine the responsible Offering Consumer Operator based on the type of the Aggregator:

- *Unbound Aggregator:* Unbound Aggregators always belong to a Consumer Operator and are usually owned by them.
- *Bound Aggregator:* Bound Aggregators usually imply that the bound Consumer agrees to the Integration Offer because resources are consumed and some application scenarios may have strict privacy requirements. Regarding the responsible Consumer Operator, two cases have to be considered:

- *Private Aggregator:* The Aggregator is a device owned by the Consumer (i.e. the Device Cloud Middleware was just installed to the device). The Consumer has to decide which Consumer Operator, if multiple ones are used, becomes the default one and is used by the Aggregator to perform Integration Offers.
- *Operator Aggregator:* The Aggregator is owned by and belongs to one Operator (e.g. a router that was handed to a Consumer by an ISP due to a contract)

Aggregator Agent → Offering Consumer Agent → Offering Consumer Operator

Offer the integration using the *EntityID* of the device. Either the Offering Consumer Operator is holding an active lock for this device itself, or it has to forward the offer to the Owning Consumer Operator. In the first case the Offering and Consuming Consumer Operators are equal and the offer can be evaluated immediately. However, in case of non-exclusive devices, the Offering Consumer Operator can decide to forward the offer anyway.

Offering Consumer Operator → Owning Consumer Operator

Based on the state of the Device Instance, the Owning Consumer Operator must decide how to further process the offer. Note that an offer does not contain a Device Category parameter, which means that all entries in *CategorySets* as well as all active locks have to be considered. If the Owning Consumer Operator forwards the offer to Consuming Consumer Operators, it has to reply the IDs of the Operators the offer was forwarded to (besides possibly accepting the offer itself).

- *Manufactured, Announced, Owned, Disabled, Deleted:* The Integration offer cannot be accepted.
- *Operated:* The Integration Offer can be accepted, if the Owning Consumer Operator is equal to the *EntityOwner* property of the Device Instance because according to Section 5.2.1, only the *Device Owner* is allowed integrating a device that is not pooled.
- *Idle:* In general, the Integration Offer can be accepted. However, no Consumer is bound, which most likely means that the Owning Consumer Operator will not benefit from agreeing to the offer. In case of a non-exclusive device, active locks that are just bound to a Consumer can be considered, too. If the Consumer refers to a Consumer Operator (i.e. Consuming Consumer Operator), the offer can be forwarded.
- *Consumer Bound:* The limit of simultaneously active locks has been reached. The Integration Offer can be accepted, if locks exist, that are bound to a Consumer belonging to the Owning Consumer Operator. Additionally, each lock referring to a Consuming Consumer Operator can be considered by forwarding the offer.
- *Aggregator Bound:* The device can only be provisioned, if one of the current bindings to an Aggregator is released in advance. This can include forwarding the offer to any other Consuming Consumer Operator.

Owning Consumer Operator → Consuming Consumer Operator

The Owning Consumer Operator forwards the offer to possibly interested Consuming Consumer Operators.

Consuming Consumer Operator ↔ Offering Consumer Operator
If a Consuming Consumer Operator is interested in the offer, it can either be accepted directly or a negotiation phase has to be entered. The negotiation phase basically includes evaluating and adapting the conditions (e.g. price) of the offer using the Offering Consumer Agent and the Consuming Consumer Agent. If the integration is accepted, an access token is issued. This does not necessarily require interaction with the Owning Consumer Operator because the Consuming Consumer Operator already possesses an valid access token and can attach another one as described in Section 5.1.3.

Offering Consumer Operator → Aggregator Agent
The access token is transmitted to the Aggregator Agent and the device can be integrated, which includes deploying the Platform Module orchestration as defined by the remote Consumer Profile (i.e. the Consuming Consumer Agent). Only the Consumer Profile entry corresponding to the integrated device is deployed. The Aggregator Agent will not consider the newly linked Consuming Consumer Agent for Integration Requests.

For the purpose of sharing the virtual representations of a device, as discussed in Section 5.2.4, the Owning Consumer Operator will most likely accept Integration Offers, even if the Device Instance is in Idle state.

Decision Policies

Some application scenarios can require an adaptation of the standard, state based provisioning behaviour described by the Integration Request interaction. An example, introduced in Section 4.1.1, is given by medical sensors used to monitor a patient. Usually, the sensors will be integrated by an Aggregator belonging to the patient (e.g. a smart phone) for the purpose of remote monitoring (i.e. telemedicine). In case of an emergency, the patient's smart phone may not be available while the sensors are still operating and the patient is transported to a hospital. Thus, an Aggregator within the ambulance and an Aggregator at the hospital may successively request access to the sensors while they are still locked by another Aggregator. This results in the requirement to implement handover mechanisms. Basically, two handover types have to be considered:

Device Handover Classification

Aggregator Handover
The Aggregator (i.e. *Device Integrator*) is changed while the *Device Consumer* remains. Hence, an active Device Lock granted to the requesting Consumer must already exist.

Consumer Handover
The Consumer is changed. Usually, the Aggregator is changed too, but may also remain. No active Device Lock granted to the requesting Consumer exists.

The E-Health scenario sketched above refers to a Consumer Handover. Both Consumer and Aggregator change while an optionally defined binding to the *Device Target* (i.e. the patient) remains. Algorithm 5.2.2 shows the basic steps of evaluating an Integration Request. Note that for simplification not all boundary conditions are shown. It is assumed, that the given Consumer and Aggregator do not already hold an active lock.

Algorithm 5.2.2 Evaluation of a Device Integration Request

```
function EvalIntegrationRequest(Principal consumer, Aggregator agg, DeviceInstance
device, DeviceCategory rC)
    cID ← device.DeviceType.getCategorySetID(rC)
    RootState ← device.DeviceInstanceState.RootState
    cState ← device.DeviceInstanceState.getCategoryState(cID)

    if RootState = Operated then
        if device.EntityOwner = consumer then
            return true
        end if
    else if RootState = Idle then
        if cState = Idle then
            return true
        else if cState = ConsumerBound ∨ cState = AggregatorBound then
            activeLock ← ∅
            for all x ∈ device.getActiveLocks() do
                if x.LockingEntity = consumer then activeLock ← x
                end if
            end for
            if activeLock ≠ ∅ then
                if activeLock.Aggregator = ∅ then
                    return true
                else
                    returnEvalReleaseLock(consumer, agg, device)
                end if
            else
                returnEvalReleaseLock(consumer, agg, device)
            end if
        end if
    end if

     return false

end function
```

The behaviour of the function *EvalReleaseLock(Principal consumer, Aggregator agg, DeviceInstance device)* is not defined because it depends on the requirements of the use case and may differ from Operator to Operator. An important parameter is given by the

location of the device. If a device is in the state Aggregator Bound, but was discovered at a location out of range of the locking Aggregator, a handover is likely to be reasonable. Additionally, Operators may monitor the activity of a device and provide an assessment regarding the connection status of a device. However, the E-Health scenario described above can involve situations, where decisions based on the location or the activity are not sufficient. If the patient enters the emergency ward, the Aggregator of the ambulance might still be in range, which means the Operator has to decide between two or more overlapping sensor access points. As not all communication protocols used by the sensors provide mechanisms to assess the quality of a connection like mobile telecommunication networks do, traditional handover approaches are difficult to apply. A possible solution would be to introduce classes of *Device Consumers* and assign priorities to them. Hospitals or other authorities providing emergency or safety-critical services could belong to a preferred class. In general, this problem is related to the definition of Quality of Service (QoS) policies for shared resources (e.g. networks). Each application domain must define a set of contextual data items required to implement sufficient decision policies.

From a formal point of view, making a decision is related to the problem of pre-emptive multitasking or scheduling. In case of a Consumer Handover, there are two processes (i.e. Consumers) A and B that compete for the exclusive device resource D. Furthermore, a scheduler C (i.e. Consumer Operator) has to decide, whether A can continue consuming D or A has to be interrupted for the purpose of assigning D to B. Similar to priority-based scheduling, C has to dynamically compute a priority given the contextual data provided by A and B. The priority reflects the benefit the *Device Owner* of D gains by either assigning A or B to D (i.e. the goal of C – the Consumer Operator – is to optimize the benefit of the *Device Owner*). Thus, the *EvalReleaseLock(...)* function first has to select the active Device Lock with the lowest priority and subsequently compare it to the priority calculated from the contextual data provided by B. The set of contextual data applicable for a particular device resource D and a function mapping a numerical value to each data item could be defined by the Device Categories. Possible contextual data items are:

Device Handover Indicators

Price:
The price A and B are willing to offer to the *Device Owner*. In case of an Integration Request the highest offer will win the competition while in case of an Integration Offer, the lowest one will be preferred.

Proximity:
The proximity of D to A and B.

Aggregator Capabilities:
Aggregators involved in the provisioning can be rated regarding their mobility,

privacy (e.g. Bound or Unbound Aggregators), available resource (i.e. possible Aggregation Platform Modules) or, reliability.

Device Target:
An unchanged *Device Target* may take precedence (e.g. the binding to a patient is kept). Composite devices may be able to serve multiple Consumers but only monitor one *Device Target* simultaneously. Thus, a request including a *Device Target* differing from the current one would lead to a revocation of other existing Device Locks.

Consumer Priority:
As mentioned, some use cases (e.g. E-Health) will likely define static priorities for the Consumers (e.g. the emergency ward of a hospital takes precedence over a home doctor).

Advanced decision policies are likely to additionally take the Idle state into consideration. Even if, for instance, a device is ready to be provisioned, the policy may reject an allocation attempt if it can be estimated that the benefit will be higher due to a high probability of a subsequent allocation attempt with better conditions. The problem is, that provisioning or allocation strategies known from the Cloud- or Grid Computing domain are difficult to apply because the unit of resource is not a virtual machine, it is the host (i.e. the device). Moreover, the set of hosts is highly heterogeneous (i.e. the type of a host is given by the Device Category) and, for instance, compared to Grid Computing the proximity between the host and the Consumer has to be considered (i.e. a model for hosts that move in space is required). However, if mobility is not considered, tools like GridSim [30] or CloudSim [31] can be used to estimate the performance of a decision policy with regard to criteria like *Device Owner* benefit or overall device utilization. Basically, the procedure would be to model devices as hosts and Integration Requests as VMs or Jobs consuming exactly all resources of one host for the whole provisioning period. If two devices belong to different Device Categories, they must be represented by different host types. Moreover, only one VM or job can be allocated to one host in parallel. Subsequently, each host just provides one single core CPU and a space shared allocation policy has to be used.

5.2.4. Sharing Virtual Representations

Although the Device Cloud is about sharing physical devices, Sensor Virtualization and the associated concept of sharing virtual representations, as introduced in Section 2.2.1, are important topics in the area of Sensor-Cloud integration. Therefore, the Sensor Virtualization approach can be easily added to the Device Cloud concept as an additional feature. It is based on the Integration Offer interaction. An Operator, willing to offer

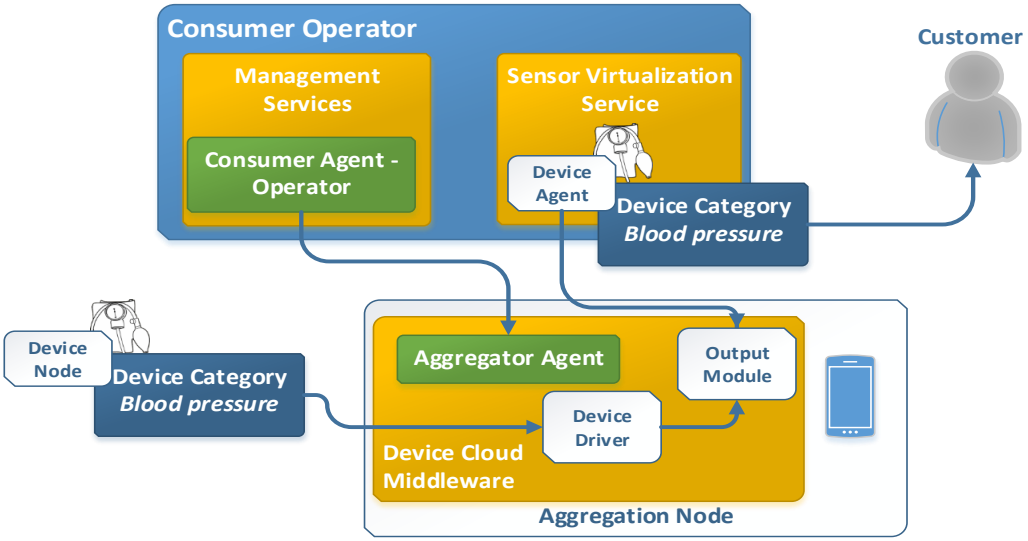


Figure 5.9.: Sensor virtualization based on the Device Cloud infrastructure.

services for virtualized sensors, can accept each Integration Offer depending on the requirements of its Customers. Regardless, whether the Operator uses own devices or devices provisioned by other Operators, it acts as the *Device Consumer*. As shown in Figure 5.9, the Consumer Operator offers a Sensor Virtualization service, that exposes a virtual representation of the integrated device to its customers. From the perspective of the Device Cloud Middleware, the virtualization service is an application. Hence, the Operator needs to deploy a Consumer Agent linking to its own profile. The profile defines output modules for each sensor that shall be virtualized. The output modules act as a bridge between the physical device and its virtual representation offered by the virtualization service. The Device Agent shown usually aligns to the interface defined by the Device Category corresponding to the device. Thus, methods invoked on the Device Agent can simply be forwarded to the device driver integrating the physical device using remote procedure calls directed to the output module.

6. Device Cloud – Architecture

Contents

This chapter describes the architecture of the main building blocks of the Device Cloud infrastructure. According to the Device Cloud concept, this includes the Device Cloud Middleware and the Backend Information System offered by Domain and Consumer Operators. The presented architecture is an exemplary approach. Other solutions compliant with the concept requirements can exist.

6.1. Backend Information System

The Backend Information Systems are used to maintain and provide access to the entities described in Section 4.4 (presuming proper authorization). The Device and the User Directory are provided by Domain Operators while the Management Services are provided by Consumer Operators. The Directories act as resource servers (i.e. data bases) that are capable of protecting their resources from unauthorized access. The Management Services basically provide the logic required to negotiate the on-demand provisioning of devices.

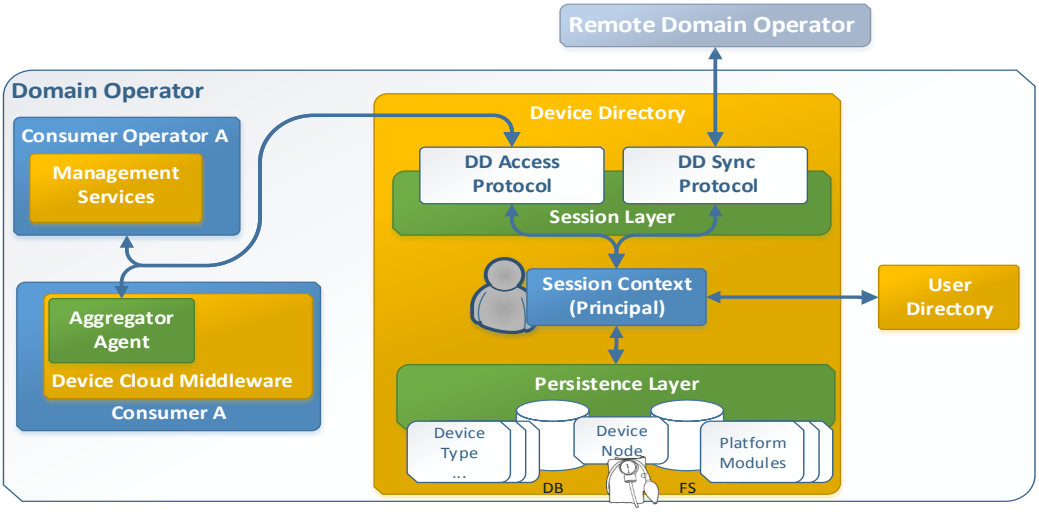


Figure 6.1.: High level Device Directory architecture.

6.1.1. Device Directory

The implementation of the Device Directory is generic, which means that no distinction is made between the Global and Local instances. Instead, the Global Instance can be elected and it is simply deployed without configuring a parent. As shown in Figure 6.1, the prototype of the Device Directory basically consists of a session and a persistence layer.

The session layer provides the implementations of the DD Access and DD Sync Protocol. Currently, a WebSocket based approach is used [54]. As discussed in Section 5.2.2, the Directories use a custom token format to authenticate the principal opening a session. However, OAuth2.0 and OpenId Connect can be integrated easily. After successful authentication, the Device Directory creates a session context, which identifies the principal and defines the scope of the session (i.e. DD Sync or DD Access protocol). The context is used to authorize access to an entity maintained by the Device Directory. As discussed in the previous chapter, access is guarded by the *EntityOwner* and *EntityOperator* properties of each entity.

The persistence layer provides access to the device knowledge base. Common for directory services, the knowledge base is organized using a tree based approach (see Figure 6.2). The Device Directory facilitates a hybrid approach by using a relational data base for storing entities and a file system repository for storing their attachments, which also includes Platform Modules. The object-relational mapping framework Hibernate is

used to translate the entities between their object-oriented and relational representation. Moreover, Hibernate [19] allows using several data base implementations (e.g. MySQL, PostgreSQL) without adapting the implementation of the Device Directory. The file system repository is organized similarly to an Apache Maven[10] repository. This is basically motivated by the format of the *EntityID* property of Platform Modules, which is similar to the identifiers used by Maven artifacts. In order to keep consistency between the Device Directories, proper management of the *EntityVersion* property is important. The version is always managed by the master copy and is usually incremented automatically in case of modifications.

Besides maintaining entities that describe a device (e.g. Device Instance, Device Type, Device Category), one of the core functionalities is to provide the Platform Modules used by the Device Cloud Middleware to integrate and handle devices. Refining the classification of Platform Modules given in Section 4.4.2, the following types are defined:

Platform Module Types

Core Platform Modules:
The class of Core Platform Modules is not further refined.

Discovery Modules:
Class: Device Integration Platform Module
The Device Cloud separates the integration from the discovery process. Although even the discovery processes are subject to a huge heterogeneity, discovery modules usually can be implemented much more generally than device drivers. Separating discovery from driver logic leads to a more efficient resource utilization.

Device Driver Modules:
Class: Device Integration Platform Module
Device Driver Modules implement the device control logic. According to the concept of Device Categories, Device Driver Modules provide the implementation to communicate with a device using its category definition.

Device Category Modules:
Class: Device Integration Platform Module
As defined in Section 4.4.2, Device Category Modules provide the category definition, which usually refers to an Interface.

Input Modules:
Class: Aggregation Platform Module
Input Modules are used by the aggregation layer of the Device Cloud Middleware. Based on the Device Category, they attach to one or more Device Drivers and translate between the category based interaction model of the device integration layer and the event based model of the aggregation layer. Thus, an Input Module polls the sensing operations offered by a Device Driver (based on its configuration) and wraps the result into a container-based generic data representation used by the aggregation layer.

Utilization Modules:
Class: Aggregation Platform Module
Utilization Modules refer to any kind of Device Cloud internal applications that take data streams emitted by the Input Modules and perform analysis, aggregation, or visualization, if the Aggregator device hosting the Device Cloud Middleware is capable of displaying user interfaces.

Transformation Modules:
Class: Aggregation Platform Module
Transformation Modules are a specialization of Utilization Modules and act as an enabler for syntactic and semantic interoperability. They are used to transform the payload of the container representation from an input to an output format. Transformation Modules are distinguished from Utilization Modules because they can be injected automatically, based on the requirements of the module composition. This is related to the *InputFormat* and *OutputFormat* properties of the Platform Module entity.

Output Modules:
Class: Aggregation Platform Module
Output Modules act as residential gateways transmitting the data streams to applications hosted outside of the Device Cloud. Similar to Input Modules, Output Modules can attach to Device Drivers in order to forward control commands based on the Device Category. This is required if an application needs to interact with a device offering actuating capabilities.

The *InputFormat* and *OutputFormat* properties of Platform Modules only apply to Aggregation Platform Modules. They describe the format of the payload contained in the generic container format. In case of Core Platform Modules, these properties are not defined. In case of Device Integration Layer Modules, these properties are statically defined as *CATEGORY_BASED* because drivers expose a service defined by an interface, which belongs to a Device Category. Moreover, the generic container format is only used by Aggregation Platform Modules, which can be based on any kind of application dependent data format (e.g. HL7, IEEE 11073). Successive modules participating in the processing of a data flow need to have matching output and input formats. Otherwise, the Device Cloud can try to automatically inject a Transformation Module. If this fails, the Aggregation Module composition is invalid and cannot be deployed. In general, Input Modules act as a mediator between the *CATEGORY_BASED* format used by Device Integration modules and the container-based format used by the Aggregation Modules.

