Institute of Architecture of Application Systems

Internet of Things Patterns for Communication and Management

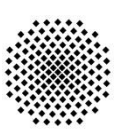
Lukas Reinfurt[1,2], Uwe Breitenbücher[1], Michael Falkenthal[1], Frank Leymann[1], Andreas Riegg[2]

[1]Institute of Architecture of Application Systems, University of Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

[2]Daimler AG, Stuttgart, Germany
{firstname.lastname}@daimler.com

BibTeX:

```
@article{reinfurt2019internet,
  author    = {Reinfurt, Lukas and Breitenb{\"u}cher, Uwe and Falkenthal,
Michael and Leymann, Frank and Riegg, Andreas},
  title     = {Internet of things patterns for communication and management},
  booktitle = {Transactions on Pattern Languages of Programming IV},
  pages     = {139--182},
  year      = {2019},
  doi       = {10.1007/978-3-030-14291-9_5},
  series    = {Lecture Notes in Computer Science (LNCS)},
  volume    = {10600}
  publisher = {Springer-Verlag}
}
```

University of Stuttgart
Germany

Internet of Things Patterns for Communication and Management

Lukas Reinfurt[1,2(✉)], Uwe Breitenbücher[1], Michael Falkenthal[1], Frank Leymann[1], Andreas Riegg[2]

[1]Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

[2]Daimler AG, Stuttgart, Germany
{firstname.lastname}@daimler.com

Abstract. The Internet of Things is gaining a foothold in many different areas and industries. Though offerings vary in their scope and implementation, they often have to deal with similar problems: Constrained devices and networks, a vast amount of different vendors and technologies, security and privacy issues, etc. Over time, similar solutions for these problems appear, but the amount of available information makes it hard to identify the underlying principles. We investigated a large number of Internet of Things solutions and extracted the core principles into patterns. The eight patterns presented in this paper are: DEVICE GATEWAY enables devices that do not support a networks technology to connect to this network. DEVICE SHADOW allows other components to interact with offline devices. RULES ENGINE enables non-programmers to create rules that trigger actions. DEVICE WAKEUP TRIGGER informs sleeping devices that they should wake up. REMOTE LOCK AND WIPE allows lost or stolen devices to be secured. DELTA UPDATE only sends data that has changed since the last communication. REMOTE DEVICE MANAGEMENT enables remote device management with a client-server architecture. VISIBLE LIGHT COMMUNICATION uses existing lights to send messages to other devices.

Keywords: Internet of Things · Patterns · Embedded and cyber-physical systems · Device management

1 Introduction

In the last years, the *Internet of Things* (IoT) has gathered more and more attention in very different areas. It is driven by several developments, such as decreasing sensor and device sizes, energy consumption, or cost of chips and sensors. Additionally, widespread broadband connectivity and new communication technologies are also pushing the IoT forward. A future where many things will be connected to the internet seems increasingly palpable. This, in turn, would allow us to collect and analyze data about practically all aspects of our lives. The gathered knowledge could then be used for widespread improvements and automation.

adfa, p. 1, 2011.

There are a few core components that are combined to realize IoT systems, as shown in Figure 1. Central to the IoT are the things, which usually resemble some kind of *device*. These devices are often limited in their capabilities due to cost, size, energy, or technological constraints. The typical device contains a combination of *sensors* and/or *actuators*, a *processing component*, some means of *communication*, and an *energy supply*. Sensors are used to translate changes in the environment to electrical signals, whereas actuators are used to act on the environment by translating electrical signals into some kind of physical action [1]. They are controlled by the processing component, which can range from a simple circuit to complex chips. A device can also communicate with other components through wired or wireless communication technologies. These other components could be, for example, other devices or a backend server that runs in a data center or in the Cloud. A *backend server* is usually used to aggregate and process data from many devices. It uses this data to gain new insights and knowledge as well as to send commands to the actuators connected to the devices. It is also used to manage all the connected devices, e.g., for registering new devices, updating software and firmware, or managing security credentials. It might also communicate with *other components*, such as web services for analytics or data storage provided by other companies.

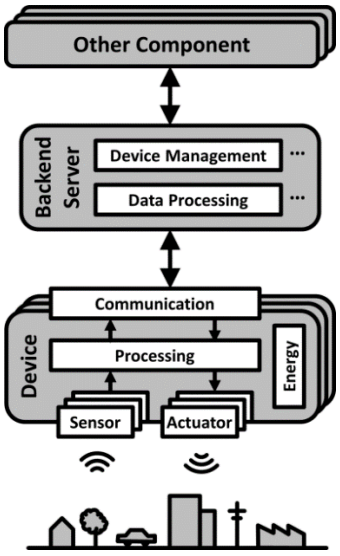


Fig. 1. Components Overview

As the IoT is not particular to any specific industry or domain, many different movements or solutions have developed over time that in some way incorporate the IoT. These include *Smart Homes*, *Smart Offices* [2, 3], *Smart Grids* [4], or the *Smart City* concept [5, 6], as well as initiatives like *Industrie 4.0* in Germany [7], or the *Industrial Internet* [8]. They all do essentially the same on a different scale: They integrate a manifold of independent, distributed, and, sometimes, also physically accessible sensors in

public environments to achieve two things: (i) to enable analyzing the gathered data and (ii) to use the processed analysis results to automate control of domain specific actuators. All these solutions share some significant similarities but have been developed mainly in closed off silos in the past. Several standardization efforts have been initiated that try to break up these silos on different levels. They include network connectivity standards [9–12], protocols [13–15], device management [16–18] or device communication frameworks [19–21]. It remains to be seen if all of these efforts can lead to a more unified IoT.

Getting to grips with all these developments is a challenge for companies. Because of the fragmented nature of the IoT space, it is not enough for them to look at different providers, solutions, and technologies in one IoT sector. Instead, they have to look in multiple separate sectors to find the most appropriate solution. Most corporations will come in contact with the IoT on one or multiple levels. A company might realize that it has to produce IoT-enabled products in the future to stay competitive. It might be able to save costs by introducing *Smart Factory* or *Smart Office* capabilities. It might find entirely new business opportunities that are connected to the IoT. When trying to build a good IoT solution, IT architects and developers at these companies are faced with the problems of:

- how to conceive application architectures to be robust for IoT challenges, i.e., how to receive and process data from a huge amount of sensors at the same time,
- how to assure security in terms of communication of devices as well as physical access to these devices,
- how to deal with energy and processing limitations of devices, and
- how to integrate multiple proprietary protocols supported by heterogeneous devices, sensors, and actuators into an IoT platform.

However, the prerequisite to tackling these issues is to understand the core design principles for developing IoT solutions. It is, therefore, valuable to extract and author a collection of proven design principles from production ready IoT solutions, which are already established in many IoT-platforms and related technologies.

Patterns have been used before to describe proven best practices that have stood the test of time in a specific domain. Examples include patterns for architecture [22], Cloud Computing [23], software design [24], or messaging systems [25]. Their abstraction of very similar and often reoccurring solutions into a structured form can be helpful to dissect and understand complex fields. They are also useful for comparing different solutions and solution providers for suitability for a specific task. Last but not least they can be used as a guideline for new implementations.

In this paper, which is an extended version of our former work that we have presented at the *21st European Conference on Pattern Languages of Programs (EuroPLoP)*, we describe eight patterns for the IoT, as seen in Table 1. The new contributions of this extended version to the original paper [26] are the three patterns, marked in Table 1 as new: DELTA UPDATE, REMOTE DEVICE MANAGEMENT, and VISIBLE LIGHT COMMUNICATION. As these patterns are also interlinked with the five original patterns, some minor adjustments have been made to the original patterns to reference

the new patterns. We also added a pattern map showing the relations between the patterns in this paper, as well as an overview of the evolving IoT Pattern Language.

The patterns are aimed at IT architects and developers. We have abstracted them from a systematic information collection process focusing on IoT-platforms and related technologies. We believe that these patterns help IT architects and developers working on IoT application with selecting, designing, and building better solutions. Although these patterns are presented as IoT patterns, some of them may also be applicable in other areas. For example, a RULES ENGINE may also be used in an IT system that does not involve things connected over the internet. However, these patterns are listed here as IoT patterns as they often play a vital role in IoT system.

Table 1. Overview of the presented patterns.

 **DEVICE GATEWAY** (p.11)	Some devices cannot directly connect to a network because they do not support the required communication technologies. These devices can be connected through a gateway.
 **DEVICE SHADOW** (p.15)	Other components can interact with currently offline devices by communicating with a persistently stored virtual representation of the device that is synchronized once the device reconnects.
 **RULES ENGINE** (p.18)	Users can define simple rules without needing to program. These rules tell the system with what action it should react to incoming events.
 **DEVICE WAKEUP TRIGGER** (p.22)	A device that is not currently connected to the backend server can be informed to do so by sending a message to a low-power communication channel where the device listens for such messages.
 **REMOTE LOCK AND WIPE** (p.26)	When a device is lost or stolen, its functionality can be remotely locked or data on it can be wiped, either fully or partially, to protect it from possible attacks.
 **DELTA UPDATE** (new) (p.29)	Only the values that have changed since the last communication are sent in a message to reduce the required traffic.
 **REMOTE DEVICE MANAGEMENT** (new) (p.32)	A central device management server allows remote management by sending management commands to management clients located on the devices who translate and execute these commands locally.
 **VISIBLE LIGHT COMMUNICATION** (new) (p.32)	Visible light is modulated to send messages, which can be received by photodiodes or cameras. Normal use of the lights is still possible.

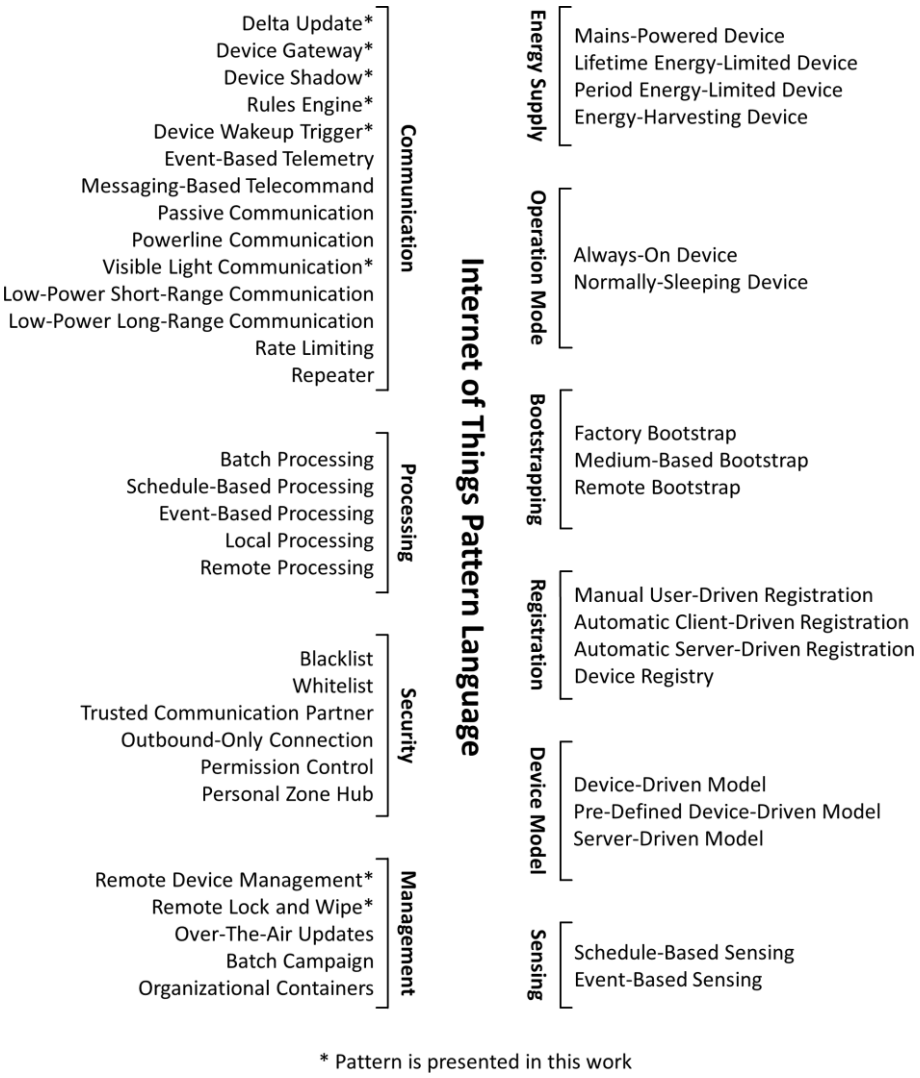


Fig. 2. The presented patterns and other well-founded pattern ideas as part of an evolving IoT pattern language.

These patterns do not stand alone but are part of an evolving pattern language, which we want to expand in the future with more patterns in several categories. Figure 2 shows the patterns presented in this paper in the context of a larger pattern language. Several potential categories and additional well-founded pattern ideas are also shown. All these pattern ideas are based on examples (several each) that we collected during the information collection process.

