Building as a Service - BaaS

Deliverable

D02 - State of the Art

Editor:

Ingo Lück, Materna

ITEA 2 Project 12011

October 18th, 2016

Document properties

Distribution	Confidential
Version	Version 2
Editor	Ingo Lück, Materna
Authors/ Contributors	Özgür Devrim Orman, Bor Software Miguel Díez, Everis Alejandro López, Everis Borja Ortiz, Everis Andrés Páez, Everis Selahattin Gökçeli, Istanbul Technical University, Defne Güneş Karabulut Kurt, Istanbul Technical University, Defne Michael Christmann, Kieback&Peter Jürgen Maaß, Kieback&Peter Peter Michael Schmidt, Kieback&Peter Emrah Açıkalın, KocSistem Ozan Gürsoy, KocSistem Christoph Fiehe, Materna Ingo Lück, Materna Jannis Müthing, Materna Niklas Röder, Materna Daniel Gastón Iglesias, Prodevelop Christophe Joubert, Prodevelop Vicente Sanjaime, Prodevelop Darko Anicic, Siemens Michael Bahr, Siemens Nicolas Gümbel, Siemens Sebastian Käbisch, Siemens Jelena Mitic, Siemens Christoph Niedermeier, Siemens Norbert Vicari, Siemens Malte Burkert, Technische Universität Dortmund Oliver Dohndorf, Technische Universität Dortmund

	Heiko Krumm, Technische Universität Dortmund Benjamin Hof, Technische Universität München Marc-Oliver Pahl, Technische Universität München Svenja Borchers, TWT Andreas Müller, TWT Martin Neubauer, TWT Björn Butzin, Universität Rostock Frank Golatowski, Universität Rostock
Pages	177

Changes in this document compared to version 1:

Type of change	Location
added	8.3 Bluetooth
added	8.4 RFID Reader and Tags
added	8.7.5.10 RFM22 Module
added	8.7.5.11 Arduino Uno
added	8.7.5.12 Intel Galileo
added	8.7.5.13 Temperature Sensor
added	8.7.5.14 HC-05 Bluetooth RF Transceiver Module
updated	8.7.2 Operating systems
added	8.8 Pixage - Digital Signage
added	8.9 Keycloak - IDM
added	6.2.4 The Smart Appliances REFerence (SAREF) Ontology
added	6.2.5 QUDT - Quantities, Units, Dimensions and Data Types Ontologies
added	6.2.6 ifcOWL - Industry Foundation Classes Ontology
added	6.3 W3C Web of Things and Thing Description
updated	3.10 Constrained Application Protocol (CoAP)

Abstract

This document describes the State of the Art analysis of technologies, modeling- and architecture-approaches, which will be relevant for the BaaS project. Since the project schedule plans one iteration of the state of the art analysis, the deliverable D02 appears in two versions. This is the first version. It identifies, collects and analyses approaches which potentially may be of relevance for the project since they may deliver conception and solution element input to the development of the BaaS architecture, model, platform, building services and demonstrator applications.

After an introduction to this document an overview over the domain of building automation describes basic concepts and current trends. It is followed by chapters dealing with base technologies, building automation data models and middleware architecture. Since the BaaS project intends to follow a model based approach this documents proceeds with two chapters about semantic modeling and domain specific modeling. Finally supporting technologies considered to be relevant for BaaS are introduced.

Table of contents

1. Introduction

The BaaS project targets the need for comprehensive and open cross-domain management and control services in today's buildings and focusses on the four technical objectives of conceptualizing and developing

- a ***flexible open building service platform*** facilitating the generation and deployment of value added building services at a considerably lower cost compared to the state of the art;
- a ***BaaS data model*** providing additional meta-information to simplify the engineering of value added services and applications for the BaaS system and the integration of legacy systems;
- **model-based mechanisms** for analysis, aggregation and transformation of data according to the meta-information provided in the BaaS data model;
- ***methods for the integration of existing and novel sources of information*** to create a "building information sphere" considering all stakeholders of the building.

These objectives imply that the distributed systems paradigm of the Service-Oriented Architecture (SOA) shall be followed up in the project.

Moreover, a series of challenging properties of complex distributed systems like flexibility, dynamic adaptability and extendibility are addressed by the objectives directly. Others, like security, privacy, dependability, safety, robustness, self-healing/self-management and real-time capabilities follow from the requirements of the application domain.

The objective of low implementation and deployment costs pleads for the application of enhanced software engineering approaches. Particularly methods of model-driven and model-based software system development are of interest.

Finally one has to consider that the objective of easy integration of existing systems and information sources demands for a thorough analysis of the application domain and a comprehensive research of the techniques, methods, terms, conceptions and models applied there so far in order to prepare the design of compliant solutions.

Thus the project objectives result in a wide range of topics which is reaching from abstract system properties over supporting implementation approaches, as well as engineering, modeling and description techniques to the different technologies and conceptions of the building automation domain.

Against this background, Task 2 of Working Package 2 of the project proposal is devoted to the corresponding state of the art analysis. This document, Deliverable D02, reports the results. Since the project schedule plans one iteration of the state of the art analysis, the deliverable D02 appears in two versions, the first one is due after the second quarter of the first year and the second one is due at the end of the project.

The content of the first version of the state of the art report identifies, collects and analyses approaches which potentially may be of relevance for the project since they may deliver conception and solution element input to the development of the BaaS architecture, model, platform, building services and demonstrator applications.

Later on, the second version will take a closer look on the approaches actually applied.

Particularly in the first version, due to its early deadline, the selection and structuring of topics, analysis work and contributions cannot profit from rich project experiences and thus mainly resorts on the existing expertise of the project partners. Table 1 summarizes the contributing partners and their contributions, as they are reflected in the first version of the document.

Partner	Contributions
everis	8.1 NFC, 8.2 Zigbee, 8.3 Machine to Machine (M2M)
Kieback und Peter	2.1 Introduction, 2.2 System Architecture and Communication Standards, 2.3 Current Trends
Masaryk University	8.5 Embedded Systems
Materna	3.1 Self-Adaptive Systems (3.1.1 - 3.1.3 and 3.1.8- 3.1.10), 3.3 Device SOA
Prodevelop	8.4 Web GIS Services
Siemens	3.5 Secure Authorization using OAuth 2.0, 3.8 RESTful Web Services, 4 Building Automation Data Models, 6.2 Semantic Web, 7 Domain Specific Modeling
TU Dortmund	3.6 Functional Safety and Reliability in Service Systems, 3.7 OSGi
TU München	3.4 Privacy and data security, 5 Middleware Architecture, 6.1 Introduction
TWT	3.1 Self-Adaptive Systems (3.1.4 - 3.1.7), 3.2 Data Mining for Building Automation
University of Rostock	3.9 Efficient XML Interchange (EXI), 3.10 Constrained Application Protocol (CoAP)

Table 1: Contributing partners, fields of expertise and contributions

The document is structured as follows:

- Section 2 provides an overview over the domain of Building Automation, its major requirements and solution approaches.
- Section 3 is devoted to those base technologies which contribute to the provision of challenging required properties and the implementation of solutions.
- Section 4 gives an overview over and a first analysis of appropriate data models in order to prepare the easy integration of existing components and systems.
- Section 5 enters into the consideration of software architecture issues, particularly, a series of middleware approaches for distributed and pervasive systems are discussed.
- Section 6 focusses on the description and modeling of the semantics of services, operations and components.
- Section 7 investigates software engineering, tool support and domain-specific modeling approaches.
- Finally, Section 8 presents communication and information representation technologies, which may be used as base technologies in the project.

2. Building Automation Overview

2.1. Introduction

Building automation technology improves the security and comfort of buildings and helps saving energy. Additionally, it reduces maintenance efforts and the need for manual control. Building automation enables effective supply of large buildings and saves costs at the same time.

For about 100 years now buildings are controlled by building automation systems (BAS). First BAS used pneumatic sensors and controllers to mainly control temperature in the building. Today BAS controls a wealth of different systems and zone controllers are used to control individual rooms separately. Nowadays BAS are found primarily in functional buildings (office buildings, hospitals, industry…) and increasingly used in residential buildings.

In the 1980s pneumatic sensors and controllers were replaced by electric and analog electronic circuits. Large ventilation systems were needed to control the air pressure, the air quality for hygienic or industrial (clean room) reasons, the air temperature and had to ensure certain level of air mass without transgressing speed limits within the air ducts. These multivariate controlling was not fulfilled by analog controllers, which use physical laws to generate a certain output signal from an input signal. Through the 1990s digital controllers came into play and transform the measured data into digital signals, process and output the signal to the actuator. The digital controlling units are called Direct Digital Controller (DDC) and many products of various manufacturers have the abbreviation DDC in their name.

The lack of the standardization for the corresponding digital communications soon led to the manifold of proprietary communication protocols on the market. Thus, interoperability and combination of products from different manufactures were not possible. Therefore, in the 1990s there were moves afoot to open standardized communication protocols like BACnet [1] and LonWorks [2].

In the last two decades evaluation of BAS has benefited heavily from rapid development of computers, communication and information technologies. The evaluation of BAS from pneumatic transmission towards IT standardizing information models is shown in the Figure 1.

Building automation relieves the user of controlling the room. When a room is empty lights are not needed anymore and heating or cooling can be reduced. A user can adjust the settings manually (by turning off the lights and setting the thermostat) or the technique takes over. For example: some room controllers enable occupancy-based heating control. Additionally, they learn and will preheat the room in advance to the predicted return of the user. Ideally, the user does not even recognize the function. By this way, energy can be saved without losing comfort.

Building automation helps user and owner of buildings to lower the energy cost and manage centrally. In public buildings (for example hospitals) users don’t have a direct interest in energy saving because they don’t participate in the cost. The operator of the building can benefit from building automation and lower the influence of the users. If open windows are detected, the controller can automatically decrease the heating or cooling power where a thermostatic valve would rather increase the heating power.