This is also related to the general permission policy applied to Platform Modules based on their type and class:

- **Core Platform Modules:** Core modules provide system services and get all necessary

permissions (e.g. installing other bundles, managing the lifecycle of other services, accessing network or file resources). Moreover, most system services are protected from direct access of non-core modules (e.g. a Device Driver cannot access and trigger action of the Aggregator Agent).

- **Device Driver Modules:** Device Driver modules are allowed to register Device and Driver services. Additionally, drivers can access network resources within the local subnet of the Aggregator device and may make use of the Java Native Interface (JNI) to interface with OS dependent stacks, as is required for Bluetooth.
- **Discovery Modules:** Discovery Modules can register discovery services and are allowed to consume Device Services and generate events upon discovery of a device.
- **Device Category Modules:** Device Category modules are API-bundles providing the interfaces of Device Categories. Hence, specific permissions are not required.
- **Input Modules:** All Aggregation Platform Modules can register services. Input Modules are allowed to access Device Services.
- **Utilization and Transformation Modules:** No specific permissions are applied to these modules. Input data streams are pushed by system services and output streams are forwarded using events.
- **Output Modules:** Similar to Input Modules, Output Modules are allowed to access Device Services. Additionally, output modules can consume network resources.

In addition to this policy, each module gets the permissions necessary for normal operation like accessing its private bundle storage area (i.e. permissions implied by the OSGi specification [120]). Another important property regarding the set of permissions is given by the signature of a bundle, which is discussed in Section 5.1.1. The current architecture approach assumes, that at least Core modules are signed by the Root Domain. If configured appropriately, the Device Cloud Middleware may accept deploying Device Integration and Aggregation Modules signed by other authorities.

Replication of device knowledge between directories is subject to traversing the device knowledge tree, which is shown in Figure 6.2. The tree basically contains one Device Instance, one Device Type entity, several Device Category entities, and several Platform Module entities. An initial replication is triggered by requesting the knowledge tree with the *EntityID* property of the Device Instance. This results in a list containing triples of the shape $\{ EntityType; EntityID; EntityVersion\}$, which can be used by the requesting Device Directory to identify the missing and outdated entities (entities like Device Categories or Platform Modules may have been already replicated due to a previous request). A replication request aiming at updating an existing Device Instance already contains the result of traversing the local knowledge tree. After replicating the knowledge tree, a Device Directory can check the *EntityDomain* property of the replicated entities. Basically, the Device Cloud encourages the Global Device Directory (i.e. the Root Domain Operator) to maintain all master copies of the knowledge tree (except the Device Instance). However, it is not forbidden that Local Device Directories maintain master copies, too. Thus, if a Platform Module for instance is not managed by the

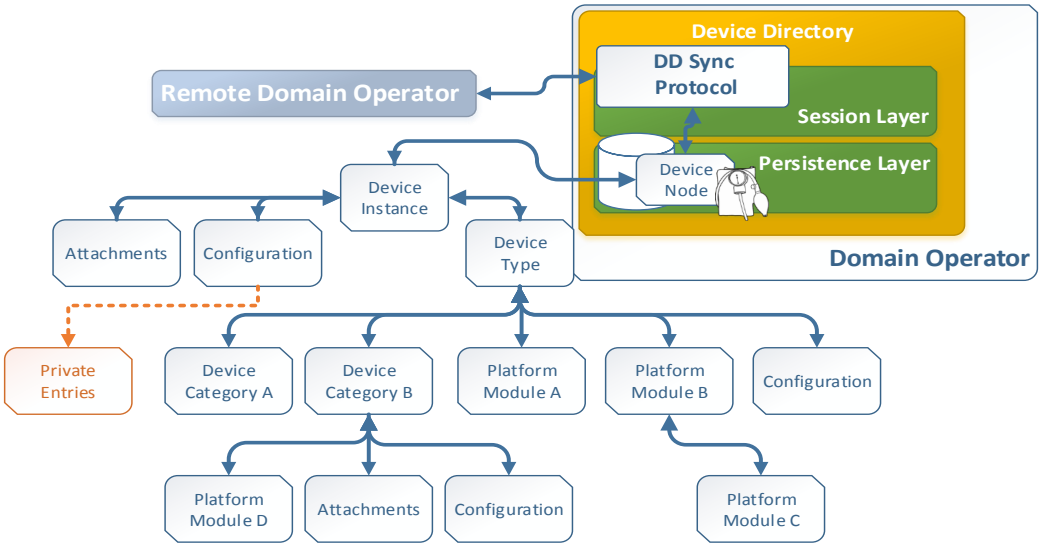


Figure 6.2.: Simplified example of a knowledge tree representing a device.

Global Device Directory, the corresponding local copy may not be up to date and the managing domain has to be queried for the master copy. As discussed in Section 4.4.1, knowledge marked as private will not be replicated.

Replication and any other read access to entities stored by a Device Directory presumes that the entity is not locked due to a write access. The locking mechanism used by the Device Directory is based on the Java *ReadWriteLock* specification. Multiple simultaneous read locks may exist, while the write lock is exclusive. Moreover, the Device Directory must provide the possibility to lock a Device Instance while a provisioning is negotiated. This can be achieved by simply requesting a write lock and releasing the lock again if the negotiation failed.

6.1.2. User Directory

The User Directory, as shown in Figure 6.3, is likely to be implemented as a wrapper around an existing Identity and Access Management (IAM) solution (e.g. an LDAP-based directory service), translating between Auth Protocol requests and the internal Identity and Access Management (IAM) representation of principals. If the User Directory is not wrapped around an existing IAM solution, a database which is able to maintain the Principal entities and their corresponding public keys is required.

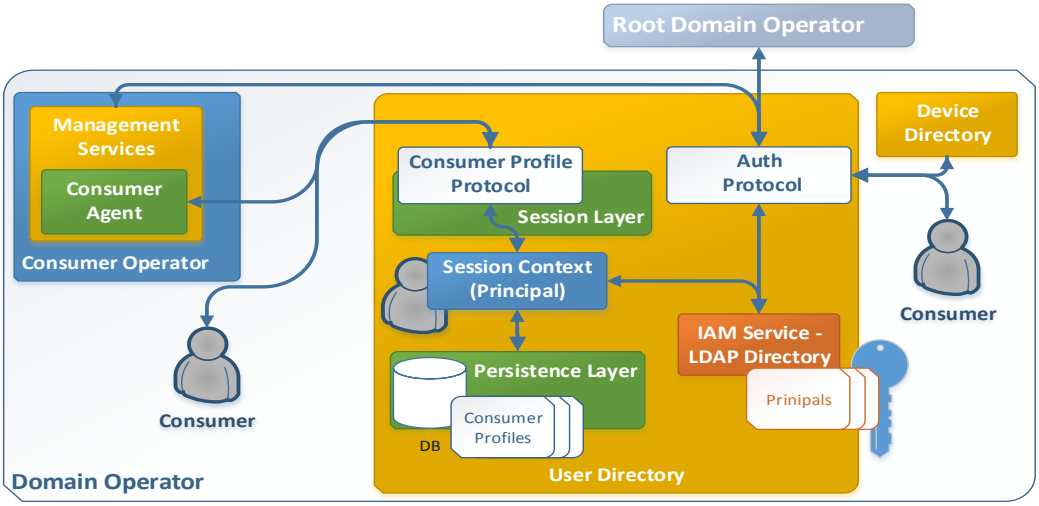


Figure 6.3.: High level User Directory architecture.

Besides the Auth Protocol, the User Directory offers the Consumer Profile Protocol and provides access to the device integration profile of a Consumer. Therefore, the User Directory provides a session and a persistence layer similar to the Device Directory. The Consumer Profile contains a set of entries that allow the Consumer Agent to evaluate whether a particular device is of interest. An entry can be envisioned as a directed acyclic graph which defines the processing of a data stream. The Consumer Profile, as depicted exemplary in Figure 6.4, must comply to the following format:

Consumer Profile

Aggregation Platform Modules:
A set of Aggregation Platform Modules used to process the incoming data. A node ID, unique within the scope of the profile, must be assigned to each module. A configuration can be attached to each module. In order to support data aggregation of several devices, the modules are defined within the scope of the overall profile (i.e. a using the node ID, the same module can appear in several entries).

Each entry must comply to following format:

1. **Entry State:** The state of the entry (e.g. is a – or possibly multiple – fitting device integrated and linked to the Consumer). The state may also include information about the *Device Target*.
2. **Device Category:** The Device Category this entry is applicable for (e.g. a blood pressure monitor). The category is matched against the *CategorySets* property of

the Device Type corresponding to the device being evaluated.

3. **Guard Condition:** A generic condition being able to be evaluated by the Consumer Agents. Basically, the condition has to return true or false in case a device of the given Category was discovered. The evaluation can include all contextual information available to the Consumer Agent. Examples are:
   - the costs or the offered duration of the provisioning
   - the current state of the entry – i.e. integrate multiple devices
   - the constraints of the provisioning – are forced withdrawals allowed; is the integration allowed to be supervised by another Consumer Operator
   - an integration schedule – e.g. a medical sensor is required to successfully monitor a patient's condition three times per day (i.e. must be integrated three times a day and transmit a complete measurement each time)
   - a static binding to a particular Device Instance by specifying the *EntityID*
   - a required *DeviceTarget*
4. **Module Paths:** A set of module paths defining the processing graph – i.e. a set of node ID concatenations. Additional, path-specific, configurations can be defined for the Aggregation Platform Modules. Each path must comply to the following rules:
   - If the path includes Utilization Modules, an Input Module is required. Additionally, if the *OutputFormat* property of the Output Module is not equal to the *CATEGORY_BASED* format, an Input Module is required because a Transformation Module, which expects input in the generic container based format, probably needs to be deployed.
   - Each path must be terminated by an Output Module or a Utilization Module that produces no output (e.g. a visualization).
   - Each module must fit the output format of its predecessor.
   - Input Modules and Output Modules using the *CATEGORY_BASED* input format, must define a set of compliant Device Categories.
   - If multiple Input and Output Modules accessing the device directly are deployed, no guarantee is given that the same data is delivered to each path.

Similar to the Device Directory, access to the Consumer Profile is protected by the *EntityOwner* property. As discussed in Section 5.1.5, the Consumer has to create appropriate entries in its profile, based on the applications he is willing to use. This process can be automated up to a certain degree, by either mandating a Consumer Operator to take care of the profile management (i.e. setting the *EntityOperator* property of the Consumer Profile) or by granting access to the profile to a set of application providers using the *PermissionSet* property. However, the latter approach may not be feasible because it requires IAM services of the Domain Operator to additionally manage principals rep-

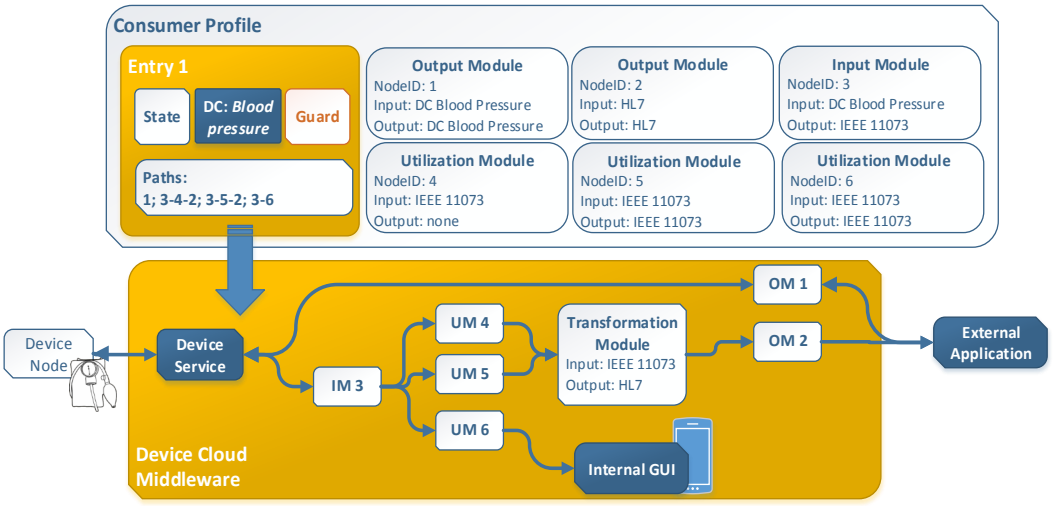


Figure 6.4.: Representation of a Consumer Profile. The deployment view shows an automatically injected Transformation Module.

resenting application providers. The first approach would basically require Consumer Operators to provide an appropriate service that links applications to the Device Cloud (e.g. as part of an app store). Regardless of the employed mechanism, Output Modules are the connectors between applications and the Device Cloud.

Each defined Consumer profile entry, has to be validated according to the rules described above. If Aggregation Platform Modules with non matching input and output formats are linked to each other, it has to be checked, whether an appropriate Transformation Module is available. Otherwise the whole entry, or at least the concerned module path, has to be rejected.

A challenge regarding the given integration profile (i.e. Consumer Profile) definition is given by gateway devices, that belong to the class of Device Nodes (i.e. no Device Cloud Middleware can be deployed). Gateway devices usually do not directly provide sensing or actuating capabilities a Consumer or one of its applications benefit from (nor can they be considered composite devices). Instead, they offer a bridge to another Machine Communication Network (MCN) (e.g. KNX or EnOcean). However, integration is required in order to access the devices connected by the network that is bridged. Adding respective entries to the Consumer Profile is not feasible because there is no knowledge about the devices made available by a gateway. On the contrary, the Device Category abstraction tries to hide the technical details from the Consumer and focuses on the

functional capabilities and not the technical details of a device integration process. As a mitigation to this problem, the following assumptions are made:

1. Gateway devices can be envisioned as external dongles increasing the technical capabilities of an Aggregation Node and therefore should be integrated whenever possible.
2. Exceptions from this behaviour are given if an Aggregator is not configured to serve Integration Offers or is currently not linked to Consumer Agents that have unaccomplished entries in their corresponding Consumer Profiles.
3. Gateway devices are mostly stationary, often used in application domains like smart homes or automation. Hence, the Aggregator integrating them will be most likely stationary, too, which means an unlimited access token can be issued and the amount of gateway provisionings is reduced.

In summary, the User Directory provides capabilities similar to a RADIUS-Server [132] commonly used in Triple-A systems (authentication, authorization, accounting) [3]. The User Directory allows services (i.e. Consumer Operators) to authenticate principals and provides a central repository for parameters necessary for efficient service delivery (i.e. the Consumer Profile). As discussed in the following section, accounting is managed by the Consumer Operators themselves.

6.1.3. Management Services

Management Services, provided by Consumer Operators, represent the functional backend infrastructure required to implement the device provisioning. As shown in Figure 6.5, a session layer, core services, and a set of supporting services are defined. Similar to the Directories, the session layer is used, upon request to an offered protocol, to authenticate a principal and create a session context. According to Section 5.2.2, two protocols are offered by a Consumer Operator: the Aggregator Management and the Provisioning Protocol.

The basic interactions have already been discussed in Section 5.2.3. However, upon discovery of a device, the Aggregator Agent uses the Management Protocol to connect to a Consumer Operator and evaluates whether the device is of interest or not, using the core services. If the Aggregator Agent is linked to a Consumer Agent with an appropriate entry in its Profile, an Integrations Request is triggered. Otherwise, it connects to a Consumer Agent representing the Consumer Operator itself and an Integration Offer can be triggered. This Consumer Agent does not necessarily require a Consumer Profile because the Consumer Operator can provide an internal policy to decide about Integration Offers. Both cases lead to an invocation of the Provisioning Service, which contains the logic required to negotiate a provisioning or possibly revoke an existing one. This

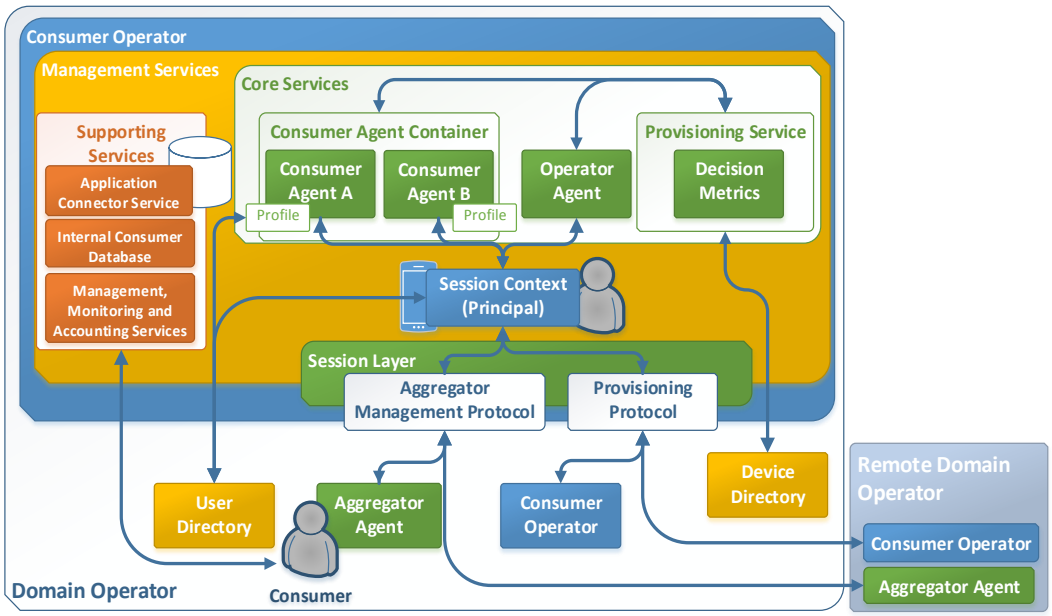


Figure 6.5.: High level Consumer Operator Management Services architecture.

basically includes considering the device state information provided by the Device Directory. As discussed in Section 5.2.3, additional information like the context (e.g. location of the device and Consumer, priority of the Consumer) or quality parameters (price, QoS) may be considered based on the application domain. If the device is operated by the Consumer Operator, the Provisioning Service is not required to further interact with other Operators. Otherwise, a negotiation phase is initiated using the Provisioning Protocol. An offer made by the remote Consumer Operator is subsequently forwarded to the involved Consumer Agent. In case of Integration Requests, the offer is matched against the *Guard Condition* of a respective Consumer Profile entry. Otherwise, an internal decision policy used by the Consumer Agent representing the Consumer Operator is applied. In general, the Consumer Agent is required to evaluate the offer. If the offer cannot be accepted, it either has to be rejected or a counteroffer has to be made (which can be based on a SLA framework [145]).

Additionally, Consumer Operators can provide a Provisioning Protocol client to their Consumers in order to involve them during a device access negotiation, which can be required if a Consumer has requirements beyond the scope of the expressiveness of the *Guard Condition*. It has to be considered that this feature introduces manual interaction and can introduce significant delay to the provisioning process. Regardless of the outcome of a negotiation, the Consumer Operator can decide, usually based on the application domain, whether or not the corresponding Device Instance entity will be locked until the negotiation is completed successfully. Even if the negotiation was successful, the latter case can result in a rejection of the request (e.g. if another request was processed in parallel).

As discussed in the previous section, a *Guard Condition* may also contain information about an integration schedule, or possibly about the amount of sensed data. This requires the Aggregator Agent to monitor the integrated device and collect respective meta data. Although the meta data will usually not contain data directly generated by the integrated devices, and thus transmitting this data to the Consumer Operator does not violate the security constraints discussed in Section 5.1.5, privacy needs to be considered (e.g. a Consumer may not want his Operator to know when a measurement was taken). Therefore, a Consumer willing to integrate a device has to agree on the collection of meta data, which in turn can be expressed using the *Guard Condition*. From a technical point of view, the Device Driver Module to be monitored has to provide appropriate functionality (i.e. generating meta data events). The terminology required to define and interpret these events is not further specified.

The Operator Agent is used in case of provisionings that require supervision by the Consumer Operator that has granted the access token (see Section 5.1.4). Basically, the Operator Agent can utilize the Provisioning Service to access private data attached to

a Device Instance and to verify if the requesting Aggregator Agent is allowed to access this information.

