# Chapter 1

# Introduction

*Just looked like a "thing", didn't it? People don't question "things". They just say, "oo... it's a thing."*

The Doctor, Doctor Who

In the past years we witnessed a development leading from the data-collecting *Wireless Sensor Networks* (WSNs) towards the *Internet of Things* (IoT). This term, which is believed to be first publicly introduced in a presentation of Kevin Ashton in 1991 [Ash], nowadays stands for different ideas, depending on the source of its definition. Ashton's definition strives to connect real-world objects to the Internet, much in the spirit of earlier term *Ubiquituos Computing* [Wei91]. The "things" in his definition do not refer to embedded computing devices, but to actual real-world objects they observe and control. Another accepted definition [HTM+14] stresses a new level of connectivity between devices, exceeding that of existing *Machine-to-Machine* (M2M) communication approaches and allowing machines and humans to exchange data. Both definitions of the term are inherently related to embedded and mobile technologies and devices such as wireless sensor nodes, smart phones, smart watches, wearables, smart TVs, home appliances and a wide array of new applications and devices focusing on observation of and interaction with the physical world. Thomas Liesner quotes in his article "The Internet of Things – next Revolution or Smooth Transition?" [Lie] data from Gartner, Inc. [Gar] and others predicting a rapid growth of in the number of connected devices of this type (see Figure 1.1).

We see a large variety of examples of (consumer-targeted) IoT devices presented in the last years fitting Ashtons definition, some shown in Figure 1.2: The Netamto Weather Station allows to measure several environmental properties such as air temperature, humidity, $CO_2$ level and noise [PD]. The Belkin WeMo Insight Switch allows to remote-control the power supply to power-plugged devices and assess their energy consumption [Bel]. The Jawbone UP24 wrist band can measure

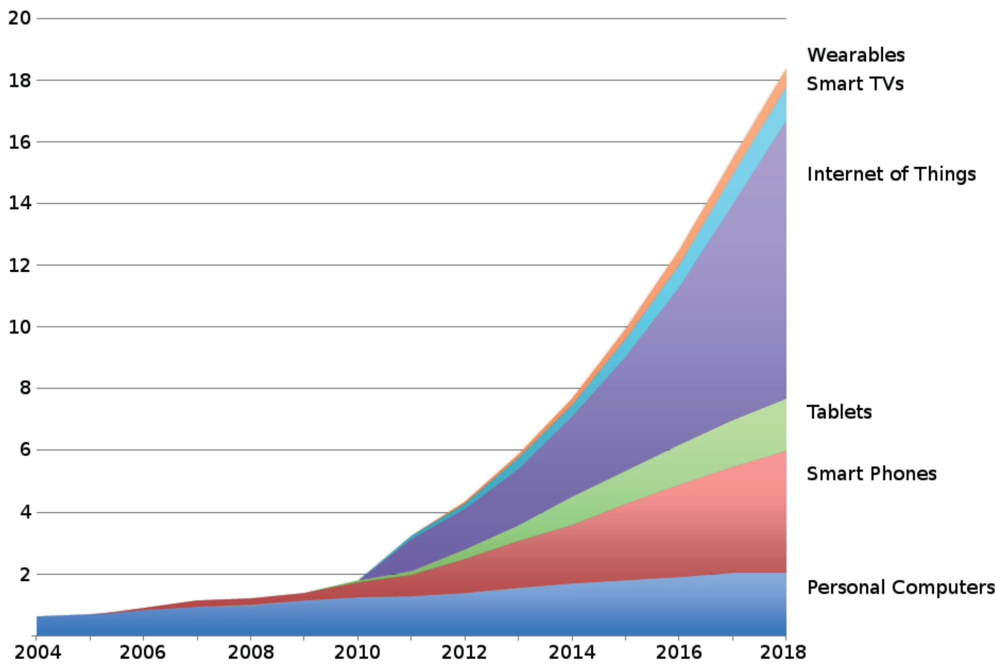


Image: *http://www.kaizen-factory.com/2013/10/26/the-internet-of-things-next-revolution-or-smooth-transition/*
*(labels edited for readability)*

Figure 1.1: Connected IoT devices (in billion). Values from 2013 onwards are predictions.

personal activity during night- and daytime and monitor sleeping behavior as well as aid in working out [Jaw]. All these devices come bundled with smart phone- or cloud-based applications that make them more useful to the end user: The Netatmo Weather Station application uses the measured values to provide warnings about air quality and a sophisticated user interface that allows to analyze weather history and forecast future weather conditions. The WeMo Insight application allows to track energy consumption of a connected appliance and can, for example, estimate the monthly energy cost for that appliance. Additionally, devices can be power-cycled remotely via smart phone or automatically by the time of day. The UP24 smart phone application is specifically designed to set personal goals for improving habits in terms of sleeping, diet or training.