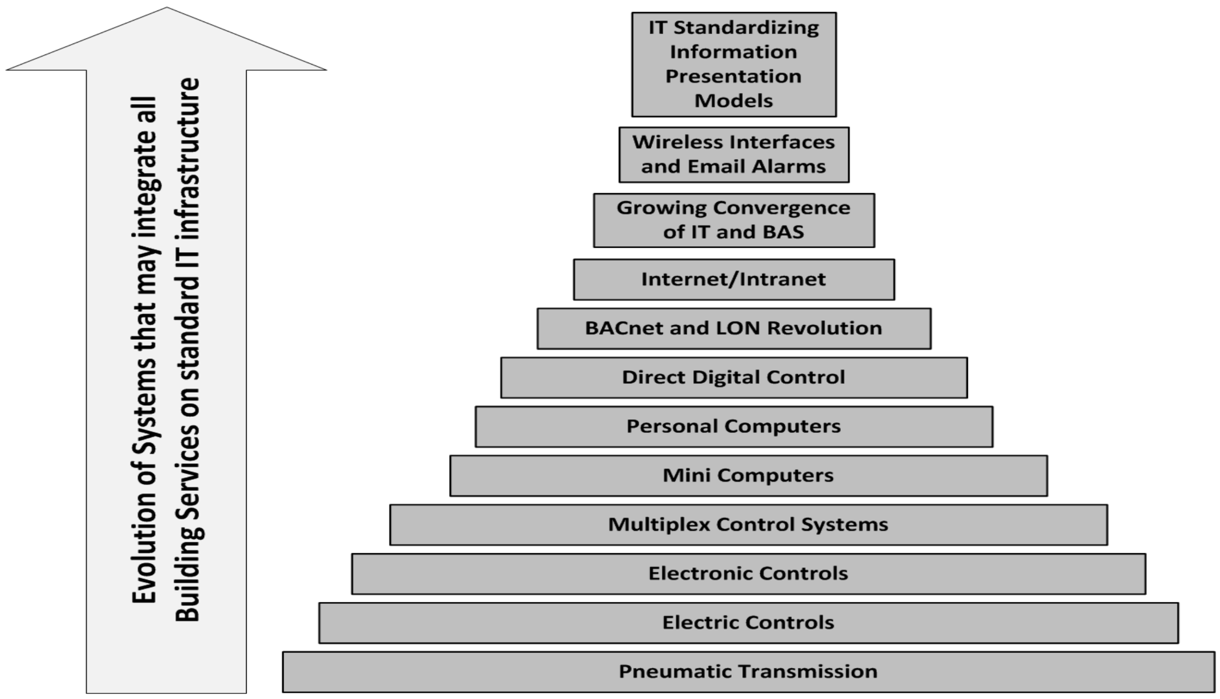

Evolution of Systems that may integrate all Building Services on standard IT infrastructure
IT Standardizing Information Presentation Models
Wireless Interfaces and Email Alarms
Growing Convergence of IT and BAS
Internet/Intranet
BACnet and LON Revolution
Direct Digital Control
Personal Computers
Mini Computers
Multiplex Control Systems
Electronic Controls
Electric Controls
Pneumatic Transmission

Figure 1: Evolution of Building Automation and Control Systems [3]

The safety of people and material is increased with building automation technology. For example when fire detectors report within milliseconds to a central building control system, an immediate evacuation can be started. Another example from residential buildings is a use of motion sensor under the bed to switch on the lamp as soon as you get up. Also, after leaving the house you can check by your smart phone whether critical appliances (cooker, electric iron) are still turned on and if necessary switch them off from the distance.

Building automation systems make buildings intelligent, make everyday life easier and lower the energy costs.

From initially controlling only heating, ventilation, and air-conditioning (HVAC) over the decades BAS have been added a manifold of different service domains listed in the Table 2.

Domain	Typical building services
Climate Control	HVAC, humidity, air quality
Visual Comfort	Artificial lighting, daylighting (motorized blinds/shutters), constant light control
Personal Safety	Fire alarm, gas alarm, emergency sound system, emergency lighting, CCTV (closed circuit television)
Building Security	Intrusion alarm, access control, water leak detection, CCTV, audio surveillance
Transportation	Elevators, escalators, conveyor belts
One-way Audio	Public address/audio distribution and sound reinforcement systems
Energy Management	Energy efficiency, peak avoidance, integration of renewable

Domain	Typical building services
	energy sources (RES)
Supply and disposal	Power distribution, waste management, fresh water/domestic hot water, waste water
Communication and information exchange	IT Networks, PBX (Private Branch Exchange), Intercom, shared WAN access, wireless access (WLAN)
Other special domains	Clock systems, flextime systems, presentation equipment (e.g., video walls), medical gas, pneumatic structure support systems (for airhouses)

Table 2: Domains of building services [4]

Currently different BAS functions are executed mostly in separate technology silos without any or with very little data exchange in between. Today's requirements on buildings to be more energy efficient, cost less to operate, provide a better indoor environment for occupants, and have a smaller environmental footprint make integration between these systems necessary. New regulations for energy performance of buildings [5] add additional optimization goals that can be achieved only with proper mechanisms for comprehensive facility management, fine-grained energy monitoring and benchmarking. Thus, modern BAS ought to provide value added cross-domain services while minimizing engineering efforts.

2.2. System Architecture and Communication Standards

The automation pyramid as drawn in Figure 2 is basic for many BAS. It shows the management level, automation level and field level.

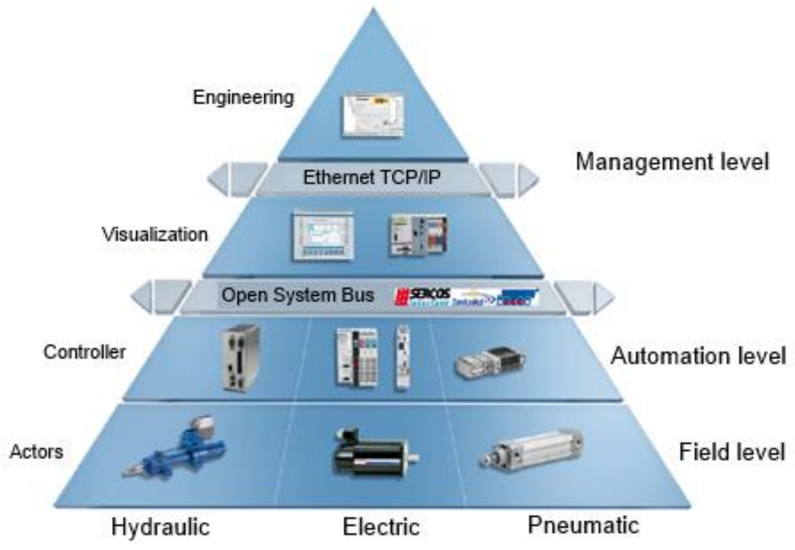
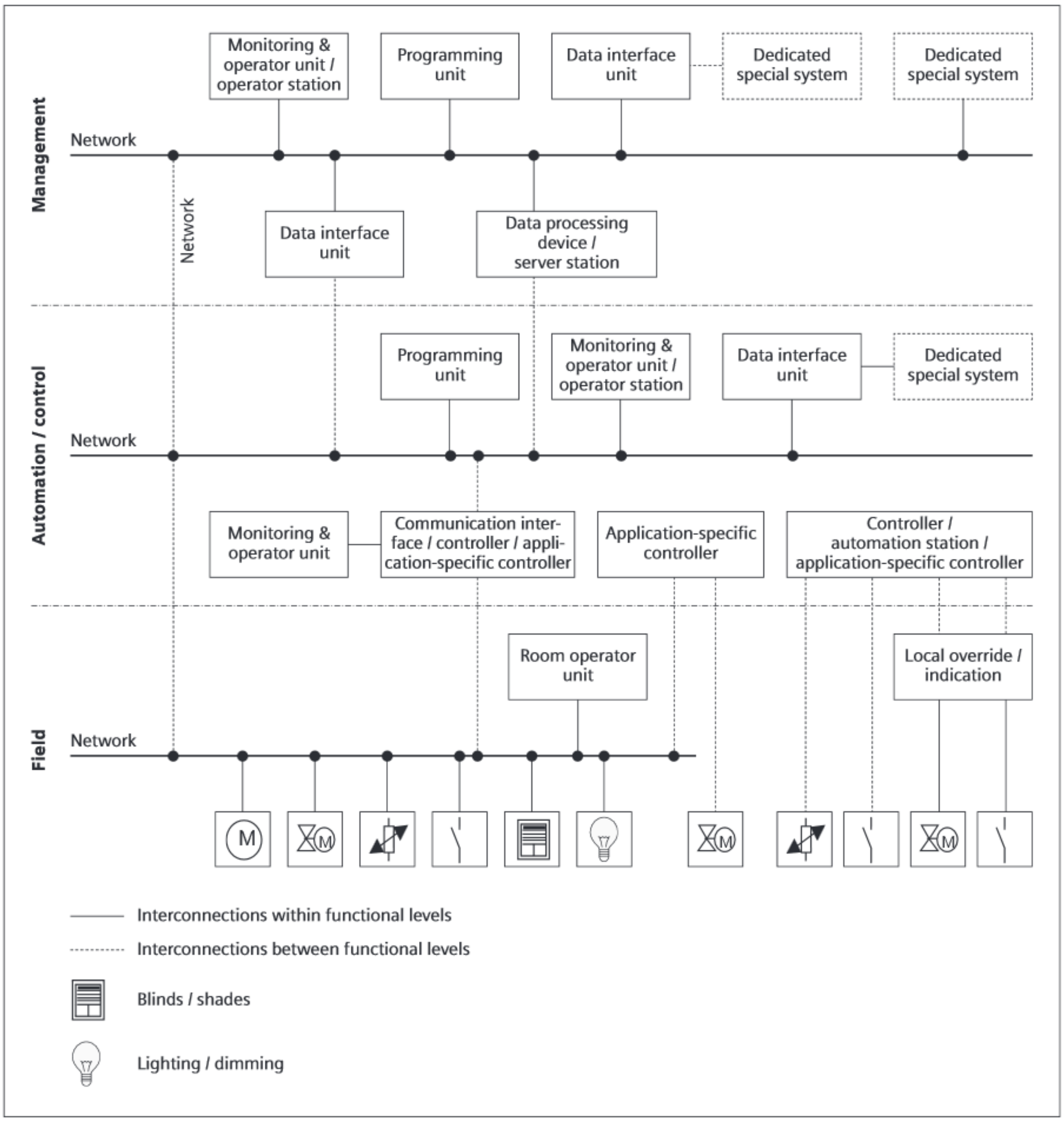


Figure 2: Automation pyramid (Source: Rexroth, Bosch Group)

The automation pyramid shows the management level separated in the engineering and the visualization. Different communication media (and protocols) such as Ethernet (TCP/IP) or various open system buses can be used. The automation and field levels are separated as well in 3 typical crafts: hydraulic (e.g. heating, cooling, ventilation etc.), electric (e.g. lights, blinds etc.) and the lesser used pneumatic.

The standard [6] defines rather a generic system model then precisely defined system architecture in order to ensure flexibility in terms of application. The model presented in Figure 3 can accommodate the different types of Building Automation and Control Systems (BACS) and their interconnections.



Management
Automation / control
Field
Network
Network
Network
Network
Monitoring & operator unit / operator station
Programming unit
Data interface unit
Dedicated special system
Dedicated special system
Data interface unit
Data processing device / server station
Programming unit
Monitoring & operator unit / operator station
Data interface unit
Dedicated special system
Monitoring & operator unit
Communication interface / controller / application-specific controller
Application-specific controller
Controller / automation station / application-specific controller
Room operator unit
Local override / indication
M
Interconnections within functional levels
Interconnections between functional levels
Blinds / shades
Lighting / dimming

Figure 3: Possible interconnections in three-level functional hierarchy

The system topology of building automation systems from different manufacturers may vary but because of the standardized open system buses (e.g. LON, BACnet, KNX) the schemes look very alike. Figure 4 and Figure 5 show the system topology of the companies Kieback&Peter and Siemens, though the field level is not shown.

Different communication standards provided the use of building automation beyond the borders of proprietary systems. A widely spread standard is the BACnet protocol as well as the LON protocol and the (more European) EIB/KNX protocol [7]. These protocols enable a universal application of different technologies on a basis of a bus system. Bus systems reduce the total cable length because not every communication node has to have a separate connection to every other node but they all share one data net. Even the data exchange between the protocols is possible, for example using a LON-BACnet-Gateway.