Besides the core services, the Management Services can provide a set of supporting services, used to support the Consumer Operator in its decision making and other processes required along with the provisioning of devices:

- **Application Connector Service:** This service can be used in case a Consumer has granted a Consumer Operator access to its Consumer Profile by setting the *EntityOperator* property. The service could maintain a set of application providers and define which provider is allowed to access which Consumer's profile. Additionally, a callback mechanism can be introduced to notify application providers about state changes of entries they have added to the profile.
- **Internal Consumer Database:** All principals of a domain are managed by the User Directory. However, a Consumer Operator needs to maintain an internal user database in order to define which Consumers actually have a customer relationship to the Operator. This is required as an enabler for accounting and billing processes.
- **Management, Monitoring and Accounting Services:** A database used to monitor the infrastructure a Consumer Operator is responsible for (e.g. which Aggregator Agent is linked to which Consumer Agent, which devices are integrated by which Aggregator Agent, meta data about an integrated device). Although status information like the binding between Aggregator Agents and devices can be gathered from the Device Directory, an internal cache can improve the overall performance. Monitoring services can additionally be used to provide contextual information considered during provisioning decisions.

6.2. Middleware

The Device Cloud Middleware is an execution engine for Platform Modules. According to the classification of the modules, the middleware consists of two major parts: the Device Layer Engine (DLE) and the Aggregation Layer Engine (ALE). Outside of the scope of the Device Cloud, both parts can be deployed independently (e.g. an ALE as an external aggregation application deployed on a dedicated server to execute resource intensive tasks). However, within the scope of the Device Cloud, both parts are required to properly link the DLE to applications and vice versa.

As shown in Figure 6.6, several core services, i.e Aggregator Agent Services, are required to manage the overall operation of the Device Cloud Middleware:

Device Cloud Middleware Core Services

Aggregator Agent:
The Aggregator Agent itself acts as the bridge to the Device Cloud infrastructure by providing client implementations of the required communication protocols (Ag-

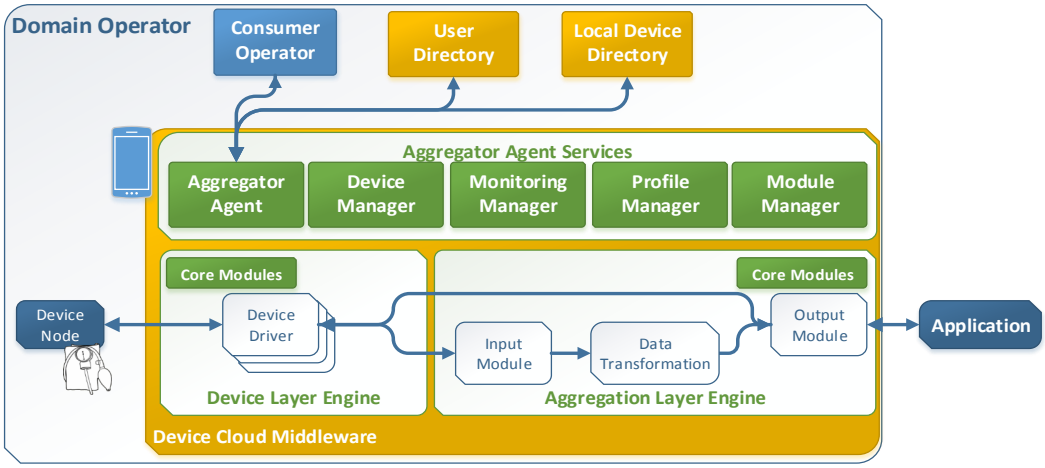


Figure 6.6.: Overview of the Device Cloud Middleware architecture.

gregator Management, DD Access Protocol). It maintains the private key of the Aggregator Principal and ensures that each request of its sub services is properly mapped to the protocols and authenticated. The Aggregator Agent is contained in a separate bundle because this bundle becomes the admin for managing OSGi permissions.

Device Manager:
The Device Manager lists the set of physical devices and Device Categories currently offered by the DLE of the Aggregator. The set of Categories can be greater than the actual set of physical devices integrated because one device may be described using several Device Categories or can refer to the composite class.

Monitoring Manager:
The Monitoring Manager is used to collect meta data and status information about integrated devices as discussed with regard to the *Guard Condition* in the previous section. Therefore, the Monitoring Manager subscribes for meta data events using the OSGi Event Admin Service.

Profile Manager:
The Profile Manager maintains the Consumer Agents linked to the Aggregator Agent, whereas the internal representation only contains the *entityID* of the Consumer Agent and a subset of the entries contained in the corresponding Consumer Profile. Regarding linked Consumer Agents, two sets are defined: local consumers and remote consumers. Local consumers refer to Consumer Agents that are directly linked to the Aggregator and are considered for Integration Requests (i.e. Consumers belonging to the Consumer Operator responsible for the Aggregator, or a

Consumer owning the Aggregator device – Bound Aggregator). Remote consumers refer to Consumer Agents that were linked due to an Integration Offer, which means they are not considered for Integration Requests, and only the particular profile entry corresponding to the Integration Offer is stored. For each Consumer Agent linked, the corresponding access tokens are maintained. Additionally, in case of an Operator supervised integration, the Profile Manager maintains private data which are possibly required .

Module Manager:
The Module Manager controls the lifecycle of the deployed Platform Modules. It maintains status information about each already deployed module, can resolve dependencies if new modules shall be deployed, and is able to uninstall modules in case of resource bottlenecks. If the Aggregator Agent has decided to install a Platform Module, it forwards the corresponding Platform Module entity to the Module Manager, which takes care of downloading the attachments (i.e. bundles) an dependencies (if necessary) and installs the bundle. Additionally, the signature of a bundle is verified. Thus, similar to the Aggregator Agent, the Module Manager is also contained in a separate bundle because it is granted the permission to manage permissions of other bundles.

6.2.1. Middleware Deployment

Deployment means that the Device Cloud Middleware is installed to an appropriate Aggregation Node as discussed in Section 5.1.2. It is presumed that the Aggregation Node provides a Java Virtual Machine (JVM). The bootstrap package contains an implementation of the OSGi platform (including all required dependencies), a bootstrap module, and additional resources like a configuration file or the public key of the Domain Operator the Agent shall register with. The bootstrap module will scan the configuration and download the publicly available Aggregator Agent core module from the Device Directory specified in the configuration file. The Aggregator Agent then employs the following steps to set up the middleware:

1. Scan the OSGi runtime for any existing services that conflict with core modules of the middleware.
2. Identify the Device Type of the Aggregator (e.g. using an entry in the configuration file or scanning the system with a Discovery Bootstrap Module).
3. Perform Aggregator registration as defined in Section 5.1.2. This can include user interaction (e.g. in case the Aggregation Node is a private Aggregator owned by a Consumer).
4. The Module Manager bundle is loaded and admin permissions are granted.

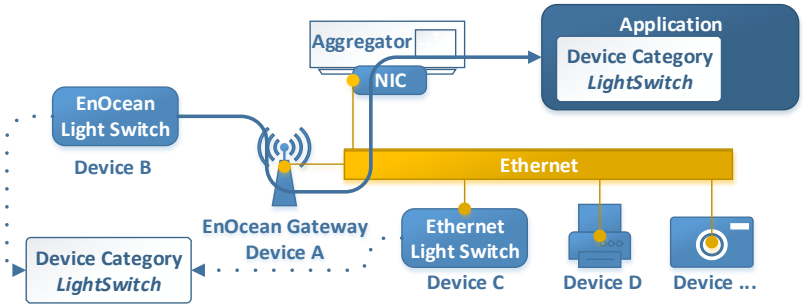


Figure 6.7.: Device integration use case based on two networking protocols.

5. The Module Manager is instructed to load the other core modules (DLE, ALE, and other Aggregator Agent Services) and properly initialize them.
6. After setting up all core modules, a Discovery Bootstrap Module is deployed (if not already done due to system identification). The Discovery Bootstrap Module aims at moving the DLE into the operational state by triggering the internal discovery and device integration processes.
7. If the Aggregation Node is a composite one, the Aggregator Agent can modify the *CategorySets* property, if it is willing to add its embedded devices to the device pool.

6.2.2. Device Integration & Abstraction

The Device Layer Engine (DLE) is based on the OSGi Device Access Specification (DAS) introduced in Section 2.4.2. Its main capability is to dynamically compose device control logic based on Device Categories (i.e. the interface abstraction model). Figure 6.7 illustrates a typical example, found in many application scenarios, which underlines the requirement to represent device control logic in a modular fashion while providing different levels of abstraction. Device B (light switch) is connected to the Aggregator based on two different networks that follow different protocols with respect to the OSI layers. However, a light switch will usually have the same control semantics regardless of whether it is attached using Ethernet or another networking technology. Thus, an application should only recognize the presence of a device compliant to a certain Device Category. Moreover, the application is usually not interested in the technical details of the integration and the underlying protocols used. However, the challenging issue raised by the use case consists in the fact that a gateway device can introduce a whole network which may follow completely different addressing schemes than the network the

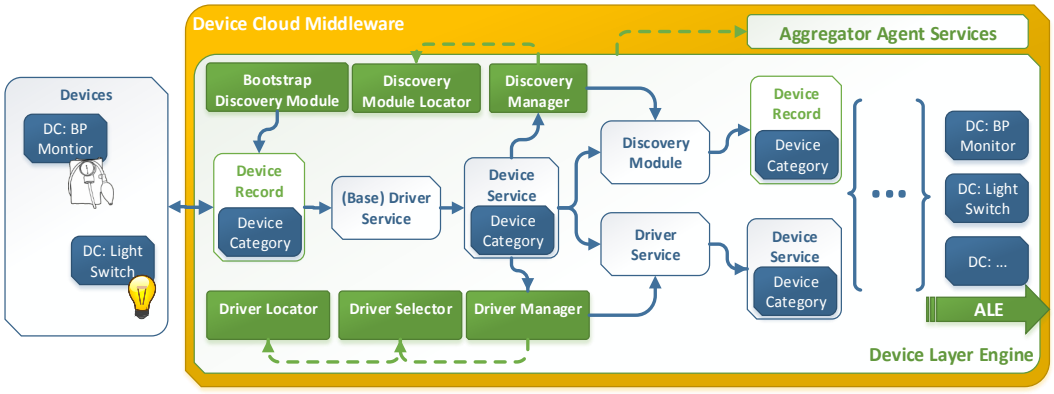


Figure 6.8.: Device Layer Engine (DLE) architecture based on the OSGi Device Access Specification (DAS).

Aggregator is attached to. In order to represent such kinds of topologies, a mechanism is required which is able to compose several device and protocol drivers in a transparent way. Given an ideal scenario, one single driver could possibly be used to integrate both light switches. However, this requires that the underlying protocol drivers (EnOcean, Ethernet) abstract the transport medium in an appropriate way. If this cannot be reached (e.g. due to protocol characteristics that cannot be abstracted), then at least the compliance with the Device Categories will guarantee the necessary level of abstraction required for interoperability at the application level.

As introduced in Section 2.4.2, the DAS models the composition of device control logic by establishing a relationship between Driver Services and Device Services based on the Device Category. If a new Device Service is registered, the Driver Manager can attach a suitable Driver Service, which in turn can register new Device Services. Driver and Device Services must specify the Device Category they belong to. Only one Driver Service can be attached to a Device Service. The first Device Service in the system appears as a result of the Discovery Bootstrap Module discussed in the previous section. In general, the DLE architecture is based on the DAS and extends it where necessary. As shown in Figure 6.8, the following components are defined:

Device Layer Engine Components

Bootstrap Discovery Module:
The initial Discovery Module delivered along with the deployment. Upon deployment it can scan the Aggregator and determine the physical host system (identify

the Device Type) if required. After the set up process has finished, it creates a Device Record corresponding to a Device Category, which represents the host system (e.g. Android smart phone xy). Based on the level of granularity given by the Categories, this can but does not have to already reflect the Operating System (OS). It is preferred to provide separate Categories in case of frequent OS updates.

Device Record:
A Device Record is the result of a discovery process conducted by a Discovery Module. It is an extension to the DAS, which defines how base drivers are deployed to the DLE. It must at least contain the Device Instance ID, the Category of the discovered device, and the Category of the parent device used during discovery (this does not apply for the bootstrap Device Record). Additionally, it can contain properties that describe how to connect to the discovered device (e.g. an IP address in case of Ethernet based devices). Which properties are mandatory and how they have to be represented, must be defined by the Category. A Discovery Module is not allowed to hold an open connection to the device because usually no access token exists during the discovery process. For instance, the first Discovery Record, describing the host system, does not require any properties. This is because the base driver, corresponding to the host system Category, knows that it has to identify the OS, and this is possible without additional discovery properties.

Base Driver Service:
Device and Driver Services are registered by Device Driver Platform Modules. As discussed in the DAS, base drivers cannot be deployed using the standard Device Manager attachment process, because no initial Device Service exists, the Driver Service could be attached to. Thus, a base driver refers to the first piece of device control logic interacting with a newly discovered physical device. Given the Category defined by the Device Record, a base driver can be located either using the *PlatformModuleSet* property of the Device Category or by inspecting the Platform Modules linked to the Device Type entity.

Device Service:
A Device Service represents the implementation of a Device Category. It can either be further refined by attaching another Device Driver or it can be consumed by ALE Aggregation Platform Modules. Additionally, it can allow to be consumed by Discovery Platform Modules, which must be indicated using a flag, a service property, or an interface. However, if multiple service consumers are bound (e.g. one Driver Service, multiple Discovery Modules, and multiple Aggregation Platform Modules), the Device Service must provide appropriate synchronization mechanisms. Driver and Device Services must maintain a service property representing the list of parents. Figure 6.9 illustrates a composition of Device Services, Driver Services and Discovery Modules.

Driver Service:
A Driver Service according to the DAS. Regular Driver Services can be attached using the Device Manager and the mechanism specified by the DAS.

Discovery Module:
Platform Discovery Modules that can be attached to Device Services using the Discovery Manager.

Discovery Manager:
Manages the process of attaching Discovery Modules to Device Services.

Discovery Module Locator:
Similar to the Driver Locator, the Discovery Modules Locator locates Discovery Modules appropriate for the Device Services present.

Driver Manager:
Specified by the DAS, the Driver Manager manages the process of attaching Driver to Device Services. Upon registration of a Device Service, the Driver Manager tries to attach Driver Services based on a matching algorithm, which basically utilizes the match values defined in the Device Category to judge which Driver Service is most appropriate. The decision can be refined using the Driver Selector. The Driver Locator is used to load Device Driver Modules from an external repository (i.e. the Device Directory).

Driver Selector:
The implementation of the Driver Manager is provided by a DAS implementation. If a further refinement of the matching process is required, the Driver Selector can be used to introduce possibly application domain dependent logic.

Driver Locator:
Specified by the DAS, the Driver Locator locates Device Driver Modules appropriate for the Device Services present. Only modules contained in the *PlatformModuleSet* property of the Device Category will be considered. Other potential drivers are already present due to the action of the Module Manager which has inspected the Platform Modules linked to the Device Type in advance.

The DLE is mostly based on events (i.e. using the OSGi Event Admin Service). Creation of a Device Record is announced using an event the Device Manager listens for. The Device Instance ID is used to trigger a Device Identification Interaction (through the Aggregator Agent). The result is stored by the Device Manager, which maintains records for each Device Instance known to the Aggregator. By inspecting the Consumer Agents linked using the Profile Manager, the Aggregator Agent now has to decide whether an Integration Request or Offer is triggered. An integration is triggered, if either a Request or an Offer resulted in a device access token being transmitted to the Aggregator Agent. The token is always bound to a Consumer and thus maintained by the Profile Manager. Upon registration of a new token, the Profile Manager generates an event indicating that a device with the given Device Instance ID has to be integrated (if the token was not issued due to a renew operation and the device is already integrated). The Module Manager inspects the Device Type corresponding to the Device Instance (using the

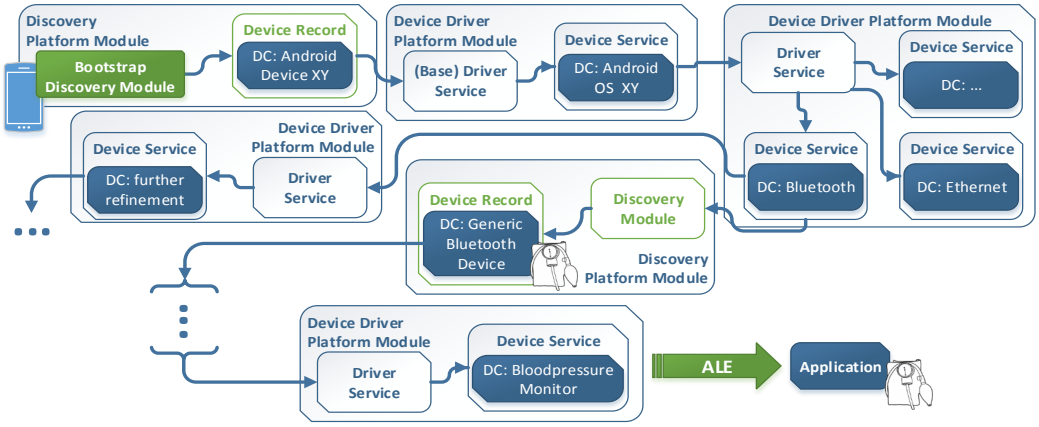


Figure 6.9.: Illustration of device control logic orchestration based on Device Categories.

Device Manager) and loads and deploys the Device Integration Platform Modules linked (if not already present).

In order to integrate the device, a base driver has to be initialized with the Device Record. Thus, after receiving the device integration event, the Device Manager has to identify the base driver required, using the following policy:

- Check the Platform Modules linked to the Device Type for Device Driver Modules compliant with the Device Category of the Device Record. If several exist, use the set of matching values to select the most appropriate one.
- If no such Device Driver Module exists, scan the *PlatformModuleSet* property of the Device Category. It must be verified that there is at least one path to the Device Driver Modules specified by the Device Type. Otherwise, an exception has to be thrown. An example where this fallback can become necessary is given by a Bluetooth Health Device Profile (HDP) blood pressure monitor. The Device Type only specifies a HDP-based driver, while the discovery has lead to a generic Bluetooth device, as shown in Figure 6.9. Hence, a generic Category must first be refined to an HDP-based Category.

After the appropriate base driver was selected, the Device Record is passed and the device is integrated. This triggers the device refinement and driver attachment mechanisms defined by the DAS. The Driver Manager listens for registrations of Device Services. If a new service has registered, it tries to attach a Driver Service. The Driver Locator is used to search for appropriate ones. The locator operates similar to the identification of

the base driver and triggers the Module Manager to load required modules. If multiple Driver Services are suitable for a Device Service, the Driver Selector is triggered. The Selector basically prefers modules linked to the Device Type, or validates that at least one path to the modules linked to the Device Type exist (which is a shortest path problem). Since each Device Service defines its parents, the relationship to the initial Device Record and thus the physical device is always given. This is also exploited by the Device Manager, in order to maintain its list of Device Categories currently bound to a Device Instance. Therefore, the Device Manager just listens for registrations of Device Services and updates its list.

Besides the Driver Manager, the Discovery Manager also listens for registrations of Device Services. If a Device Service tagged as suitable for attachment of Discovery Modules is found, the Discovery Module Locator is used to search for suitable Discovery Platform Modules. In turn, either the *PlatformModuleSet* property of the Device Category or the Platform Modules linked to the Device Type can be considered.

If the access token expires or the device integration was cancelled by the Consumer or due to an Operator supervised integration, an event is triggered either by the Profile Manager or the Aggregator Agent. The Device Manager receives the event and disconnects the base driver from the device.

Device Management

As introduced in Section 2.4.2, device management can be used to introduce a uniform representation of the device object models. Instead of linking the Aggregation Platform Modules directly to the Device Services, a management layer could introduce a uniform data structure to access the available device. However, this approach is only conceptional, because currently no Open Source implementation of the DmtAS is available.