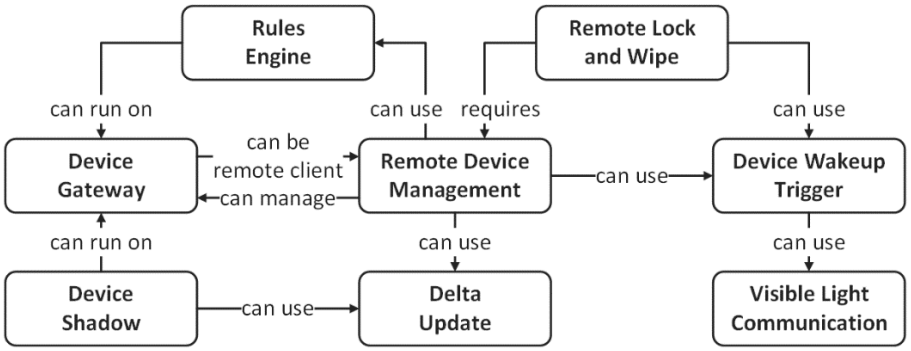


Fig. 3. Relations between the patterns presented in this work.

The patterns presented here are interconnected with each other. Figure 3 shows a pattern map, where each connection from one pattern to another pattern is labeled. Most of these connections, e.g., *can use* or *can run on*, describe how one pattern can be optionally enhanced or combined with another pattern. However, some connections, like *requires*, describe that one pattern is a mandatory prerequisite to be able to apply another pattern. These connections can therefore provide a guideline for a sensible reading and application order of the patterns. The connections are described in more detail in the patterns themselves.

The remainder of this paper is structured as follows: For a better comprehension, the original contribution of our previous work [26] is repeated in Sections 2, 3, 4, 5.1-5.5, and 6. Section 2 elaborates on how the patterns presented in this paper have been identified. Section 3 briefly describes the pattern format used for these patterns. Section 4 introduces definitions that are helpful in the scope of the IoT and that are frequently used in the pattern descriptions. Sections 5.1-5.5 present the five IoT patterns that we identified for the original paper [26]. Sections 5.6-5.8 contain the new patterns. Section 6 presents related work in the field of patterns and the IoT. Section 7 summarizes the paper and gives an outlook on planned future research.

2 Pattern Identification Process

The patterns presented in this paper have been identified by collecting and reviewing information from existing products and technologies following the pattern authoring process defined by Fehling et al. [27]. They divide the pattern writing process into three iterative phases: *Pattern Identification*, *Authoring*, and *Application*. They further divide each phase into several steps [27]. The patterns presented here are the result of the first two phases. The pattern language is currently applied in the SePiA.Pro[1] project. In the first step of the first phase, *Domain Definition*, we started by defining our understanding

[1] http://projekt-sepiapro.de/en/ (last accessed on 13.06.2018)

of the IoT domain by identifying common knowledge in this area. Some results of this step are the IoT overview presented in Figure 1 and the terminology and definitions in Section 4. In the next step, *Coverage Consideration*, we decided that we do not want to be limited to a specific sector or user group when looking for IoT patterns. Thus, commercial and open-source solutions for enterprises, developers, and end users alike were included. The exact sources for each pattern are detailed in the respective pattern's example section and include:

- **Product pages** that describe the functionality of IoT solutions
- **User manuals** that explain how to use IoT solutions
- **Technical documentation** of IoT solutions intended for developers
- **Standard documents** of technologies used in IoT solutions
- **Whitepapers** of companies that provide IoT solutions
- **Research Papers** that investigate technologies used in IoT solutions

During the *Information Format Design* step, we decided to use Citavi[2] to collect and manage all information. Citavi's quoting features can be used to extract information out of all kinds of documents and store them in a database. These quotes can then be tagged to assign them to a category, creating groups by similarity. These features where used in step four, *Information Collection*. A random selection of possible sources was created based on searches on Google and Google Scholar for IoT related keywords. These sources were scanned briefly. If the content of a source seemed relevant, it was added to the Citavi database and all relevant sections where marked as quotes and tagged, using the same tags for similar quotes. In the final step of the first phase, *Information Review*, the resulting collection of tagged quotes was reorganized. Some tags with quotes describing very similar concepts were combined, some tags with quotes describing disparate concepts were split up. Each tag then represents a rough pattern indicator or idea. Several tags belonging to a similar area were organized under a larger parent category. This resulted in the hierarchy shown in Figure 2.

During the second phase, *Pattern Authoring*, each tag containing at least three different examples [28] was authored into a pattern. In the first step, Pattern Language Design, the pattern format described in Section 3 was created based on other existing formats (see Section 3). In the next step, *Primitive Design*, additional graphical features commonly used in our patterns were defined. For example, each pattern's problem and solution section are put into a gray box to make them easy to find at a glance. Or, to ensure a unified look of the pattern sketches, graphical primitives for commonly used objects (such as *Device* or *User*) were created.

In the *Composition Language Definition* step, design rules for the pattern sketches were created to further ensure a consistent look. This includes defining common colors, line thicknesses, line types, etc. Then, in step four, *Pattern Writing*, the essence and core principles contained in the considered sources were abstracted to form a high level, provider independent description of the particular solution. Using these descriptions, patterns were authored following the advice of [29–32]. Finally, in step five, *Pattern*

[2] https://www.citavi.com/en (last accessed on 13.06.2018)

Language Revision, the links between the resulting patterns and pattern candidates (future patterns that have not yet been fully formulated) were checked for completeness. As this is an iterative process, phases and steps were revisited when new information was found, feedback of workshops was implemented, or ideas for other improvements appeared.

3 Pattern Format

This section describes the pattern format that is used to describe the patterns presented in this paper. It is based on pattern formats, approaches, and guidelines described in several publications about pattern writing or publications that contain patterns [23, 29–33]. While some elements are required in every pattern description, others are optional and are only used when necessary.

The **Name** is used to identify the pattern. Other names by which the pattern might be known in the industry are listed under **Aliases**. Additionally, the **Icon** adds a graphical representation of the pattern that is intended to be used in architecture diagrams or sketches [23]. The **Problem** section captures the core problem that is resolved by the pattern in an abstract manner, i.e., independent from a concrete domain or technology since the general problem might exist in many different use cases. Thus, more technical patterns [34] are out of the scope of this work. The **Context** then further describes the circumstances in which the problem typically occurs, which might impose constraints on the solution. Next, the **Forces** state the considerations that must be taken into account when choosing a solution to a problem. These can often be contradictory.

The **Solution** states the core steps to solve the problem and is often closed with a sketch depicting the architecture of the solution. Then, the **Result** section elaborates the solution in greater detail and describes the situation we find ourselves in after applying the pattern. **Variants** of the pattern are listed if they do not differ enough to need their own separate pattern description. Connections between patterns, such as patterns that are often applied together or patterns that exclude each other can be listed in the **Related Patterns** section. A final **Example** section lists concrete examples that illustrate the application of the pattern and could also contain links to concrete solution artifacts as conceptually introduced in [35] and validated for different domains in [36].

4 Terminology and Definitions

In this section, we define the basic terminology used to describe the IoT Patterns following Bormann et al. [37], who presented a terminology for constrained-node networks. The terminology defines different (i) device types, (ii) device energy supply types, and (iii) device operation modes. The following is a short summary to provide a clear understanding of the presented patterns.

4.1 Device Types

Devices in the IoT can be categorized into groups according to their computational and communication capabilities.

Unconstrained Devices have no significant constraints regarding their computational and communication capabilities. They are able to run arbitrary software and can use communication technology that is not specifically designed for low energy consumption, limited storage, or limited performance.

Semi-Constrained Devices are constrained in their computational power and/or storage space in such a way that they cannot use a common full protocol stack to communicate over the internet. However, they can use protocol stacks that are specifically designed for *Semi-Constrained Devices*, such as the Constrained Application Protocol (CoAP)[3], IPv6 over Low-power Wireless Personal Area Networks (6LoWPAN)[4], or Open Platform Communications Unified Architecture (OPC UA) Binary[5]. This enables them to act as fully integrated peers in a network without the help of a gateway or similar components. These nodes often also have a limited energy supply.

Constrained Devices are severely constrained in their computation, storage, and communication capabilities, often caused or accompanied by strong limitations of their energy supply. Therefore, they do not have the resources to support direct internet communication. Consequently, they use communication technology specifically designed for *Constrained Devices*, such as Bluetooth Low Energy[6], ZigBee[7], or Z-Wave[8].

4.2 Device Energy Supply Types

The energy supplies available for devices in the IoT can be divided into four groups.

Mains-Powered devices have no direct limitation to available energy, i.e., they are plugged into a wall socket. Unless there is an outage, they can use all the power they need.

Period Energy-Limited devices have a power source that has to be replaced or recharged in regular intervals, such as easily replaceable or rechargeable batteries or fuel in some kind of generator.

---

3 https://tools.ietf.org/html/rfc7252 (last accessed on 13.06.2018)
4 https://tools.ietf.org/html/rfc4944 (last accessed on 13.06.2018)
5 https://opcfoundation.org/ (last accessed on 13.06.2018)
6 https://www.bluetooth.com/bluetooth-technology/radio-versions (last accessed 13.06.2018)
7 http://www.zigbee.org/ (last accessed on 13.06.2018)
8 http://www.z-wave.com/ (last accessed on 13.06.2018)

Lifetime Energy-Limited devices contain a non-replaceable and non-rechargeable power source, such as a battery that is directly soldered onto the circuit board. Once this power source is depleted it cannot be easily replaced.

Energy Harvesting devices convert ambient energy into electrical energy. Ambient energy can be in form of radiant energy (solar, infrared, radio-frequency), thermal energy, mechanical energy, or biomechanical energy. The energy available to the device depends on the ambient energy available at the location of the device and might vary significantly over time. Energy harvesting can supply a device with perpetual power in some cases, but the available amount of energy is usually very small. Often, these devices will be mostly sleeping while they collect enough energy for short bursts of activity.

4.3 Device Operation Modes

Devices can operate in different modes depending on their communication frequency and their need to save energy.

Always-On devices have no reason to change operation modes to save power. They can stay connected and operational all the time.

Low-Power devices usually need to operate on small amounts of power but are still required to communicate frequently. They will sleep for short periods of time between communicating, but will generally stay connected to the network. This requires optimized hardware and communication solutions.

Normally-Off devices will be asleep most of the time and reconnect to the network at specific intervals to communicate (duty cycling).

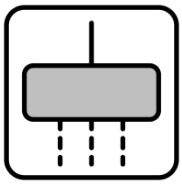
5 Internet of Things Patterns

In this section, we present eight IoT Patterns that were identified following the procedure described in Section 2. The format follows the definition presented in Section 3.

5.1 Device Gateway

Aliases: Gateway, Field Gateway, Intermediate Gateway, Physical Hub, Protocol Converter.

Context: A number of devices have to be connected to a network. These might include *Constrained Devices* or *Semi-Constrained Devices* that are limited in their processing power and do not support the communication methods of the network. These might also include *Unconstrained Devices* from legacy systems that cannot connect to the network due to outdated technology. A backend server reachable over this network is intended to process data from these devices.



Problem: You want to connect many different devices to an already existing network, but some of them might not support the networks communication technology or protocol.

Forces:

- **Connectivity:** Devices have to be connected to a network because you want to access their data and functionality regularly. Doing this manually is not an option.
- **Upgradability:** Changing or building up a network so that it supports the communication technology required by the device is often not possible. You might not control the network, or the purpose of the network cannot be realized with the device's technology, e.g., you need a long-range network but the device only supports short-range communication.
- **Effort:** Adding communication capabilities that are supported by the network to all device types would mean a high investment in time and resources, or might not be possible at all because of technological limitations.
- **Diversity:** Other devices with different communication technology might also have to be connected to the same network and will face the same problem.
- **Device Numbers:** Your network can only support a certain amount of simultaneous connections. The number of devices you want to connect exceeds this limit. Extending the network is not an option.

Solution: Connect devices to an intermediary DEVICE GATEWAY that translates the communication technology supported by the device to communication technology of the network and vice-versa.

Result: A DEVICE GATEWAY is usually a dedicated hardware appliance that can translate between a number of heterogeneous communication technologies. In many cases, it will be located at the edge of the network, close to the devices that it connects to the backend. It is possible to integrate a DEVICE GATEWAY into the backend, but this is often not practical. It is often used to translate low-power short-range communication to IP communication, so it has to be located close to these devices, whereas the backend is usually located far away in a data center.

For communication translation, it has to support at least two, but more commonly multiple communication technologies. On the interface towards the backend it usually supports IP communication over Ethernet, Wi-Fi, or mobile networks. On the interfaces towards the devices, it usually supports some kind of low energy communication technology. Depending on its application, it might also contain additional interfaces supporting other protocols. A translation layer converts messages received from either the backend or the devices to messages that can be sent to the respective opponent interface and vice versa. To be able to route the messages to their intended receivers the messages have to container some kind of identifier.