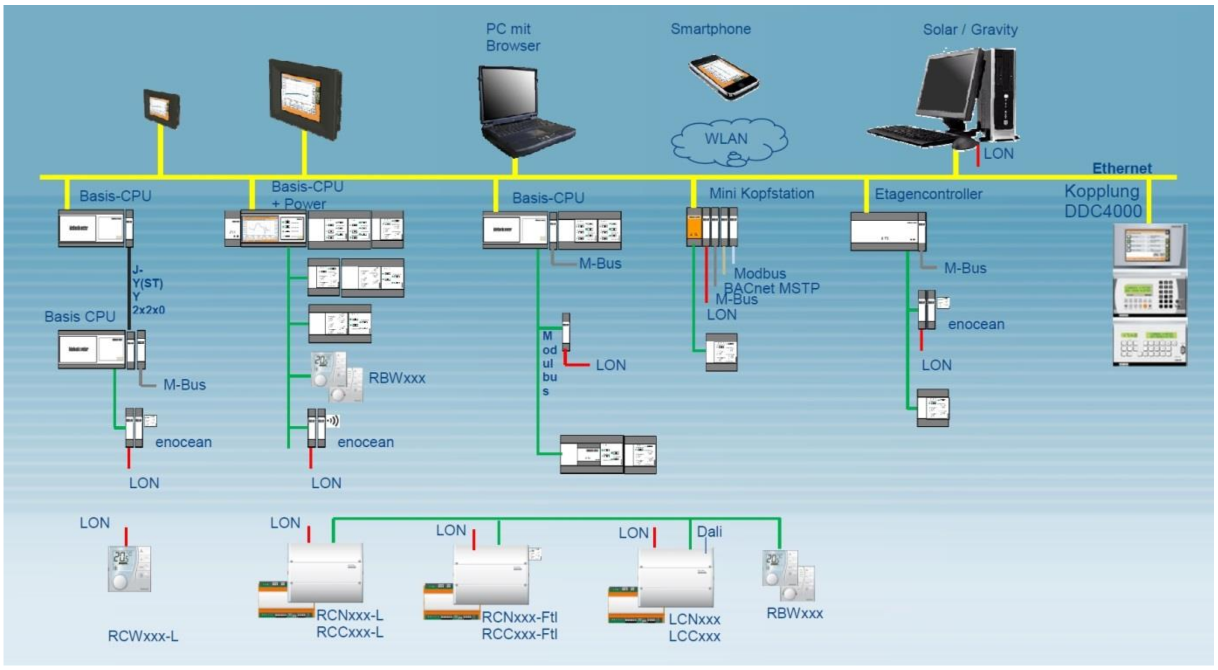


Figure 4: System topology within the Kieback&Peter company (new automation system, Source: Kieback&Peter GmbH & Co. KG)

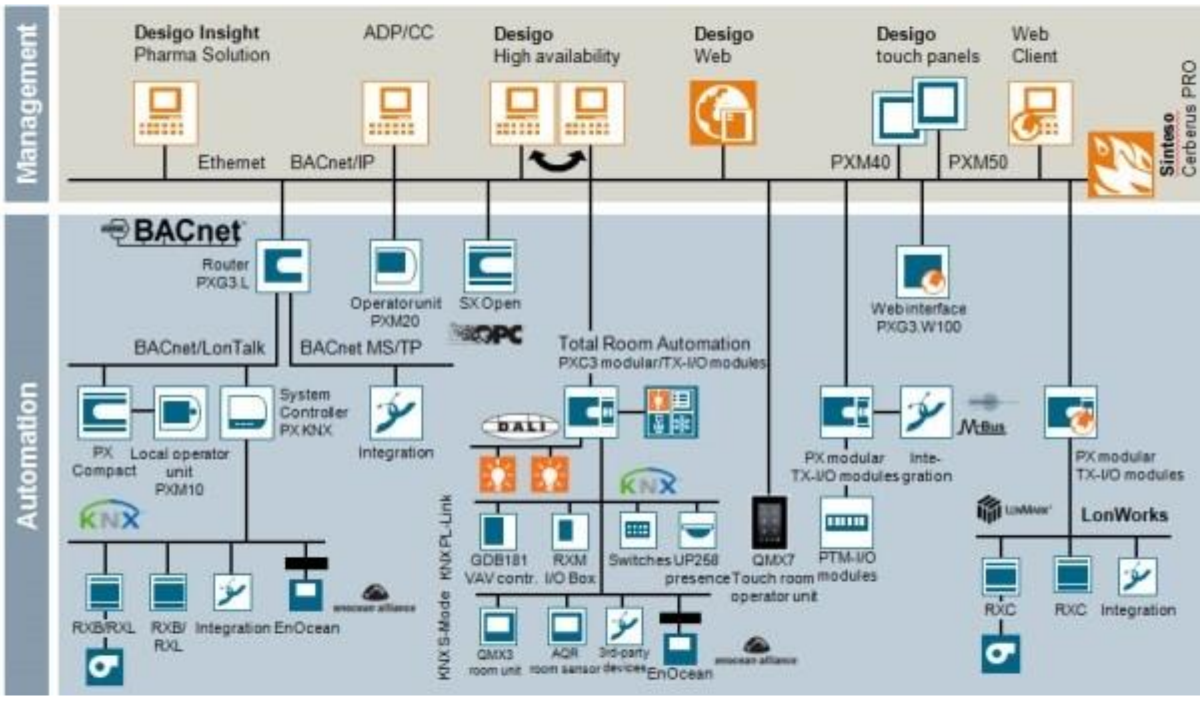


Figure 5: System topology of DESIGO BAS from Siemens [8]

Other protocols for the field level are M-bus, Modbus, DALI (lighting control), EnOcean and Web technologies. The development of latter was pushed by the requirements towards using data from the filed within enterprise applications and led to different technologies for use in building automation like OPC UA, oBIX or BACnet/WS [9]. The Figure 6 depicts on which levels the different protocols can be used.

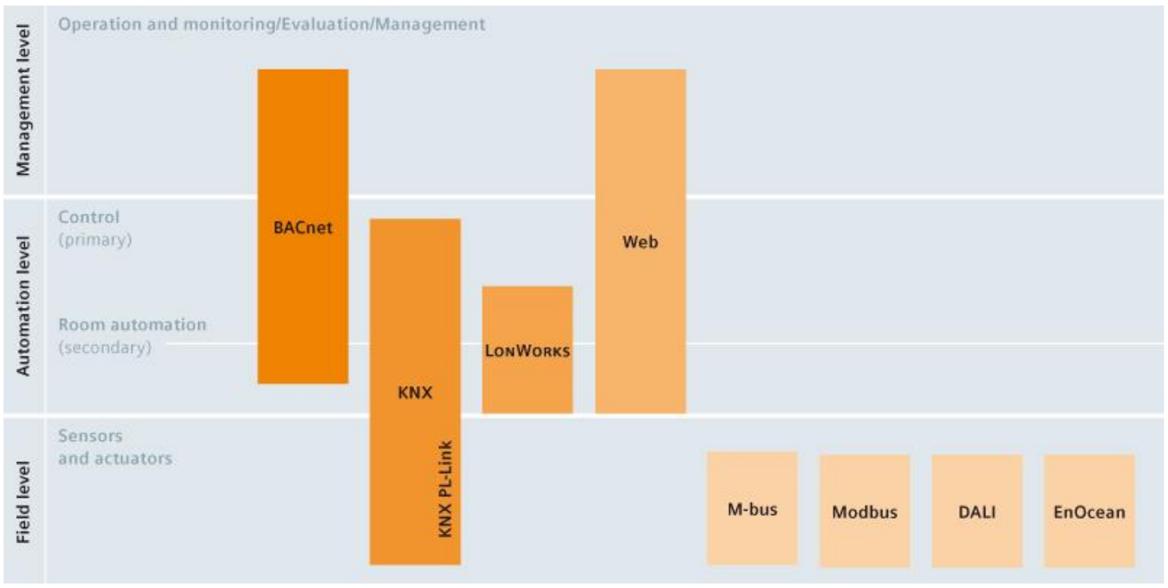


Figure 6: Communications protocols as used in the different levels of the automation pyramid [10]

2.3. Current Trends

The Web technologies based on Transmission Control Protocol (TCP), Hyper-Text Transfer Protocol (HTTP) and Extensible Markup Language (XML) often represent a bottleneck in building automation and control systems. Therefore a novel approach using a User Datagram Protocol (UDP) based Constrained Application Protocol (CoAP) and efficient XML interchange – EXI [11] is discussed.

Interesting for the future of building automation is also a present trend that is pursued by the project “haystack” [12]. Haystack tries to semantically structure information and to develop models so that the huge amount of data automation systems have to deal with can be analyzed effectively. The Haystack community recognized the threat of a data overload in large automation facilities. More about Haystack Project is to be found in Chapter 4.

Another interesting development is Niagara AX Framework [13], an open Java-based framework for building device-to-enterprise applications and Internet-enabled products. It provides a unified development platform to easily build Internet-enabled products and software applications for controlling and managing diverse “smart” devices across an enterprise in real time.

In the last years the Building Information Modeling (BIM) is changing the way of the design process of a building. BIM “represents the process of development and use of a computer generated model to simulate the planning, design, construction and operation of a facility” [14]. BIM aims at collecting, processing and updating digital data over the whole life cycle of a building – from the planning stage to the operating to the refurbishing to the demolition. The data could possibly be the basis for model predictive control (MPC). BIM data includes a 3D-model of the building with information about the building physics as well as the plumbing and electricity network. A standard for BIM files are for example the Industry Foundation Classes (IFC). While BIM has proven its value in design and construction phases, the necessary technologies for using BIM through operation including the integration with BAS are still missing. Nevertheless, there are several approaches in this direction, including ELASSTIC Project [15].

3. Base Technologies

3.1. Self-Adaptive Systems

Systems that operate under changing conditions and environments often require human supervision in order to adapt their behaviors correspondingly. It is the human himself that decides which actions have to be taken to achieve a desired goal. This approach refers to the *Human-in-the-loop* principle [16] leading to costly and time-consuming procedures during the operating phase. Therefore, there is a high demand for reduction of management complexity, management automation, and robustness to maintain the quality criteria within a time-limited range and that with only minimum costs. Self-adaptive systems are a solution to these problems. They react immediately to changing conditions and requirements and adapt relevant system parameters automatically. Therefore, such a system has to monitor itself and its surrounding context continuously, detect changes, decide how to react, and execute these actions. This results in a modified behavior that ensures a proper system operation. These processes depend on adaptation properties, domain characteristics, and preferences of stakeholders. It is widely believed that new models and frameworks are needed to design self-adaptive systems. Today, we are struck by the trend of increasing complexity in the design, development, and maintenance of technical systems. *Organic Computing* (OC) [17], like other initiatives such as IBM's *Autonomic Computing* [18] or *Proactive Computing* [19], postulates the necessity of a paradigm shift in the design of future technical applications underlined by [20]:

> *"It is not the question whether self-organized and adaptive systems will arise but how they will be designed and controlled."*

Traditionally, a significant part of the research on handling complexity and achieving quality goals has been focused on software development and its internal quality attributes. However, in recent years, there has been an increasing demand to deal with these issues at runtime. The primary causes for this trend include an increase in the heterogeneity level of software components, more frequent changes in the context, goals, and requirements especially at runtime. In fact, some of these causes are consequences of the higher demand for ubiquitous, pervasive, embedded, and mobile applications, mostly in the Internet and ad-hoc networks.

3.1.1. Definition

A definition of self-adaptive software is provided in a DARPA Broad Agency Announcement (BAA) [21]:

> "*Self-adaptive software evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible*."

A similar definition is given in [22]:

> "*Self-adaptive software modifies its own behavior in response to changes in its operating environment. By operating environment, we mean anything observable by the software system, such as end-user input, external hardware devices and sensors, or program instrumentation"*.