The Device information base maintained by the Device Management Tree (DMT) is managed by the Tree Manager. The Tree Manager listens for device integration events and registration of Device Services. Therefore, the Tree Manager is notified every time a Device Service appears or disappears and gets access to the service reference of the respective Device Service. Based on the Device Category and the service property denoting the parent of the Device Service, the Tree Manager is able to modify the DMT data structure properly (i.e. adding, removing, or modifying nodes based on the operations and properties of the Device Category). As mentioned in Section 2.4.2, the structure of the DMT has an impact on the capabilities of how information can be accessed (e.g. searching for specific devices, or for all devices compliant with a certain category). The structure is not defined by the DMT Admin Service Specification (DmtAS) and is managed by plugins. The Tree Manager registers such a plugin that takes responsibility

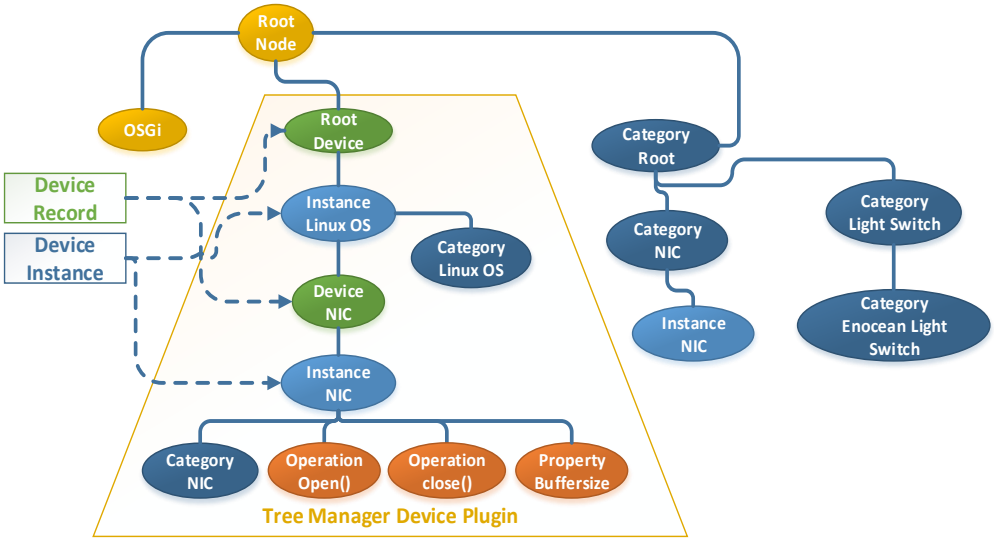


Figure 6.10.: Overview of the Device Management Tree structure.

for all device related subtrees of the DMT. Two subtrees are maintained by the plugin: a device-related subtree that reflects the actual topology of integrated devices and a category-related subtree that allows grouping and searching for devices based on Device Categories. An example is illustrated in Figure 6.10. Operations and properties of a device are represented by leaf nodes. The Tree Manager plugin is able to create these nodes based on the Device Category interface corresponding to Device Records and Instances. Whether a node refers to an operation (i.e. is executable) or to a readable and/or writable property can be defined using meta nodes. Additionally, meta nodes can be used to store extended knowledge (e.g. node descriptions), provide information about the data types of a node or offer validation information to verify constraints on the node values. The Tree Manager plugin is responsible for mapping the nodes to the corresponding Device Category implementation offered by the Device Driver that created the respective Device Service.

The category-related subtree reflects an important capability with regard to device abstraction. Categories are allowed to inherit from other categories. For example, a specialization of a light switch, additionally offering a dimmable feature, will still be compliant to the base light switch category and can thus be accessed by each application able to interact with regular light switches. A Device Service exposing a category that inherits from a base category is required to implement the methods offered by the base category, too. This is similar to the concept of interface inheritance used in many programming

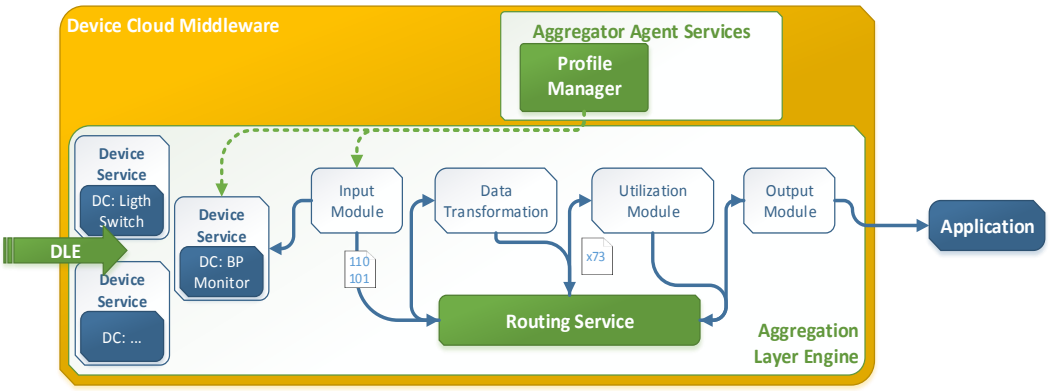


Figure 6.11.: Aggregation Layer Engine architecture.

languages.

6.2.3. Data Aggregation

The Aggregation Layer Engine (ALE) is the execution environment for Aggregation Platform Modules. The only additional core module introduced by the ALE is a Routing Service, which is responsible to properly route the data containers through the Aggregation Platform Modules.

As discussed, the device integration is triggered if the Profile Manager receives a valid access token. Besides triggering the integration, the Profile Manager uses the Module Manager to load all Aggregation Platform Modules required. Aggregation Platform Modules must support multiple sessions (i.e. processing of multiple data flows in parallel). As discussed in Section 6.1.2, each entry in a Consumer Profile consists of one or more paths. Each path must start with an Aggregation Platform Module that expects a certain Device Category as its input. The Profile Manager is notified by the Device Manager each time a new Category becomes available for an integrated Device Instance. The Profile Manager checks whether the Category matches the input requirements of one of the path entries. Additionally, in order to avoid "stealing" Categories introduced by devices granted to other Consumers, it is verified that one of the access tokens belonging to the Consumers matches the originating Device Instance. If the path entry is covered by the corresponding device access token (i.e. the *CategorySet* property), the Profile Manager links the Device Instance to the first Aggregation Platform Module in the path (either an Input or an Output Module) and notifies the Routing Service that the path was activated. Additionally, an empty session is initialized and linked to the

ID of the path. This is required because Aggregation Platform Modules can process multiple paths simultaneously, but each session may involve a different configuration. If the path has already been active, the *Guard Condition* decides, whether an additional session is created (i.e. the path is then applicable for multiple simultaneously integrated devices).

If an Input Module initiates the path, it translates the category based format to the generic container based format introduced in Section 6.1.1. Besides the payload, each container includes the path ID, a sequence ID, the ID of the module it has passed, and a references to the Device Instance the data originated from and the Consumer Profile the path originated from. Each module uses an event, to emit data containers. The event is received by the Routing Service, which determines the next Aggregation Platform Module in the path. Processing of a path is always terminated by an Output Module, which can forward the data to sinks outside of the Device Cloud.

Data Transformation

Transformation modules allow us to achieve semantic interoperability at the application layer. The ALE is able to dynamically load required transformation modules from the Device Directory. The approach behind the transformation modules can be referred to as template mapping [135]. Template mapping is an approach for data transformation between a source and a target model in order to achieve semantic interoperability. Template mapping assumes that two model instances, a source and a target instance, exist (e.g. specific medical device specializations). The predefined target instance then acts as a template, which is filled with dynamic values from the source instance (e.g. measurements). An example would be a blood pressure monitor, using a proprietary protocol which is translated to IEEE 11073. The Transformation Module is aware of the data formats used by the proprietary as well as 11073 protocol. 11073 defines a set of device type related object models. The Transformation Module knows this object model (i.e. the template) and can appropriately insert the values, upon the transmission of a measurement.

Compared to ontology-mapping approaches, template mapping requires providing multiple pairs of templates and their mappings. However, the approach is more lightweight and allows for better modularity, and it fits into the Platform Module-based device integration model. If a device driver module is provided by a vendor, a mapping module can be added easily. Ontology mapping would be the preferred approach if an openly disclosed and well documented abstract meta-model for each incoming data stream (proprietary or standard based) would exist, which is not the case for every vendor. Moreover, the adoption of required technologies like the Web Ontology Language [20] or the Resource Description Framework [89] is only partially supported or does not exist for

mobile and embedded systems since considerable resources are required. Thus, the template mapping approach fits to the general system model because it allows splitting the overall problem into several of lightweight modules, which can be deployed on demand.

6.3. Conclusion

In order to take the generic Device Cloud building blocks to operation, the following components need to be defined and provided:

- The Device Categories necessary to describe the interfaces offered by the devices. Based on the application domain, Device Categories can be derived from standardization activities (e.g. ISO/IEEE 11073 for medical devices).
- The Platform Modules necessary to integrate devices and pre-process the data streams.
- The Device Cloud assumes that a globally unique device ID can be reconstructed during the device discovery process. However, not all vendors will provide such capabilities. It may be necessary to provide mapping tables that translate from properties like vendor, product number, and serial number to the ID range used by the Device Cloud.
- The decision policies used by the Provisioning Service of a Consumer Operator must be defined. Multiple decision policies may coexist and can be applied, based on the application domain a device belongs to.
- As discussed in Section 6.1.3, additional monitoring parameters can be defined. These parameters can be used to refine the provisioning decisions or to enhance the expressiveness of the *Guard Condition*. As discussed in Section 5.2.3, the basic parameters like device location or device status may not always be sufficient for proper provisioning decisions.
- Tools that allow Consumers to create and update their Consumer Profiles need to be provided.

7. E-Health Application Scenario

Contents

> "e-health is an emerging field in the intersection of medical informatics, public health and business, referring to health services and information delivered or enhanced through the Internet and related technologies. In a broader sense, the term characterizes not only a technical development, but also a state-of-mind, a way of thinking, an attitude, and a commitment for networked, global thinking, to improve health care locally, regionally, and worldwide by using information and communication technology." - Eysenbach, 2001 [53]

7.1. E-Health Systems

Sufficient health coverage and health service delivery are crucial social and economic requirements for the evolution of a country's population and the improvement of living conditions. Leading industrial nations like the United States of America or Germany allocate about 19% of their governmental resources to the health care sector [118]. Therefore, it is important that the health service providing systems (i.e. health systems) properly distribute and utilize the available resources in view of the needs of the society. Key requirements of effective health systems, as defined by the World Health

Organization (WHO) [117], are to improve the health status and to defend the population against health risks, to protect people against financial consequences of ill-health, and to provide access to people-centred care. While health care basically can be defined as the diagnosis, treatment, and prevention of diseases, people-centered care puts emphasis on the health requirements of people and communities instead of focusing on the diseases themselves. This is related to an upcoming change of paradigm in patient treatment, where modern Information and Communications Technology (ICT) systems like telemedicine applications allow transforming the hospital-centered way of treatment to a more patient-centered one. This change is driven by challenges raised from social issues like the ageing society and urbanization, and by economic aspects like increasing costs. Health systems will have to adapt to these challenges in order to efficiently deliver quality health services in a way that preserves the quality of life and the independence of the patients.

One building block to boost effectiveness and quality of health service delivery is to introduce ICT systems that allow an efficient distribution and delivery of health information to the places where it is needed. This can be referred to as E-Health. Besides employing ICT systems to boost efficiency and productivity in the delivery of healthcare services, E-Health solutions are expected to provide and enhance [49]:

- interaction between patients and health-service providers
- institution-to-institution transmission of data
- peer-to-peer communication between patients and/or health professionals

Examples of E-health systems are health information networks in general, Electronic Health Records (EHRs), telemedicine services, and mobile and wearable personal health system (sometimes also referred to as mHealth) [49]. Although the Device Cloud basically deals with the integration and provisioning of medical devices and thus has a high relation to telemedicine and mHealth systems, all areas of E-Health are covered. From a general point of view, the Device Cloud covers the integration of ICT-based data sources (i.e. medical devices) into health information networks, regardless of whether the source is mobile or stationary, or refers to a device used in hospitals or in the scope of personal health systems.

7.2. The Data Dissemination Problem in E-Health

The evolution of ICT in the healthcare domain is heavily influenced by upcoming distributed architectures that integrate and facilitate medical sensors in a ubiquitous fashion [154]. Streams of medical data emitted by integrated medical devices can support physicians in their decision-making process. However, a large variety of heterogeneous

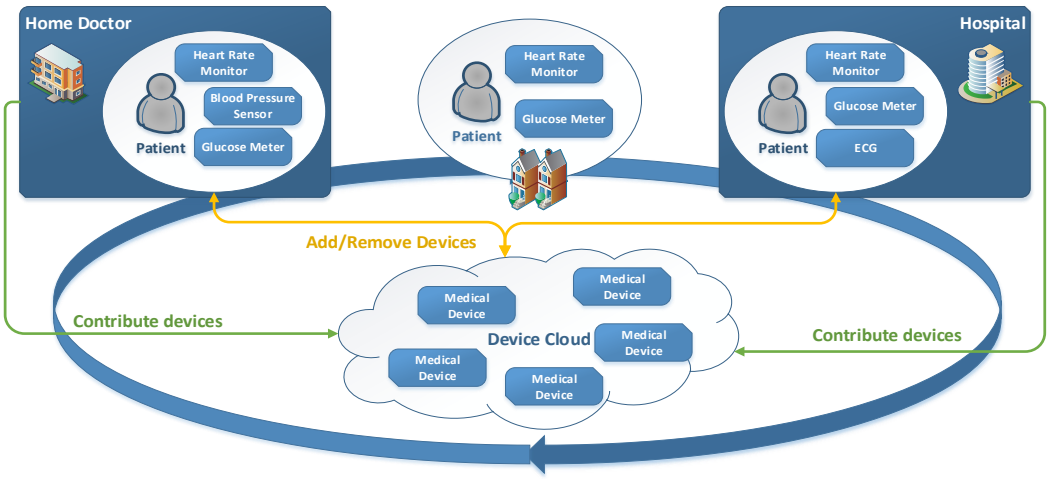


Figure 7.1.: Patient monitored by a set of devices dynamically provisioned from the Device Cloud based on the requirements

sensors has to be considered in order to get a meaningful survey of a patient's condition. Moreover, treatment decisions often have to be made under time constraints, which requires an aggregated view of the available data streams. Thus, besides the device integration challenge, data availability has to be considered with respect to the E-Health domain.

Treatment processes usually include several steps and institutions (i.e. CDOs), ranging from monitoring at home (i.e. telemedicine) to emergency transportation or different hospitals, where each location might be managed by a different authority. According to the brief introduction of the E-Health application scenario given in Section 4.1.1, patients can be already equipped with a set of wearable medical devices which are organized in a Body Area Network (BAN). Real time access to the emitted data streams could provide better knowledge to physicians. As shown in Figure 7.1, the BAN can grow or shrink at each location (i.e. new medical devices are integrated), in order to fit the set of medical devices to the current treatment situation. However, to prevent reattachment or replacement of the already given medical devices, it is required that the data streams can be accessed by each CDO that is involved in the treatment process. Providing access using sensors-virtualization or other Cloud-based services may not be feasible according to the constraints (e.g. real-time, privacy) discussed in Section 4.1. The capabilities offered by the Device Cloud allow CDOs to directly access the physical sensors and temporarily take control. This can be referred to as sharing the data sources instead of the data.

However, physical sensors do usually not provide persistent memory to store the recorded measurements. Thus, no history can be provided when relying only on the Device Cloud. As a result, a hybrid approach is required including systems that provide access to the treatment history of a patient and the Device Cloud, which provides real time data reflecting the current condition. The history of a patient can be maintained by EHRs. Currently, Cloud Computing approaches are investigated to solve the problem of efficiently and securely distributing EHRs among CDOs involved in a treatment process.

7.2.1. EHR Clouds

The adoption of Cloud Computing concepts for the e-Health domain both raises opportunities and challenges. Governmental initiatives and research funding for Cloud based e-Health services [50] show that Cloud Computing already found its way into the healthcare domain and is not just a concept under discussion any more. EHR-Clouds try to solve the data dissemination problem by increasing the data availability while paying respect to the serious privacy and security issues related to sharing health records in clouds [104].

Patient treatment nowadays is organized in a multi-tenant fashion, where multiple CDOs have to collaborate. Each participant in the treatment process must take knowledge about the patient's history and past treatments into consideration, while making own decisions. Knowledge about a patient is stored in patient records, where according to the HIMSS definitions [60], one has to distinguish between Personal Health Records (PHRs), Electronic Medical Records (EMRs) and Electronic Health Records (EHRs). A PHR should provide a complete summary of the health status and the medical history by gathering information from various sources, like EMRs or EHRs. These records are usually maintained by an individual (i.e. the patient) and allow making the health status information available for those who are involved in the treatment process. EMRs are maintained by CDOs and are used to represent and document the health care services delivered to a patient by the maintaining CDO. In most cases each CDO hosts its own database to store EMRs. In order to share this knowledge between CDOs involved in a treatment process, EHRs can be used. An EHR is a subset of the knowledge maintained by EMRs of the involved CDOs. This means that an EHR is used to provide the knowledge required for present and future health care decisions and to exchange this knowledge between participating CDOs. Based on the EHR definition, the primary purpose of an EHR Cloud Application is to obtain relevant knowledge from the EMRs located at different CDOs and to distribute it among involved health care providers. Therefore, the main challenges for EHR Cloud Applications are related to security and privacy [105].

As the content of an EHR is usually collected from several EMRs, it has to be defined how the access to EMRs by an EHR application can be managed. In the matter of privacy, it has to be considered that a patient might only want to make parts of the EHR available to physicians, who, for instance, are only involved in a specific subset of the overall healthcare services delivered during the treatment process. Another crucial requirement is the authenticity of the data represented by EHRs. It has to be ensured that the author (i.e. a physician or a CDO) can be verified, which basically refers to the process of data authentication [40]. Treatment decisions based on altered or non-authentic data can cause serious damage to the health of a patient. Seen from a physician's point of view, the capability to collect data from multiple EMR/EHR repositories in a scalable and secure way is important, since a physician might have to treat patients whose data originate in different EMR systems. This is related to the EMR access management challenge. Moreover, gathering access to patient data stored by multiple CDOs requires an access control model that involves multiple entities because both the patient's and the respective CDO's authorization are required. Finally, ensuring the data integrity is important, since undesired changes to the data or any loss of information have to be avoided. This is a critical issue when considering multiple CDOs that are updating an EHR, as knowledge might get lost or lose accuracy if update processes are not properly managed.

7.2.2. Application Scenario

In order to extend the EHR Cloud capabilities with access to real time data, the device Cloud needs to link the EMRs with the moving medical devices. Medical devices are treated as resources of patient-related data. In general, a data resource can be considered as required if it becomes visible to the network of a CDO involved in the treatment of a patient. Thus, the patient takes the role of the *Device Target* while the *Device Consumer* role, based on the treatment process, is transferred between the participating CDOs and the patient himself (i.e. devices are linked to EMRs or to a PHR). The required data resource can be booked and integrated, based on the Pay-as-you-go usage model offered by the Device Cloud. Based on the type of the deployment, each hospital or other large medical facilities can become Consumer Operators. The Domain Operator could be given by an independent third party (e.g. a health ministry or a governing body of CDOs). Regular domains could be organized on the basis of federal states, while the Root Domain could refer to the country.

According to Section 4.1, two medical device-sharing principles can be distinguished based on the *Device Target* (i.e. the patient).

Medical Device Sharing Principles

Optimizing Device Utilization:
Based on the Device Cloud provisioning capabilities, a medical institution (or even a single individual) can optimize the utilization of owned medical devices. It is assumed that a pool of devices exist and that each device can be provisioned dynamically based on the demands of the patients. A hospital could efficiently pool the medical devices even among several sites. This sharing principle is based on a changing *Device Target* (i.e. medical devices are provisioned to multiple patients based on their requirements).

Optimizing Data Availability:
In contrast to optimizing the utilization, optimizing the data availability is based on an unchanged *Device Target*. This principle is reflected by the application scenario introduced in Section 4.1.1. A patient, already bound to a set of medical devices, is moving between several CDOs involved in the treatment process. As shown in Figure 7.2, the patient holds the *Device Target* role, while each CDO is acting as a *Device Consumer*. *Device Owner* and *Device Operator* can refer to each CDO, the patient himself (at least the *Device Owner*) or even a third party (e.g. a health insurance or a telemedicine provider). The initial binding of the devices to the patient can be due to participation in a telemedicine program or because of any other type or treatment (e.g. a patient is transferred from a general hospital to a specialized clinic while the devices remain attached).

Accordingly, the definition of a valid *Device Target* is of crucial importance for E-Health application scenarios. Without a *Device Target* defined for a Device Instance, a CDO would not be able to link an integrated device to a patient or its corresponding EMR. Hence, the data would be useless. If only one CDO is involved in the treatment, the mapping from devices to patients could be maintained externally (e.g. the clinical information system). However, as soon as multiple CDOs are involved, the mapping has to be maintained by the Device Cloud. As discussed in Section 4.4.2, the *Device Target* can be attached in form of configuration entries to the Device Instance or the Device Locks. For the sake of simplicity, it is assumed that the information is attached to the Device Instance because usually most medical devices refer to the functional device class of Exclusive Transducer Devices. The *Device Target* must be expressed as an identifier common to all participating CDOs (e.g. health insurance number or identity card ID).