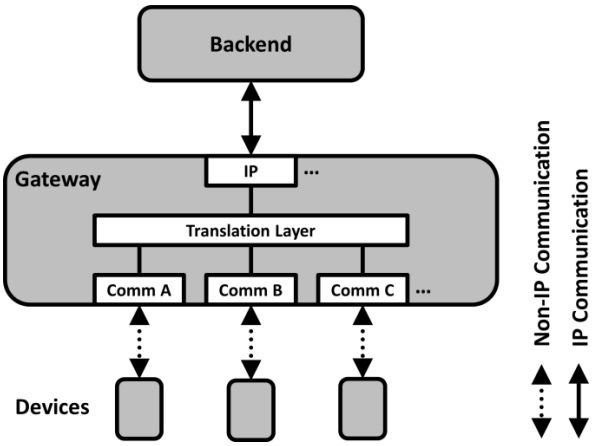


Fig. 4. Exemplary sketch of the DEVICE GATEWAY pattern used to transform to and from IP communication.

Benefits:

- **Connectivity:** Devices that do not directly support the networks communication technology can be connected to the network.
- **Separation of Concerns:** Device implementations can focus on only one arbitrary protocol or technology, which makes them simpler. On the other hand, the DEVICE GATEWAY can be optimized for protocol translation.
- **Effort:** One DEVICE GATEWAY can support multiple different communication technologies. The devices do not have to be modified.

- **Cost:** Many devices can be connected to a network via one DEVICE GATEWAY, without needing to support multiple communication technologies over the whole network, which saves costs.
- **Reusability:** It might be possible to reuse existing hardware as a DEVICE GATEWAY, for example, smartphones or routers, which might further decrease the effort and cost needed.
- **Technological Limitations:** Devices can use very limited communication technology in the form of a specifically reduced software stack. Therefore, they can exploit their limited power elsewhere, while still being able to connect to a network that requires more sophisticated technology through a DEVICE GATEWAY.
- **Additional Functionality:** A DEVICE GATEWAY might have enough resources to be able to implement additional functionality, such as management or monitoring capabilities, data aggregation or filtering, or enhanced security mechanisms.
- **Resilience:** A DEVICE GATEWAY with additional local functionality, like a RULES ENGINE and a backup battery, can add a layer of resilience and keep local processes running regardless of power or network outages and backend server failures.

Drawbacks:

- **Connectivity:** The DEVICE GATEWAY might become a single point of failure for the network connectivity of the connected devices. Adding redundant DEVICE GATEWAYS with a failover mechanism could alleviate this problem, but at an increased cost.
- **Security:** As a single point of attack, the DEVICE GATEWAY also poses a security risk. If compromised, an attacker could gain access to all attached devices or the backend server.
- **Complexity:** Another layer of components is introduced that has to be managed and maintained. This becomes even more difficult if multiple kinds of gateways are used.
- **Cost:** The DEVICE GATEWAY usually has to support multiple communication technologies and, thus, needs more processing power, which makes it expensive. In addition, if devices are distributed, possibly multiple DEVICE GATEWAYS are required to connect all of them. Costs might be reduced by using a modular DEVICE GATEWAY design, where only the required technologies can be added with extension boards.
- **Compatibility:** Some technologies might be incompatible on a conceptual level. A DEVICE GATEWAY might only be able to create a partial translation between these technologies, or it might not be able to translate between certain technologies at all.

Variants: Common variants of a pure DEVICE GATEWAY usually include some kind of local processing power. Some examples are listed below. They are not mutually exclusive and can be combined.

- **Aggregating Device Gateway:** Besides translating communication technologies this gateway also aggregates the messages it receives from the devices in some meaningful way. For example, it might average the temperature readings of several devices and send it on once a minute. This is usually done to reduce the number of individual messages that have to be sent to the backend.

- **Local Processing Device Gateway:** In addition to translating communication technologies, this gateway also contains some local processing functionality, which could mirror or replace functionality located in the backend. For example, it could contain a local RULES ENGINE, which decides some actions directly on the gateway. This is usually done to minimize communication with the backend and therefore reduce latency or to insulate from connection loss between gateway and backend.

Related Patterns:

- **MESSAGE GATEWAY:** The MESSAGE GATEWAY pattern is similar to the DEVICE GATEWAY, but describes how one or more gateways can be used to combine several different messaging technologies in a single machine [38].
- **ADAPTER:** The DEVICE GATEWAY can be seen as a physical version of the ADAPTER pattern that describes how two incompatible interfaces can work together by converting one interface to the other [24].
- **RULES ENGINE:** A DEVICE GATEWAY might contain a RULES ENGINE to trigger actions locally. This can prevent unnecessary round trips to a remote server and might decrease latency.
- **REMOTE DEVICE MANAGEMENT:** A DEVICE GATEWAY might act as a management client for *Constrained Devices* connected to it when using the *Remote Client* variant of REMOTE DEVICE MANAGEMENT.

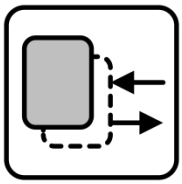
Examples: Central hubs are a common occurrence in the product portfolios of home automation companies. Here, they often act as an indispensable central point for integrating and managing the actual home automation devices. Examples are the Samsung SmartThings Hub [39] which supports ZigBee, Z-Wave, and IP, or the Wink Hub [40] that additionally supports Bluetooth Low Energy and Lutron Clear Connect[9]. The SmartThings Hub v2 also introduced local processing capabilities, which is also supported by other DEVICE GATEWAYS like the THNGHUB [41]. Various companies offer development kits and appliances to implement DEVICE GATEWAYS for industrial use, such as Intel, Dell, or Nexcom [42–44]. The Eclipse Kura project is an Open Source framework for building the software side of DEVICE GATEWAYS [45]. Zachariah et al. [46] proposed to use smartphones with Bluetooth Low Energy as universal gateways for other devices. In a way, smartphones are already used as DEVICE GATEWAYS for many wearable devices, like fitness trackers or smartwatches, which completely rely on the smartphone to communicate the data they collected to the backend. Many IoT platform documentations mention physical hubs or field gateways as a way to connect devices to their platforms that cannot connect to the internet on their own, even though they do not offer any products or solutions in this space [47–51]. Thus, these follow the idea of DEVICE GATEWAYS.

[9] http://www.lutron.com/en-US/Residential-Commercial-Solutions/Pages/Residential-Solutions/IntegrationConnectivity.aspx (last accessed on 13.06.2018)

5.2 Device Shadow

Aliases: Thing Shadow, Virtual Device

Context: Devices, such as *Constrained Devices*, *Semi-Constrained Devices*, and *Unconstrained Devices*, might operate in *Normally-Off*, *Low-Power*, or *Always-On* modes. Either because of their operation modes or because of external circumstances, these devices might be offline at various times.



Problem: Some devices will be only intermittently online in order to save energy or because of network outages. Other components want to interact with them but do not know when they will be reachable.

Forces:

- **Availability:** Sending commands to or reading state from offline devices is not possible.
- **Timeliness:** Waiting for currently offline device to come online again to send or receive data in a synchronous fashion can lead to long idle times and should be avoided.
- **Consistency:** Often a slightly out-of-date state is better than no state.

Solution: Store a persistent virtual representation of each device on some backend server. Include the latest received state from the device, as well as commands not yet sent to the device. Do all communication from and to the device through this virtual version. Synchronize the virtual representation with the actual device state when the device is online.

Result: By storing persistent virtual representations of the devices on the backend server and communicating only through those, device communication can be decoupled. This allows reading device state as well as sending device commands even if the device is offline. Essential to this is a persistent storage on the backend that can store virtual device representations reliably for many devices and that can handle read and write access from multiple sources. If commands are saved, they should be queued, unless only the newest command is regarded as relevant. When a device reconnects to the backend, which can happen according to a schedule or based on certain events, it can retrieve and process the stored command and update the last known state. To let other components know that a device is online, a flag can be stored with the device shadow. When a device connects or gracefully disconnects it enables or disables this

flag itself. Otherwise, the flag is set to false after a certain time of inactivity or by another mechanism, for example by the last will and testament of the Message Queue Telemetry Transport (MQTT)[10] protocol.

Conceivably, DEVICE SHADOW functionality could also be implemented on DEVICE GATEWAYS to allow localized decoupling between devices connected to one DEVICE GATEWAY. This would bring the benefits of a DEVICE SHADOW to these devices, even if the Gateway might be disconnected from the rest of the network from time to time. A problem here could be that a DEVICE GATEWAY might not be able to provide the reliable persistent storage that is needed.

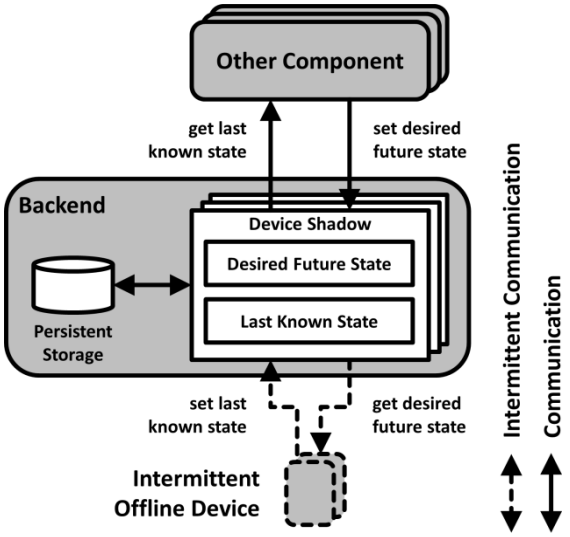


Fig. 5. Sketch of the DEVICE SHADOW pattern.

Benefits:

- **Unified Handling:** The communication with devices can be handled as if they are *Always-On*, even if they really are not. Therefore, time autonomy between backend and devices is established.
- **Additional Functionality:** If all communication goes through a DEVICE SHADOW, additional functionality can be implemented, such as batch messaging, filtering, or caching.
- **Security:** By only communicating with a single, well-known target, security can be increased, because devices can categorically deny communication attempts from any other source.

[10] http://mqtt.org/ (last accessed on 13.06.2018)

Drawbacks:

- **Eventual Consistency:** The virtual device representation is only eventually consistent with its actual state.
- **Synchronization Issues:** State updates could be lost if a new state update is written to the device shadow that is based on a state that is older than the current last known state. One way to avoid such issues is versioning the states and using OPTIMISTIC OFFLINE LOCK [52]. Other issues include transactional synchronization and issues depending on the semantics of the synchronized data.
- **Obsolescence:** By the time an offline device reconnects and receives stored commands, these commands might have become obsolete. In the same way, sensor values or other data send by the device to the Device Shadow may be too old to be useful by the time the device synchronizes. To avoid stale commands and data, the MESSAGE EXPIRATION pattern [25] can be used.
- **Quality of Service**: If all communication is forced through the backend server, latency and decreased availability for communication that could be done locally can be a problem.

Related Patterns:

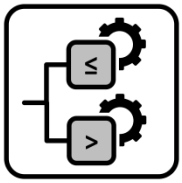
- **Remote Proxy:** Gamma et al. describe remote proxy as one application of the PROXY pattern. Here, the remote proxy locally represents an object in another address space to hide the fact that the object is remote [24]. DEVICE SHADOW can be seen as a device specific version of a remote proxy.

Examples: AWS IoT stores a persistent virtual version of each connected device that includes the last reported state and the desired future state of the device. This allows applications to read and write device state irrespective of the actual availability of the device [53, 54]. Azure IoT Suite stores device models in a device registry that is an eventually consistent view of device data [55]. Kii IoT Platform's Thing Interaction Framework saves the latest state of registered things on the backend server. Applications that request a device's state get the state stored on the server [56].

5.3 Rules Engine

Aliases: Action Engine, Trigger Conditions

Context: A wide range of differing messages from devices and other components are received at the backend server. These might include measurements from sensors, errors, a heartbeat, registration information, etc. These messages can arrive regularly or irregularly. There are different kinds of actions that have to be executed depending on the type of the received message, its content, the time it is received, or other factors.



Problem: Throughout its operation, a system receives a wide range of messages from devices and other components. You want to react in different ways to these messages.

Forces:

- **Flexibility:** The actions to trigger might change over time, new actions might be added, old ones removed, or you might want to temporarily test or disable an action. Hard-coding them into some software component would be possible, but is not flexible enough.
- **Data Sources:** In some cases, additional data apart from the device message might be needed to decide if a particular action should be taken.
- **Diversity:** The type of action to be triggered can vary significantly depending on the circumstances. In some cases, you might want to add an entry into a log file or send an email. In other cases, you might want to route a message to another service for further processing or store it in some kind of database.

Solution: Pass all messages received from devices through a RULES ENGINE. Allow non-programmers to define and manage rules using a graphical user interface. Provide an API for programmers. Use these rules to evaluate the content of incoming messages or metadata about the message against a set of comparators. Allow external data sources to be included in these comparisons. Let users associate a set of actions with these rules. Apply each rule on each message and trigger the associated actions if a rule matches.