Prior to formalizing the concept of self-adaptive software, there has been a related point of view regarding the adaptive programming principle as an extension of object-oriented programming [23]:

> *"A program should be designed so that the representation of an object can be changed within certain constraints without affecting the program at all."*

According to this view point, an adaptive program is considered as [24]:

> *"A generic process model parameterized by graph constraints which define compatible structural models (customizers) as parameters of the process model."*

This view on adaptation is similar to reflection and meta-programming techniques. In another point of view, adaptation is mapped to evolution. A taxonomy is provided by [25] based on the object of change (*where*), the system properties (*what*), the temporal properties (*when*), and the actions to perform (*how*). This classification is mapped by [26] to the self-adaptive software domain, they propose a conceptual model for adaptation changes based on *Activity Theory*. Static and dynamic adaptations, related to the temporal dimension of this view, are mapped to compile-time evolution and run-time evolution. For this reason, dynamic adaptation is sometimes called *dynamic evolution*. In fact, self-adaptivity is directly linked to feedback and feedback control, whereas self-adaptive software can be aligned with the laws of evolution. Self-adaptive systems are autonomic on the one hand and self-managing on the other hand. Many researchers use the terms self-adaptive, autonomic, and self-managing interchangeably [27]. In the context of self-adaptive systems, we can consider a layered model that consists of: applications, services, components, middleware, network, and devices. The key point in self-adaptive software is that its lifecycle does not end after its development. It must be continued in order to respond to changing constraints and requirements at runtime.

3.1.2. Self-X Properties

The term self-x properties was initially characterized by IBM in the context of autonomous systems. They comprise key features exhibited by adaptive systems. At the very beginning IBM just defined the so called s*elf-CHOP* functions: self-<u>c</u>onfiguration, self-<u>h</u>ealing, self-<u>o</u>ptimization und self-<u>p</u>rotection. In the course of time, several additional self-x properties were defined. The most popular ones are listed below.

3.1.2.1. Self-Management

The system must be able to manage its own functionalities without actions from outside the system. The complexity of the system management task can be decreased by increasing the management capability of single components.

3.1.2.2. Self-Configuration

The configuration of complex systems is performed by experts. By enhancing a system with self-configuration capabilities, it is possible to find a feasible configuration in a distributed and autonomous way. Thus, the manual and error-prone configuration process can be omitted.

3.1.2.3. Self-Healing

The autonomous diagnosis of the current system state enables the detection of invalid system states. Afterwards, a valid system state is restored by means of self-healing. The self-healing process is supported by the self-configuration capabilities of the system. To achieve the complete "healing" of the system a certain degree of redundancy is assumed.

3.1.2.4. Self-Protection

Self-protection of specific elements is necessary if the system is operating in a dynamic environment. An autonomic application/system should be capable of detecting and protecting its resources from both internal and external attacks and maintaining overall system security and integrity.

3.1.2.5. Self-Optimization

The proactive search of a specific element for new opportunities to optimize its own behavior helps to reach the optimal system state. But to achieve such an optimization, resources are continuously needed. It is necessary to evaluate carefully if this effort for self-optimization is justifiable.

3.1.3. Adaptation Loop

The reference standard from the IBM Autonomic Computing Initiative [28] comprises an external feedback control loop which is called the *MAPE-K* loop. It includes monitoring, analyzing, planning and executing functions together with an additional shared knowledge base.

3.1.3.1. Adaptation Process

- The monitoring part provides the mechanisms that collect, aggregate, filter and report information (such as metrics and topologies) collected from managed resources.
- The analyzing part contains the mechanisms that correlate and model complex adaptation situations.
- The planning function encloses the mechanisms that construct the actions needed to achieve goals and objectives. The planning mechanism uses adaptation policies information to guide its work.
- The execute function groups the mechanisms that control the execution of an adaptation plan with considerations for dynamic updates.
- Knowledge represented by symptoms and policies is the relevant data shared among the monitoring, analyzing, planning and execute activities of the Autonomic Manager.

3.1.3.2. Sensors and Effectors

Sensors monitor software entities to generate a collection of data reflecting the state of the system, while effectors apply changes. Sensors and effectors are essential parts of a self-adaptive software system. The first step in realizing self-adaptive software is instrumenting sensors and effectors to build the adaptable software. Some of the protocols, standards, and formats that can be utilized are: *Web-Based Enterprise Management* (WBEM) [29] together with the *Common Information Model* (CIM) [30], *Scalable Internet Event Notification Architectures* (SIENA) [31], and the *Open Mobile Alliance Device Management Model* (OMA

DM) [32]. Another noteworthy standard for sensing is *Application Response Measurement* (ARM) [33], which enables developers to create a comprehensive end-to-end management system with the capability of measuring the application's availability, performance, usage, and end-to-end response time. The ideas behind the *Simple Network Management Protocol* (SNMP) [34] for network and distributed systems are also applicable to self-adaptive software. Software management frameworks, such as *Java Management eXtensions* (JMX) [35] provide powerful facilities for both sensing and effecting. Another notable idea along this line is *pulse monitoring* [36] adopted from Grid Computing, which is an extension of the heartbeat monitoring process.

Some of the effectors are based on a set of design patterns that allow the software system to change some artifacts during run-time. For instance, wrapper (adapter), proxy, and strategy are well-known design pattern [37] for this purpose. Moreover, microkernel, reflection, and interception are architectural patterns suitable for enabling adaptability in a software system [38], [39]. Furthermore, several design patterns are mentioned, namely goal-driven self-assembly, self-healing clusters, and utility-function-driven resource allocation for self-configuring, self-healing, and self-optimizing, respectively [40].

3.1.4. Sense-Plan-Act

The sense-plan-act (SPA) is an approach for autonomous robots consisting of three functional components: a sensing system translating raw sensor inputs into a world model, a planning system generating a plan to achieve a specific goal with the help of the world model, and an execution system generating the actions provided by the plan [41]. The characteristics of the SPA approach are that the flow of control among these elements is unidirectional and linear and that the acting component, i.e., the execution of a plan, is built of orderings, conditionals and loops. Thus, the intelligence of the system is entailed in the planning component that generates the plan. However, the SPA architecture entails the major difficulty that planning is time-consuming. Since the world may change quickly, the resulting plan might be rendered invalid already during the planning process. Thus, these time-consuming computations induce the risk of internal states that are not synchronized with the reality that it is intended to represent and therefore execution steps might be executed in an inappropriate context [41].

3.1.5. Collect-Analyze-Decide-Act

Autonomic systems form a feedback loop collecting information from several sources, analyzing them, forming a decision based on the analysis and reporting this result to users or acting in a similar way. This process is also often referred to as the autonomic control loop [42]. Specifically, in the collection phase relevant knowledge information about the current state is collected, e.g., via environmental sensors or network instrumentations. This data must be analyzed in the next step constructing a model of the situation using inferences and distinct rules. At this state, it needs to be clarified how the systems state is inferred and which data is relevant for validation. The basis of the inferences is useful knowledge for the decision making process in the next step. In the acting phase, the decision is attempted to be realized by performing the adaptation or by reporting the result to users or administrators. For the next control cycle, the impact of the decisions can be fed back and used as relevant knowledge.

3.1.6. Observer/Controller

Numerous sensors, processors and embedded systems provide safety and comfort functions as well as regulation or motor control functions. These embedded systems will be interconnected and form a complex communication network. A system consisting of many interacting components may exhibit new properties emerging from new configuration possibilities that are not yet anticipated in the design stage but need to be dealt with at run-time. This requires adaptive systems with optimization techniques in order to learn adequate responses to unforeseen conditions. A generic observer/controller is required to assess the behavior of an organic computing (OC) system and to control its dynamics [43]. A number of sensors and actuators are used in order to measure system variables and to influence the System under Observation and Control (SuOC) characterizing the system's global state and dynamics. The observer measures and quantifies the current state of the SuOC. The monitored data needs to be preprocessed, analyzed and a prediction of future developments will result in situation parameters that characterize the observed or future system state. Based on these situation parameters that are computed by the observer and being transferred to the controller, an evaluation will be performed with respect to the user-defined goal leading to a decision of the controller whether an intervention will be beneficial. This decision is made by mapping the situation parameters to respective actions and evaluating the corresponding performance changes. Previous situation-action mappings will be stored in order to determine the reaction to known situations. Using these mappings and estimations, the controller will basically act as a learning component. In particular, the controller is designed in two levels consisting of an on-line learning level and an offline optimization level. This design provides several advantages: using simulation based evaluations, appropriate situation-action mappings can be found without having to test different alternatives and this approach is significantly faster than the realization of evaluations in the SuOC. Combining the slower level 2 approach with the faster memory-based level 1 approach enables a quick reaction by situation-action mappings for known or similarly known situations while in parallel an optimized situation-action mapping will be available in the future [44]. Therefore, the observer/controller architecture framework is widely applicable to a large range of technical systems including building automation systems.

3.1.7. Operator-Controller Module

Another related approach is the Operator-Controller Module (OCM) developed by the Collaborative Research Centre 614 [45]. The OCM is an autonomic system following its own objectives. It is specialized for mechatronic systems combining mechanical and electrical engineering with a strong focus on real-time constraints [46]. The controller represents the continuous part of the system and the operator comprises the time-discrete parts of information processing, which includes functions like emergency routines, controller monitoring and optimization. In particular, a reflective operator may modify the controller and induce switches between control strategies, while a cognitive operator gathers information about the system and its environment improving the system's behavior. The agent could, e.g., use simulation runs of alternative future behaviors and evaluate them selecting the most promising alternative concerning the optimization goal. Thus, the structure of the OCM is especially useful for model-based optimization and due to its modular composition it is easily possible to add other methods or functions of the agent theory. During execution of the plan and the monitoring of the real world, inductive and reinforcement learning is used in order to adapt the behavior-based environment and system models to the real world. The use of learning procedures enhances the assessment of an optimal starting point for the optimization and the convergence of the optimization technique. Hence, this knowledge base can be used for similarity analysis enabling the detection of frequently reoccurring scenarios [46].

3.1.8. Software Engineering

A lot of research areas in software engineering are directly or indirectly related to self-adaptive software. Many of the self-x properties correspond to quality factors and deal with *non-functional requirements* (NFR) concerning e.g. safety, security, and performance. According to this, most of the ideas developed in the context of software quality for realizing and measuring quality are applicable to self-adaptive software. Despite of non-functional requirements, also functional ones have to be considered bringing the requirements engineering into play. The combination of software with its specification allows the formal proof of correctness regarding requirements and self-x properties [47]. Due to various differences between traditional and self-adaptive software, the existing models and methods developed for non-adaptive software systems are not directly applicable. This means that new approaches based on formal models, such as *Model-Integrated Computing* (MIC) [48], are required for this purpose.

Software Architecture models and languages, such as *Architectural Description Languages* (ADL), can certainly be helpful in software modeling and management, particularly at runtime [49]. A survey on several ADLs based on graphs, process algebras, and other formalisms for dynamic software architectures is provided by [50]. Acme ADL is used to describe the architecture of adaptable software and to detect violations from defined constraints [51].

Component-Based Software Engineering (CBSE) supports the development of self-adaptive software in two ways. It is easier to implement adaptable software systems relying on established component models. Secondly, an adaptation engine needs to be itself modular and reusable. Moreover, component models can be used in adaptive systems as a means of incorporating the underlying services for dynamic adaptation and adaptation management. Another related area, *Aspect-Oriented Programming* (AOP) can also be used in realizing self-adaptive software. This facilitates encapsulating adaptation concerns in the form of aspects through dynamic runtime adaptation.

Service Computing and *Service-Oriented Architecture* (SOA) can also support realizing self-adaptive software by facilitating the composition of loosely coupled services. Web service technology is often an appropriate option for implementing dynamic adaptable business processes and service-oriented software systems, due to their flexibility for composition, orchestration, and choreography.