Each of these devices are sophisticated, useful products on their own, some of which connect to the Internet to increase the value of the application by incorporating additional data sources or making use of centralized cloud storage. Vendors of consumer articles seem to agree on the use of established protocols like 3G, Bluetooth, Zigbee and WLAN on the low layers and thus usually integrate easily into existing home networks. To that end, there is a general trend of the usage of standardized communication protocols established in practice. However, on higher layers the formats of data exchange are usually vendor-specific, not always publicly documented and targeted at machine-to-user communication.

Image: *http://netatmo.com* Image: *https://belkin.com* Image: *https://jawbone.com*

Figure 1.2: Examples of current IoT products. From left to right: The Netatmo Weather Station, the Belkin WeMo Insight Switch, the Jawbone UP24 wrist band.

At the same time, there is a strong and growing interest for integration of these diverse IoT components with each other and with services and applications on the Web on a data layer. This trend is witnessed by recent practical integration approaches (Figure 1.3): The web service *“If this then that”* (IFTTT) [IFT], provides its users with the means to create simple rules that connect triggers from different *channels* to actions on other channels. At the time of this writing, IFTTT supports 104 distinct channels, including popular web services such as Facebook [Fac], YouTube [You] or GitHub [Git] and IoT appliances such as the Belkin WeMo Insight Switch, the Jawbone UP24 wrist band and Netatmo Weather Station introduced above. Example rules could switch off a WeMo plug whenever the UP24 detects sleep, or post a Facebook update when the user achieves its step goal as measured by UP24. Another practical integration approach is presented by the *Ninja Sphere* [Nin]: The Ninja Sphere can track objects and users and provide its users with location-based services. It allows, for example, to inform the user about a left-on heating device and give him/her (via smart phone) the possibility to turn it off remotely, or for a user to control the light in the room he/she currently is in via his/her smart watch. The Ninja Sphere supports a variety of IoT devices such as the Belkin WeMo, the Phillips Hue [Kon] and the Pebble smart watch [Peb].

In addition to this movement of integrating existing IoT appliances with each other we also witness a development towards more user-controlled, multi-purpose IoT sensing devices, targeted at user-customized installation, tinkering and even user application development. The *Ninja Blocks* [Nin] system provides a variety of wireless sensors that can sense temperature, motion or button presses. The vendor specifically encourages tinkering with the devices and installation of user-provided embedded or centralized applications and provides full compatibility with the pop-

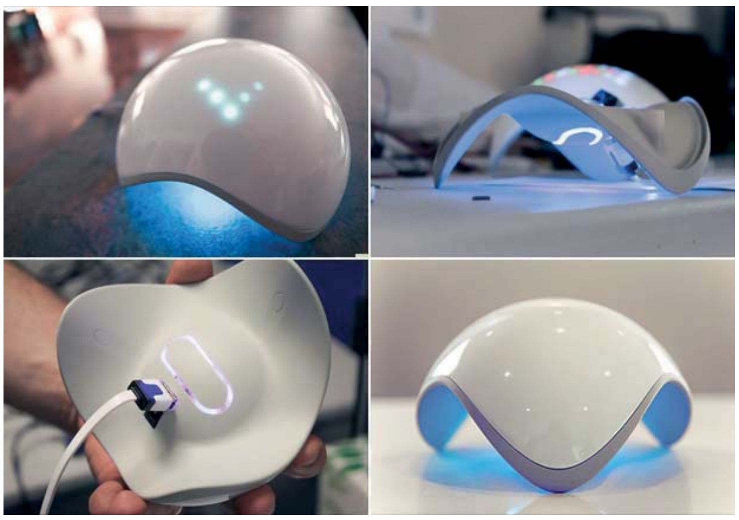
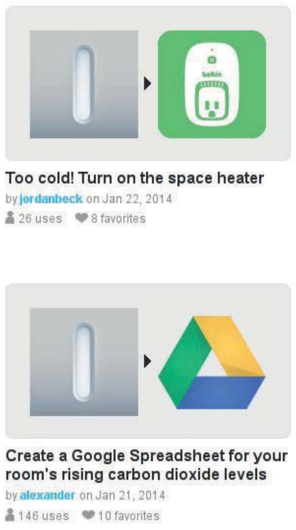