Result: A RULES ENGINE contains a set of rules and actions that should be executed if a particular rule is met. Usually, these rules and associated actions are user definable through a graphical user interface on the backend server, which allows non-programmers to implement and manage these rules. But an API may provide developers with more flexible options. Another possibility is to provide a domain specific language (DSL) for creating the rules. During operation, each incoming message is compared against these rules. If a rule matches, the associated action is triggered. RULES ENGINES

are often located on a central backend server but can also be located on a DEVICE GATEWAY.

The rules usually allow comparing incoming data to static values, historical data, data from other sources, or a combination thereof. Different comparators allow a user to check if incoming data is, e.g., equal to, unequal to, larger than, or smaller than a certain value, or if it contains a certain value. Regular expressions or SQL statements might be allowed for comparisons that are more complex. Rule matching for a particular message could be stopped after the first match, or it could be continued until all rules are evaluated. It could also be possible to let a rule trigger only once and never again, or only once in a specific time window.

Actions can vary in their scope and complexity. Simple actions might trigger some functionality that is built into the platform that is used, such as sending an alert to a user. They might also act as a router that passes data on to services on the same backend server or to external services of other companies for further processing. One rule could only trigger one action, but it could also be possible to associate multiple actions to one rule that then could be executed in serial or in parallel.

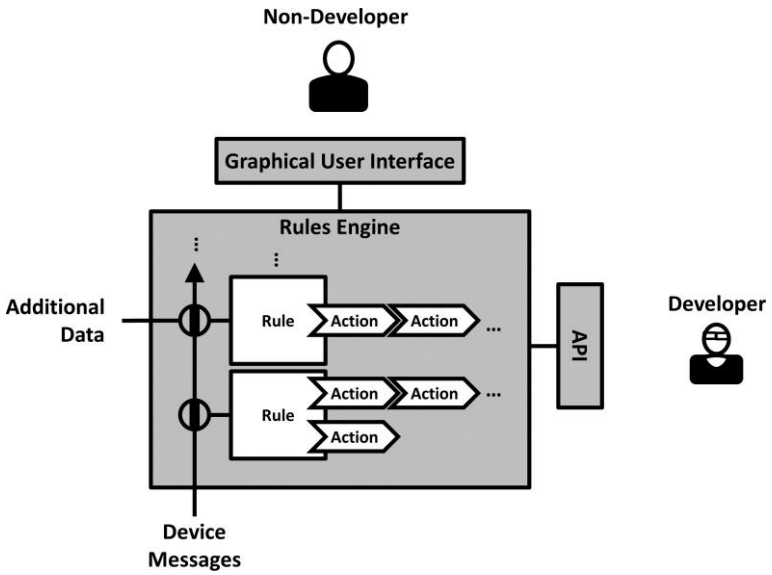


Fig. 6. Sketch of the RULES ENGINE pattern.

Benefits:

- **Flexibility:** Rules can be flexibly added, changed, temporarily disabled, or removed, because they are not hard-coded into software.
- **Ease of Use:** A graphical user interface allows non-programmers to manage rules.
- **Configurability:** The RULES ENGINE usually offers a wide range of options for how to evaluate the rules and trigger the actions, but users without extensive programming knowledge can configure simple rules.

- **Automation:** A RULES ENGINE allows creating automatic responses for certain situations.
- **Analytics:** A RULES ENGINE might track certain values to enable monitoring and analytics on the messages it receives and the rules and actions that are or are not triggered.

Drawbacks:

- **Suitability:** Depending on the functionality offered by the inbuilt rules and actions, a RULES ENGINE might not be suitable for certain complex transformation or routing tasks. A possible way to mitigate this drawback is to support user defined rules and actions via some scripting language.
- **Configurability:** While simple rules are easy to configure, complex rules might require more insight or special training.
- **Security:** A compromised or misconfigured RULES ENGINE can be a security risk.
- **Single Point of Failure:** If all messaged are passed through a RULES ENGINE it becomes a single point of failure.
- **Effort:** Creating and maintaining good rules might be a lot of work. Creating a marketplace for rules could be one solution to decrease effort and duplication and increase efficiency.

Related Patterns:

- **PRODUCTION RULE SYSTEM:** As described by Fowler [57], a PRODUCTION RULE SYSTEM organizes logic into a set of rules, where each rule has a condition and an action. While the PRODUCTION RULE SYSTEM is just a formalism to represent and organize logic into rules and conditions, the RULES ENGINE is the component that controls the execution of these rules.
- **CONTENT BASED ROUTER:** A RULES ENGINE can be seen as an extended CONTENT BASED ROUTER as described by Hohpe et al. [25]. A CONTENT BASED ROUTER examines only the message content and then routes the message to exactly one system. A RULES ENGINE can route to multiple systems based on the message content, other data, or a combination thereof.
- **REMOTE DEVICE MANAGEMENT:** A RULES ENGINE can be used on the management server to automate certain management tasks.

Examples: The AWS IoT Platform includes a RULES ENGINE that can transform and deliver inbound messages to other devices or Cloud services. Its rules can be applied to multiple data sources at once and multiple actions can be triggered in parallel. The rules can be created in an SQL-like syntax [53]. IBM IoT Real-Time Insights has an action engine that lets users define automated responses to detected conditions. Inbuilt actions include sending an email, triggering an IFTTT[11] recipe, or executing a Node-RED[12] workflow. Arbitrary other web services can be included with webhooks [58]. Many other IoT Platforms also include a Rules Engine [56, 59–63]. There are also standalone

[11] https://ifttt.com/ (last accessed on 13.06.2018)
[12] http://nodered.org/ (last accessed on 13.06.2018)

services like Waylay, IFTTT, and Zapier or apps like Stringify that offer RULES ENGINE functionality without a complete IoT platform [64–67]. Some RULES ENGINES, like EVRYTHNG's Reactor, can be located on DEVICE GATEWAYS to enable low latency message processing close to the devices [41].

5.4 Device Wakeup Trigger

Aliases: Update Trigger, Device Triggering

Context: You have a *Constrained Device* or *Semi-Constrained Device* that is *Lifetime Energy-Limited* or *Period Energy-Limited* and operates in a *Low-Power or Normally-Off* mode. You have a backend server where the device is registered, i.e., the server knows its identity and other metadata. From time to time, you have a situation where you want to contact the sleeping device immediately. For example, this could be the case if a critical security fix has to be applied, if you need current sensor values or send commands for one-off time critical situations, or if the device has been lost or stolen and you want to use REMOTE LOCK AND WIPE immediately.

Problem: Some devices might go into a sleep mode to conserve energy and only wake up from time to time to reconnect to the network. During sleep, they are not reachable on their regular communication channels. In some instances, other components may have to contact a sleeping device immediately.

Forces:

- **Irregularity:** You need to establish a connection at non-regular times.
- **Predictability:** You do not know the point in time when you need to connect to the device in advance.
- **Timeliness:** The device might reconnect on its own, but you cannot wait that long.
- **Power Consumption:** The device has to maintain low power consumption in terms of entering *Low-Power* or *Normally-Off* operation modes to save energy.

Solution: Implement a mechanism that allows the server to send a trigger message to the device via a low energy communication channel. Have the device listening for these triggering messages and immediately establish communication with the server when it receives such a message.

Result: A triggerable device can be in a *Low-Power* or *Normally-Off* [13] operation mode, where most of its functionality is dormant. However, it is still listening on a specific communication channel for triggering messages using a low energy communication module. When a server wants to wake up a device, it has to know the device and this channel in advance. Therefore, a prerequisite for the DEVICE WAKEUP TRIGGER is that the device has previously registered some kind of identifier and its listening channel with the backend server. This can either be done manually when the server or the

[13] The device can be NORMALLY-OFF when a passive trigger mechanism, such as passive RFID is used [68].

device is provisioned, or it could be done automatically by the device when it communicates with the server.

If the server wants to initiate communication with a triggerable device, it looks up the device in its registry and uses the stored information to send a trigger message to the channel that the device is listening on. The trigger message can contain a payload, e.g., to trigger some specific action on the device after the wake-up. Depending on the content and the existence of a payload, the triggered device might react in two ways: (i) If a payload was sent, it can process it and send a response to the server without establishing a long lasting connection (piggybacked response). (ii) If no payload was sent or the payload indicates that further communication is needed, the device can establish a long lasting connection to the server and wait for further instructions. The maximum time to wait for further instructions can be configured by a timeout, either directly on the device or in the payload of the wake-up message.

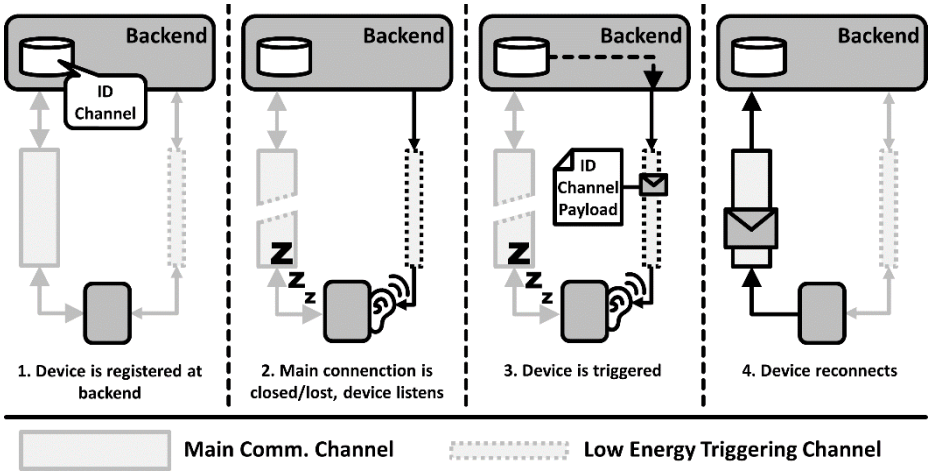


Fig. 7. Sketch of the DEVICE WAKEUP TRIGGER pattern.

Benefits:

- **Efficiency:** If no constant connection has to be kept alive, it allows the device to operate in a *Low-Power* or *Normally-Off* mode where the only active component is a low energy communication module listening for trigger messages.
- **Responsiveness:** Even though the device can be in a *Low-Power* or *Normally-Off* operation mode most of the time, it can be triggered to reconnect at any time if needed.

Drawbacks:

- **Efficiency:** At least the communication module has to be active to listen for triggering messages. To maximize efficiency, a very low power communication module should be used to listen for trigger messages. There are also passive RFID-based

modules that eliminate this drawback but at a loss of range compared to active modules [68].

- **Cost:** There might be costs associated with sending a trigger message, for example, when using SMS to trigger devices.
- **Infrastructure:** New infrastructure might be needed on the server side for a low energy communication channel, which is only used for device triggering.
- **Effort:** The device needs a second communication circuit which increases cost and complexity.
- **Responsiveness:** Although it is possible to wake up the device when needed, the wakeup procedure itself takes some time that should be accounted for.
- **Security:** The wakeup triggering channel is another vector for potential attacks on the device. For example, a denial of service (DoS) attack could be used to repeatedly wake up a device and, thus, quickly drain its battery.

Related Patterns:

- **CORRELATION IDENTIFIER:** A CORRELATION IDENTIFIER can be used when sending and replying to a DEVICE WAKEUP TRIGGER so that the server from which the trigger message originated knows to which trigger message the answer it received belongs [25].
- **VISIBLE LIGHT COMMUNICATION:** One way to implement a low-energy communication channel for a DEVICE WAKEUP TRIGGER is a circuit with a photodiode and VISIBLE LIGHT COMMUNICATION.

Examples: There have been several studies proposing active and passive wake-up receivers [68]. In general, active receivers provide remote wakeup capabilities at higher ranges while using some energy, while passive receivers use no energy but sacrifice range. One example is RFID, which is passive and low range. It's range can be extended by adding a power source to the receiver and, thus, turning it into active RFID [68, 69]. Device Triggering was introduced in release 11 of the 3rd Generation Partnership Project (3GPP) as a way to allow server initiated communication with UMTS or LTE devices when their IP address is not known. SMS is used as triggering mechanism, but a direct response to the payload is not supported [70]. 3GPP2 also supports Device Triggering using SMS, broadcast SMS, or IP transport [71]. OneM2M uses these mechanics to trigger devices to wake them up, to force them to establish a connection to the server, or when their IP address is not known [72]. Starsinic et al. [73] argue that LTE devices always have an IP address and using SMS as triggering mechanism makes applications using a DEVICE WAKEUP TRIGGER more platform dependent, because they always need to support SMS. Additionally, the lack of direct response to a trigger message requires devices to always establish a connection, which may be inefficient in cases where a simple reply to the trigger messages would have been sufficient. They propose an IP-based triggering method that is LTE backwards compatible and utilizes UDP packages. It supports direct responses to triggering messages, for example by using CoAP confirmable data packages. Open Mobile Alliance Lightweight Machine to Machine (OMA LWM2M) supports an update trigger mechanism where the server can wake up devices via SMS. An LWM2M client can disconnect if it does not receive a message

after a certain time but stays reachable via SMS. The LWM2M server queues operations for the client while it is offline. The server can send an update trigger message via SMS to the client. After the client received the SMS it reconnects and receives the queued operations [17]. The CPE WAN Management Protocol, also known as TR-069, includes a mechanism called asynchronous auto-configuration server-initiated notifications. It allows a configuration server to instruct a device to establish a connection with the server when a new configuration is available [18]. Examples of products are the PawTrax pet trackers. They stay in a sleep mode to save energy until activated by SMS. As a piggyback response, they send the current location of the pet, but they can also be switched to periodically send the location to an app or web platform [74].