3.1.9. Control Theory

Techniques used in network and distributed computing can be extensively applied to self-adaptive software. *Policy-based management* (PBM) is one of the most successful approaches followed in network and distributed computing [52]. Policy-based management specifies how to deal with situations that are likely to occur (e.g., priorities and access control rules for system resources). Most of the definitions given for policy emphasize on providing guidance in determining decisions and actions. The policy management services normally consist of a policy repository, a set of *Policy Decision Points* (PDP) for interpreting the policies, and a set of *Policy Enforcement Points* (PEP) for applying the policies [53]. The most widely used policy type in networks is the action policy (in the form of event-condition-action rules) which is also applicable to self-adaptive software. In addition, other policy types like goal policy (specifying a desired state), and utility policy (expressing the value of each possible state) can also be exploited in self-adaptive software [54]. The adaptation policies may need to be changed based on new requirements or conditions.

Quality of Service (QoS) management, another successful area in networking and distributed systems [55], is closely related to policy management [56]. QoS requirements are related to non-functional requirements of a system, and consequently, they can be linked to self-x properties in distributed software systems. In this context, QoS management methods rely on either modeling the application, e.g., queuing models, or using well-understood components, e.g. *Prediction-Enabled Component Technology* (PECT) [57]. Therefore, QoS management can assist in modeling the quality factors of a self-adaptive software system and also in realizing adaptation processes.

One of the well-established areas in networks and distributed systems is resource management. In specific, virtualization techniques can have a significant impact on the quality of self-adaptive software. *Virtualization* reduces the domain of an adaptation engine to the contents of a virtual machine [58]. It also provides an effective way for legacy software systems to coexist with current operational environments.

3.1.10. Research Challenges

Self-adaptive software creates new opportunities, and at the same time, poses new challenges to its development and operation. Functional and non-functional requirements, self-x properties, and quality factors must be considered as a whole. Besides that, it is a challenging task to capture the stakeholders' expectations, translate, refine and relate them to adaptation requirements and goals satisfying at run-time. A model at design-time is used as a fundamental basis for answering the adaptation requirements' questions: where, when, what, why, who, and how. A significant challenge for monitoring different attributes in adaptable software is the cost and load of the sensors, respectively. In most cases, the monitoring process does not need all details of an event, while in the case of deviating from the "normal" behavior, more data is required. Therefore, the monitoring and management process must be adaptable itself. Most of the existing solutions are not able to represent policies and goals explicitly. Typically high-level policies must be decomposed and refined into lower-level ones that are understandable by the technical system elements. It is difficult to accomplish this task without an explicit model with comprehensive tool-support. Commonly, lower-level policies at runtime are represented as rules that are hand-coded on the basis of informal descriptions of constraints and objectives restricting the system behavior. This is insufficient, especially in the context of high dynamic service systems that depend on an overall management process respecting high-level goals and constraints.

3.2. Data Mining for Building Automation

3.2.1. Introduction

Data Mining is the process of analyzing and identifying patterns in large data sets in order to extract information and transform it into a meaningful structure for further use. When information about energy management, physical security, environmental conditions and facility operations etc. is converged for building automation, not only redundant control infrastructure can be eliminated but also communication between different systems is enabled allowing for data collection and analysis. It was shown, e.g., that building occupancy can be predicted using data mining [59] combined with Bayesian modeling [60] enabling more efficient control of HVAC systems. Overall, analyzing this large amount of data may open new possibilities as flexible management control and automation mechanisms leading to risk reduction, energy efficiency, operational effectiveness and occupant satisfaction. However, before being able to analyze this data, it needs to be pre-processed in order to minimize errors. The pre-selection of relevant data, it's preprocessing as well as transformations are critical for the quality of the overall result and consequently take up 75-80% of the total effort in the data mining process [61]. In addition to these three stages, the data mining process, also known as Knowledge Discovery in Databases (KDD) contains several other stages: the selection of data mining methods and their application, the interpretation and evaluation of results, and finally the application of the results [62]. Basically, a parallel can be drawn between the data mining process and the MAPE loop [63] containing the phases: observe (measure), analyze (analyze), learn (plan) and apply (execute) (Figure 7).

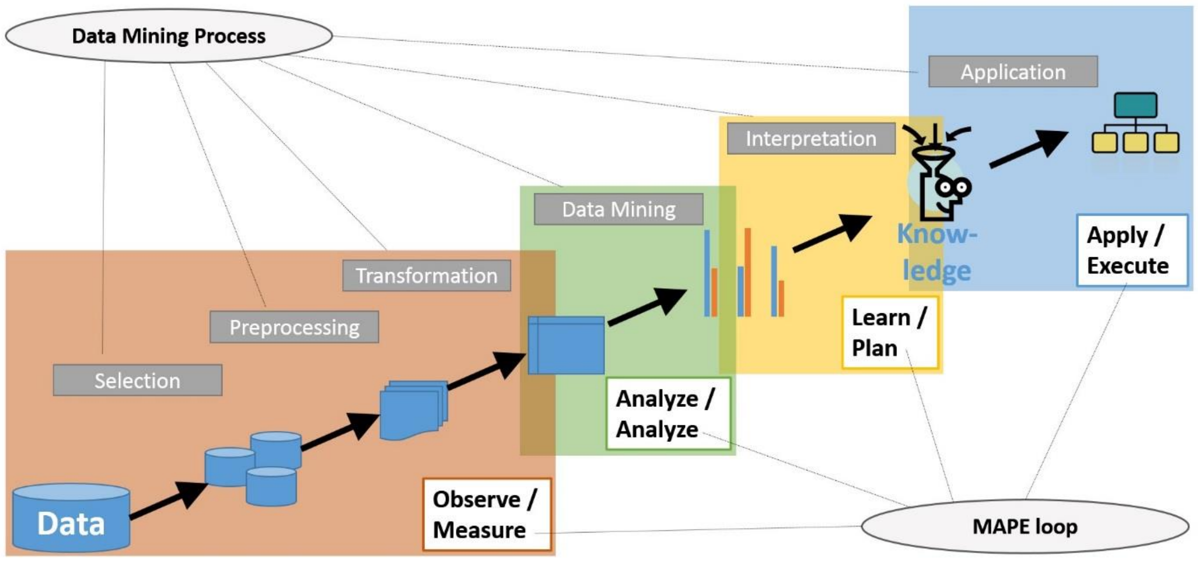


Figure 7: Data Mining Process adapted from [62] compared with the MAPE loop.

Since in BaaS, the target is an automated process, the major challenge of the data mining process is the automatic minimization of errors during pre-processing of data and the preparation of meaningful quantifications, while the automatic interpretation and evaluation of the data and the application of the results might be realized via adaptive systems (3.1). Nevertheless, the interpretation of processed data might also be computed further using Bayesian networks as proposed by [60].

Another challenge for the application of data mining for Building Automation is that resource limitations may impose restrictions on applicable data mining techniques. For example, if data mining techniques are implemented on building automation devices distributed throughout the building then these are likely to be designed cost-effectively, i.e. tailored to their focused purpose. Consequently, resource consuming data mining algorithms might not be possible to realize on these devices but might be relocated to the central control center of the building.

On the other hand it was shown that data mining techniques can also assist in detecting communication faults in control networks, preventing problems by detecting early symptoms of potential problems [64].

3.2.2. Observe

Since the measurement and collection of huge amounts of data in order to continuously track changes can be error-prone, the preprocessing of data minimizing errors and selecting important data for interpretations is an important step in the data mining process. Data formats need to be transformed, data needs to be analyzed in an exploratory manner, appropriate data needs to be selected and aggregated, samples need to be selected for further analysis reducing the amount of data, the dimensionality of the data needs to be reduced, missing data needs to be dealt with and incorrect data needs to be detected. Techniques like anomaly/outlier detection, clustering or regression can be used in order to detect missing or incorrect data. Finally, features need to be coded in order to be analyzed choosing a subset of features that ideally and sufficiently describe the target concept. This step of feature selection is of paramount importance since it determines the quality of the data mining process. When the quality of feature selection, usually performed by machine learning, pattern recognition, or statistical algorithms is poor, incomplete information might be extracted or noisy or irrelevant features might be detected. Here, the separability of features in feature space (e.g., in different categories) is targeted to unambiguously describe the target concept [65].

3.2.3. Analyze

In the next step the preprocessed data is analyzed for being interpretable. That means that data mining involves the fitting of models to observed data or to determine patterns from data. The goal of data mining is either the verification of existing hypothesis or the autonomous discovery of new patterns either for predictions or for the presentation in human-understandable form. Data is being analyzed using methods like (1) classifications for discrete variables, (2) regression analyses identifying dependencies between continuous variables and predicting new values based on the past ones, (3) segmentations or clustering for finding homogenous objects within a group, or (4) other exploratory statistically based data analyses [62], [66]. These methods are applied in a variety of data mining approaches. The most commonly applied techniques for data mining are decision trees, artificial neural networks, and nearest neighbor classification. These data mining techniques are often repeatedly applied in an iterative manner [62].

Decision trees are based on classification techniques. During training sessions data is successively divided into disjoint subsets that within each subset represent a homogenous group. This division is subsequently verified during test sessions, where the quality of the model is determined by the number of incorrect classifications. The classification model is represented in a tree structure making the model relatively easy to comprehend for the user but also limiting the type of classification boundaries. Therefore, often multivariate hyperplanes are introduced making the model more powerful for predictions but more complex to comprehend [62], [66]. This technique was also used by a recent study examining the use of data mining techniques for the understanding of energy performance of a building [67].

Inspired by the biological nervous system, artificial neural networks are non-linear, predictive, but rather complex models learning through training. They contain modelled neurons, each being a processing unit within a network. Each neuronal unit has an input function calculating a weighted mean of all inputs. This value is propagated to the activation function, which determines whether the neuron is activated when a certain threshold is reached. An output function determines the value that will be propagated to other neurons when the activation of the neuron was successful. When modelling a neural network the number, type as well as the configuration of neurons and the weightings of their connections need to be fixed at the beginning. The model is then adjusted according to the dataset refining the weightings of the connections by learning. Supervised learning methods, where input and desired output data is known at the time of training, rely basically on classification mechanisms, while unsupervised learning, which is not provided with the correct results during training, works through clustering and prediction. Back propagation is mostly used as learning technique where the output value is compared with the target and the error is fed back through the network enabling an adjustment of the weights [62], [68].

The nearest neighbor classifier is a supervised learning mechanism trying to classify datasets based on similar data in a historical datasets, i.e., where their classification is known. It works therefore better for extrapolation rather than for predictive enquiries [66], [69]. Vectors in a multidimensional feature space are used to separate the classifications of data, while a commonly used distance metric is the Euclidean distance for continuous variables and the Hamming distance for discrete variables [66]. It was shown that using this technique a high performance data mining can be achieved [69].

Regardless of the technique used, data mining always involves the process of building a model based on specified criteria from already captured data. Once a model is built, it can be applied in similar situations for predictions or discovery of new pattern.

3.2.4. Learn and apply

Finally, when patterns have been found in datasets, they need to be interpreted. Often, an iterative process of data mining techniques is necessary before data can be interpreted. Interpretation of the extracted patterns and models usually involve visualization techniques. However, since in building automation, the learning and application from data mining should be performed in an automated way, parameters might directly be adjusted of an existing model or other appropriate parameters will be selected and applied to the model. Parameters need to be meaningfully quantified, e.g., to assess the required energy to achieve a certain increase in (room) temperature. In order to map these results on adaptations in building automation, an integration of autonomous adaptive techniques like the operator/controller approach (3.1.6) will be conceivable for the assistance in planning but as well for the application of the results.