In case of the E-Health application scenario, the Device Cloud requires that each Integration Request, resulting in a state transition from Idle to Consumer Bound, contains a definition of the *Device Target*. This basically covers the utilization sharing principle. In case of the availability sharing principle, where the medical devices are moved between CDOs but are still bound to the patient, an Integration Request presumes that the Device Instance is already in the Consumer Bound state. Note, that the *Device Consumer* role differs from the patient (i.e. the Device Target) in this case. This can be envisioned

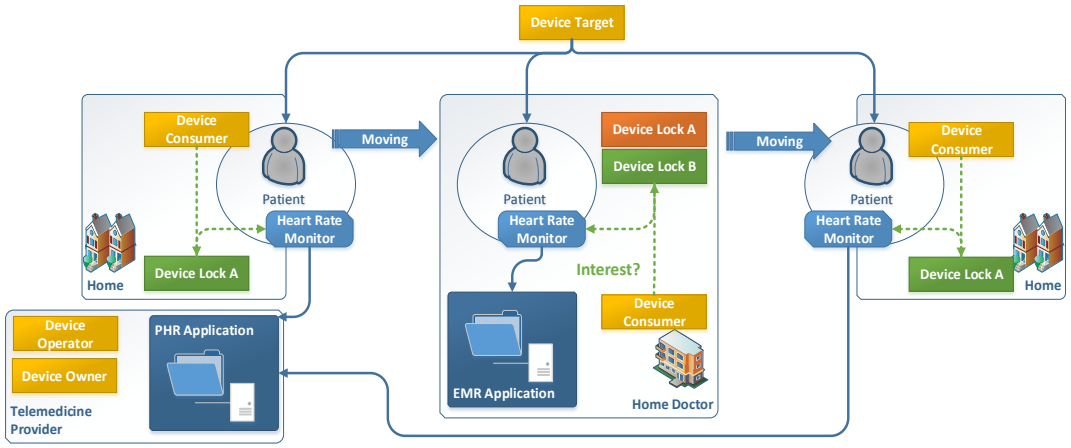


Figure 7.2.: Optimizing data availability sharing principle – fixed Device Target and changing Consumer

as follows:

- A set of devices is bound to a patient for a certain period of time by requesting a device lock and defining the *Device Target*. The initial *Device Consumer* could be the patient himself (i.e. its PHR application), a home doctor or a telemedicine provider.
- A CDO involved in the patient's treatment discovers devices that are either in Consumer or Aggregator Bound state and are linked to the patient through the *Device Target*.
- An access request would lead to a temporary revoke of the initial device lock (based on the *Device Operator's* decision policies). Subsequently, a new lock is issued to the CDO, which becomes the *Device Consumer*.
- The E-Health use case requires that the decision policies of the *Device Operator* remember the initial device lock that was revoked. If the CDO releases its lock, the *Device Operator* has to restore the initial lock.

The definition of the *Device Target* can be established automatically, semi-automatically or manually, which depends on the Aggregator and the medical device involved. If the Device Type describing the medical device denotes that the device is capable of providing a patient identifier, definition of the *Device Target* is not mandatory. ISO/IEEE 11073 based devices provide an appropriate entry within their object model. However, the implementation of this entry is optional. If not defined by the device itself, the *Device Target* needs to be defined during the provisioning process. Bound Aggregators (i.e.

Aggregators bound to one Consumer – see Section 4.4.4) belonging to the patient, can establish the binding in a semi-automated fashion. Manual interaction is required if multiple devices of the same type are in close proximity to the patient (i.e. it has to be ensured that the patient actually uses the device that was provisioned to him). In case of unbound Aggregators (e.g. in hospitals), the definition of the *Device Target* has to be established manually. Therefore, an unbound Aggregator must be configured to display a *Device Target* dialogue upon the discovery of an Idle Device Instance (either using an integrated display or another control panel). This is basically triggered during evaluation of the *Guard Condition* by the Consumer Agent (after the provisioning has been negotiated successfully).

7.3. Medical Device Interoperability – x73

The development of the ISO/IEEE 11073 family of standards (x73) started in 1982 in order to provide interoperability and Plug-and-Play functionality for medical devices. The main application domains of the first versions were hospital and clinical environments. Due to the dissemination of mobile and wearable medical devices, effort was made on improving the original standard towards telemedicine and Personal Health Devices (PHDs) environments [166]. Currently, x73 is promoted by a huge industry consortium (Continua Health Alliance [37]) to become the major standard in the area of personal health devices and has also been elected as the basis for the Bluetooth Health Device Profile (HDP). Out of the x73 standard family, the following parts are important for this approach:

- *11073-10101 Nomenclature:* A basic nomenclature to enhance semantic interoperability by providing a common meaning of numeric values throughout components in the system [78].
- *11073-10201 Domain Information Model (DIM):* Describes an object-oriented self-descriptive approach to model medical devices, their configuration and, capabilities [79].
- *11073-20601 Optimized Exchange Protocol:* Defines the transformation of an x73-DIM to an interoperable transmission format optimized for PHD environments [75].
- *11073-104xx:* Device specializations composed of a subset of available classes and services in the DIM (e.g. blood pressure monitor, weighing scale). These specializations can act as the foundation for the definition of Device Categories. If a proprietary device is not covered by the existing specializations, a new one has to be defined.

In terms of x73, medical devices are called agents, and devices in the aggregation layer (e.g. a smart phone) are called managers. As shown in Figure 7.3, the basic concept

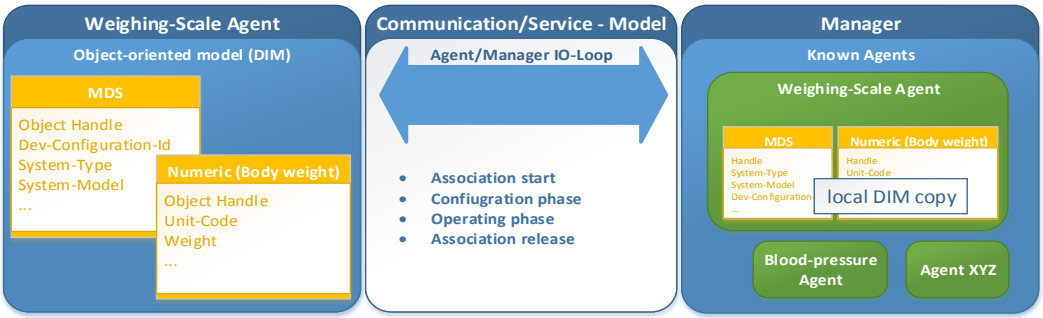


Figure 7.3.: Overview of the x73-20601 protocol between an Agent (Device Node) and a Manager (Aggregation Node)

of medical data exchange is to establish a connection between an agent and a manager and to create a local copy of the agent's DIM at manager side by using a service and communication model. The invocation of defined services allows the manager to keep its local copy up-to-date when new measurements are provided by the agent. The manager provides the recorded data to higher application layers (e.g. GUI components). In terms of the Device Cloud, a manager is represented by a Device Driver Platform Module, which provides an implementation of a Device Category corresponding to the respective 11073-104xx specialization. Besides integrating an x73-compliant device, two possibilities to integrate proprietary devices using x73 and the Device Cloud exist:

Mapping between proprietary devices and x73 representations

DLE based:
A refining Device Driver Module can be attached to the Device Service integrating the proprietary device. In case of, for instance, a blood pressure monitor this basically results in a transformation of the basic BP monitor category to a x73 BP monitor category.

ALE based:
An ALE execution path containing a Transformation Module must be defined. The Transformation Module basically translates between the proprietary input format of the device and x73.

Implementing the DLE-based approach is usually more complex because a complete x73 based manager needs to be provided. It has the advantage that the exposed Device Category is aligned to x73 and thus allows triggering control commands defined in the specification (e.g. through an Output Module). The second approach is more lightweight and easier to implement. It has the disadvantage that only the control commands

provided by the Base Category can be used because transformation to x73 only applies to the payload of the corresponding data stream within the ALE.

The nomenclature defined in x73-10101 (i.e. medical data information base (MDIB)) provides a common data dictionary applicable to a broad range of vital signs ranging from intensive care (e.g. ECG) over laboratory to common parameters, like weight or blood pressure. In x73, the nomenclature is primarily used to specify attributes that can appear in data streams (i.e. protocol data units) and are not statically defined. This allows communication partners to obtain a common semantic understanding of the exchanged data. An entry (i.e. term) in the nomenclature basically consists of a term code and a human readable reference identifier. For efficiency reasons, all nomenclature terms are organized in partitions (e.g. dimensions), where each partition has a set of private term codes that allow for vendor specific extensions. Using terminology management concepts like the Rosetta Terminology Management project [73], started by the Integrating the Healthcare Enterprise (IHE), allows one to extend the nomenclature in case of proprietary devices that are not completely covered.

Besides the nomenclature, x73 defines a DIM, which consists of several classes and attributes that are used to model medical devices in an object-oriented fashion. Each class and attribute is referenced using nomenclature codes, thus interoperability is ensured through preserving the same semantic meaning among different implementations. A model of a medical device (i.e. agent) is composed of a set of objects that refer to the data sources accessible by manager device drivers. The set of objects and the corresponding attributes are usually defined by device specializations each of which correlate to a specific medical device (e.g. blood pressure monitor). Each specialization picks out a defined subset of objects and attributes available in the DIM to define its intended functionality. In terms of the template mapping approach used by ALE Transformation Modules it is important, that specializations define a static (e.g. system type) and dynamic (e.g. measurement value) set of attributes. It is assumed that a Transformation Module has predefined knowledge about the DIM of a specialization (i.e. the template) based on the Device Category. In case of proprietary medical devices, only the dynamic attributes are exchanged and can be merged into the existing static part of the DIM (i.e. the template). Therefore, each incoming proprietary data stream to be transformed has to be matched against a device specialization available in x73, or to be added to the system subsequently.

7.3.1. x73 Implementation

I order to easily support the DLE and ALE-based device integration approaches, a Java based x73-20601 implementation was developed. The implementation covers the whole x73-20601 standard and allows for a quick integration of x73 device specializations.

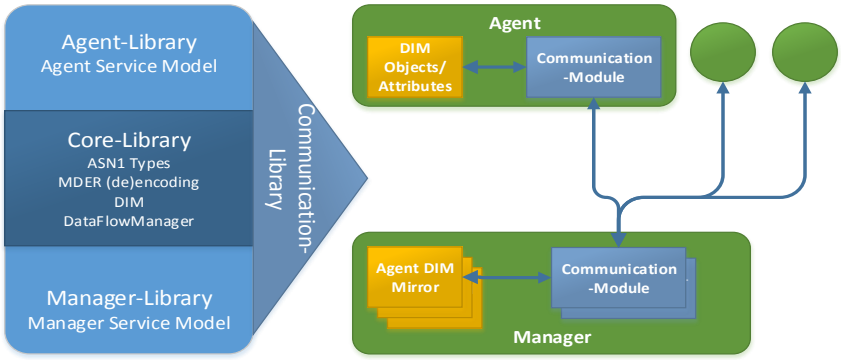


Figure 7.4.: Overview of the Device Cloud x73-20601 implementation

Moreover, the definition of own specializations in order to cover device types which are not already part of x73 is supported by plugging together existing DIM types. As shown in Figure 7.4, a set of libraries provide the necessary functionality to compose a x73 compliant manager (i.e. Device Driver Module).

- *Core-library:* implements all ASN.1 based data types, the Medical Device Encoding Rules (MDER), the entire DIM and interfaces and modules to execute services on the data contained in the DIM
- *Agent-library:* implements the agent-side functionality defined in the x73-20601 service model, the agent state machine, and some utility methods to quickly create agents and manage their communication
- *Manager-library:* implements the manager-side functionality defined in the service model, the manager state machine, and utility methods to store configurations of already known agents, manage agent discovery and the communication
- *Communication-library:* this library implements communication modules that can easily be bound to either agent or manager, allowing them to communicate over Sockets or Web Services (based on DPWS).

The Agent and Manager libraries provide a set of services used to access or modify the DIM and an execution environment for these services. Basically, the DIM, representing the object model of a medical device, is linked to an execution environment. Upon receiving a control command defined by x73, the appropriate service is selected and executed on top of the DIM. In order to test extensions made to the x73-104xx device specializations, an agent and a manager simulator were developed. As shown in Figure 7.5, the simulator allows triggering control commands and inspecting the DIM and the current state of an agent to manager association.

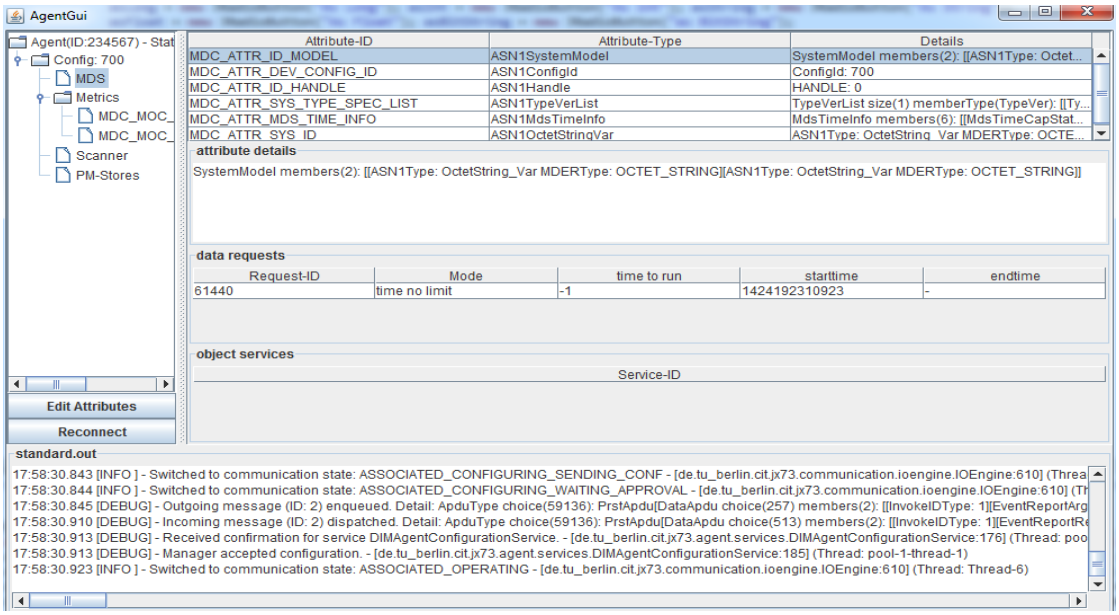


Figure 7.5.: x73 Agent (medical device) simulator

7.4. Device Cloud Deployment

This section will discuss the steps required to take the Device Cloud into operation in line with the conclusion in Section 6.3. As discussed in Section 7.2.2, it is assumed that larger medical institutions (e.g. hospitals) act as Consumer Operators while an independent third party acts as the Domain Operator (e.g. a governing body of CDOs). All other participants (patients, home doctors) act as regular Consumers. CDOs will usually install unbound Aggregators because multiple patients are served. Rather, patients themselves can be equipped with bound Aggregators (e.g. smart phones) to monitor their vital signs. The sink of the data streams is defined by the Consumer Profile, while the profile either belongs to the patient himself or a CDO having requested access to the medical devices. Unbound Aggregators (i.e. Aggregators used by CDOs) will usually forward the data to a sink within the scope of the CDO they belong to (e.g. the clinical information system or the EMR system). Bound Aggregators will forward the data to a sink defined by the patient (e.g. a PHR application hosted by a telemedicine provider).

7.4.1. Medical Devices

Based on the requirements of the research projects the present Device Cloud concept evolved from (in particular Smart Senior, Forschungscampus Connected Living, RehaIn-

teract), several devices have been integrated. Due to the ongoing progress in developing the Device Cloud concept, not all implemented device drivers align already strictly to the current specification. The following ones are related to E-Health use cases:

- *Blood Pressure Monitor:* A Bluetooth 2.0 based blood pressure monitor manufactured by Boso. A proprietary protocol is used to transmit measurements. The listing 7.1 shows a simplified Device Category for blood pressure monitors.
- *Weighing Scale:* A weighing scale compliant to the ISO/IEEE 11073-10415 weighing scale device specialization. The Bluetooth Health Device Profile (HDP) is used to transmit measurements.
- *Position and Pressure Sensors:* Within the scope of the RehaInteract project, which targets the support of rehabilitation processes using medicals sensors embedded in training devices, position and pressure sensors were integrated (e.g. a prototype of a shoe embedding both sensor types).

Listing 7.1: Blood Pressure Monitor Device Category

```
public interface BaseCategory{
        public String getSerialNumber();
        public String getModelNumber();
        public String getVendor();
        ...
}

public interface BP_Category extends BaseCategory{
        //match values used by DLE (DAS) driver attachment
        public static int MATCH_DEVICE_CLASS = 1;
        public static int MATCH_DEVICE_VENDOR = 2;
        public static int MATCH_DEVICE_MODEL = 3;
        public static int MATCH_DEVICE_MODEL_REVISION = 4;

        public BPMeasurement getBloodPressure();
        //either MDC_DIM_MMHG or MDC_DIM_KILO_PASCAL
        public int getUnitCode();
        //push new measurements to handler
        public void registerCallbackHandler(BP_InputModule im);
        ...
}

public interface BP_Pulse_Category extends BP_Category{
        //unit fixed to MDC_DIM_BEAT_PER_MIN
        public double getPulseRate();
```

```
        ...
}

public class BPMeasurement{
        double MDC_PRESS_BLD_NONINV_SYS;
        double MDC_PRESS_BLD_NONINV_DIA;
        double MDC_PRESS_BLD_NONINV_MEAN;
        Calendar MDC_ATTR_TIME_STAMP_ABS;
}
```

The Category definition in 7.1 is based on the assumption that each blood pressure monitor can deliver a systolic, diastolic and mean value. According to the ISO/IEEE 11073-10407 specialization, some devices may optionally provide the pulse rate. Hence, an extended Category can be defined.

The proprietary devices were integrated and aligned to x73 using the ALE-based approach introduced in Section 7.3. Thus, Data Transformation Modules were provided. Since x73 does not define a specialization for position or pressure sensors, the proprietary payload was aligned to x73 by adding appropriate terms to the private section of the x73 nomenclature. Therefore, the Transformation Module uses a static x73 template for each sensor type and maps the dynamic measurements into it, using the added nomenclature terms as semantic identifiers. The actual values are mapped to x73 data types already existing in the 20601 specification. Hence, standard x73 data (de)encoders can be used to process the data after transmission. For most sensor types the data types already provided by x73 will be sufficient and only nomenclature terms have to be added. Basically, x73-20601 defines all kinds of primitive types (bit strings, signed and unsigned floats/integers, octet strings) as well as compound and list structures required to express numerical, wave form, or textual measurements.

7.4.2. Medical Device Sharing

Based on the defined Device Categories, added Platform Modules, and created device descriptions (Device Type, Device Instances), the Consumer Operators have to define the decision policies as introduced in Section 5.2.3 (i.e. provide an implementation of the Provisioning Service discussed in Section 6.1.3). Before the policies can be applied due to an Integration Request, a CDO (i.e. Consumer) has to evaluate its Consumer Profile and decide whether it is interested in a device.

In case of the E-Health application scenario, the evaluation must include the *Device Target* (i.e. the CDO has to decide whether it participates in the treatment process of

the patient). The challenge is given by the fact that an unbound Aggregator discovering a device does not have any knowledge whether the device itself, the device type, or the patient represented by the device are of interest. Moreover, the discovery process will usually not provide sufficient knowledge that allows mapping the device to a patient. Hence, the *Device Target* has to be identified using the Device Identification Interaction. The E-Health application scenario presumes that all CDOs registered as Consumers are allowed to access the respective property, which is protected by the Device Directory. If the *Device Target* cannot be accessed, is not defined or cannot be interpreted, the device can only be integrated if it is in Idle state. Having identified the device and its *Device Target*, the Aggregator hands over the results to its corresponding Consumer Agent (i.e. the Agent representing the CDO).