5.5 Remote Lock and Wipe

Aliases: Remote Factory Reset, Remote Locking, Remote Wiping

Context: A device is connected to a backend server and is in danger of being lost or stolen. This might be the case because it is installed at an easily accessible public location, or a remote and unmonitored location. The device might have functionality that must not be accessed by a thief. It might also contain classified data that has to be kept protected. The data might or might not be encrypted. The device might be retrievable when it is lost or stolen, but it might also vanish forever.

Problem: Some devices might be lost or stolen. You want to prevent attackers from misusing the functionality of the device, or from gaining access to the data on the device or to the network through the device.

Forces:

- **Long-term Data Security:** If the device is irretrievably stolen, an attacker might have ample time to break encryptions if data on the device is encrypted.
- **Fine-grained Control:** Depending on the situation, the type of device and the content on the device, different actions might be necessary in the case of loss or theft.
- **Reversibility:** A lost or stolen device might eventually be returned, so any actions taken should be reversible if possible.
- **Remote Control:** Since the device is no longer physically available, the activation of additional security mechanisms has to work remotely.

Solution: Make the device a managed device that can receive and execute management operations from the backend server. Allow authorized users to use the backend server to trigger functionality on the device that can delete files, folders, applications or memory areas, revoke or remove permissions, keys, and certificates, or enable additional security feature. Execute triggered functions as soon as the device receives them and provide acknowledgment to the backend.

Result: To be able to offer REMOTE LOCK AND WIPE functionality, a device has to be a managed device that is connected to a management backend, which is a component on the backend server that can remotely execute management functionality on the device. This can be achieved by applying REMOTE DEVICE MANAGEMENT. Once an authorized user successfully authenticated to the backend, he or she can choose between different lock or wipe options depending on the circumstances. Which exact options are provided depends on the particular device. The device should provide a list of lockable or deletable data and functionality to the backend server.

In some circumstances, it might be enough to disable only some functionality but leave on location tracking to facilitate the retrieval of a lost or stolen device. The user might also only erase certain sensitive data to prevent data theft. In more severe cases, he or she might reset the device to its factory state, which would leave it operational but without any data on it. He or she might also completely disable the device to make it unusable.

Wiping data can be done by utilizing existing functionality to delete files and folders, or by directly deleting certain memory areas. Data can also be encrypted with a key stored on the device, which is used by applications to access this data. When this key is deleted, access to this data is effectively revoked. Functionality can be locked by revoking permissions, keys, or certificates that are required for execution, or by enabling security checks that were previously not enabled. Functionality could also be completely removed by deleting the associated code from the device. Once the requested operations are executed, the device should send back an acknowledgment to the backend server if possible.

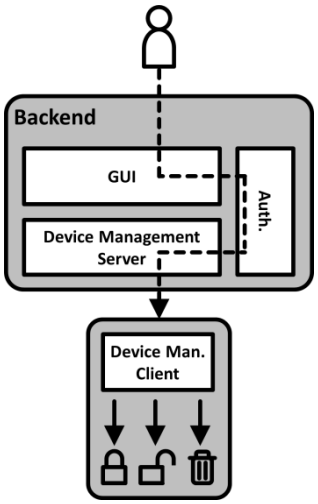


Fig. 8. Sketch of the REMOTE LOCK AND WIPE pattern.

Benefits:

- **Long-Term Data Security:** Wiping sensitive data from the device prevents an attacker from stealing the data, even when he has enough time to circumvent some kind of encryption.
- **Fine-grained Control:** Partially or fully locking or wiping and full factory reset allow reactions appropriate to the situation and the sensitivity of the data on the device, or its functionality.
- **Reversibility:** Locked device functionality can be unlocked if the device is retrieved.

- **Remote Action:** To execute lock and wipe functionality the device does not have to be under physical control. It only has to be connected to the backend so that the lock and wipe functionality can be triggered.

Drawbacks:

- **Reversibility:** Wiped data and a factory reset cannot be reversed. A backup mechanism could be used to be able to restore at least some data.
- **Connectivity:** The device has to be connected to receive the REMOTE LOCK AND WIPE commands. A DEVICE WAKEUP TRIGGER could be used to get the device to connect to the backend server.
- **Security:** If attackers gain access to the REMOTE LOCK AND WIPE functionality they could lock devices for ransom or wipe or disable them to cause damage. Proper authentication and authorization mechanisms, as well as end-to-end encryption, should be used at all times.

Related Patterns:

- **REMOTE DEVICE MANAGEMENT:** REMOTE LOCK AND WIPE is a specific use case of REMOTE DEVICE MANAGEMENT.
- **DEVICE WAKEUP TRIGGER:** A DEVICE WAKEUP TRIGGER could be used to get the device locked or wiped as soon as possible if it is currently not connected to the backend server.

Examples: Functionality to remotely locate, lock or wipe a phone is common on modern smartphones. Android phones can be located, set to ring, locked, or erased remotely with the Android Device Manager website or app [75]. Apple offers similar functionality through the iCloud [76, 77]. Options for other kinds of devices do also exist. The OMA LWM2M standard specifies a Lock and Wipe object. It supports functionality for partially or fully locking a device, for partially or fully wiping data on a device, and for doing a factory reset. These operations can be performed with or without user confirmation or notification [17]. The Kii IoT Platform allows users to lock and unlock devices over their web interface. When locked, the device is not able to access its data resources in the Cloud, while the owner and admin users still have access to these resources [78]. TR-069 and the IBM IoT Foundation Platform both support remote factory reset functionality [18, 79].

5.6 Delta Update

Aliases: Delta State, Delta Records

Context: You have devices with which you communicate using messages. The network they use to communicate has limited bandwidth. You want to add new devices to the network but you do not want to overwhelm the network. You cannot extend or change the network.

Problem: You want to reduce the size of messages containing sensor data without losing any information.

Forces:

- **Message Size:** You want to reduce the size of messages to fit more messages in existing connections but you do not want to lose any information.
- **Compression:** Compression alone does not give the desired results or using compression is impossible because you use severely *Constrained Devices*.
- **Structure:** Messages can have structured or unstructured content but in your case they have a common structure with multiple identifiable fields.
- **Repetition:** The messages may contain values that have been sent before without a change.

Solution: Store the last message sent. Calculate the delta from the current data to this message. Also, calculate a hash of the current full data set. Send only the delta and the hash to the receiver. Let the receiver merge the delta with its current state and check, if it matches the received hash.

Result: DELTA UPDATES reduce message size without losing information as they contain only the data that has changed since the last communication, but not more. As such, they need messages to have a common structure, which has multiple identifiable fields whose values do not change at once. Examples are devices that periodically send multiple sensor values. Other examples are backend servers that send configuration messages to devices to adjust their settings. If such messages are sent with values that have not changed, DELTA UPDATES reduce their size by omitting these unchanged values.

To send a DELTA UPDATE, the sender first has to calculate the delta, as shown in Figure 9, step 1 and 4. The sender does this based on the data it wants to be sent, for example, a set of sensor values or configuration parameters. The delta is the difference between the latest full data set and the last data set that the sender communicated. Thus, the sender has to store two or more full data sets: The current data set and the last

communicated data set for each communication partner. The exact algorithm for calculating the delta depends on the format of the data set. The resulting delta is empty if there were no changes to the data since the last communication. The delta is equal to the current data set if every value changed since the last update. If a value disappeared since the last update, the DELTA UPDATE has to include this change. One solution is to mark such a value with a reserved word to let the receiver know it has to delete this value.

Besides the delta, the sender calculates the hash value of its current full data set. It sends this value together with the delta to the intended receiver, as shown in step 2 and 5. The receiver merges the delta into its latest version of the dataset, as shown in step 3 and 6. The exact algorithm for merging the delta depends on the format of the data set. To make sure the data is consistent the receiver calculates the hash value of the resulting merged data set. If this hash value is equal to the hash value in the DELTA UPDATE the update was successful. Otherwise, the receiver asks the sender for a full update to synchronize their states.

The communication frequency for DELTA UPDATES varies depending on the use case. One way is to send DELTA UPDATES periodically at fixed intervals, regardless of changes. If there have been no changes since the last update, an empty Delta State message is comparable to a heartbeat and is thus no unnecessary overhead. Another way is to send DELTA UPDATES event-based. In this case, the sender emits a DELTA UPDATE when an event occurs, for example, if a sensor value has changed. A third way is to send a DELTA UPDATE once a parameter reaches a threshold. This limits communication to those situations where a relevant change has happened.

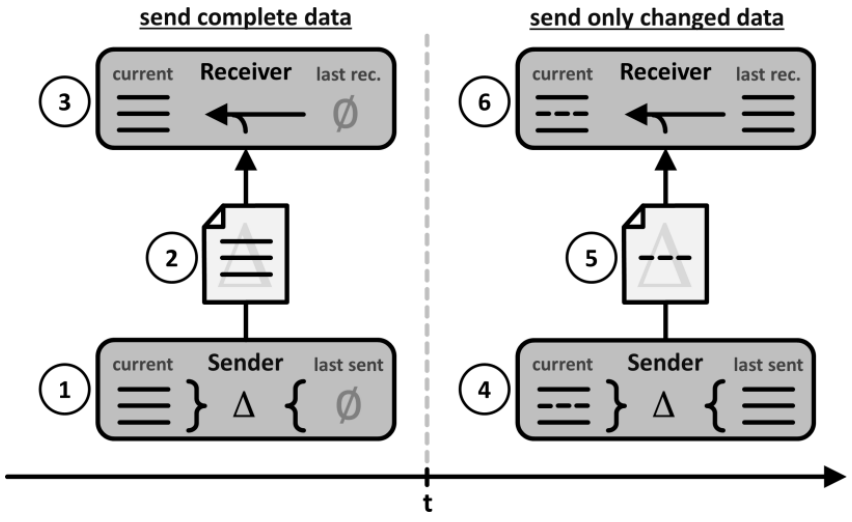


Fig. 9. Sketch of the DELTA UPDATE pattern.

Benefits:

- **Message Size:** Messages are smaller as they no longer contain any unnecessary data.
- **Information:** Messages do not loose information. They still contain the data that has changed since the last communication and only omit data which has not changed.
- **Bandwidth:** The decreased message size lowers the bandwidth required on involved components, such as devices, DEVICE GATEWAYS, etc.
- **Energy Consumption:** Communication is the biggest energy consumer on *Constrained Devices*. DELTA UPDATES are smaller than full updates and thus devices need less time to send them. This, in turn, allows devices to switch off communication modules for longer periods, which lowers their energy consumption.

Drawbacks:

- **Suitability:** DELTA UPDATES may not be suited for data that does not contain an identifiable structure which allows an algorithm to create and merge in deltas.
- **Data Consistency:** If messages containing DELTA UPDATE get lost, and the receiver merges a later message, the updates from the lost message are not present in the dataset. This leaves the receiver in an inconsistent state. To prevent this, use hash values to check for data consistency or reliable messaging technologies for transportation.
- **Increased Storage:** The sender has to store the last sent data set for each of its communication partners. Depending on the number of communication partners, this increases the storage space required by orders of magnitude. *Constrained Devices* with severe storage limitations may not have the required storage space.
- **Incompatibility:** Sending DELTA UPDATES leads to problems with communication patterns where receivers are not directly addressable. For example, a PubSub network distributes one message to multiple receivers. If a hash check fails and the receiver requests a full update, the network distributes the update to every receiver. If one hash check fails with each update this leads to more messages than before. Therefore, if multiple receivers are involved, they need a way to communicate with senders directly in case of data inconsistencies.

Examples: Libelium [80] mentions DELTA UPDATE in its Waspmote programmer guide. They describe sending only changed data as one way to lower the power consumption of the communication module. The OMA LWM2M standard [81] describes a client registration update operation. When executed, this operation sends the parameters that have changed since the last update. The Kaa IoT platform [82] uses delta messages to send configuration updates to endpoints. They make sure the data is consistent by comparing hashes between endpoint and server.

5.7 Remote Device Management

Aliases: Device Management

Context: You have a large number of devices, which need to be managed throughout their lifecycle. From time to time, you have to update the firmware or software installed on the device, or adjust configuration values. The locations of the devices are remote, or hard- or dangerous-to-reach.

Problem: You want to manage a large number of devices remotely.

Forces:

- **Location:** You have devices located in remote or dangerous areas. The placement of the devices makes management on location difficult.
- **Scalability:** You have to manage a large number of devices. Manual work does not scale for these numbers.
- **Outsourcing:** Device management is not part of your core business. You want the ability to have a third party to do the device management for you.
- **Security:** Managing devices includes handling sensitive information such as passwords. You want this information to stay secure.