Image:
*http://www.kickstarter.com/projects/ninja/ninja-sphere-next-generation-control-of-your-envir*



Too cold! Turn on the space heater
by jordanbeck on Jan 22, 2014
26 uses 8 favorites

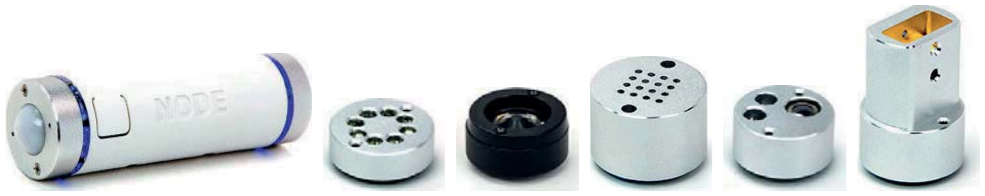
Create a Google Spreadsheet for your room's rising carbon dioxide levels
by alexander on Jan 21, 2014
146 uses 10 favorites

Image:
*https://ifttt.com/netatmo*

Figure 1.3: Integration of IoT components. Left: Prototype of the "Ninja Sphere". Right: Example IFTTT rules involving the Netatmo Weather Station.

ular Arduino [Ard] platform. The *VARIABLE NODE+* [Var] provides a wireless sensor platform with exchangeable sensor- and actuator plugs and can be used for a variety of applications such as reading bar codes, measuring light, air- or surface temperature and air composition, shown in Figure 1.4. Additionally, the vendor provides an *Input / Output* plug, that provides controllable I/O pins for custom user applications and extension. These approaches of empowering the end users in using their IoT devices and building custom applications pose new demands on integrability of devices and applications: Whereas in the formerly presented approaches devices and software applications were shipped as an atomic bundle, for this type of devices the application it not known a priori. What data will the user or application access? Where should it be communicated? What other devices is the device going to interact with? These questions stress the second meaning of the term "Internet of Things" we introduced: A high degree of connectivity between different devices and applications. The idea of carefully integrating the multitude of upcoming IoT appliances into platforms such as IFTTT or the Ninja Sphere one by one however does not scale in the long term: Code has to be added specific for the API of each new appliance and user-defined applications (such as IFTTT rules) are not reusable in that they still refer to specific products and vendors.

Consider a simple, user-defined home automation application: A contact sensor observes the open–closed state of a window. Whenever the window is being opened, the smart power plug should cut the power to the air conditioning device to avoid a waste of energy. A possible, IFTTT-like formulation could be *"IF sensors 42*



Images: *https://variableinc.com*

Figure 1.4: VARIABLE NODE+ with some of the available exchangeable sensor plugs.

*measures contact loss THEN turn off power plug 23"*. Ideally, we would rather like to express our applications in the spirit of *"IF a window is open in any room $x THEN turn off all AC devices in room $x"*. Note how much more generic and reusable this second formulation is: It can work with any number of window sensors and any number of AC devices and matches them to the same room. More important, it does not include (a) any implicit assumptions about where sensors are installed and what they are observing (b) access to the raw sensor information ("contact loss"). This level of abstraction, considered at a general level, requires the following preconditions:

1. A description of the sensor and actuator devices: What are they observing? What does that mean for the application? Where are they located? How can they interact with the real world?

2. An infrastructure that allows to obtain the information asked for in the query: As the second query does not address sensors explicitly, other means of accessing this information need to be provided.

We pose the following questions: How can we express—on the data layer—such information in a way that any thinkable future application scenario remains possible, yet still have it be descriptive enough to enable reuse of knowledge across different applications and devices? The simple query shown above only accesses information about sensors, how can we incorporate knowledge from remote databases or data publicly available on the Web such as company time tables, large-scale location information or weather services? What is the cost of such a knowledge representation, and where can this knowledge reside? If queries do not address specific devices anymore, how can we still make this knowledge accessible?