3.3. Device SOA

3.3.1. Introduction

There is a great number of technologies available to have functionality exposed to users on a network. We will talk about the approach to have services offered by specific devices. This is different from the device independent approach to implement a service oriented architecture (SOA) [70] using, for example, Java's remote method invocation (RMI) [71] or representational state transfer based (REST) web services [72].

In service oriented architectures services are self-contained functional entities [73]. Devices on the other hand are containers for services or other devices. In Device SOAs, devices announce their presence on the network and/or can be actively discovered. Both services and devices are enabled to let third parties know their capabilities and metadata information. For devices, this may include hosted services and friendly names for example. A service's metadata can respectively include offered operations. This is illustrated in Figure 8.

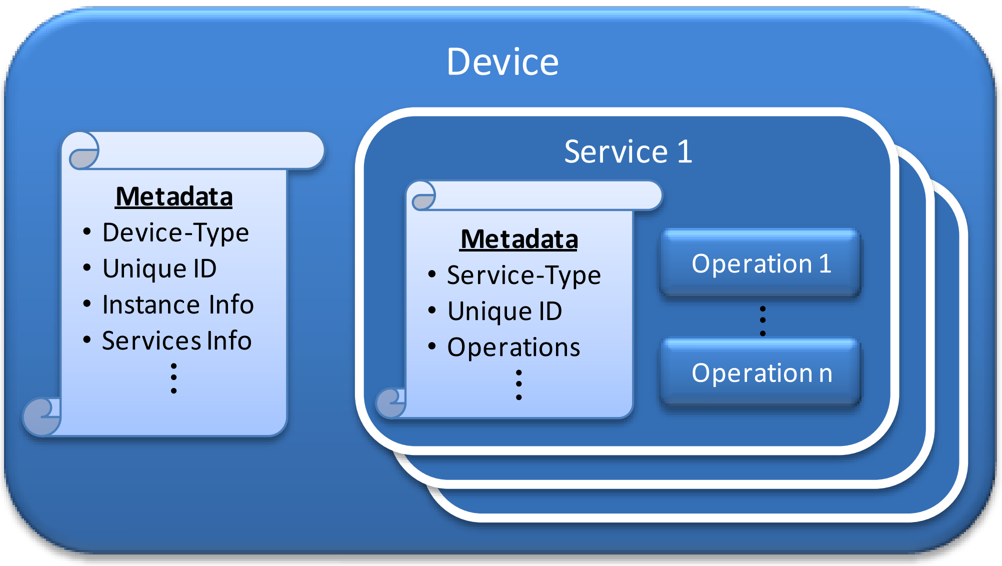


Figure 8: Abstract Device SOA scheme

To talk about similarities and differences between Device SOA technologies a common terminology is helpful. Important concepts are listed below.

Discovery: As mentioned, this includes devices announcing their presence on a network and clients being able to actively search for them.

Description: The ability of devices and services to describe themselves by means of metadata exchange.

Control: Makes it possible for clients on the network to use operations of a service. This includes providing input and getting output back from operations.

Eventing: Enables clients to subscribe for a predetermined time span for interesting data from a service. The service is responsible for delivering the subsequent events to all its subscribers.

These represent a subset of the six categories identified by the SIRENA project [74].

3.3.2. Technologies

Some of the most important technologies enabling a Device SOA approach are Devices Profile for Web Services (DPWS) [75], Universal Plug and Play (UPnP) [76], Bluetooth (BT) [77], the building automation and control networks protocol (BACnet) [1], the local operating network (LonWorks) [2] and the building automation field bus KNX [78]. These technologies are specialized to serve the needs of their respective domains but are very similar from a more abstract point of view.

The following subsections shortly present these technologies with a focus on the four basic concepts, described in the last chapter. Because the description capabilities of the technologies differ, description is split up into device/service metadata and dynamic service description. Dynamic service description is the ability of a service to describe itself without relying on static information (i.e. templates or profiles). This enables such services to be defined as needed and makes them much more flexible.

The four basic concepts have different names and characteristics in the respective technology. Table 3 offers a short overview for each technology and its capabilities. Notable is that only DPWS and UPnP offer dynamic service description.

	DPWS	UPnP	Bluetooth	BACnet	LonWorks	KNX
Discovery	X	X	X	X	X	X
Device/Service Metadata	X	X	X	X	X	X
Dyn. Service Description	X	X				
Control	X	X	X	X	X	X
Eventing	X	X		X	X	

Table 3: Abstract comparison of technologies focusing on Device SOA functionalities

3.3.2.1. DPWS

DPWS fully implements the Device SOA principle. Mechanisms for Discovery, Description, Control and Eventing are all defined and come in alignment with web service standards. Services, operations and parameters of operations are described using the Web Service Description Language (WSDL).

Discovery is implemented according to the WS-Discovery standard. It uses Hello, Bye, Probe and Resolve multicast messages to enable devices to announce when they are entering a network (Hello), when they are leaving a network (Bye) and when they have changed their metadata (Hello with an updated metadata version). Metadata exchange is done over HTTP using Get messages when talking to devices and GetMetadata messages when requesting metadata from services. Operations in DPWS can be one-way operations (only input) or request-response operations (input and output). DPWS is based upon WS-Eventing and supports simple (one-way) notification for events as well as solicit responses from the client.

3.3.2.2. UPnP

Like DPWS, UPnP implements the Device SOA principle. It therefore offers mechanisms for Discovery, Description, Control and Eventing.

Discovery is based on HTTP over UDP and Simple Service Discovery Protocol (SSDP), which uses M_Search and Notify messages. UPnP offers Description functionality, which relies on

XML templates. Eventing is based and limited to the observation of status variables (multicast notifications are available).

UPnP comes with the special ability for devices to contain other devices, calling the former root device and the latter embedded devices. Embedded devices are published by the root device.

As stated in [79], “UPnP was a choice for SIRENA basic technology but has the disadvantage of supporting only smaller networks. With an increasing amount of services/devices, the amount of broadcast messages grows exponentially in an UPnP network. Furthermore UPnP supports IPv4 only.”

3.3.2.3. Bluetooth

Bluetooth uses predefined profiles. These represent certain functionalities, like the service discovery profile. This facilitates easy to use communication between devices with the ability for devices to advertise all of the services they provide. Because all service descriptions are known in advance and defined by their respective profiles, the logic to discover and use a service is not as complex and service descriptions are not send over the network, which helps to keep the network’s congestion low.

Bluetooth’s Device Manager relies on Inquiry and Inquiry-Response messages for device-discovery, the Link Manager Protocol provides device-description leveraging name request messages, while the Service Discovery Protocol provides service description (using ServiceSearchRequest, ServiceAttributeRequest and ServiceSearchAttributeRequest messages). Control is being provided by the OBEX protocol (using Connect, Get, Put, SetPath messages). Bluetooth provides no eventing capabilities at all.

3.3.2.4. BACnet

“BACnet’s device functionality is based on an object model to represent the functioning of building automation and control systems (BACS).” [80]

While the protocol currently provides eight device profiles defining their capabilities, BACnet defines a set of 38 services as a basis for all messages between devices. This communication is based on a client-server model, which uses standardized objects for information exchange (service requests and responses).

The service set can be differentiated into five broad categories. These categories do not match with the Device SOA functionalities one-to-one and one category has become deprecated through technological advancements. The Remote Device Management Services offer the Who-Is and Who-Has services which provide device and object Discovery mechanisms while also offering a variety of Control functionalities. The File Access Services and Object Access Services provide other Control functionalities (e.g. CreateObject) whereby the Object Access Services offers access to object properties for Description, i.e. the ReadProperty service. Eventing functionalities are provided by the Alarm and Event Services.

This standardization of sets of profiles, services and objects facilitates high interoperability across manufacturers. It is also a common practice within the building automation domain to use BACnet systems for managing KNX- and LonWorks networks because of its focus on the management level in contrast to their focus on the field level.

3.3.2.5. LonWorks

LonWorks' devices must run an application, which may contain network variables and configuration properties. Device templates are being used that contain all the attributes of a given device type and the device publishes information of the running application. As for Discovery an automatic discovery process can be executed to search for devices on the network. LonWorks' self-identification and self-documentation mechanisms provide Description functionality.

While communication itself uses a client-server model, (standardized) network variables are being defined to create logical connections between devices. "LonWorks uses bindings which offer a process that defines connections between devices including the data that devices share with each other." [81]

LonWorks also provides basic eventing functionalities, i.e. subscription and one-way event notification.

3.3.2.6. KNX

Within a KNX network devices communicate over a single event bus system (which can be coupled to an Ethernet network). On this bus, special data-telegrams are being used with Service Type Identifiers like SEARCH_REQUEST for Discovery functionality, DESCRIPTION_REQUEST for Description and DEVICE_CONFIGURATION_REQUEST for Control functionality. No Eventing is provided.

3.3.3. Not-Device-centered SOA gateway/middleware projects

There is software building on the abstract similarity of the presented technologies and many others, functioning as a middleware or gateway layer between two or more of them.

Many of the more sophisticated approaches use a modularized solution utilizing web service standards or more often OSGi as an underlying service platform.

For example, EnTiMid "A service-based middleware for home appliances" which uses OSGi in a model driven engineering approach "to address the challenges of the development and deployment of building automation applications over an evolving, large-scale distributed computing infrastructure." [82] The framework integrates high-level service technologies like DPWS, UPnP and web services as well as low-level service like BACnet, LonWorks, KNX and others into a service architecture to offer a neutral middleware solution.

Another OSGi based architecture is presented in [83], where the evaluation of the platform, in a similar context as the one presented by the previous example, proves that it serves as an effective bridge across disparate networking technologies (DPWS, UPnP, Bluetooth, Jini, and Zigbee).

The study in [84] discusses possibilities for building automation system based on web services while a web service adapter approach has been evaluated in [85], integrating DPWS, UPnP, Bluetooth, Jini and WS (Web Services).

As one of the most recent and the most promising solutions, mainly focusing on smart home technologies, the open Home Automation Bus (openHAB) [86], using a domain model centered approach, has been developed. It integrates a vast range of building automation technologies based on, once more, the OSGi platform for modularization. It primarily implements an event bus, which adds new technologies through technology-specific bindings, which come as OSGi bundles. In contrast with the other examined projects, it tries to be an offline solution, thereby ignoring web service interoperability resulting in the lack of technologies like DPWS, UPnP etc. and marketing itself as the "Intranet of Things". The project currently evolves into the open-source project Eclipse SmartHome [87].

The next chapter will present the Device SOA based Java Multi Edition DPWS Stack (JMEDS) framework.

3.3.4. Abstraction provided by JMEDS

The general similarity of the concepts underlying all of the technologies mentioned above was the motivation behind the development of JMEDS beyond its single technology orientation to be a dynamic Device SOA framework. In the following chapter, we will explain how the framework enables the development of devices and services independently of the underlying protocol.

First, the most important concepts from the device/service perspectives will be explained in Section 3.3.4.1. The same will be done from the client's perspective in the following Section 3.3.4.2. Please refer to the diagram in Figure 9 when reading these sections. Finally, some of JMEDS' cross technology security capabilities will be explained in Section 3.3.4.3.