According to the general definitions, the Consumer Agent will evaluate its Consumer Profile before triggering an Integration Request. A naive approach to cover devices of a particular patient would be to access the patient's Consumer Profile and copy all entries related to medical devices. This is not feasible because the entries in the patient's Profile may be aligned to different requirements (e.g. Transformation and Output Modules) or may contain devices that are not applicable in general (e.g. because the clinical information system of the CDO cannot process the data). Therefore, the Consumer Profile of a CDO needs to contain default entries for all suitable medical devices. The *Guard Condition* can be used to express that multiple devices per entry are accepted. Each default entry contains the definition of Aggregation Platform Modules and Module Paths required to properly forward the recorded data to the clinical information system in the desired format. However, the *Guard Condition* has to additionally evaluate the *Device Target* (i.e. patient identifier), which is only possible by linking to the clinical information system of the CDO. The Device Cloud itself is not designed to maintain any patient specific knowledge and thus cannot determine whether or not a CDO is involved in the treatment of a patient.

If the patient and the device are of interest, the Consumer Agent will trigger an Integration Request. The decision policies employed within the scope of the E-Health scenario can be based on a simple priority mechanism. Therefore, the following basic classification of Aggregators is proposed:

Classification of Aggregators

1. **Normal Operation:** The Aggregator is bound to the patient. The data is forwarded as specified by the patient's Consumer Profile (e.g. telemedicine provider).
2. **Priority Operation:** An Aggregator belonging to a medical institution visited by the patient due to non-critical issues (e.g. a medical practice).
3. **Critical:** An Aggregator belonging to a preferred medical institution (e.g. a hospital), but not deployed in an emergency area/ward.

4. **Emergency:** Aggregators deployed in an emergency area/ward. The Emergency priority may have several levels. A vehicle (e.g. ambulance, helicopter), used to carry a patient to an emergency ward may have a lower level than the emergency ward itself.

The priorities can be further refined based on a classification of medical institutions or even wards within one hospital. If a patient was admitted by a CDO due to an emergency and is subsequently moved to a ward with lower priority (i.e. the installed Aggregators have the Critical or the Priority Operation priority assigned), a handover is still possible because the Consumer is not changed. However, some exceptional cases have to be considered regarding the priority-based decision policy.

Medical Device Sharing - Exceptional Cases

Non-involved CDO:
Judging the interest in a device by the CDO itself is of notable importance. This cannot be achieved by simply checking whether the patient is known to the clinical information system but must additionally consider if an active participation in the current treatment is required. Otherwise, conflicts may arise if two CDOs reside at the same location (i.e. building) or a patient is just walking by a CDO knowing him due to a previous treatment.

Unexpected/Emergency admission of patient:
Judging the interest is challenging in case of emergencies like accidents or disasters. The patient may not be known to a CDO at all. Furthermore, in case of several patients involved in an accident, the Aggregator of an emergency physician may not be able to determine the set of devices the physician is interested in because several devices belonging to different patients may be in close proximity. The unexpected admission of a patient can be treated as a specialization of the non-involved CDO case.

Paired devices:
Some device types, especially Bluetooth based devices, cannot be discovered while already being paired with an Aggregator. Thus, if a patient moves to a CDO while his Aggregator is still active, the CDO may not be able to notice the device.

A possible solution to these exceptional cases can be realized based on Near Field Communication (NFC) enabled marker devices, such as RFIDs. As shown in Figure 7.6, these marker devices stay discoverable and allow identifying the *Device Target* (i.e. the patient). Similar to solutions like micro-payment, knowledge of the patient identifier, gathered due to the close range to the patient, in conjunction with the priority of the CDO could be treated as an authorization to access the medical devices of a patient and simultaneously validate the interest of the CDO. Searching the Device Cloud infrastructure for medical devices bound to the identified *Device Target* can be employed

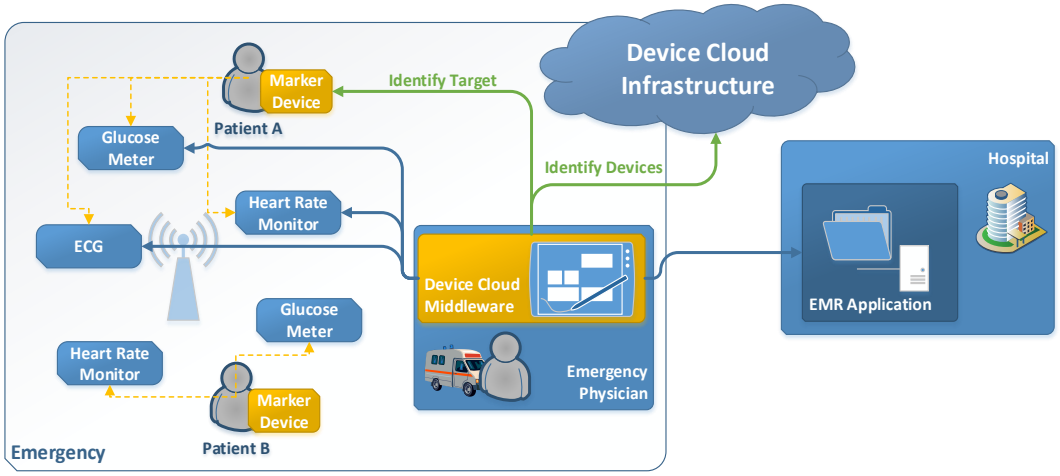


Figure 7.6.: Device Target identification using NFC enabled marker devices

to check for paired/non-discoverable devices (either by accessing the Consumer Profile of the patient or using a separate *Device Target* to device mapping service). Marker devices can additionally help to mitigate the problem of manually defined *Device Targets* as discussed in Section 7.2.2.

Finally, Figure 7.7 shows a simple decision policy derived from the priority based Consumer classification. In case an Operator serves several application domains, the appropriate decision policy can be selected based on the Device Category. For simplicity reasons, the shown decision tree assumes, that access to an exclusive device is requested and that the device is already locked. If the Consumer does not already hold the lock, the Device Target is evaluated. In case the Target remains, the *Optimizing Data Availability* sharing principle is applied and the priorities of the existing lock and the request are compared. Otherwise, the request is only accepted, if the requesting Consumer is equal to the *Device Owner* and hence always preferred. As discussed in Section 7.2.2, the *Optimizing Device Utilization* principle is based on a single Consumer provisioning its devices to several Targets (i.e. a hospital provisioning its devices to several patients). Thus, this principle is covered by simply switching the Aggregator or extending the Device Lock as shown by the right hand side of the decision tree.

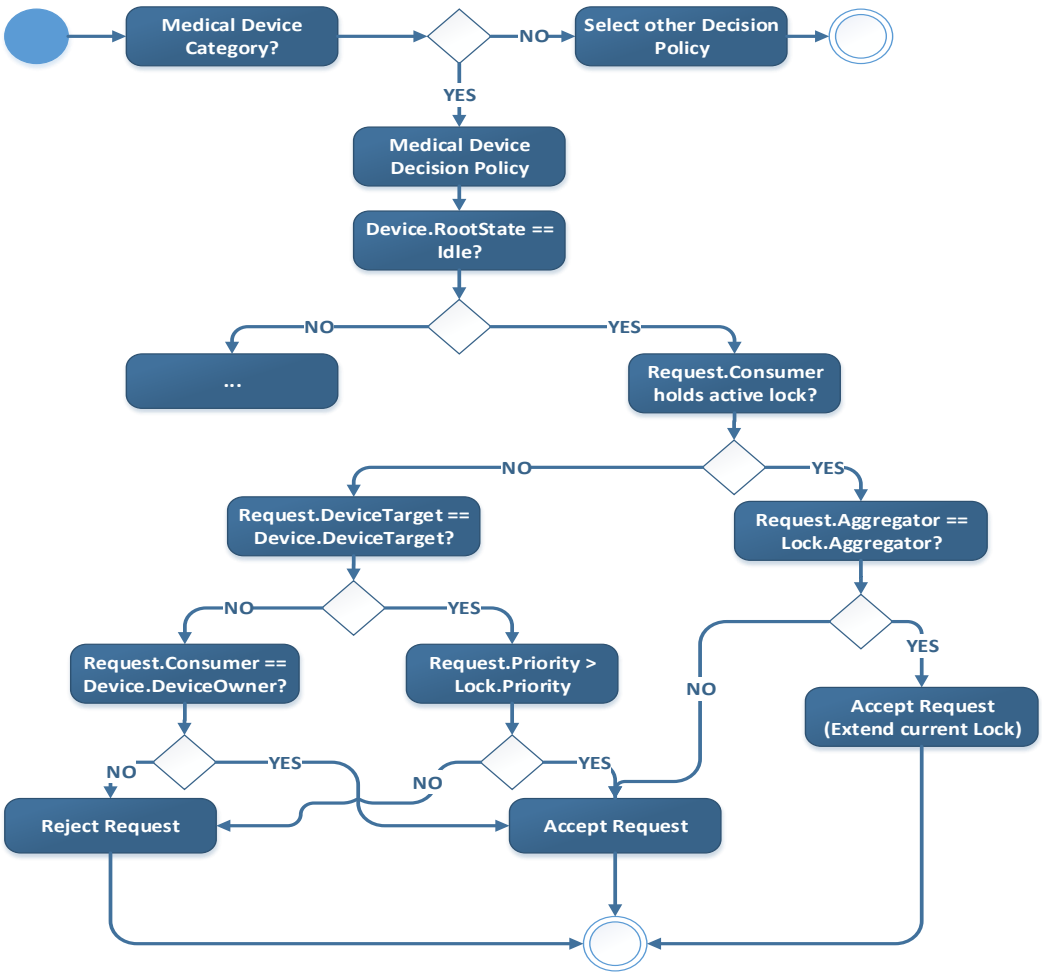


Figure 7.7.: Priority based medical device decision policy

7.5. Conclusion

This chapter demonstrated the application of the generic Device Cloud approach to a specific use case, which was chosen from the E-Health domain. As a refinement of the general sharing principles discussed in Section 4.1, the *Optimizing Data Availability* and the *Optimizing Device Utilization* strategies were discussed. Basically, the strategies cover the handling of the *Device Target* (remaining Target – Optimizing Data Availability; altered Target – Optimizing Device Utilization).

Taking the Device Cloud into operation requires specifying and developing Device Categories and Platform Modules for the medical devices to be deployed. Furthermore, a decision policy used to resolve access conflicts must be defined. Similar to scheduling mechanisms, an approach based on static priorities was proposed. Moreover, it was pointed out that judging the interest in a device and specifying the binding between a device and a *Device Target* (i.e. patient) is of crucial importance for the E-Health use case. A solution based on NFC enabled marked devices was proposed in order to cope with situations like the emergency admission of a patient or competing Consumers in close proximity.

8. Conclusion

Contents

Following the Ubiquitous Computing vision, the basic IoT assumption is that people are no longer supported by a single monolithic computing system, such as a PC, but rather use all the small embedded systems surrounding them to fulfill their needs (e.g. sensors, actuators, smart devices). Currently, most of these smart devices act like closed "boxes" and barely interconnect or collaborate with each other. Moreover, usually an application domain oriented segmentation of IoT related solutions can be observed (i.e. one box for entertainment, one box for smart home control, one box for e-health services). Justified by the proliferation of IoT solutions, the increasing amount of devices, and the increasing capabilities offered by them, appropriate measures to dynamically manage and provision these IoT resources in an application domain independent manner are required.

On demand allocation and provisioning of resources are key capabilities offered by the Cloud Computing domain. Users can consume resources on demand on a Pay-As-You-Go (PAYG) basis without having to worry about the details of the underlying physical infrastructure.

This thesis contributes to the field of IoT research by describing a framework and architecture for on demand provisioning of physical devices to users, called the Device Cloud. The Device Cloud mitigates the problem of static, application domain dependent bindings between users and devices by applying Cloud Computing paradigms to the IoT domain. Devices are considered as resources that should be provisioned to users based on their requirements. Based on a discussion of related approaches, such as sensor virtualization, and possible application scenarios, a theoretical foundation enabling the provisioning and sharing of physical devices between users was presented. This includes the fundamental principles of sharing physical devices as well as an Entity-, an Interaction-, and a Security model. The Entity Model aims at providing a uniform description of knowledge required to establish an ad-hoc collaboration between the participants of the Device Cloud without having any pre-defined knowledge about a communication partner. The Security Model discusses specific issues that arise when physical devices from different owners are organized in a federated resource pool and provisioned among users that are most likely not known to each other. Based on the Entity- and Security Model,

the Interaction Model defines basic interactions required to provision devices from the pool (e.g. Integration Offer or Integration Request).

A generic, application domain independent Device Cloud architecture, consisting of a backend information system and a Device Cloud Middleware, was derived from the models. By employing modularity, abstraction, and spontaneous interoperability features, the middleware was designed in a technology and protocol agnostic manner, being able to adapt to the requirements of the environment at runtime.

Finally, the applicability of the Device Cloud approach was demonstrated using a specific use case from the E-Health domain.

8.1. Future Work

Similar to the requirement analysis, future work can be generally classified into functional and non-functional enhancements of the Device Cloud infrastructure.

By exploiting the Device Cloud ability to provision physical device on demand, the integration of Mobile Grid concepts into IoT environments can be achieved. So far, the device Cloud treats physical devices as resources of data and just provides basic data processing capabilities. Applications are considered to be hosted by external infrastructures (e.g. PaaS Clouds). Based on the assumption that further development of embedded systems hardware will lead to increasing resources, physical devices could be additionally considered as compute- and storage resources. Based on the capabilities of Aggregation Nodes, the Device Cloud Middleware could be extended with a service execution environment more comprehensive than given by the ALE. Compute- and storage resources could be provisioned to the users, similar to the devices themselves. Applications could be deployed close to the data sources and migrated between instance of the Device Cloud Middleware, which leads to a distributed middleware platform as conceptually shown in Figure 8.1.

Another functional enhancement is related to the orchestration of Aggregation Platform Modules. Except for Transformation Modules, the orchestration of Aggregation Platform Modules currently has to be specified manually. Giving the pre-specified input and output formats, a directed graph can be created from the Platform Module specification. Hence, existing paths between the Category based output format of devices and Output Modules could be identified automatically. A related issue is given by devices, that consume data streams of other devices as an input. In general, such kind of data dispatching can be managed by applications, which may use Output Modules to forward data streams to other devices (similar to the process of controlling actuator devices). However, if both, providing and consuming device, are integrated by the same Device

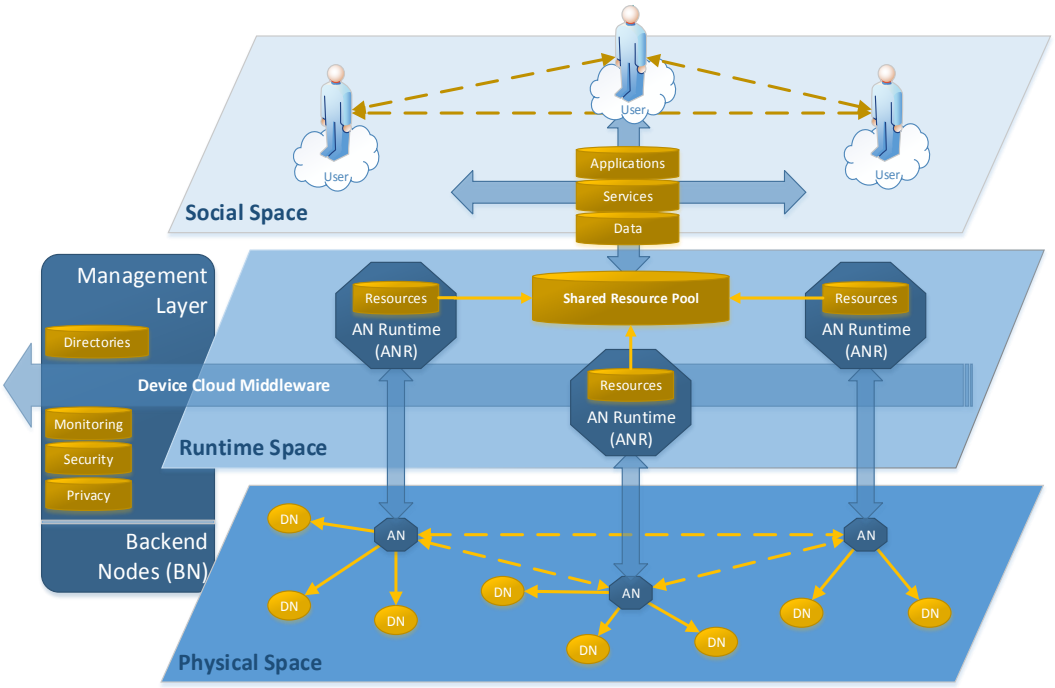


Figure 8.1.: Overall sharing of compute-, storage-, and data-resources based on a distributed Aggregation Node Middleware platform.

Cloud Middleware instance, internal handling would be more efficient. Although the general design of Output Modules allows for bridging between two integrated devices, dedicated Bridging Platform Modules could be introduced to tackle this issue in a more convenient way.

Further functional enhancements may cover a more sophisticated and user-friendly application integration with regard to modifications of the Consumer Profile or tackle decision making processes regarding the device interest. For instance, recommender systems that support Consumers in areas with a large amount of available devices or the integration of Big Data engines to handle the large amount of data generated by the Device Cloud could be examined.

The priority objective for non-functional enhancements are measures supporting application domains with high QoS or real time (RT) requirements. Preliminary performance investigations within the scope of the RehaInteract project have shown that the delay introduced by the ALE preprocessing is suitable for sensors with transmission rates around 25Hz (introduced delay around 10ms given three integrated position sensors). However, safety-critical application domains may require more specific measures to specify RT requirements or achieve reliability. Due to the application domain independent design of the Device Cloud, several devices with different RT requirements are allowed to be integrated by the same Device Cloud Middleware. Thus, the resources of the respective Aggregation Node device can be considered as shared, which means the Platform Modules need to be properly scheduled with respect to the requirements of the linked device and application. Moreover, the utilization of an Aggregation Node could be used as an indicator for the device integration decision policies.

Accordingly, another challenge to be tackled is generalizing the decision policies and respective evaluation metrics. As discussed, existing resource allocation strategies usually do not take the mobility of the resources and the required proximity to the users into account. Providing a formal model will help to assess and to improve the decision policies used by Consumer Operators. Another important aspect related to the decision policies are nomenclatures that allow describing the contextual data used for decision making, especially the *Device Target* property, in a machine processable manner. In case of large Device Cloud deployments with a majority of devices deployed in public areas, the decision policies can be further enhanced by introducing reputation systems that allow dynamically ranking the trustworthiness of a formerly unknown Consumer.

A. List of Acronyms

ALE Aggregation Layer Engine

BAN Body Area Network

CoAP Constrained Application Protocol

CDO Care Delivery Operator

CPS Cyber Physical Systems

DAS Device Access Specification

DEP Data End Point

DIM Domain Information Model

DIP Data Integration Point

DLE Device Layer Engine

DMT Device Management Tree

DmtAS DMT Admin Service Specification

DPWS Device Profile for Web Services

EHR Electronic Health Record

EMR Electronic Medical Record

HDP Bluetooth Health Device Profile

IaaS Infrastructure as a Service

IAM Identity and Access Management

ICT Information and Communications Technology

IETF Internet Engineering Task Force

IHE Integrating the Healthcare Enterprise

IOCS IO Connector Service Specification

IoE	Internet of Everything
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
JNI	Java Native Interface
JVM	Java Virtual Machine
LCIM	Levels of Conceptual Interoperability Model
M2C	Machine-to-Cloud
M2M	Machine-to-Machine
MANET	Mobile Ad-hoc Network
MCN	Machine Communication Network
MDER	Medical Device Encoding Rules
MDM	Mobile Device Management
NFC	Near Field Communication
NIST	National Institute of Standards and Technology
OS	Operating System
PaaS	Platform as a Service
PAYG	Pay-As-You-Go
PC	Personal Computer
PHD	Personal Health Device
PHR	Personal Health Record
QoS	Quality of Service
SaaS	Software as a Service
SLA	Service Level Agreement
SOA	Service Oriented Architecture
UC	Ubiquitous Computing
UPnP	Universal Plug and Play

USB Universal Serial Bus

VSN Virtual Sensor Network

WHO World Health Organization

WPAN Wireless Personal Area Network

WSN Wireless Sensor Network

WSAN Wireless Sensor and Actor Network

B. List of Figures

Bibliography

References

[1] S. Abdelwahab et al. "Enabling Smart Cloud Services Through Remote Sensing: An Internet of Everything Enabler". In: *Internet of Things Journal, IEEE* 1.3 (2014), pp. 276–288. ISSN: 2327-4662. DOI: 10.1109/JIOT.2014.2325071.