Solution: Set up a management server on the backend. Add management clients to the device which you want to manage. Send management command from the server to the client and have the client execute these commands locally on the device.

Result: REMOTE DEVICE MANAGEMENT enables a remote party to execute management procedures on devices. If desired, even a third party is able to manage the devices. Otherwise, the device user, operator, owner, vendor, or a combination of them handles management. For example, router modems provided by internet service providers to customers include functionality, which enables the provider to set up the router remotely. However, users are able to override these settings with a local web interface or remote management interfaces accessible via the web or mobile apps.

Available management operations vary from device to device. The ability to create, read, update, and delete configuration values remotely enables managers to initially configure devices and adjust them to changing surroundings. Functionality for downloading and updating firmware and software on the device keeps versions up-to-date and allows quick reaction to security vulnerabilities. Besides, it allows the introduction of new features after you have installed the device. Other functionalities, such as remote rebooting and factory resets, are helpful for troubleshooting.

REMOTE DEVICE MANAGEMENT provides this remote management functionality with a client-server architecture. It involves three key components as shown in Figure 10: One or several management server, management clients, and a connection between them. The management servers are used to send management commands to the management clients. They handle authentication and authorization to ensure that only users or applications with proper authorization are able to trigger management commands. Other components are able to trigger management commands if a management server offers an external application programming interface (API). Users access the server through a graphical user interface, which allows them access to the management operations. The management server has the ability to store multiple device configurations and to give an overview of manageable devices. Users select existing configurations or create new ones, change them if needed, and apply them to one or more devices.

The messages containing the management commands are sent using a device management protocol. It is a bi-directional protocol where the server sends a command and receives a response when the device has processed the command. These commands are not timed out because the server cannot predict the time that a device needs to process a command. It has to be secure since management messages involve confidential data. End-to-end encryption is one solution that offers this security. Usually, there are many different devices connected to one management server and these devices may support different management protocols. Thus, the management server also has to be able to support different management protocols. This can be done with a plugin architecture, where a common internal representation of the management commands is translated into the required management protocols.

A device management model defines the parameters and functions, which are manageable and executable by the management server. It comprises a set of generic features common to devices, such as changing configuration values, updating the firmware, or doing a factory reset. Besides, vendors are able to extend and customize it for vendor-specific functionality.

The management client is a piece of software, which runs directly on a device and receives management messages from the server. The client translates the generic message format into the specific actions, which are necessary to execute the management operations on the device. It executes these operations on the device and sends a response to the server. A device managed by such a client is a *Managed Device*.

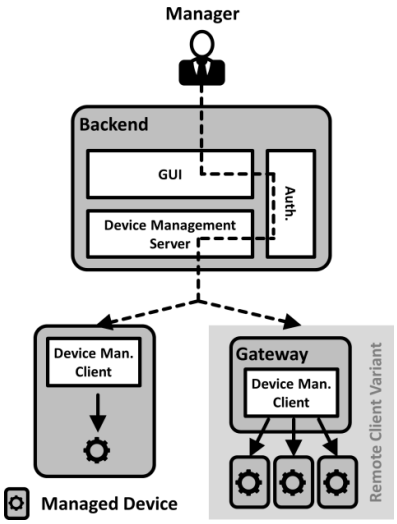


Fig. 10. Sketch of the REMOTE DEVICE MANAGEMENT pattern.

Benefits:

- **Remote Management:** Managers do not have to physically go to a device to change its configuration.
- **Decoupling:** The employed management protocol and model can hide differences between devices. This decouples the device management commands on the server from their implementation on the devices.
- **Regular Updates:** REMOTE DEVICE MANAGEMENT enables managers to apply regular updates to firmware and software on devices. This reduces the risk of devices with outdated and insecure software on them.
- **Bulk Management:** Managers do not have to manage devices individually. REMOTE DEVICE MANAGEMENT enables bulk management of devices, where the server applies changes to a large number of devices.
- **Automation:** Managers do not have to trigger management operations manually. A RULES ENGINE allows automatic management by triggering management operations, for example, when a device connects for the first time.

Drawbacks:

- **Security:** The management server, the clients, and the communication between them create new attack vectors. One solution to limit exposure is to use outbound-only communication for devices where they start communication and do not accept connection requests.
- **Scalability:** Sending bulk management operations to a large number of devices without limitations overwhelms networks with limited bandwidth. Bulk campaigns prevent this as they divide large bulk operations into smaller parts and issue these commands in stages. Besides, if the server sends pending notifications to a device that reconnects after a longer period of time, the number of notifications may overwhelm

the device. One solution is to queue these notifications and have the device retrieve them one by one when it is ready.
- **Connectivity:** *Constrained Devices* cannot keep up connections because of power constraints. A DEVICE WAKEUP TRIGGER is one solution to tell devices when they need to connect to the server.
- **Compatibility:** Multiple device management standards exist and vendors create custom implementations. Managing a heterogeneous group of devices requires a device management solution that handles different standards. However, this is complex to implement.

Related Patterns:

- **REMOTE LOCK AND WIPE:** REMOTE LOCK AND WIPE uses REMOTE DEVICE MANAGEMENT to implement security features. It allows authorized managers to remotely lock or wipe a device when it is no longer physically reachable due to loss or theft.

Variants:

- **Remote Client:** Severely *Constrained Devices* do not have the capabilities to run a resource-intensive management client. In such a case, other more capable devices in their vicinity host the client for them. For example, a DEVICE GATEWAY running a management client manages the *Constrained Devices* connected to it.

Examples: Multiple IoT platforms include remote device management functionality. IBM's Watson IoT platform includes a device management service. It communicates via a device management protocol based on Message Queuing Telemetry Transport (MQTT) with management agents located on the devices. The available management operations include location updates, firmware updates, as well as reboot and factory reset functionality [83]. Oracle mentions lifecycle management in its IoT Cloud Service documentation [84].

Multiple specifications and standards for device management exist. The Broadband Forum created the Customer Premise Equipment (CPE) Wide Area Network (WAN) Management Protocol, known as TR-069, for remote management of end-user devices. It includes an auto-configuration server, which automatically and dynamically provisions and configures devices based on criteria. These criteria include device specific requirements or general criteria, such as vendor, model, or software version. Besides, the protocol has software and firmware management, monitoring, and diagnostics functionality [18].

The Open Mobile Alliance (OMA) defines the Device Management (DM) protocol for remote management of mobile devices. It includes functionality for provisioning, configuration, software upgrades, and fault management. A DM server controls these functions on the DM clients. It has the ability to trigger sessions, but the clients start the sessions themselves. The protocol defines a common core parameter set, but vendors who need specific functionality are able to extend the protocol [16].

OMA Lightweight Machine to Machine (LWM2M) is a protocol optimized for remote management of *Constrained Devices*. An LWM2M server communicates with

LWM2M clients to reach management goals. These goals include device bootstrapping, registration, and management [81].

Nokia Motive Connected Device Platform is one solution that combines TR-069, OMA DM, and OMA LWM2M and other protocols into a single device management system. The platform automatically detects and configures devices. It is able to manage devices attached through gateways. Besides, its functionality includes remote firmware and software updates, diagnostics, fault management, and REMOTE LOCK AND WIPE [85, 86].

5.8 Visible Light Communication

Aliases: LiFi, Free-space Optical, Optical Wireless

Context: Today, wireless communication uses the radio spectrum, which is a part of the electromagnetic spectrum, because of its beneficial properties. Humans cannot perceive this spectrum and its waves are not harmful to the environment. Depending on their frequency, they travel long distances and through objects. Organizations that manage the radio spectrum further divide it into frequency bands for specific purposes, for example, broadcasting, air, and marine communication, or radar. This leaves a limited amount of frequencies for wireless communication technologies such as WiFi or Bluetooth. As more and more devices communicate wirelessly, these limited radio bands become increasingly crowded. This is problematic in areas with dense connectivity, such as office or apartment buildings, or high traffic public areas.

Problem: You need to use wireless communication in a crowded area, but the limited radio spectrum and the interference from many other devices lead to performance problems, which you want to avoid.

Forces:

- **Wireless:** You have to use wireless communication because devices are mobile or because using cables is not an option.
- **Limited Spectrum:** The radio spectrum used for wireless communication is limited and increasingly crowded.
- **Safety:** Wireless communication technology has to be safe to use near humans and other lifeforms. Besides, the technology has to minimize adverse effects on machinery that is sensible to electromagnetic effects.
- **Security:** Wireless communication has to be secure to avoid attackers from eavesdropping or tampering with messages.
- **Speed:** You need communication speed comparable to wireless communication technologies such as Bluetooth or WiFi.
- **Cost:** You want to keep costs for wireless communication low.
- **Infrastructure:** Building up a new wireless communication network requires investment into infrastructure.
- **Communication Distance:** Communication technologies have different distances at which they are usable. Finding a suitable technology often requires trade-offs in other areas.

Solution: Use visible light for short distance wireless communication. Modulate messages into the light by turning the light on and off. Do it fast to not impede normal light usage and to be invisible to the human eye.

Result: VISIBLE LIGHT COMMUNICATION (VLC) uses light in the visible spectrum between 380 to 720nm wavelength to transport messages. A sender encodes a message into a sequence of binary states, similar to Morse code. It sends out the sequence by turning a light source on and off in rapid succession, thereby modulating the sequence into the light. A receiver near the light decodes the sequence and reads the message.

Senders for VISIBLE LIGHT COMMUNICATION make use of existing infrastructure. A large amount of today's wireless communication happens in densely populated and frequented areas, in particular in buildings. In these areas, light fixtures are densely deployed and suitably positioned for data transmission. Lights using Light Emitting Diodes (LED) are now increasingly used as light sources because of their beneficial properties. LEDs are cheap and reliable and support the rapid modulation which encoding messages into the light requires. Using other components, such as lasers, is an option, but is not a concern of this pattern.

Multiple options exist for the form factor of the sender. One option is to integrate the sender into the fixture, but this requires large infrastructure investments. Another option is to have the sender as a separate part and connect it to the fixture, which allows retrofitting of VISIBLE LIGHT COMMUNICATION capabilities. A third option is to integrate the sender directly into LED lights.

The sender encodes messages into the lighting with intensity modulation. The modulation is not visible to the human eye since its frequency exceeds the flicker fusion threshold. Lights are dimmable by changing the relative periods of light and darkness. It is even workable to communicate data while the human eye perceives the light as off. Thus, VISIBLE LIGHT COMMUNICATION does not impede normal light functionality.

The receiver consists of a photodiode and a circuit for decoding the messages. Another option is to use a camera, for example, on a smartphone. Photodiodes support high-speed data reception while their framerate limits cameras to lower speeds. Cameras are able to extract data from multiple senders at once, which makes them great for positioning. They support positioning in three dimensions, including orientation, with high accuracy. Both photodiodes and cameras work with indirect light reflected from a surface, but direct line of sight to the sender is helpful. LED lights with integrated photodiodes act as both sender and receiver and thus enable bi-directional communication.

Hybrid systems exist which combine VISIBLE LIGHT COMMUNICATION with other wireless communication technologies, such as WiFi or Bluetooth. In these scenarios, the traditional wireless communication technologies act as backup when light communication is not working because of obfuscation. On the flip side, VISIBLE LIGHT COMMUNICATION frees up the wireless spectrum.

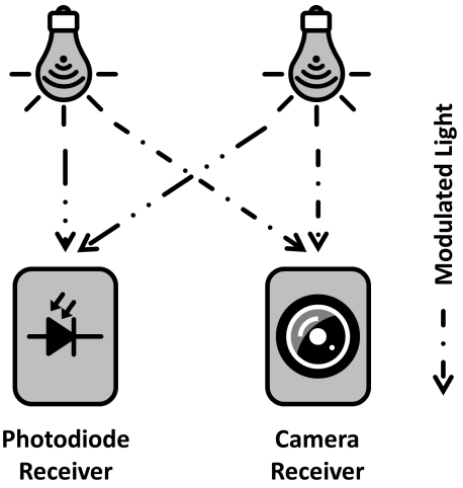


Fig. 11. Sketch of the VISIBLE LIGHT COMMUNICATION pattern.

Benefits:

- **Datarate:** The datarate benefits from the speed of light and is, thus, higher than in other wireless communication technologies that use radio waves.
- **Cost:** VISIBLE LIGHT COMMUNICATION uses low-cost off-the-shelf LEDs. Besides, it reuses existing lighting infrastructure, further reducing infrastructure cost. Energy cost is lower because the light has a dual functionality as it now additionally transports data.
- **Power:** Lights have a direct connection to power lines and, thus, do not need extra power sources. Besides, this allows you to combine them with Powerline Communication, where data is communicated over electrical wires [87].
- **Directional Propagation:** Directional propagation of light allows high spatial reuse and larger total network capacity.
- **Interference:** The visible light spectrum does not interfere with the radio spectrum. Opaque materials are one way to control and limit self-interference.
- **Safety:** Unlike other wireless communication technology, VISIBLE LIGHT COMMUNICATION does not cause electromagnetic interference. This makes it safe in areas where such interference is harmful and where authorities have banned other technologies, such as airplanes or hospitals. Besides, visible light does not pose a health risk.
- **Security:** Eavesdropping needs line-of-sight to the light source or a surface it illuminates. By confining the VISIBLE LIGHT COMMUNICATION to one room, security is controllable. Besides, security problems are identifiable as the communication channel is visible to the eye.
- **Localization:** Receivers are able to locate themselves using VISIBLE LIGHT COMMUNICATION. They use the strength of the received signal to calculate the distance to the transmitter.