A lot of research is currently in progress that aims to address these and similar questions from different perspectives: The European Union project *Internet of Things Architecture* (IoT-A) [BBB$^+$12] was started in 2010 and concluded in November 2013. IoT-A's mission was to create architectural foundations for the Internet of Things for addressing questions like integration, self configuration and orchestration. Figure 1.5 gives an overview over a fraction of their outcomes,

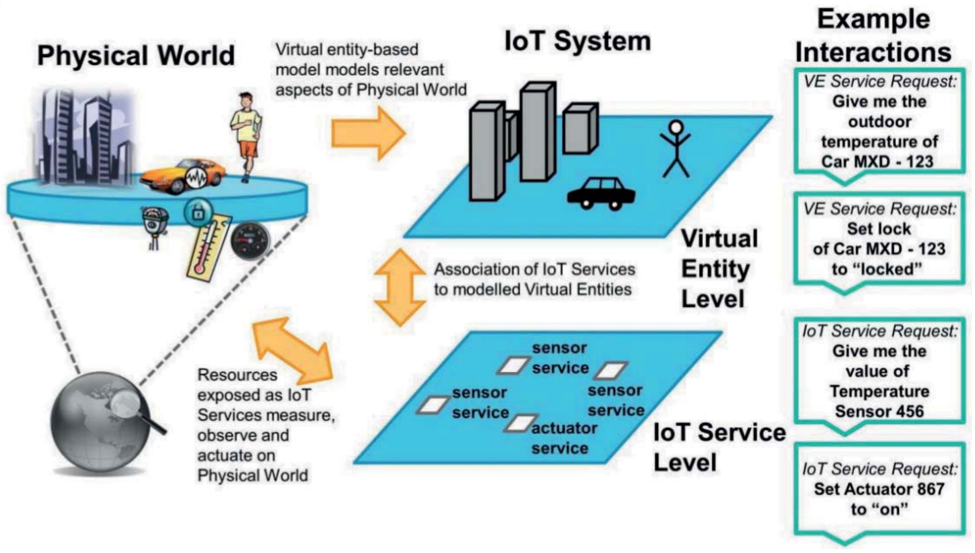

Physical World
Virtual entity-based model models relevant aspects of Physical World
IoT System
Example Interactions
VE Service Request: Give me the outdoor temperature of Car MXD - 123
VE Service Request: Set lock of Car MXD - 123 to "locked"
Virtual Entity Level
Association of IoT Services to modelled Virtual Entities
IoT Service Request: Give me the value of Temperature Sensor 456
IoT Service Request: Set Actuator 867 to "on"
sensor service
sensor service
sensor service
actuator service
IoT Service Level
Resources exposed as IoT Services measure, observe and actuate on Physical World

Image: [BBB$^+$12]

Figure 1.5: IoT-A deliverable D1.5: IoT service and Virtual Entity abstraction levels.

namely the conceptual integration of the physical world, the several IoT services and the IoT system with its *Virtual Entities*—virtual representations of observed or controlled real-world objects. The EU project *iCore* [iCo] implements a realization of the IoT-A architecture, abstracting over device heterogeneity and individual devices towards a representation of the observed real-world objects.

In parallel, the European Union funded the project *Semantic Service Provisioning for the Internet of Things using Future Internet Research by Experimentation* (SPITFIRE) [SPIa]. SPITFIRE addresses this set of questions by consequently connecting the IoT to the *Semantic Web* [BLHL01] and emphasizing the elevation of embedded devices to self-describing first-class citizens of the future Internet of Things.

**Thesis Outline.** This thesis addresses the question of how Semantic Web technologies can be used to describe IoT devices efficiently to provide a better integration. In Chapter 2 we analyze existing standards for describing embedded devices with respect to their expressiveness and universality. We introduce the Semantic Web and show how the description methodology it offers can improve on these existing standards. Chapter 3 discusses the storage of semantic descriptions directly on the resource-constrained embedded devices with a focus on determining the overhead, especially in terms of energy consumption of such a data model. In Chapter 4 we address the question of how we can further abstract from the description of embedded devices discussed so far. In particular, we raise the question of how these device descriptions can be converted into knowledge about the real-world objects an application programmer is interested in. This abstraction, as we demonstrate, can then be utilized for semantically-informed energy conservation schemes and thus compensate for some of the overhead introduced for the more verbose semantic descriptions. Finally, Chapter 5 discusses how to approach querying of the descriptions on the devices when it is not feasible to stream all data proactively out of the embedded network.

Throughout these chapters we focus on the use case of a home application scenario in which it is useful for devices to be able to communicate directly with each other without the necessity for a centralized service. However, the ideas and techniques we discuss are of much broader scope and are applicable in many IoT scenarios.