[2] Karl Aberer, Manfred Hauswirth, and Ali Salehi. "A Middleware for Fast and Flexible Sensor Network Deployment". In: *Proceedings of the 32Nd International Conference on Very Large Data Bases*. VLDB '06. Seoul, Korea: VLDB Endowment, 2006, pp. 1199–1202. URL: http://dl.acm.org/citation.cfm?id=1182635.1164243.

[3] Bernard Aboba and Jonathan Wood. *RFC 3539 – Authentication, Authorization and Accounting (AAA) Transport Profile*. https://tools.ietf.org/html/rfc3539. [Online; accessed 17-February-2015]. 2003.

[4] Ian F Akyildiz and Ismail H Kasimoglu. "Wireless sensor and actor networks: research challenges". In: *Ad hoc networks* 2.4 (2004), pp. 351–367. ISSN: 1570-8705. DOI: 10.1016/j.adhoc.2004.04.003. URL: http://www.sciencedirect.com/science/article/pii/S1570870504000319.

[5] Y. Al-Hazmi et al. "An automated health monitoring solution for future Internet infrastructure marketplaces". In: *Teletraffic Congress (ITC), 2014 26th International*. 2014, pp. 1–6. DOI: 10.1109/ITC.2014.6932979.

[6] S. Alam, M.M.R. Chowdhury, and J. Noll. "SenaaS: An event-driven sensor virtualization approach for Internet of Things cloud". In: *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*. 2010, pp. 1–6. DOI: 10.1109/NESEA.2010.5678060.

[7] Atif Alamri et al. "A survey on sensor-cloud: architecture, applications, and approaches". In: *International Journal of Distributed Sensor Networks* 2013 (2013). DOI: 10.1155/2013/917923.

[8] Cristina Alcaraz et al. "Wireless sensor networks and the internet of things: Do we need a complete integration?" In: *1st International Workshop on the Security of the Internet of Things (SecIoT'10)*. IEEE, 2010.

[9] EC Amazon. *Amazon elastic compute cloud (Amazon EC2)*. http://aws.amazon.com/de/ec2/. [Online; accessed 26-December-2014]. 2010.

[10] Apache Maven Project. *Apache Maven Project – Welcome to Apache Maven*. http://maven.apache.org/index.html. [Online; accessed 02-March-2015]. 2015.

[11] Michael Armbrust et al. "Above the clouds: A Berkeley view of cloud computing". In: *Communications of the ACM* 53.4 (2010), pp. 50–58.

[12] Kevin Ashton. "That "internet of things" thing". In: *RFiD Journal* 22.7 (2009), pp. 97–114.

[13] CamlonH. Asuncion and MartenJ. van Sinderen. "Pragmatic Interoperability: A Systematic Review of Published Definitions". English. In: *Enterprise Architecture, Integration and Interoperability*. Ed. by Peter Bernus, Guy Doumeingts, and Mark Fox. Vol. 326. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2010, pp. 164–175. ISBN: 978-3-642-15508-6. DOI: 10.1007/978-3-642-15509-3_15. URL: http://dx.doi.org/10.1007/978-3-642-15509-3_15.

[14] Luigi Atzori, Antonio Iera, and Giacomo Morabito. "The Internet of Things: A survey". In: *Computer Networks* 54.15 (2010), pp. 2787 –2805. ISSN: 1389-1286. DOI: http://dx.doi.org/10.1016/j.comnet.2010.05.010. URL: http://www.sciencedirect.com/science/article/pii/S1389128610001568.

[15] Jan Axelson. *USB complete: everything you need to develop custom USB peripherals*. 2005. ISBN: 978-0965081955.

[16] Soma Bandyopadhyay et al. "Role of middleware for internet of things: A study". In: *International Journal of Computer Science & Engineering Survey (IJCSES)* 2.3 (2011), pp. 94–105.

[17] Alessandro Bassi et al. *Enabling Things to Talk*. Springer Berlin Heidelberg, 2013. ISBN: 978-3-642-40402-3. DOI: 10.1007/978-3-642-40403-0.

[18] M. Batty et al. "Smart cities of the future". English. In: *The European Physical Journal Special Topics* 214.1 (2012), pp. 481–518. ISSN: 1951-6355. DOI: 10.1140/epjst/e2012-01703-3. URL: http://dx.doi.org/10.1140/epjst/e2012-01703-3.

[19] Christian Bauer and Gavin King. *Java Persistance with Hibernate*. 2006. ISBN: 978-1932394887.

[20] Sean Bechhofer. "OWL: Web Ontology Language". English. In: *Encyclopedia of Database Systems*. Ed. by LING LIU and M.TAMER ÖZSU. Springer US, 2009, pp. 2008–2009. ISBN: 978-0-387-35544-3. DOI: 10.1007/978-0-387-39940-9_1073. URL: http://dx.doi.org/10.1007/978-0-387-39940-9_1073.

[21] Fran Berman, Geoffrey Fox, and Anthony JG Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Vol. 2. John Wiley & Sons, 2003. ISBN: 978-0-470-85319-1.

[22] Bluetooth SIG. *Enabling wireless communication between devices*. `https://www.bluetooth.org/en-us/training-resources/technology`. [Online; accessed 30-December-2014]. 2014.

[23] I. Bojanova, G. Hurlburt, and J. Voas. "Imagineering an Internet of Anything". In: *Computer* 47.6 (2014), pp. 72–77. ISSN: 0018-9162. DOI: `10.1109/MC.2014.150`.

[24] David Boswarthick, Omar Elloumi, and Olivier Hersent. *M2M communications: a systems approach*. John Wiley & Sons, 2012. ISBN: 978-1-119-99475-6.

[25] Broadband Forum. *TR-069 CPE WAN Management Protocol*. Tech. rep. [Online; accessed 8-January-2015]. 2014.

[26] Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e.V. *Das Internet schafft eine Kultur des Teilens*. `http://www.bitkom.org/de/presse/78284_75237.aspx`. [Online; accessed 13-December-2014]. 2013.

[27] Michael Burrows, Martin Abadi, and Roger Needham. "A Logic of Authentication". In: *ACM Trans. Comput. Syst.* 8.1 (1990), pp. 18–36. ISSN: 0734-2071. DOI: `10.1145/77648.77649`. URL: `http://doi.acm.org/10.1145/77648.77649`.

[28] Betsy Burton and David A. Willis. "Gartner's Hype Cycle Special Report for 2014". In: (2014).

[29] Peter Buxmann et al. "The Standardization Problem – An Economic Analysis of Standards in Information Systems". In: *Proceedings of the 1st IEEE Conference on Standardization and Innovation in Information Technology SIIT 99*. 1999, pp. 157–162.

[30] Rajkumar Buyya and Manzur Murshed. "GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing". In: *Concurrency and Computation: Practice and Experience* 14.13-15 (2002), pp. 1175–1220. ISSN: 1532-0634. DOI: `10.1002/cpe.710`. URL: `http://dx.doi.org/10.1002/cpe.710`.

[31] Rodrigo N. Calheiros et al. "CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms". In: *Software: Practice and Experience* 41.1 (2011), pp. 23–50. ISSN: 1097-024X. DOI: `10.1002/spe.995`. URL: `http://dx.doi.org/10.1002/spe.995`.

[32] M. Castro, A.J. Jara, and A.F. Skarmeta. "An Analysis of M2M Platforms: Challenges and Opportunities for the Internet of Things". In: *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. 2012, pp. 757–762. DOI: `10.1109/IMIS.2012.184`.

[33] Erdal Cayirci and Chunming Rong. *Security in wireless ad hoc and sensor networks*. John Wiley & Sons Ltd, 2009. ISBN: 978-0-470-02748-6.

[34] A. Celesti et al. "How to Enhance Cloud Architectures to Enable Cross-Federation". In: *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*. 2010, pp. 337–345. DOI: 10.1109/CLOUD.2010.46.

[35] Sheng-Tzong Cheng, Chi-Hsuan Wang, and Gwo-Jiun Horng. "OSGi-based smart home architecture for heterogeneous network". In: *Expert Systems with Applications* 39.16 (2012), pp. 12418 –12429. ISSN: 0957-4174. DOI: http://dx.doi.org/10.1016/j.eswa.2012.04.077. URL: http://www.sciencedirect.com/science/article/pii/S0957417412006744.

[36] Gao Chong, Ling Zhihao, and Yuan Yifeng. "The research and implement of smart home system based on Internet of Things". In: (2011), pp. 2944–2947. DOI: 10.1109/ICECC.2011.6066672.

[37] Continua Health Alliance. *About Continua*. http://www.continuaalliance.org/about-continua. [Online; accessed 17-February-2015]. 2015.

[38] Silviu S Craciunas et al. "Information-acquisition-as-a-service for cyber-physical cloud computing". In: *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*. USENIX Association. 2010, pp. 14–14.

[39] M. Darianian and M.P. Michael. "Smart Home Mobile RFID-Based Internet-of-Things Systems and Services". In: *Advanced Computer Theory and Engineering, 2008. ICACTE '08. International Conference on*. 2008, pp. 116–120. DOI: 10.1109/ICACTE.2008.180.

[40] Premkumar Devanbu et al. "Authentic Third-Party Data Publication". English. In: *Data and Application Security*. Ed. by Bhavani Thuraisingham et al. Vol. 73. IFIP International Federation for Information Processing. Springer US, 2001, pp. 101–112. ISBN: 978-0-7923-7514-2. DOI: 10.1007/0-306-47008-X_9. URL: http://dx.doi.org/10.1007/0-306-47008-X_9.

[41] E. W. Dijkstra. "Solution of a Problem in Concurrent Programming Control". In: *Commun. ACM* 8.9 (Sept. 1965), pp. 569–. ISSN: 0001-0782. DOI: 10.1145/365559.365617. URL: http://doi.acm.org/10.1145/365559.365617.

[42] Hoang T. Dinh et al. "A survey of mobile cloud computing: architecture, applications, and approaches". In: *Wireless Communications and Mobile Computing* 13.18 (2013), pp. 1587–1611. ISSN: 1530-8677. DOI: 10.1002/wcm.1203. URL: http://dx.doi.org/10.1002/wcm.1203.

[43] A. Dunkels, B. Gronvall, and T. Voigt. "Contiki - a lightweight and flexible operating system for tiny networked sensors". In: *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*. 2004, pp. 455–462. DOI: 10.1109/LCN.2004.38.

[44] S. Duquennoy, G. Grimaud, and J.-J. Vandewalle. "Consistency and scalability in event notification for embedded Web applications". In: *Web Systems Evolution (WSE), 2009 11th IEEE International Symposium on*. 2009, pp. 89–98. DOI: 10.1109/WSE.2009.5631249.

[45] Dynastream Innovations Inc. *What is ANT+*. http://www.thisisant.com/consumer/ant-101/what-is-ant/. [Online; accessed 30-December-2014]. 2014.

[46] Claudia Eckert. *IT-Sicherheit: Konzepte-Verfahren-Protokolle*. Oldenbourg Verlag, 2008. ISBN: 978-3-486-58270-3.

[47] Markus Eisenhauer, Peter Rosengren, and Pablo Antolin. "HYDRA: A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems". English. In: *The Internet of Things*. Ed. by Daniel Giusto et al. Springer New York, 2010, pp. 367–373. ISBN: 978-1-4419-1673-0. DOI: 10.1007/978-1-4419-1674-7_36. URL: http://dx.doi.org/10.1007/978-1-4419-1674-7_36.

[48] EnOcean GmbH. *EnOcean Wireless Standard*. https://www.enocean.com/en/enocean-wireless-standard/. [Online; accessed 6-January-2015]. 2015.

[49] European Commission. *eHealth Action Plan 2012-2020 – Innovative healthcare for the 21st century*. https://ec.europa.eu/digital-agenda/en/news/ehealth-action-plan-2012-2020-innovative-healthcare-21st-century. [Online; accessed 24-Feburary-2015]. 2012.

[50] European Commission. *Living Healthy, Ageing Well*. http://ec.europa.eu/digital-agenda/en/life-and-work/living-healthy-ageing-well. [Online; accessed 17-February-2015]. 2013.

[51] Kosmatos Evangelos A, Tselikas Nikolaos D, and Boucouvalas Anthony C. "Integrating RFIDs and Smart Objects into a Unified Internet of Things Architecture". In: *Advances in Internet of Things* 2011 (2011). DOI: 10.4236/ait.2011.11002.

[52] Dave Evans. *The Internet of Everything - How More Relevant and Valuable Connections Will Change the World*. https://www.cisco.com/web/about/ac79/docs/innov/IoE.pdf. [Online; accessed 25-December-2014]. 2012.

[53] Gunther Eysenbach. "What is e-health?" In: *Journal of medical Internet research* 3.2 (2001). [Online; accessed 01-April-2015].

[54] Ian Fette and Alexey Melnikov. *RFC 6455–The WebSocket Protocol*. Tech. rep. [Online; accessed 9-Feburary-2015]. The Internet Engineering Task Force (IETF), 2011.

[55] FI-PPP - Future Internet Public-Private Partnership. *Internet-Enabled Innovation in Europe*. http://www.fi-ppp.eu/about/. [Online; accessed 25-December-2014]. 2013.

[56] Roy T. Fielding and Richard N. Taylor. "Principled Design of the Modern Web Architecture". In: *ACM Trans. Internet Technol.* 2.2 (May 2002), pp. 115–150. ISSN: 1533-5399. DOI: 10.1145/514183.514185. URL: http://doi.acm.org/10.1145/514183.514185.

[57] I. Foster et al. "Cloud Computing and Grid Computing 360-Degree Compared". In: *Grid Computing Environments Workshop, 2008. GCE '08*. 2008, pp. 1–10. DOI: 10.1109/GCE.2008.4738445.

[58] Ian Foster and Carl Kesselman. "What is the Grid? A Three Point Checklist". In: (2002).

[59] Foundation for Intelligent Physical Agents (FIPA). *FIPA Device Ontology Specification.* http://www.fipa.org/specs/fipa00091/SI00091E.html. [Online; accessed 17-February-2015]. 2002.

[60] Dave Garets and Mike Davis. "Electronic medical records vs. electronic health records: yes, there is a difference". In: *Policy white paper. HIMSS Analytics* (2006).

[61] Anastasius Gavras et al. "Future Internet Research and Experimentation: The FIRE Initiative". In: *SIGCOMM Comput. Commun. Rev.* 37.3 (July 2007), pp. 89–92. ISSN: 0146-4833. DOI: 10.1145/1273445.1273460. URL: http://doi.acm.org/10.1145/1273445.1273460.

[62] Peter Gilbert et al. "YouProve: Authenticity and Fidelity in Mobile Sensing". In: *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems.* SenSys '11. Seattle, Washington: ACM, 2011, pp. 176–189. ISBN: 978-1-4503-0718-5. DOI: 10.1145/2070942.2070961. URL: http://doi.acm.org/10.1145/2070942.2070961.

[63] Carles Gomez and Josep Paradells. "Wireless home automation networks: A survey of architectures and technologies". In: *IEEE Communications Magazine* 48.6 (2010), pp. 92–101.

[64] Jayavardhana Gubbi et al. "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future Generation Computer Systems* 29.7 (2013), pp. 1645–1660. DOI: 10.1016/j.future.2013.01.010.

[65] D. Guinard, V. Trifa, and E. Wilde. "A resource oriented architecture for the Web of Things". In: *Internet of Things (IOT), 2010.* 2010, pp. 1–8. DOI: 10.1109/IOT.2010.5678452.

[66] D. Hardt. *RFC 6749–The OAuth 2.0 Authorization Framework.* Tech. rep. [Online; accessed 9-January-2015]. The Internet Engineering Task Force (IETF), 2012.

[67] Mohammad Mehedi Hassan, Biao Song, and Eui-Nam Huh. "A Framework of Sensor-cloud Integration Opportunities and Challenges". In: *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*. ICUIMC '09. Suwon, Korea: ACM, 2009, pp. 618–626. ISBN: 978-1-60558-405-8. DOI: 10.1145/1516241.1516350. URL: http://doi.acm.org/10.1145/1516241.1516350.

[68] George T. Heineman and William T. Councill, eds. *Component-based Software Engineering: Putting the Pieces Together*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 0-201-70485-4.

[69] John H Howard. *An overview of the andrew file system*. Carnegie Mellon University, Information Technology Center, 1988.

[70] Guoqiang Hu, Wee Peng Tay, and Yonggang Wen. "Cloud robotics: architecture, challenges and applications". In: *Network, IEEE* 26.3 (2012), pp. 21–28. ISSN: 0890-8044. DOI: 10.1109/MNET.2012.6201212.

[71] Dijiang Huang et al. "Mobile cloud computing". In: *IEEE COMSOC Multimedia Communications Technical Committee (MMTC) E-Letter* 6.10 (2011), pp. 27–31.

[72] "IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries". In: *IEEE Std 610* (1991), pp. 1–217. DOI: 10.1109/IEEESTD.1991.106963.

[73] Integrating the Healthcare Enterprise (IHE). *PCD Profile Rosetta Terminology Mapping*. http://wiki.ihe.net/index.php?title=PCD_Profile_Rosetta_Terminology_Mapping. [Online; accessed 17-February-2015]. 2011.

[74] Md Motaharul Islam et al. "A survey on virtualization of wireless sensor networks". In: *Sensors* 12.2 (2012), pp. 2175–2207.

[75] "ISO/IEC/IEEE Health informatics–Personal health device communication–Part 20601: Application profile–Optimized exchange protocol". In: *ISO/IEEE 11073-20601:2010(E)* (2010), pp. 1–208. DOI: 10.1109/IEEESTD.2010.5703195.

[76] "ISO/IEC/IEEE Information technology – Smart transducer interface for sensors and actuators – Common functions, communication protocols, and Transducer Electronic Data Sheet (TEDS) formats". In: *ISO/IEC/IEEE 21450:2010(E)* (2010), pp. 1–350. DOI: 10.1109/IEEESTD.2010.5668466.

[77] "ISO/IEC/IEEE Systems and software engineering – Architecture description". In: *ISO/IEC/IEEE 42010:2011(E) (Revision of ISO/IEC 42010:2007 and IEEE Std 1471-2000)* (2011), pp. 1–46. DOI: 10.1109/IEEESTD.2011.6129467.

[78] "ISO/IEEE Health Informatics - Point-Of-Care Medical Device Communication - Part 10101: Nomenclature". In: *ISO/IEEE 11073-10101:2004(E)* (2004), pp. 1–492. DOI: 10.1109/IEEESTD.2004.95741.

[79] "ISO/IEEE Health Informatics - Point-Of-Care Medical Device Communication - Part 10201: Domain Information Model". In: *ISO/IEEE 11073-10201:2004(E)* (2004), pp. 1–169. DOI: 10.1109/IEEESTD.2004.95742.

[80] A.J. Jara, M.A. Zamora, and A.F.G. Skarmeta. "An Architecture Based on Internet of Things to Support Mobility and Security in Medical Environments". In: *Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*. 2010, pp. 1–5. DOI: 10.1109/CCNC.2010.5421661.

[81] A.P. Jayasumana, Qi Han, and T.H. Illangasekare. "Virtual Sensor Networks - A Resource Efficient Approach for Concurrent Applications". In: *Information Technology, 2007. ITNG '07. Fourth International Conference on*. 2007, pp. 111–115. DOI: 10.1109/ITNG.2007.206.

[82] Michael Jeronimo and Jack Weast. *UPnP design by example: a software developer's guide to universal plug and play*. Intel Press, 2003. ISBN: 978-0971786110.

[83] Jiong Jin et al. "An Information Framework for Creating a Smart City Through Internet of Things". In: *Internet of Things Journal, IEEE* 1.2 (2014), pp. 112–121. ISSN: 2327-4662. DOI: 10.1109/JIOT.2013.2296516.

[84] Stamatis Karnouskos. "The cooperative internet of things enabled smart grid". In: *Proceedings of the 14th IEEE international symposium on consumer electronics (ISCE2010), June*. 2010, pp. 07–10.

[85] Artem Katasonov et al. "Smart Semantic Middleware for the Internet of Things." In: *ICINCO-ICSO* 8 (2008), pp. 169–178.

[86] T. Kindberg and A. Fox. "System software for ubiquitous computing". In: *Pervasive Computing, IEEE* 1.1 (2002), pp. 70–81. ISSN: 1536-1268. DOI: 10.1109/MPRV.2002.993146.

[87] C. Kirsch et al. "Cyber-physical cloud computing: The binding and migration problem". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*. 2012, pp. 1425–1428. DOI: 10.1109/DATE.2012.6176587.