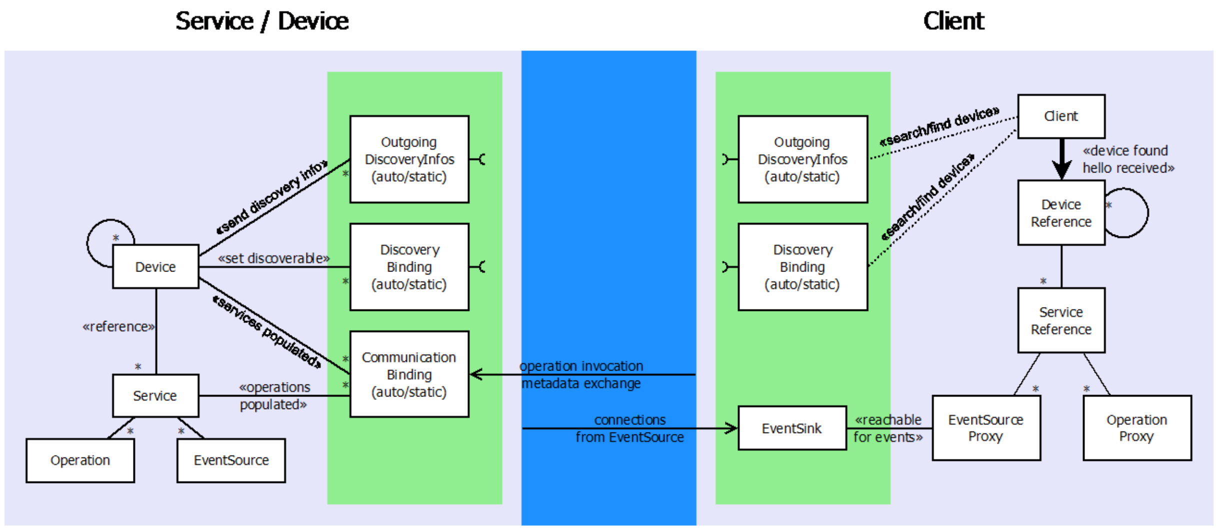


Figure 9: JMEDS frame structure

3.3.4.1. Service/Device

It does not come as a surprise to find devices and services represented in the internal structure of JMEDS. A device traditionally holds references to services. An exception to the traditional case is the UPnP protocol where it is possible to have devices hosting other devices. The first case is generally supported by the framework, the second only when using the UPnP module. The diagram in Figure 9 shows both containment variants.

Services contain operations. Operations, like methods in Java or functions in C, have predefined input and output parameters. A special kind of operation is an event (also called "evented operation"). Events can send messages to subscribers, after those have subscribed themselves.

All those entities need to be discoverable on the network. Discoverability in this context means nothing more than announcing the presence of a device on a network or probing for a device on a network making use of multicast technologies. DPWS does this utilizing WS-

Discovery (WS-DD), UPnP using the Simple Service Discovery Protocol (SSDP). JMEDS does not expose these protocols directly. Instead, it only has to be provided with the necessary information about the interfaces to use during this process. A device in JMEDS can be provided with so-called “Discovery Bindings” and “Outgoing Discovery Infos”. The former specify an interface and an address (for example something like eth0 and IP/PORT in case it is a UDP binding) that the device is going to listen on to receive messages (i.e. probe messages), the latter specify the interface to be used when sending discovery messages.

Both of the previously mentioned constructs come in two flavors. The concept will be explained with a focus on the bindings, but the “Outgoing Discovery Infos” work very similar. First, there is the static binding, which has to be provided with everything (interface, address, port etc.) in advance. The second flavor has auto in its name and thus can do more on its own. The so called “auto bindings” require only interfaces to be supplied to them. Ports and addresses are chosen automatically. When one of those interfaces goes down or comes up, the ”auto binding” takes care of the specifics of removing or (re-)adding the device to the corresponding network. In fact, it is even possible to make an interface known to the binding that does not yet exist. It will be used by the “auto binding” as soon as it becomes available.

As discovery bindings are needed for the discovery of devices, devices and services also need bindings to be reachable for metadata requests, operation invocations etc. The concept behind these bindings in JMEDS is very much analog to that behind the discovery bindings. As discovery bindings, the so called “communication bindings” also come as static and automatic bindings. For example, if a devices gains reachability through one if its auto-bindings (an interface becomes available) JMEDS takes care of the logistics of changing the devices metadata and making this change public (in case of DPWS it, for example, sends a new hello message).

3.3.4.2. Client

The most important entity in JMEDS on the client side is the main client (the “default client”). It offers abilities to actively search for interesting devices and to be notified about new devices that appear on the network (i.e. when receiving a DPWS hello message). The client uses the “discovery bindings” and “outgoing discovery info” concepts (presented in the previous section) to provide these abilities. When a search in a network is successful or a new device appears on the network, a device reference is provided by JMEDS. It is important to remark that a device reference does not contain any device metadata initially. This metadata is obtained only when an actual device (technically a device proxy) is requested from the device reference. This segregation between discovery and metadata exchange exists in many of the supported technologies such as DPWS and UPnP. Even if the segregation does not exist in the technology (for example, if there is only a limited number of profiles and those are all present on the client side - as it is the case in BT) the JMEDS API has proofed to be appropriate.

A device on its part can be asked for a service reference, which in turn can be asked for an actual service. Again, the further metadata exchange is triggered by the request for the service on the reference.

The services can be asked to provide operation proxies and event source proxies. These can be invoked or in case of the event source can be subscribed. To receive events an event sink has to be provided. The address of which is included in the subscription message. An event sink must be reachable for connections from the corresponding event source. It is a special kind of binding.

3.3.4.3. Security

To foster easier understanding of the basic structures, the security support was omitted in the previous sections. JMEDS supports authentication, authorization and encryption. This section will focus on authentication and authorization to some extent.

It is possible for every device, service, operation and event source to be configured to take the user's credentials and the way of communication into account when deciding to answer or to disallow the request (authorization). In JMEDS the authentication information on both client and device/service side is stored in "credential info" objects. Those are supplied, for example, as parameters when invoking an operation on a service reference. They can contain user name/password combinations or digital certificates. Another concept used in JMEDS is the "security key". It encapsulates "credential infos" and "outgoing discovery information" objects. On the device/service side, the latter are used to control the network interfaces that are to be used for discovery (e.g. hello messages, resolve messages, etc.). The former is used to optionally sign outgoing discovery messages and more importantly enable secure (SSL/TLS) connections between clients and devices, services (encrypted metadata exchange) and operations (encrypted operation invocations).

3.4. Privacy and data security

A system with the aspiration of BaaS will collect, store and process data from the environment as well as data about individual persons. In particular, this includes sensitive data with need for protection. Therefore, security considerations have to be taken into account in the BaaS system design. Where personal data is handled by the system, privacy aspects need to be examined.

3.4.1. Data security

For designing secure distributed systems, Anderson identifies four elements to be subject to analysis [88].

First, the security **policy** defines intended goals. Security policies are abstract rules a system needs to fulfill. One way to define these is to model threats to the assets considered valuable and determining the appropriate protection rules.

A policy is implemented by **mechanisms**, which are the concrete method used to achieve goals. An example could be requiring a secure channel protecting and authenticating communication content and implementing it with the transport layer security protocol (TLS).

Assurance considers the appropriate amount of confidence to be put in a mechanism, in order for the security analysis to reflect the actual properties of a deployed system in adversarial conditions.

Lastly, complex systems involving multiple stakeholders or individuals need to ensure that the actors as modeled in the system design reflect the actual interests and behavior of individuals in the deployed system. To reach this goal, **incentives** have to be appropriately engineered. This ensures that any attacker actually has to defeat the security policy as designed and cannot simply circumvent it.

One common requirement for security mechanisms is to provide secure separation of data access, while transiting less trusted systems. This goal can be addressed with the concept of *attribute based encryption* [89]. In contrast to secret-sharing approaches, where multiple parties have to cooperate, there the goal of attribute based encryption is to isolate decryption power to the appropriate parties, comparable to role based access control. Such systems allow specifying attributes, where only the entities labeled with a specific attribute may decrypt a cipher text.

3.4.2. Privacy strategies

For IT systems that handle user information, data protection laws apply in many countries. Considering privacy implications (from early development phases on) allows creating systems that fulfill their functional goals while maintaining privacy properties.

Relevant influences to privacy properties not only arise from storage and processing of personal data, but also from the power of combining data from multiple sources to infer properties not directly visible from a single source. This fact indicates that a high level of diligence is required in analyzing such systems.

Based on an analysis of European data protection legislation, OECD guidelines and the ISO 29100 privacy framework, Hoepmann identifies eight design strategies for designing privacy preserving IT systems [90]. These strategies aim to help fulfill privacy principles and the respective requirements. They can be grouped in data oriented and process oriented strategies.

Data oriented strategies:

Minimize: The minimization strategy demands that only the minimally possible amount of data shall be collected, stored and disseminated. The principle of proportionality should be applied in the design.

Hide: Personal data and relationships between data should be *hidden*. This strategy suggests not making data accessible to other entities, where it may not be needed. For example, the creation of identifiers should be scrutinized in order to reduce likability.

Separate: By separating and compartmentalizing data processing, the profiling of persons can be impeded. When possible, data should be processed locally.

Aggregate: Data should be processed at the least possible detail in which data is still useful. The amount of aggregation directly influences the sensitivity of the data.

Process oriented strategies:

Inform: In order to provide transparency, individuals should be informed which information about them is processed. Any distribution to third parties is to be disclosed as well.

Control: In complementing the information strategy, individuals need to be able to exert their will about the fact that data is processed. Without information, control has little meaning. In the same vein, information without control has little practical impact.

Enforce: A privacy policy should exist and be enforced, e.g. by technical protections and organizational structures.

Demonstrate: The ability to actively show compliance to a policy by demonstrating the expected behavior shows that the implementation is functioning correctly.

These strategies formulate very generic approaches to deal with common problems in the processing of personally identifiable data. In general, not all of the strategies can be applied to a given situation. Applicable strategies have to be identified and combined to establish privacy properties in a system. These strategies are of use to the BaaS system, e.g. where a smart building may add value by identifying specific users or storing information about these users in order to provide tailored services to them. This kind of information would need to be protected from abuse.

3.5. Secure Authorization using OAuth 2.0

In device networks, such as building automation networks or the Internet of Things (IoT), secure communication is going to become a quite crucial issue. In particular, it has to be ensured that the access to resources (data, APIs etc.) on devices is controlled, i.e. unauthorized access to these resources is prohibited. General access control includes the elements authentication, authorization, access approval and accountability. A more narrow definition of access control is focusing on access approval. In this case, a system has to decide whether to grant or reject an access request issued by an already authenticated subject. This decision is usually based on an authorization model that describes what resources an individual subject or a role that the subject may assume is authorized to access. Authentication and access control are sometimes combined into a single step where the access to a resource is automatically granted if authentication has been successful or if an appropriate anonymous access token has been presented.

OAuth is an open protocol for allowing secure API/service authorization from desktop and web applications through a simple standardized method. OAuth provides client applications a secure delegated access to server resources on behalf of a resource owner. It provides a mechanism that allows resource owners to grant third parties access to their resources (usually hosted on a Web server) without sharing their credentials with them.

3.5.1. Introduction

In the common client-server authentication model, a client requests access to a protected resource by presenting the resource owner’s credentials (e.g. username and password) to the server. To facilitate access to protected resources for third-party applications (or devices), the resource owner has to share its credentials with that third party. This kind of procedure has quite a few drawbacks as listed below.