**Collaborations.** The papers on which this work builds are the result of collaborations with other people. **Alexander Kröller** contributed many ideas and thoughts to almost all aspects of this work.

The implementations presented in this work found largely on the embedded algorithms library *Wiselib* [Bau12], formerly maintained by **Tobias Baumgartner** and developed in the course of the EU-project WISEBED [Sev08].

Many people contributed implementations in form of Wiselib components or external software that interfaces with our reference implementation in different ways. These include many partners of the SPITFIRE project, the Wiselib community as well as various students.

The work presented in Chapter 3 contains ideas and implementations that are the result of a close collaboration with **Max Pagel**. **Ioannis Chatzigiannakis** and **Dimitrios Amaxilatis** contributed ideas to the work in Chapter 4 and also several Wiselib components used by our implementations, such as the advanced neighborhood discovery mechanisms used by the implementations presented in chapters 4 and 5. Chapter 5 is the result of a collaboration with **Christian von der Weth**, **Marcel Karnstedt** and **Dennis Boldt** in terms of both ideas and implementations.

# Chapter 2

# Knowledge Integration for Embedded Systems

| | |
|---|---|
| ***Clara:*** | *When you say mobile phone, why do you point at that blue box?* |
| ***The Doctor:*** | *Because it's a surprisingly accurate description!* |
| ***Clara:*** | *Okay. We're finished now.* |

Doctor Who

In this chapter we discuss our vision on how devices in the Internet of Things (IoT) can be integrated with each other and the existing Internet. This integration poses a number of challenges, starting with heterogeneity of embedded devices hardware. As devices are deployed for different use cases and produced by different vendors, integration approaches must deal with a variety of hardware architectures. Moreover, devices may also differ drastically in terms of capabilities and resources provided to an application. Due to differing deployment demands, the problem of integration on a protocol level has to be considered. Standardization bodies such as the *Internet Engineering Task Force* (IETF) [Int] and the *World Wide Web Consortium* (W3C) [Wor] provide widely accepted protocol standards, however in some deployments other means of communication might be necessary. Thus, an IoT infrastructure has to provide the possibility of an exchangeable protocol stack. Finally, in order to exchange knowledge between devices and applications in a universal and future-proof way and to allow true plug-and-play behavior, we identify the necessity for a universal data representation format. This format should be applicable to any knowledge domain so the exchange of information of future applications is not constrained to a limited vocabulary.

We open the chapter with a brief description of a use case scenario that provides a conceptual environment for later examples in this and the following chapters.

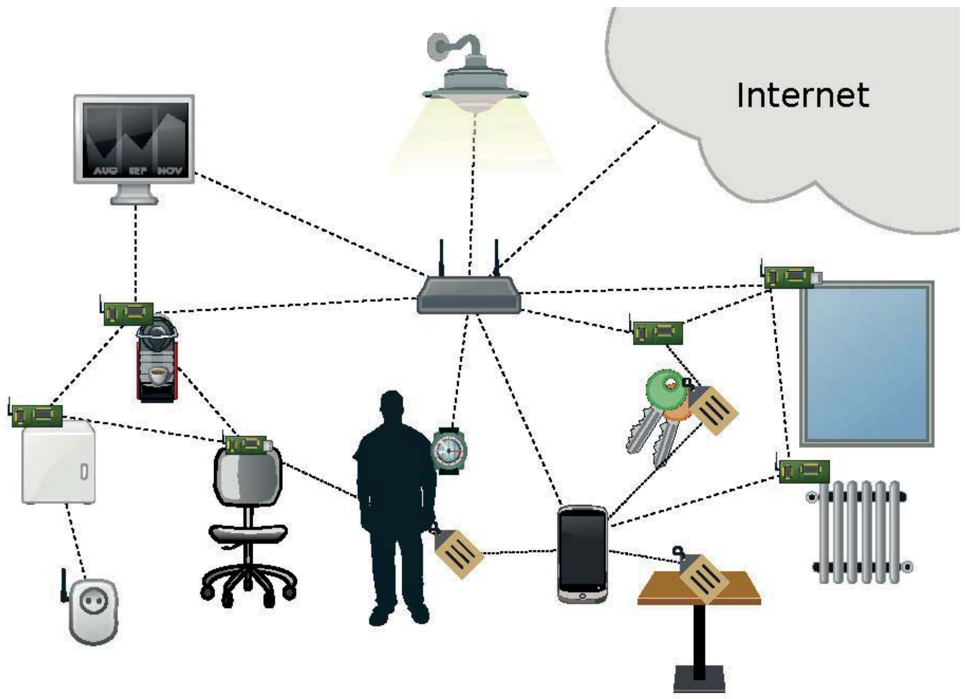


Figure 2.1: Exemplary home automation scenario. A lightweight wireless router connects some of the IoT devices to each other and with the Internet. Devices may also interact directly with each other.