[88] Kristian Ellebæk Kjær. "A survey of context-aware middleware". In: *Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering*. ACTA Press. 2007, pp. 148–155.

[89] Graham Klyne and Jeremy J Carroll. *Resource description framework (RDF): Concepts and abstract syntax*. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. [Online; accessed 17-February-2015]. 2004.

[90] JeongGil Ko et al. "Wireless Sensor Networks for Healthcare". In: *Proceedings of the IEEE* 98.11 (2010), pp. 1947–1960. ISSN: 0018-9219. DOI: 10.1109/JPROC.2010.2065210.

[91] G. Kortuem et al. “Smart objects as building blocks for the Internet of things”. In: *Internet Computing, IEEE* 14.1 (2010), pp. 44–51. ISSN: 1089-7801. DOI: 10.1109/MIC.2009.143.

[92] Peter Kostelnik, Martin Sarnovsk, and Karol Furdik. “The semantic middleware for networked embedded systems applied in the Internet of Things and Services domain”. In: *Scalable Computing: Practice and Experience* 12.3 (2011). ISSN: 1895-1767.

[93] Matthias Kovatsch, Simon Mayer, and Benedikt Ostermaier. “Moving Application Logic from the Firmware to the Cloud: Towards the Thin Server Architecture for the Internet of Things”. In: *Proceedings of the 2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*. IMIS ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 751–756. ISBN: 978-0-7695-4684-1. DOI: 10.1109/IMIS.2012.104. URL: http://dx.doi.org/10.1109/IMIS.2012.104.

[94] Nandakishore Kushalnagar, Gabriel Montenegro, C Schumacher, et al. “IPv6 over low-power wireless personal area networks (6LoWPANs): overview, assumptions, problem statement, and goals”. In: *RFC4919, August* 10 (2007).

[95] Teemu Laukkarinen, Jukka Suhonen, and Marko Hännikäinen. “A survey of wireless sensor network abstraction for application development”. In: *International Journal of Distributed Sensor Networks* 2012 (2012).

[96] Iva Lazarova. *Evolving trends in cyber-physical systems*. http://ec.europa.eu/digital-agenda/futurium/en/content/evolving-trends-cyber-physical-systems. [Online; accessed 15-August-2014]. 2013.

[97] E.A. Lee. “Cyber Physical Systems: Design Challenges”. In: *Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on*. 2008, pp. 363–369. DOI: 10.1109/ISORC.2008.25.

[98] C. Lerche et al. “Implementing powerful Web Services for highly resource-constrained devices”. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2011 IEEE International Conference on*. 2011, pp. 332–335. DOI: 10.1109/PERCOMW.2011.5766899.

[99] Minbo Li and Hua Li. “Research on RFID integration middleware for enterprise information system”. In: *Journal of Software* 6.2 (2011), pp. 167–174.

[100] Xu Li et al. “Smart community: an internet of things application”. In: *Communications Magazine, IEEE* 49.11 (2011), pp. 68–75. ISSN: 0163-6804. DOI: 10.1109/MCOM.2011.6069711.

[101] Libelium Comunicaciones Distribuidas S.L. *Top 50 Internet of Things Applications - 50 Sensor Applications for a Smarter World*. http://www.libelium.com/top_50_iot_sensor_applications_ranking/. [Online; accessed 21-November-2014]. 2014.

[102] Antonios Litke, Dimitrios Skoutas, and Theodora Varvarigou. "Mobile grid computing: Changes and challenges of resource management in a mobile grid environment". In: *5th International Conference on Practical Aspects of Knowledge Management (PAKM 2004)*. 2004.

[103] L. Liu, R. Moulic, and D. Shea. "Cloud Service Portal for Mobile Device Management". In: *e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on*. 2010, pp. 474–478. DOI: 10.1109/ICEBE.2010.102.

[104] Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. "Securing the e-Health Cloud". In: *Proceedings of the 1st ACM International Health Informatics Symposium*. IHI '10. Arlington, Virginia, USA: ACM, 2010, pp. 220–229. ISBN: 978-1-4503-0030-8. DOI: 10.1145/1882992.1883024. URL: http://doi.acm.org/10.1145/1882992.1883024.

[105] A. Lounis et al. "Secure and Scalable Cloud-Based Architecture for e-Health Wireless Sensor Networks". In: *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*. 2012, pp. 1–7. DOI: 10.1109/ICCCN.2012.6289252.

[106] M2M Alliance e.V. *Machine-to-Machine (M2M) - Whitepaper*. http://www.m2m-alliance.de/uploads/media/Whitepaper.pdf. [Online; accessed 26-December-2014]. 2007.

[107] R. Marti, J. Delgado, and X. Perramon. "Security specification and implementation for mobile e-health services". In: *e-Technology, e-Commerce and e-Service, 2004. EEE '04. 2004 IEEE International Conference on*. 2004, pp. 241–248. DOI: 10.1109/EEE.2004.1287316.

[108] CarloMaria Medaglia and Alexandru Serbanati. "An Overview of Privacy and Security Issues in the Internet of Things". English. In: *The Internet of Things*. Ed. by Daniel Giusto et al. Springer New York, 2010, pp. 389–395. ISBN: 978-1-4419-1673-0. DOI: 10.1007/978-1-4419-1674-7_38. URL: http://dx.doi.org/10.1007/978-1-4419-1674-7_38.

[109] Peter Mell and Tim Grance. "The NIST definition of cloud computing". In: (2011).

[110] Hermann Merz, Thomas Hansemann, and Christof Hübner. *Building Automation: Communication Systems with EIB/KNX, LON and BACnet*. Springer Science & Business Media, 2009. ISBN: 978-3540888284.

[111] Microsoft Corporation. *Device nodes and device stacks*. http://msdn.microsoft.com/en-us/library/windows/hardware/ff554721(v=vs.85).aspx. [Online; accessed 7-January-2015]. 2014.

[112] Nationaler IT Gipfel - Projektgruppe M2M AG2. *M2M - Querschnittstechnologie für die vernetzte Gesellschaft*. http://www.m2m-alliance.com/fileadmin/user_upload/pdf/it-gipfel-2014-ag-2-strategiepapier-m2m_property_pdf_bereich_itgipfel_sprache_de_rwb_true.pdf. [Online; accessed 26-December-2014]. 2014.

[113] C. Neuman et al. *RFC 4120–The Kerberos network authentication service (V5)*. Tech. rep. [Online; accessed 9-January-2015]. The Internet Engineering Task Force (IETF), 2005.

[114] OASIS. *Device Profile for Web Services (DPWS)*. http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01. [Online; accessed 31-December-2014]. 2014.

[115] Jan Ohlenburg, Wolfgang Broll, and Irma Lindt. "DEVAL – A Device Abstraction Layer for VR/AR". English. In: *Universal Acess in Human Computer Interaction. Coping with Diversity*. Ed. by Constantine Stephanidis. Vol. 4554. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, pp. 497–506. ISBN: 978-3-540-73278-5. DOI: 10.1007/978-3-540-73279-2_56. URL: http://dx.doi.org/10.1007/978-3-540-73279-2_56.

[116] Open Mobile Alliance. *OMA Device Management V2.0*. http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-device-management-v2-0. [Online; accessed 6-January-2015]. 2015.

[117] World Health Organization. *Key components of a well functioning health system*. 2010.

[118] World Health Organization. *WHO Global Health Expenditure Atlas*. Tech. rep. 2012. URL: http://www.who.int/nha/en/.

[119] OSGi Alliance. *OSGi Compendium Release 5*. Tech. rep. [Online; accessed 9-January-2015]. 2013.

[120] OSGi Alliance. *OSGi Core Release 5*. Tech. rep. [Online; accessed 9-January-2015]. 2012.

[121] J. Pan, S. Paul, and R. Jain. "A survey of the research on future internet architectures". In: *Communications Magazine, IEEE* 49.7 (2011), pp. 26–36. ISSN: 0163-6804. DOI: 10.1109/MCOM.2011.5936152.

[122] D.F. Parkhill. *The Challenge of the Computer Utility*. The Challenge of the Computer Utility S. 246. Addison-Wesley Publishing Company, 1966. ISBN: 978-0201057201.

[123] Adrian Perrig, John Stankovic, and David Wagner. "Security in Wireless Sensor Networks". In: *Commun. ACM* 47.6 (June 2004), pp. 53–57. ISSN: 0001-0782. DOI: 10.1145/990680.990707. URL: http://doi.acm.org/10.1145/990680.990707.

[124] Thinagaran Perumal et al. "Interoperability Among Heterogeneous Systems in Smart Home Environment". English. In: *Web-Based Information Technologies and Distributed Systems*. Vol. 2. Atlantis Ambient and Pervasive Intelligence. Atlantis Press, 2010, pp. 141–157. DOI: 10.2991/978-94-91216-32-9_7. URL: http://dx.doi.org/10.2991/978-94-91216-32-9_7.

[125] Benjamin C Pierce. *Types and programming languages*. MIT press, 2002. ISBN: 978-0262162098.

[126] Stefan Poslad. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. John Wiley & Sons, 2009. ISBN: 978-0-470-03560-3. DOI: 10.1002/9780470779446.fmatter.

[127] G. J. Pottie and W. J. Kaiser. "Wireless Integrated Network Sensors". In: *Commun. ACM* 43.5 (May 2000), pp. 51–58. ISSN: 0001-0782. DOI: 10.1145/332833.332838. URL: http://doi.acm.org/10.1145/332833.332838.

[128] N. R. Prasad et al. "Open Source Middleware for Networked Embedded Systems towards Future Internet of Things". In: *Vision and Challenges for Realising the Internet of Things, CERP-IoT cluster*. Ed. by H Sundmaeker et al. EUR-OP, 2010, pp. 153–164.

[129] H. Rachidi and A. Karmouch. "A framework for self-configuring devices using TR-069". In: *Multimedia Computing and Systems (ICMCS), 2011 International Conference on*. 2011, pp. 1–6. DOI: 10.1109/ICMCS.2011.5945613.

[130] R. Rajkumar et al. "Cyber-physical systems: The next computing revolution". In: *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. 2010, pp. 731–736.

[131] Abdelmounaam Rezgui and Mohamed Eltoweissy. "Service-oriented Sensor-actuator Networks: Promises, Challenges, and the Road Ahead". In: *Comput. Commun.* 30.13 (Sept. 2007), pp. 2627–2648. ISSN: 0140-3664. DOI: 10.1016/j.comcom.2007.05.036. URL: http://dx.doi.org/10.1016/j.comcom.2007.05.036.

[132] C. Rigney et al. *RFC 2865–Remote Authentication Dial In User Service (RADIUS)*. Tech. rep. [Online; accessed 17-February-2015]. The Internet Engineering Task Force (IETF), 2000.

[133] R. Roman, P. Najera, and J. Lopez. "Securing the Internet of Things". In: *Computer* 44.9 (2011), pp. 51–58. ISSN: 0018-9162. DOI: 10.1109/MC.2011.291.

[134] Danielle Sacks. "THE SHARING ECONOMY." In: *Fast Company* 155 (2011). [Online; accessed 26-November-2014], pp. 88 –131. ISSN: 10859241. URL: http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=60036724&site=ehost-live.

[135] Asmiza A Sani, Fiona Polack, and Richard Paige. "Generating Formal Model Transformation Specification Using a Template-based Approach". In: *Scope of the Symposium*. 2010, p. 3.

[136] Hans Schaffers et al. "Smart Cities and the Future Internet: Towards Cooperation Frameworks for Open Innovation". English. In: *The Future Internet*. Ed. by John Domingue et al. Vol. 6656. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2011, pp. 431–446. ISBN: 978-3-642-20897-3. DOI: 10.1007/978-3-642-20898-0_31. URL: http://dx.doi.org/10.1007/978-3-642-20898-0_31.

[137] R.R. Schaller. "Moore's law: past, present and future". In: *Spectrum, IEEE* 34.6 (1997), pp. 52–59. ISSN: 0018-9235.

[138] F.A. Schreiber et al. "PerLa: A Language and Middleware Architecture for Data Management and Integration in Pervasive Information Systems". In: *Software Engineering, IEEE Transactions on* 38.2 (2012), pp. 478–496. ISSN: 0098-5589. DOI: 10.1109/TSE.2011.25.

[139] Z Shelby et al. "Constrained Application Protocol (CoAP), draft-ietf-core-coap-13". In: *Orlando: The Internet Engineering Task Force–IETF, Dec* (2012).

[140] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*. Vol. 43. John Wiley & Sons, 2011.

[141] N. Skimura et al. *OpenID Connect Core 1.0 incorporating errata set 1*. Tech. rep. [Online; accessed 9-January-2015]. 2014.

[142] Tomás Sánchez López et al. "Adding sense to the Internet of Things". English. In: *Personal and Ubiquitous Computing* 16.3 (2012), pp. 291–308. ISSN: 1617-4909. DOI: 10.1007/s00779-011-0399-8. URL: http://dx.doi.org/10.1007/s00779-011-0399-8.

[143] K. Sohrabi et al. "Protocols for self-organization of a wireless sensor network". In: *Personal Communications, IEEE* 7.5 (2000), pp. 16–27. ISSN: 1070-9916. DOI: 10.1109/98.878532.

[144] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Addison-Wesley Longman Publishing Co., Inc., 1998. ISBN: 978-0201485349.

[145] Alexander Stanik, Fridtjof Sander, and Odej Kao. "Autonomous Agreement-Mediation based on WS-Agreement for improving Cloud SLAs". In: *Cloud Computing Technology and Science (CloudCom), Proceedings of the 2014 IEEE 6th International Conference on*. Vol. 1. IEEE Computer Society, 2014, pp. 583–590. ISBN: 978-1-4799-4093-6. DOI: 10.1109/CloudCom.2014.25.

[146] M. Starsinic. "System architecture challenges in the home M2M network". In: *Applications and Technology Conference (LISAT), 2010 Long Island Systems*. 2010, pp. 1–7. DOI: 10.1109/LISAT.2010.5478336.

[147] H Sundmaeker et al. *Vision and Challenges for Realising the Internet of Things, CERP-IoT cluster*. 2010.

[148] Andrew Tanenbaum and Maarten Van Steen. *Distributed systems*. Pearson Prentice Hall, 2006. ISBN: 978-0136135531.

[149] Marvin M Theimer et al. *Method for granting a user request having locational and contextual attributes consistent with user policies for devices having locational attributes consistent with the user request*. US Patent 5,555,376. 1996.

[150] A. Tolk, D Saikou, and T Charles. "Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composability for system-of-systems engineering". In: *Journal of Systemics, Cybernetics and Informatics* (2007).

[151] I. Toma, E. Simperl, and G. Hench. "A joint roadmap for Semantic technologies and the Internet of Things". In: *Proceedings of the Third STI Roadmapping Workshop, Crete, Greece*. 2009.

[152] Dieter Uckelmann, Mark Harrison, and Florian Michahelles. "An Architectural Approach Towards the Future Internet of Things". English. In: *Architecting the Internet of Things*. Ed. by Dieter Uckelmann, Mark Harrison, and Florian Michahelles. Springer Berlin Heidelberg, 2011, pp. 1–24. ISBN: 978-3-642-19156-5. DOI: 10.1007/978-3-642-19157-2_1. URL: http://dx.doi.org/10.1007/978-3-642-19157-2_1.

[153] Thomas Usländer et al. "The Future Internet Enablement of the Environment Information Space". English. In: *Environmental Software Systems. Fostering Information Sharing*. Ed. by Jiří Hřebíček et al. Vol. 413. IFIP Advances in Information and Communication Technology. Springer Berlin Heidelberg, 2013, pp. 109–120. ISBN: 978-3-642-41150-2. DOI: 10.1007/978-3-642-41151-9_11. URL: http://dx.doi.org/10.1007/978-3-642-41151-9_11.

[154] Upkar Varshney. "Pervasive Healthcare and Wireless Health Monitoring". In: *Mob. Netw. Appl.* 12.2-3 (2007), pp. 113–127. ISSN: 1383-469X. DOI: 10.1007/s11036-007-0017-1. URL: http://dx.doi.org/10.1007/s11036-007-0017-1.

[155] R.T. Vaughan, B.P. Gerkey, and A Howard. "On device abstractions for portable, reusable robot code". In: *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 3. 2003, 2421–2427 vol.3. DOI: 10.1109/IROS.2003.1249233.

[156] Ovidiu Vermesan et al. "Internet of Things Strategic Research Roadmap". In: *Internet of Things-Global Technological and Societal Trends* (2011), pp. 9–52.

[157] Pepijn RS Visser et al. "An analysis of ontology mismatches; heterogeneity versus interoperability". In: *AAAI 1997 Spring Symposium on Ontological Engineering, Stanford CA., USA*. 1997, pp. 164–72.

[158] Mark Wahl, Tim Howes, and Steve Kille. "Lightweight directory access protocol (v3)". In: (1997). [Online; accessed 18-December-2014].

[159] John Paul Walters et al. "Wireless sensor network security: A survey". In: *Security in distributed, grid, mobile, and pervasive computing* 1 (2007), p. 367.

[160] Wenguang Wang, Andreas Tolk, and Weiping Wang. "The Levels of Conceptual Interoperability Model: Applying Systems Engineering Principles to M&S". In: *Proceedings of the 2009 Spring Simulation Multiconference*. SpringSim '09. San Diego, California: Society for Computer Simulation International, 2009, 168:1–168:9. URL: http://dl.acm.org/citation.cfm?id=1639809.1655398.

[161] Yan-Wei Wang, Hui-Li Yu, and Ya Li. "Notice of Retraction Internet of things technology applied in medical information". In: (2011), pp. 430–433. DOI: 10.1109/CECNET.2011.5768647.

[162] Rolf H Weber. "Internet of Things–New security and privacy challenges". In: *Computer Law & Security Review* 26.1 (2010), pp. 23–30. DOI: 10.1016/j.clsr.2009.11.008.

[163] Mark Weiser. "The Computer for the 21st Century". In: *SIGMOBILE Mob. Comput. Commun. Rev.* 3.3 (July 1999), pp. 3–11. ISSN: 1559-1662. DOI: 10.1145/329124.329126. URL: http://doi.acm.org/10.1145/329124.329126.

[164] Falk v Westarp et al. "Information technology standards and standardization: A global perspective". In: (2000). Ed. by Kai Jakobs, pp. 168–185. DOI: 10.4018/978-1-878289-70-4.ch011.

[165] Geng Wu et al. "M2M: From mobile to embedded internet". In: *Communications Magazine, IEEE* 49.4 (2011), pp. 36–43. ISSN: 0163-6804. DOI: 10.1109/MCOM.2011.5741144.

[166] Jianchu Yao and Steve Warren. "Applying the ISO/IEEE 11073 Standards to Wearable Home Health Monitoring Systems". English. In: *Journal of Clinical Monitoring and Computing* 19.6 (2005), pp. 427–436. ISSN: 1387-1307. DOI: 10.1007/s10877-005-2033-7. URL: http://dx.doi.org/10.1007/s10877-005-2033-7.

[167] Chen Yuqiang, Guo Jianlan, and Hu Xuanzi. "The Research of Internet of Things' Supporting Technologies Which Face the Logistics Industry". In: (2010), pp. 659–663. DOI: 10.1109/CIS.2010.148.

[168] M. Yuriyama and T. Kushida. "Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing". In: *Network-Based Information Systems (NBiS), 2010 13th International Conference on*. 2010, pp. 1–8. DOI: 10.1109/NBiS.2010.32.

[169] A. Zanella et al. "Internet of Things for Smart Cities". In: *Internet of Things Journal, IEEE* 1.1 (2014), pp. 22–32. ISSN: 2327-4662. DOI: 10.1109/JIOT.2014.2306328.

[170] Arkady Zaslavsky, Charith Perera, and Dimitrios Georgakopoulos. "Sensing as a service and big data". In: *arXiv preprint arXiv:1301.0159* (2013).

[171] ZigBee Alliance. *What is ZigBee?* http://zigbee.org/what-is-zigbee/. [Online; accessed 6-January-2015]. 2014.

[172] H. Zimmermann. "OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection". In: *Communications, IEEE Transactions on* 28.4 (1980), pp. 425–432. ISSN: 0090-6778. DOI: 10.1109/TCOM.1980.1094702.

[173] M. Zorzi et al. "From today's INTRAnet of things to a future INTERnet of things: a wireless- and mobility-related view". In: *Wireless Communications, IEEE* 17.6 (2010), pp. 44–51. ISSN: 1536-1284. DOI: 10.1109/MWC.2010.5675777.