Drawbacks:

- **Distance:** Long distance VISIBLE LIGHT COMMUNICATION becomes increasingly complicated to implement. Focused light needs to hit the receiver, which long distance solutions achieve with lasers and mechanical stabilization systems. This increases the cost and complexity of such solutions.
- **Infrastructure:** For VISIBLE LIGHT COMMUNICATION to play out its advantages, a suitable light infrastructure has to exist. In areas that do not need lighting, this is not given. You also need the components that modulate the lights if they do not already exist.
- **Line of Sight:** Direct line of sight is not needed, but beneficial. In situations where direct line of sight is not given at times, performance suffers. If no line of sight to a surface illuminated by a transmitter exists, communication does not work.
- **Security:** Using visible light as communication medium may pose new security risks. It may also be possible to launch attacks to disrupt communication, for example by using additional lights to overwhelm the receivers, similar to denial of service (DoS) attacks.

Related Patterns:

- **DEVICE WAKEUP TRIGGER:** VISIBLE LIGHT COMMUNICATION is one option to implement a DEVICE WAKEUP TRIGGER. In this case, a device uses a photodiode and a simple detection circuit to catch trigger messages sent using the lights in its environment.

Examples: Bell patented the idea of using optical signals for communication in 1880 [88]. Research in this general field, called Free Space Optical (FSO) communication, has steadily advanced the technology. FSO is now used in multiple forms and applications, including infrared remote controls and communication with spacecrafts and satellites [89]. In recent years, the availability and increased usage of low-cost LEDs have made a particular form of FSO, VISIBLE LIGHT COMMUNICATION, a practical option.

Realizing VLC using modified off-the-shelf LED bulbs is workable [90]. Photodiodes and cameras work as receivers and both are addressable with the same signal. Besides, photodiodes enable always-on VLC receivers that are suitable as a low-energy channel to receive DEVICE WAKEUP TRIGGER. The power available to *Energy Harvesting* devices is insufficient for using LEDs as uplink for VLC. An alternative is to use retro-reflective materials to reflect light from the transmitter and other light sources to form an uplink. An LCD-shutter modulates messages into the reflected light [91]. OpenVLC [92] is a project that offers an open-source VLC research platform based on off-the-shelf components.

Disney researches LED to LED communication between toys. Pointing a magic wand with a VLC enabled LED at a dress activates the lights in the dress. A smartphone add-on placed in the headphone jack or other VLC enabled lights control the lights of a toy police car [93].

Light-Fidelity (LiFi) [94] extends VLC by adding common wireless networking features. These include bi-directional multiuser communication and seamless handover

between cells. PureLiFi [95] offers commercial solutions, such as the LiFi-XC system. It consists of access points that modulate existing light fixtures and stations which plug into laptops via USB.

Qualcomm Lumicast [96] is a commercial technology, which uses VLC for indoor mobile device positioning where GPS is not available. They offer a software framework that allows developers to access the location information in their apps. Position accuracy is higher than with methods that use WiFi or Bluetooth. Besides, the framework offers orientation determination and three-dimensional positioning. It allows using auxiliary positioning methods as a backup. Other companies offer VLC services based on Lumicast, such as Acuity's BiteLight [97].

The Institute of Electrical and Electronics Engineers (IEEE) has created the 802.15.7 standard for VLC [98]. It describes a physical and media access control layer for short-range optical wireless communication. IEEE designed the standard for audio and video services, mobility, and compatibility with existing light infrastructure. During design, they considered impairments due to noise and eye safety.

6 Related Work

The concept of patterns, as introduced by Alexander et al. [22] is of course nothing new. Over the years, many publications either included new patterns for a specific field or talked about the pattern creation process in general. A selection of the latter was already mentioned in Section 3 and includes [23, 29–33]. Additional publications include [27, 99, 100]. Further, research about efficient pattern application via pattern refinement and concrete solutions organized in solution repositories emerges [34, 36].

Some patterns for topics in IoT or related areas exist. Eloranta et al. [101] describe patterns for building distributed control systems for moving machinery used for foresting, mining, construction etc. These patterns focus on aspects of reliability and fault-tolerance within these large machines but are not concerned with communication between small, *Constrained Devices* [101]. Qanbari et al. [102] present four patterns for edge application provisioning, deployment, orchestration, and monitoring. In addition to their narrow focus on edge applications, these patterns use existing technologies like Docker and Git, which are not suited for all *Constrained Devices*.

Publications in other contexts exist that contain patterns that are applicable in the IoT domain. The *Messaging Patterns* by Hohpe et al. [25] contain several patterns that can be used to describe communication aspects in the IoT. For example, the COMMAND MESSAGE and EVENT MESSAGE patterns fit neatly with the two types of messages that are exchanged in the IoT, namely messages that are sent to devices that contain a command, e.g., to activate some kind of actuator, and messages that are sent from devices to the backend for further processing by other components, e.g., sensor values. Other patterns that are applicable include EVENT-DRIVEN CONSUMER, PUBLISH-SUBSCRIBE CHANNEL, or GUARANTEED DELIVERY. However, these patterns only cover some aspects of IoT communication.

The *Cloud Computing Patterns* by Fehling et al. [23] also contain some patterns that are applicable in the IoT domain. For example, a variant of the WATCHDOG pattern can

be found on DEVICE GATEWAYS where it resets the system when it detects a problem with a critical component [103]. The EXACTLY-ONCE DELIVERY and AT-LEAST-ONCE DELIVERY patterns apply to device communication, for example when the MQTT protocol is used. The different workload patterns could be used to describe workloads generated by device messages and the LOOSE COUPLING pattern discusses principles to decouple devices from other components that consume their data or trigger some actuator functionality of the device, respectively. Again, these patterns only cover some aspects that are relevant for IoT.

7 Summary and Outlook

The vision of the IoT has existed for a few years now. While not fully realized yet, recent developments have added numerous solutions, technologies, and standardization efforts to various areas of this field. However, their ever-increasing number and heterogeneity make it hard to grasp the underlying principles. To help to understand this complex field, we presented IoT patterns, which summarize recurring solutions to various problems in the IoT space. In our original work [26], we presented five patterns: DEVICE GATEWAY, which enables devices which do not support the technology of a network to communicate with this network, DEVICE SHADOW, which allows other components to interact with offline devices, RULES ENGINE, which enables non-programmers to design rules which trigger actions, DEVICE WAKEUP TRIGGER, which notifies sleeping devices when they should wake up, and REMOTE LOCK AND WIPE, which allows lost or stolen devices to be secured.

In this extended version of our original work [26], we added three new patterns: DELTA UPDATE, which only sends data which has changed since the last communication, REMOTE DEVICE MANAGEMENT, which allows remote device management using a client-server architecture, and VISIBLE LIGHT COMMUNICATION, which modulates visible light to send messages to devices. These patterns already show relations between them and the new patterns added in this extended version also added new relations. They also hint at other patterns that have not yet been published. We are working on expanding this pattern catalog into a full pattern language by adding new patterns and investigating relations between these patterns. In the future, this pattern language will guide developers towards useful pattern combinations, give companies a tool to evaluate different IoT providers and solutions, and help other interested readers to understand the different aspects of the IoT.

Acknowledgements. We would like to thank our shepherd Marko Leppänen for the discussions and comments that helped to improve this paper. This work was partially funded by the BMWi projects NEMAR (03ET4018B), SmartOrchestra (01MD16001F) and SePiA.Pro (01MD16013F).

References

1. Anjanappa, M., Datta, K., Song, T.: Introduction to Sensors and Actuators. In: Bishop, R.H. (ed.) The Mechatronics Handbook, pp. 327–340. CRC Press, Boca Raton, Florida (2002)
2. Röcker, C.: Services and Applications for Smart Office Environments - A Survey of State-of-the-Art Usage Scenarios. International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering 4, 51–67 (2010)
3. Le Gal, C., Martin, J., Lux, A., Crowley, J.L.: SmartOffice: Design of an Intelligent Environment. IEEE Intelligent Systems 16, 60–66 (2001)
4. Kopp, O., Falkenthal, M., Hartmann, N., Leymann, F., Schwarz, H., Thomsen, J.: Towards a Cloud-based Platform Architecture for a Decentralized Market Agent. In: Cunningham, D., Hofstedt, P., Meer, K., Schmitt, I. (eds.) INFORMATIK 2015, P-246, pp. 69–80. Gesellschaft für Informatik e.V. (GI), Bonn (2015)
5. Nam, T., Pardo, T.A.: Conceptualizing Smart City with Dimensions of Technology, People, and Institutions. In: Proceedings of the 12th Annual International Digital Government Research Conference: Digital Government Innovation in Challenging Times, pp. 282–291. ACM, New York, NY (2011)
6. Su, K., Li, J., Fu, H.: Smart City and the Applications. In: 2011 International Conference on Electronics, Communications and Control (ICECC), pp. 1028–1031. IEEE, Piscataway, NJ (2011)
7. Kagemann, H., Wahlster, W. and Helbig, J.: Recommendations for implementing the strategic initiative INDUSTRIE 4.0, http://www.acatech.de/fileadmin/user_upload/Baumstruktur_nach_Website/Acatech/root/de/Material_fuer_Sonderseiten/Industrie_4.0/Final_report__Industrie_4.0_accessible.pdf (2013)
8. Industrial Internet Consortium: Overview, http://www.iiconsortium.org/pdf/IIC-Overview-11-24-15.pdf (2015)
9. ZigBee Alliance: Control your World, http://www.zigbee.org/
10. Z-Wave Alliance: The Internet of Things is powered by Z-Wave, http://z-wavealliance.org/
11. Bluetooth: Bluetooth Technology Website, https://www.bluetooth.com/what-is-bluetooth-technology/how-it-works
12. Thread Group: Home, http://www.threadgroup.org/
13. IETF: The Constrained Application Protocol (CoAP), https://tools.ietf.org/html/rfc7252 (2014)
14. OASIS: MQTT Version 3.1.1. OASIS, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf (2014)
15. OPC Foundation: Unified Architecture - OPC Foundation, https://opcfoundation.org/about/opc-technologies/opc-ua/
16. Open Mobile Alliance: OMA Device Management Protocol. Open Mobile Alliance, http://www.openmobilealliance.org/release/DM/V2_0-20150122-C/OMA-TS-DM_Protocol-V2_0-20150122-C.pdf (2015)

17. Open Mobile Alliance: Lightweight M2M - Lock and Wipe Object (LwM2M Object - LockWipe), http://technical.openmobilealliance.org/Technical/Release_Program/docs/LWM2M_LOCKWIPE/V1_0-20150217-C/OMA-TS-LWM2M_LockWipe-V1_0-20150217-C.pd (2015)
18. Bernstein, J. and Spets, T.: DSL Forum TR-069. CPE WAN Management Protocol, https://www.broadband-forum.org/technical/download/TR-069.pdf (2004)
19. AllSeen Alliance: AllSeen Alliance, https://allseenalliance.org/
20. Open Interconnect Consortium: Open Interconnect Consortium, http://openinterconnect.org/
21. Object Management Group: Data Distribution Service (DDS), http://www.omg.org/spec/DDS/1.4/PDF/ (2015)
22. Alexander, C., Ishikawa, S., Silverstein, M.: A Pattern Language: Towns, Buildings, Construction. Oxford University Press, New York (1977)
23. Fehling, C., Leymann, F., Retter, R., Schupeck, W., Arbitter, P.: Cloud Computing Patterns. Fundamentals to Design, Build, and Manage Cloud Applications. Springer, Wien (2014)
24. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, Massachusetts (1995)
25. Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Designing, Building, and Deploying Messaging Solutions. Addison-Wesley, Boston, Massachusetts (2004)
26. Reinfurt, L., Breitenbücher, U., Falkenthal, M., Leymann, F., Riegg, A.: Internet of Things Patterns. In: Proceedings of the 21st European Conference on Pattern Languages of Programs (EuroPLoP). ACM (2016)
27. Fehling, C., Barzen, J., Breitenbücher, U., Leymann, F.: A Process for Pattern Identification, Authoring, and Application. In: Proceedings of the 19th European Conference on Pattern Languages of Programs (EuroPLoP). ACM, New York, NY (2015)
28. Coplien, J.O.: Software Patterns. SIGS, New York, NY (1996)
29. Meszaros, G., Doble, J.: Metapatterns: A Pattern Language for Pattern Writing. In: Third Pattern Languages of Programming Conference. Addison-Wesley (1996)
30. Wellhausen, T., Fießer, A.: How to write a pattern? A rough guide for first-time pattern authors. In: Proceedings of the 16th European Conference on Pattern Languages of Programs. ACM, New York, NY (2012)
31. Harrison, N.B.: Advanced Pattern Writing. Patterns for Experienced Pattern Authors. In: Pattern languages of program design 5, 5, pp. 433–452. Addison-Wesley, Upper Saddler River, NJ (2006)
32. Harrison, N.B.: The Language of Shepherding. A Pattern Language for Shepherds and Sheep. In: Pattern languages of program design 5, 5, pp. 507–530. Addison-Wesley, Upper Saddler River, NJ (2006)
33. Fehling, C., Barzen, J., Falkenthal, M., Leymann, F.: PatternPedia - Collaborative Pattern Identification and Authoring. In: PURPLSOC (In Pursuit of Pattern Languages for Societal Change): The Workshop 2014, pp. 252–284. epubli GmbH, Berlin (2015)

34. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F., Hadjakos, A., Hentschel, F., Schulze, H.: Leveraging Pattern Application via Pattern Refinement. In: Proceedings of the International Conference on Pursuit of Pattern Languages for Societal Change (PURPLSOC) (2016)
35. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: From Pattern Languages to Solution Implementations. In: Proceedings of the Sixth International Conferences on Pervasive Patterns and Applications (PATTERNS 2014), pp. 12–21. IARIA, Wilmington, DE (2014)
36. Falkenthal, M., Barzen, J., Breitenbücher, U., Fehling, C., Leymann, F.: Efficient Pattern Application: Validating the Concept of Solution Implementations in Different Domains. International Journal on Advances in Software 7, 710–726 (2014)
37. Bormann, C., Ersue, M. and Keranen, A.: Terminology for Constrained-Node Networks, http://www.rfc-editor.org/rfc/pdfrfc/rfc7228.txt.pdf (2014)
38. Eloranta, V.-P., Koskinen, J., Leppänen, M. and Reijonen, V.: Patterns for the Companion Website, http://media.wiley.com/product_ancillary/55/11186941/DOWNLOAD/website_patterns.pdf
39. SmartThings: Architecture, http://docs.smartthings.com/en/latest/architecture/index.html
40. Wink: Wink Hub, http://www.wink.com/products/wink-hub/
41. EVRYTHNG: THINGHUB Local Cloud Gateway, https://evrythng.com/wp-content/uploads/THNGHUB-data-sheet.pdf
42. Intel: Intel IoT Gateways, https://www-ssl.intel.com/content/www/us/en/embedded/solutions/iot-gateway/overview.html
43. Dell: Dell IoT solutions, http://www.dell.com/learn/us/en/04/oem/oem-internet-of-things
44. Nexcom: IoT Gateway, http://www.nexcom.com/Products/industrial-computing-solutions/iot-solutions/iot-gateway
45. Eclipse Foundation: Kura - Open Source Framework for IoT, http://www.eclipse.org/kura/
46. Zachariah, T., Klugman, N., Campbell, B., Adkins, J., Jackson, N., Dutta, P.: The Internet of Things Has a Gateway Problem. In: Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications - HotMobile '15, pp. 27–32. ACM, New York, NY (2015)
47. Amazon Web Services: AWS IoT FAQs, https://aws.amazon.com/iot/faqs/
48. Microsoft: Azure and IoT, https://azure.microsoft.com/en-us/documentation/articles/iot-hub-what-is-azure-iot/
49. Microsoft: Azure IoT Hub guidance, https://azure.microsoft.com/en-us/documentation/articles/iot-hub-guidance/
50. Comarch Technologies: Comarch IoT Platform. In the pursuit of becoming smart, http://technologies.comarch.com/wp-content/uploads/2015/10/CT_IoT-white-paper_22092015_WEB.pdf (2015)
51. Bosch Software Innovations: The Bosch IoT Suite. Technology for a Connected World, https://www.bosch-si.com/media/en/bosch_si/iot_platform/bosch-iot-suite_product-brochure.pdf (2015)

52. Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., Stafford, R.: Patterns of Enterprise Application Architecture. Addison-Wesley, Boston, Massachusetts (2002)
53. Amazon Web Services: How the AWS IoT Platform Works, https://aws.amazon.com/iot/how-it-works
54. Amazon Web Services: Device Shadows Documents, http://docs.aws.amazon.com/iot/latest/developerguide/thing-shadow-document.html
55. Microsoft: Overview of device management with IoT Hub, https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-management-overview
56. Kii: State Registration and Retrieval, http://documentation.kii.com/en/starts/thingifsdk/model/states/
57. Fowler, M.: Domain-Specific Languages. Addison-Wesley, Upper Saddle River, NJ (2011)
58. IBM: Getting started with IoT Real-Time Insights, http://www.ng.bluemix.net/docs/services/iotrtinsights/index.html
59. myDevices: myDevices Connected Device Platform for the Internet of Things, https://www.mydevices.com/platform
60. Wind River: Wind River Helix Device Cloud, http://www.windriver.com/products/product-overviews/wr-device-cloud_overview.pdf (2015)
61. Comarch Technologies: Digital Lifestyle & IoT Solutions, http://www.comarch.com/files-com/file_91/Comarch-Digital-Lifestyle-and-IoT-Solution-283522.pdf (2015)
62. Ayla Networks: Ayla Architecture. Focusing on the 'Things' and Their Manufacturers, https://www.aylanetworks.com/wp-content/uploads/2015/06/Ayla_Architecture_White_Paper_preview.pdf (2015)
63. EVRYTHNG: Evrythng Platform Overview, https://evrythng.com/wp-content/uploads/EVRYTHNG-IoT-Platform-Overview.pdf
64. waylay.io: Waylay.io Documentation, https://docs.waylay.io/usage/tasks-and-templates/
65. IFTTT: IFTTT, https://ifttt.com/
66. Zapier: Connect Your Apps and Automate Workflows, https://zapier.com/
67. Stringify: Home - Stringify, https://www.stringify.com/
68. Ba, H., Parvin, J., Soto, L., Demirkol, I., Heinzelman, W.: Passive RFID-based Wake-Up Radios for Wireless Sensor Networks. In: Wirelessly Powered Sensor Networks and Computational RFID, pp. 113–129. Springer (2013)
69. Ruzzelli, A.G., Jurdak, R., O'Hare, G.M.P.: On the RFID wake-up impulse for multi-hop sensor networks. In: The 1st ACM Workshop on Convergence of RFID and Wireless Sensor Networks and their Applications (SenseID) at the 5th ACM International Conference on Embedded Networked Sensor Systems (ACM SenSys 2007) (2007)
70. ETSI: 3GPP TS 23.682. Architecture enhancements to facilitate communications with packet data networks and applications, http://www.etsi.org/deliver/etsi_ts/123600_123699/123682/12.04.00_60/ts_123682v120400p.pdf (2015)
71. 3GPP2: Network Enhancements for Machine to Machine (M2M), http://www.3gpp2.org/public_html/specs/X.S0068-0_v1.0_M2M_Enhancements_20140718.pdf (2014)

72. oneM2M: Functional Architecture, http://www.onem2m.org/images/files/deliverables/TS-0001-Functional_Architecture-V1_6_1.pdf (2015)
73. Starsinic, M., Mohamed, A.S.I., Lu, G., Seed, D., Aghili, B., Wang, C., Palanisamy, S., Murthy, P.: An IP-Based Triggering Method for LTE MTC Devices. In: 2015 Wireless Telecommunications Symposium (WTS). IEEE (2015)
74. PawTrax: Welcome to PawTrax, http://www.pawtrax.co.uk/
75. Google: Remotely ring, lock, or erase a lost device - Accounts Help, https://support.google.com/accounts/answer/6160500
76. Apple: iCloud: Use Lost Mode, https://support.apple.com/kb/PH2700
77. Apple: iCloud: Erase your device, https://support.apple.com/kb/PH2701
78. Kii: Disable/Enable Things, http://documentation.kii.com/en/guides/thingifsdk/thingsdk/thing-client/things-status/
79. IBM: Device Management Operations - Device Actions, https://console.bluemix.net/docs/services/IoT/devices/device_mgmt/requests.html#requests
80. Libelium: Waspmote Programming Guide (2015)
81. Open Mobile Alliance: Lightweight Machine to Machine Technical Specification, http://technical.openmobilealliance.org/Technical/Release_Program/docs/LightweightM2M/V1_0-20151030-C/OMA-TS-LightweightM2M-V1_0-20151030-C.pd (2015)
82. CyberVision: Configuration - Kaa - Kaa documentation, http://docs.kaaproject.org/display/KAA/Configuration
83. IBM: About Watson IoT Platform, https://console.bluemix.net/docs/services/IoT/iotplatform_overview.html#about_iotplatform
84. Oracle: Using Oracle Internet of Things Cloud Service, http://docs.oracle.com/cloud/latest/iot/IOTGS/IOTGS.pdf
85. Nokia: Nokia Motive connected device platform, http://resources.alcatel-lucent.com/asset/196246 (2016)
86. Nokia: Motive Connected Device Platform. Release 6.0, http://resources.alcatel-lucent.com/asset/196247 (2016)
87. Komine, T., Nakagawa, M.: Integrated system of white LED visible-light communication and power-line communication. IEEE Transactions on Consumer Electronics 49, 71–79 (2003)
88. Bell, A.G.: Apparatus for Signaling and Communicating, called Photophone (1880)
89. Sevincer, A., Bhattarai, A., Bilgi, M., Yuksel, M., Pala, N.: LIGHTNETs: Smart LIGHTing and Mobile Optical Wireless NETworks – A Survey. IEEE Communications Surveys & Tutorials 15, 1620–1641 (2013)
90. Schmid, S., Richner, T., Mangold, S. and Gross, T.R.: EnLighting: An Indoor Visible Light Communication System Based on Networked Light Bulbs, https://s3-us-west-1.amazonaws.com/disneyresearch/wp-content/uploads/20160615205959/EnLighting-An-Indoor-Visible-Light-Communication-System-based-on-Networked-Light-Bulbs-Paper.pdf

91. Li, Jiangtao, Lie, Angli, Shen Guobin, Li, L., Sun, C., Zhao, F.: Retro-VLC: Enabling Battery-free Duplex Visible Light Communication for Mobile and IoT Applications. In: Manweiler, J., Choudhury, R.R. (eds.) Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile 2015), pp. 21–26. ACM, New York, NY (2015)
92. Wang, Q., Donne, D. de, Giustiniano, D.: Demonstration Abstract: Research Platform for Visible Light Communication and Sensing Systems. In: Proceedings of the 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN). IEEE (2016)
93. Schmid, S., Ziegler, J., Gross, T.R., Hitz, Manuela, Psarra, Afroditi, Corbellini, G. and Mangold, S.: (In)visible Light Communication: Combining Illumination and Communication, https://s3-us-west-1.amazonaws.com/disneyresearch/wp-content/uploads/20140915070828/Pub_InvisibleLightCommunication_Siggraph14_paper.pdf
94. Haas, H., Yin, L., Wang, Y., Chen, C.: What is LiFi? Journal of Lightwave Technology 34, 1533–1544 (2016)
95. pureLiFi: LiFi-XC, https://purelifi.com/lifi-products/
96. Jovicic, A.: Qualcomm Lumicast : A high accuracy indoor positioning system based on visible light communication, https://www.qualcomm.com/media/documents/files/lumicast-whitepaper.pdf (2016)
97. Acuity: Illuminating the In-Store Experience. Indoor Positioning Services Using LED Lighting Benefit Shoppers and Retailers, http://www.acuitybrands.com/-/media/Files/Acuity/Solutions/Services/Indoor%20Positioning%20White%20Papers/indoor%20positioning%20white%20paperrevised%20110315%20pdf.pdf (2016)
98. IEEE: Part 15.7: Standard for Short-Range Wireless Optical Communication using Visible Light (2011)
99. Reiners, R., Falkenthal, M., Jugel, D., Zimmermann, A.: Requirements for a Collaborative Formulation Process of Evolutionary Patterns. In: Proceedings of the 18th European Conference on Pattern Languages of Programs (EuroPlop). ACM, New York, NY (2013)
100. Falkenthal, M., Barzen, J., Breitenbücher, U., Brügmann, S., Joos, D., Leymann, F., Wurster, M.: Pattern Research in the Digital Humanities: How Data Mining Techniques Support the Identification of Costume Patterns. In: Proceedings of the 10th Symposium and Summer School On Service-Oriented Computing (SummerSOC 2016). Springer (2016)
101. Eloranta, V.-P., Koskinen, J., Leppänen, M., Reijonen, V.: Designing distributed control systems. A pattern language approach. Wiley, Hoboken, NJ (2014)
102. Qanbari, S., Pezeshki, S., Raisi, R., Mahdizadeh, S., Rahimzadeh, R., Behinaein, N., Mahmoudi, F., Ayoubzadeh, S., Fazlali, P., Roshani, K., et al.: IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications. In: Proceedings of the First International Conference on Internet-of-Things Design and Implementation (IoTDI), pp. 277–282. IEEE (2016)
103. Eclipse Foundation: Kura Documentation - Introduction, http://eclipse.github.io/kura/intro/intro.html

All links have been last accessed on 13.06.2018