We then discuss the integration of the Internet of Things and the current Internet through different abstraction layers, starting with hardware heterogeneity in Section 2.2, then working our way up to protocol-level integration in Section 2.3. Finally, we discuss the integration of the data or *knowledge* exchanged via these protocols in greater depth: Section 2.4 identifies what such a knowledge representation must accomplish to achieve universality. Section 2.5 then examines existing approaches and to what extend they comply with these demands. In sections 2.6 and 2.7 we introduce the *Semantic Web* and discuss its current connection to the IoT and to what extend it can address the demands for knowledge integration we identified.

## 2.1 Use Case Scenario

In order to see what demands for universal communication and representation of knowledge arise in Internet of Things applications, we consider the use case of building- and home automation systems. Although we draw exemplary considerations mostly from this scenario, our emphasis on universality ensure our considerations apply equal to other IoT scenarios such as Smart Cities, factory monitoring and others.

*Building automation* refers to the automation of a wide array of building-related control tasks such as climate control, presence detection and room scheduling, profiling the operating state and energy consumption of appliances, lighting and security [KNSN05]. *Home automation* describes building automation in the context of a residential home. This includes the control and/or monitoring of household devices (such as washing machines), temperature control (heating, ventilation, air condition), lighting, security (locks and alarm systems) and entertainment media (such as television). Home automation systems are usually administered by the inhabitants with little or no professional assistance. This situation naturally demands for plug-and-play solutions. That is, it should be easy for a user to install and set up the system and extend it by new, off-the-shelf devices—some examples of these devices were introduced in Chapter 1. Current building- and home automation approaches provide sophisticated features and extensions, but rarely a wide interoperability with products from other vendors or different types of systems, although that would be beneficial for the user.

Although in certain settings it is possible to connect all automation devices to a power line, in other situations this might not be practical because devices are to be attached to unfavorable locations (such as high windows), have to be relocatable on the fly or are attached to a movable object such as a desk chair. In such a case the embedded devices have to rely on battery power and energy conservation is essential. Thus, energy costly radio communication has to be avoided to maximize the devices life time.

Despite from being a fast growing application market, home automation is a particular challenging scenario for IoT integration for the following reasons:

- It is desirable to allow plug-and-play style installation of the devices by untrained users; that is, deployment of devices must not involve a complex configuration step.

- Moreover, devices cannot be expected to be installed statically, but might be occasionally repositioned. Due to urban WLAN and running appliances the network topology is subject to change.

- A user of a home automation system might be interested in successively expanding his/her system by new off-the-shelf components. These must be expected to be produced by different vendors with different applications in mind but should be integrated and form a joint IoT network.

Figure 2.1 sketches an example installation in which a number of IoT devices provide different services and allow for different interactions. A lightweight wireless router such as commonly present in most homes can serve as gateway to the Internet.

## 2.2 Hardware Abstraction

The powerful devices that we currently see in the Internet have gained their popularity—among other reasons—because they are compatible in the sense that it is possible to develop applications that are usable on a large number of those devices. This universality is made possible by integration and abstraction on multiple layers: Modern operating systems can be compiled for different CPU architectures and thus provide uniform access to hardware for many platforms. This approach has some limitations: Operating systems cannot be ported to all hardware, for either legal or marketing reasons and because the design goals of the hardware might be incompatible with the design goals of the operating system. To nevertheless provide functionality such as establishment of a secure connection or processing of certain file formats to an application developer in a uniform way, portable libraries and applications provide the same functionality on different operating systems. Probably most important for the extensibility of the Internet is the notion of standardized protocols for communication. These agreements allows the development of new services (such as databases or web services) and applications (such as browsers or email clients) in a reusable way and makes it possible to provide components that are meaningful independently of the implementation of the components they interact with.

For the IoT this integration on multiple levels provides new challenges: Hardware is much more diverse than in the common Internet, not only in terms of different processor architectures in use but also in terms of available resources and peripherals. In Chapter 1, we have already introduced some of the IoT devices currently available at the consumer market. As these devices fulfill very diverse tasks and are produced by different vendors, we can inadvertently observe a large variety in terms of employed hardware. Table 2.1 illustrates the CPU architectures and means of communication found in some of these systems, ranging from very constrained devices with few kilobytes of RAM to full-blown 32-bit machines with megabytes or even gigabytes of RAM, capable of running desktop-class operating systems such as derivatives of Linux [Linb].

This variety makes it impossible to provide a single operating system that can support all these devices fully an efficiently. Moreover, it makes it nontrivial to provide libraries which are usable on all these systems. Consider an algorithm for compression of data: Where the necessary system resources are available, such an algorithm might profit heavily from storing the document to be compressed in RAM completely so that all redundancies in the document can be identified and exploited for compression. On a less powerful machine however, that might not be possible and the algorithm can only consider a file in a streaming fashion, processing each consecutive chunk of data individually. A similar observation holds for communication protocols: Due to the large variety in terms of available

| Product | CPU |  |  | RAM | Communication |
|---|---|---|---|---|---|
|  | Type | Bits | MHz | kB | Peripherals |
| VARIABLE NODE [Var] | AVR | 8 | 16 | 8 | BT, BTLE |
| "Girogo" bank card [Inf], Fig. 2.2 | 80251 | 16 | 33 | 8 | NFC |
| TMote Sky Sensor Node [Mot] | MSP430 | 16 | 8 | 10 | IEEE 802.15.4 |
| ioBridge iota [ioB] | PIC | 16 | 8 | 16 | Eth, or WiFi |
| Pebble Smart Watch [Peb] | ARM | 32 | 120 | 128 | BT, BTLE |
| Linksys WRT54G v8.2 [Lina] | MIPS | 32 | 240 | 2,048 | Eth. x5, WiFi (AP) |
| Ninja Blocks Base Station [Nin] | ARM | 32 | 1,024 | 524,288 | Eth. |
| Wireless shield | AVR | 8 | 20 | 2 | 433MHz Radio |

Table 2.1: IoT hardware configurations. Communication peripherals neglect USB/UART (which is present in most considered systems primarily for flashing, debugging or charging of the device). *BT*: Bluetooth [Blu], *BTLE*: Bluetooth Low Energy, *NFC*: Near-Field Communication [NFC], *Eth.*: Ethernet [IEE12a].

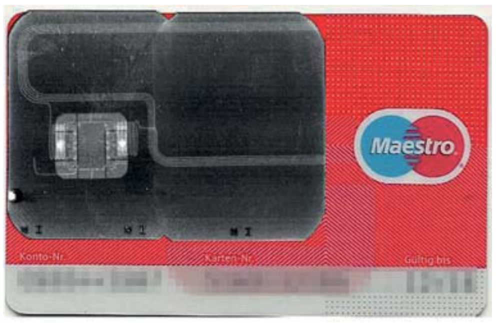


Image: *http://commons.wikimedia.org/wiki/File:Überlagert.jpg*

Figure 2.2: The "Girogo" bank card. A partial x-ray scan reveals chip and NFC antenna.

hardware there cannot possibly be a single communication protocol supported by all devices in all deployments. Still, it is desirable to not have to rewrite applications for each protocol in use; protocols should be exchangeable depending on the target platform and network configuration.

The operating systems Contiki [DGV04] and TinyOS [LMP$^+$05] have been ported to a variety of resource constrained platforms commonly used in the context of Wireless Sensor Networks (WSN) and thus provide means of abstraction that allow a development of code for all these platforms with a single code base. Both systems focus on constrained embedded devices without hardware support for dynamic memory allocation, e.g., using a *Memory Management Unit* (MMU). Unfortunately however, they offer only limited scalability in terms of available features of the platform such as using dynamic allocation mechanisms where they are available or easy ways to exchange communication primitives or data structures within an algorithm to adapt it to a different environment without changing the algorithms code.

### 2.2.1 The Wiselib

The Wiselib [Bau12] is an open source, modular algorithms library and abstraction layer for embedded devices. The Wiselib is written completely in C++ and freely available under the Lesser GNU Public License (LGPL) [Wisb]. It was developed during the WISEBED [CKM$^+$09, WISa] project which started in 2008. Since then it received continued development from different EU projects such as WISEBED, FRONTS [FRO] and SPITFIRE [SPIa] and participated in Google Summer of Code [Goob] in the years 2012, 2013 and 2014. On the lowest layer, the Wiselib offers interfaces to a variety of underlying platforms such as Contiki [DGV04], TinyOS [LMP$^+$05], Arduino [Ard], OpenWRT [Opeb] the Shawn network simulator [FKFP07] and many more. On top of that, the Wiselib offers a large fund of data structures and utility functionality and, on the highest abstraction tier—algorithms from a variety of categories including localization, routing and graph theoretic algorithms as well as a number of communication protocols.

The Wiselib relies heavily on C++ templates for abstraction in the same spirit as the C++ *Standards Template Library* (STL) [SL95], *Boost* [DAR] and the *Computational Geometry Algorithms Library* (CGAL) [FP09]. Similar to these libraries, the Wiselib provides abstraction of components by the means of *concepts*. Concepts are pieces of documentation that specify the behavior of a group of classes, such as defining which methods, members and type definitions every class implementing the concept must provide. It is thus similar to an interface definition, however is not enforced by the compiler but rather documentation for the (human) programmer. Classes adhering to one or more concepts are called *models*, we also say a class *models* a certain concept. This compile-time abstraction mechanism avoids the need for runtime dispatch, as dispatch is handled completely by static binding.

The Wiselib provides a modular architecture in several regards: Modules can be substituted with alternative implementations by the change of a template parameter. This flexibility allows not only to easily exchange used data structures and, e.g., alternate between the use of static or dynamic memory utilization but also to substitute algorithm implementations, substitute the simple radio communication with a routing algorithm or transparently en- and decode messages before sending or storing them. The Wiselib carries this idea down to the operating system abstraction itself such that the OS is—like any other module—a template parameter to the application and its used algorithms. This way, by changing a template parameter and recompiling an application can not only be altered to work with different data structures and sub-algorithms, but can effectively be ported to a different platform.

Whereas some embedded platforms such as *Coalesenses iSense* [BP07] support C++ natively, many platforms target at the use of C or (in the case of TinyOS) the C extension *nesC* [GLVB$^+$03] instead. As these platforms commonly rely on the *GNU Compiler Collection* [Fre], supporting compilation of C++ code is straight forward. However, the `libstdc++` library which provides support for features such as dynamic memory allocation, run-time type information (RTTI), exceptions and virtual inheritance is not available on these systems. Also these features would imply considerable overhead for some of these resource-constrained platforms in terms of runtime and memory consumption, so their implementation for all platforms is not useful. Thus, the Wiselib only uses an "extremely portable" subset of the C++ language, avoiding the features mentioned above and rather implement abstraction by the means of C++ template mechanisms at compile time. In contrast to most other abstraction approaches, this allows for full compile time optimization (such as method inlining) and avoids the call overhead implied by runtime dispatch. As the Wiselib is included as a set of C++ header files, it does however not restrict the application programmer from using whatever C++ features are made available to him/her by the target platform.

Among others, the Wiselib features implementations for 6LoWPAN [KMS07] and CoAP [SHBF11] (both discussed in greater detail below) and thus allows communication on a protocol level using open standards.

The Wiselib serves as foundation of our reference implementations which we present later in this work. In the course of the development of these implementations, we contributed several extensions to the Wiselib that are of general use to the Wiselib community:

**Block Memory Interface.** We contributed several modules related to block memory devices, specifically *Secure Diginal Cards* (SD cards) [SD ]. SD cards provide an inexpensive and exchangeable mass storage medium, can be accessed directly via the SPI bus and are thus available on many embedded devices. To ease developing and testing of block memory oriented algorithms and data structures, we provided two block memory implementations that are usable on PC (and possibly other) platforms even without access to physical block devices: The *RAM Block Memory* module provides a volatile block memory in RAM that can be easily inspected during runtime. The *File Block Memory* in contrast provides access to any file in a Linux file system in a block-oriented fashion. This implementation allows to either work on image files—which may be easily inspected, exchanged and manipulated with common operating system tools—or directly on device files, providing access to the blocks of a hard drive partition, USB stick or any other Unix block device. SD Card interfaces for Arduino and Coalesenses iSense have been provided.

For platforms on which both block memory and sufficient RAM is available we provide the *Cached Block Memory* module that provides transparent caching of