- Third party applications usually store the resource owner’s credentials for future use; presumably in clear text.
- Servers have to support password based authentication which has inherent vulnerabilities.
- Third party applications get full access to the protected resources; usually, there is no way to limit the duration or scope of that access.
- Resource owners are not able to revoke access from a particular third party without revoking access from all third parties because the password must be changed.

- If any third party is compromised, the end-user's password and all data protected by that password are compromised.

OAuth offers an alternative way to authenticate an application or device to a service. It is a security protocol that allows users to grant third-party access to their (web) resources without sharing their passwords. The heart of OAuth is a security token with limited rights and limited lifetime. If supported by the infrastructure, a user may revoke that security token at any time and thus prevent further access. As each client obtains a different token, revocation of a token does not affect any other client.

OAuth supports this "delegated authentication" between web apps using a security token called an "access token." To obtain access to a resource, the web app has just to present that kind of token; no other credentials are required. An OAuth token gives *one* client access to *one* API on behalf of *one* user.

Figure 10 illustrates with an example how data can be shared with an application using OAuth 2.0: The user provides the application (Game) with a token that allows it to access the user's data on the server (Facebook).

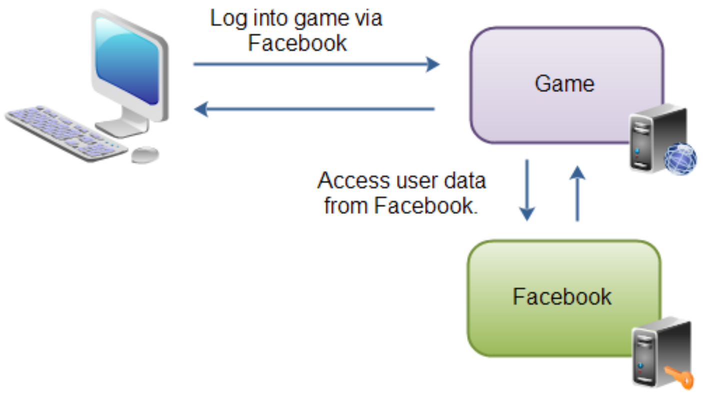


Figure 10: Example of how OAuth 2.0 is used (from [91])

The mechanisms of OAuth can be transferred to the IoT by providing an IoT device acting as a client with an access token that allows it to access the data on another IoT device acting as a server. No user must be involved in that kind of scenario; the client device itself requests the access token from an authorization server using its own credentials.

OAuth is already used by a large number of major Web players: Amazon, Dropbox, Facebook, Twitter, Google, Flickr, GitHub, Instagram, LinkedIn, MySpace, PayPal, Xing etc. A more complete list of OAuth service providers is given by Wikipedia [92].

Information, documentation and code regarding OAuth 2.0 is provided on the OAuth Community site [93]. An introduction to OAuth 1.0 is also available there [94]. There are several tutorials on OAuth 2.0, for instance [91] or [95], a video tutorial in 8 lessons.

Recently, several books on OAuth 2.0 have been published or will be in the near future: A guide to OAuth 2.0 for beginners [96], a comprehensive guide to OAuth 2.0 providing practical information for building clients and servers [97], and several eBooks on different aspects of OAuth 2.0 [98]–[100].

3.5.2. OAuth 2.0 Architecture

In the following, the architecture and communication scheme of OAuth are explained.

The OAuth architecture is based on the following roles / entities:

- **Resource Owner**: An entity capable of granting access to protected resources. The resource to be shared is usually data owned by the resource owner, but can also be an API providing access to some service. The resource owner may either a person or an application. OAuth 2.0 allows both possibilities.
- **Resource Server:** The server hosting protected resources. It is capable of handling client requests asking for access to the protected resource. In particular, it must be able to verify the validity of the access tokens presented with the request. This may include a check if the token has been revoked since it has been issued.
- **Client Application**: An application making protected resource requests on behalf of the resource owner and with its authorization. The request includes an access token which is presented to the resource server providing the protected resource. If the access token proves to be valid, the application gains access to the resource.
- **Authorization Server**: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization. Authorization server and resource server may be collocated on one machine. The OAuth 2.0 specification does not describe how the two servers should interact, if they are separate.

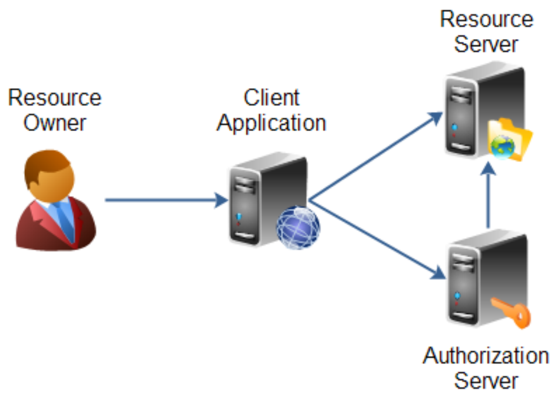


Figure 11: OAuth 2.0 roles as defined in the specification (from [91])

Figure 11 shows the roles/entities used by OAuth and the relationships between them. In case of device networks, such as building automation systems, the application acting as client is hosted on an embedded device and usually has its own schedule when to access another embedded device acting as a resource server; a resource owner is usually not participating in the procedure.

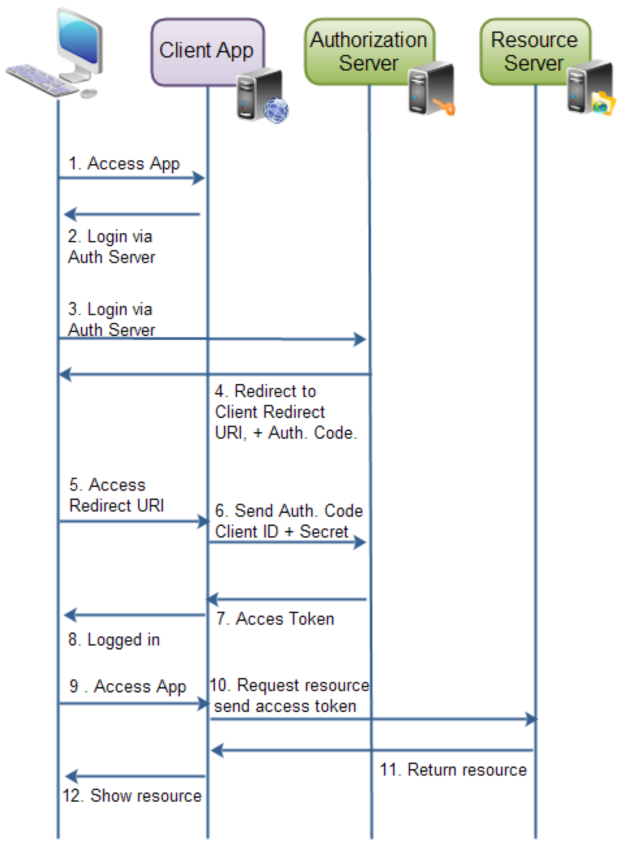


Figure 12: OAuth 2.0 authorization sequence (from [91])

Figure 12 describes the sequence of interactions required to obtain an authorization grant and steps are listed below.

1. The resource owner (user) accesses the client application.
2. The client application advises the user to login to it via an authorization server, and redirects the user to such a server. The client application provides its ID to the authorization server to inform him who requests access to the protected resource.
3. The user performs a login via the authorization server. After successful login, the user is asked if the client application should be granted access to the protected resource.
4. After granting access to the protected resource to the client application, the user is redirected back to a specific redirect URI of the client application that it has registered previously at the authorization server. Together with the redirection, the authorization server sends an authorization code.
5. When the redirect URI in the client application is accessed, it connects directly to the authorization server.
6. The client application sends the authorization code along with its own credentials.
7. If the authorization server is ready to accept these values, it sends an access token back to the client application.
8. The login procedure is reported to be complete.
9. The user can now access the client application to request an action on the resource.
10. The client application can now use the access token to request the protected resource from the resource server.
11. The resource server validates the access token. If this is successful, the resource server returns the resource to the client application.
12. The client application can now present the obtained resource to the user.

3.5.3. OAuth 2.0 Standards

Work on OAuth started around 2006 as a complementary activity to the definition and implementation of OpenID [101]. In April 2007, a discussion group was created to write the draft proposal for an open protocol. On October 3, 2007, the OAuth Core 1.0 final draft was released [102]. After the decision in November 2008 to bring OAuth into the IETF for further standardization work, the IETF OAuth Working Group [103] was started. The working group already released a number of RFCs:

- RFC 5849 "The OAuth 1.0 Protocol" [104]
- RFC 6749 "The OAuth 2.0 Authorization Framework" [105]
- RFC 6750 "The OAuth 2.0 Authorization Framework: Bearer Token Usage" [106]
- RFC 6819 "OAuth 2.0 Threat Model and Security Considerations" [107]

OAuth 2.0 differs considerably from OAuth 1.0; and there is no backward compatibility. OAuth 2.0 targets to simplify client development while offering specific authorization flows for web applications, desktop applications, mobile phones, and living room devices.

The IETF working group is still active and working on a number of Internet drafts. The titles of the active working group drafts are given below; the full list of drafts is available on the status page of the IETF OAuth working group [103].

- OAuth 2.0 Token Revocation
- OAuth 2.0 Dynamic Client Registration Protocol
- JSON Web Token (JWT)
- JWT Profile for OAuth 2.0 Client Authentication and Authorization Grants
- SAML 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants
- OAuth 2.0 Message Authentication Code (MAC) Tokens

3.5.4. OAuth 2.0 Features

This section describes major features of OAuth 2.0 that discriminate it from OAuth 1.0.

3.5.4.1. Specific Authorization Flows

OAuth 2.0 provides specific authorization flows for particular types of clients and use cases.

- **User-Agent Flow**: for clients running inside a user-agent (e.g. a web browser).
- **Web Server Flow:** for clients that are part of a web server application, accessible via HTTP requests. This is a simpler version of the flow provided by OAuth 1.0.
- **Username and Password Flow:** used in cases where the user trusts the client to handle its credentials but still does not allow the client to store its username and password. Only applicable if there is a high degree of trust between user and client.
- **Client Credentials Flow:** the client uses its credentials to obtain an access token. This flow supports what is known as the 2-legged scenario. This flow is appropriate for authorization in the device networks like building automation systems.
- **Assertion Flow:** the client presents an assertion such as a SAML (Security Assertion Markup Language) assertion to the authorization server in exchange for an access token.

3.5.4.2. Bearer tokens

OAuth 2.0 provides a cryptography-free option for authentication that is based on existing cookie authentication architecture. Instead of sending signed requests using Keyed-Hash Message Authentication Code (HMAC) and token secrets, a so-called bearer token is used as secret. Any party possessing such a bearer token can use it to get access to the respective resources (without the need of a cryptographic key). To prevent misuse, bearer tokens need to be protected from disclosure in storage and transport. Bearer token transport can be secured by using HTTPS exclusively.

The advantage of this approach is that if HTTPS is used for secure transport in a system, no other cryptographic mechanism beside the Transport Layer Security (TLS) protocol is required. However, some people consider this as a dangerous feature because it's "putting all your eggs in one basket" [108].

3.5.4.3. Short-lived tokens with long-lived authorizations

Instead of issuing a long lasting token (typically good for a year or unlimited lifetime), the server can issues a short-lived access token and a long lived refresh token. This allows the client to obtain a new access token without having to involve the user again, but keeps the lifetime of access tokens limited.

The advantage of short-lived access tokens is that no complicated revocation mechanisms are required to withdraw a granted authorization from a client.

3.5.4.4. Separation of roles

OAuth 2.0 separates the role of the authorization server responsible for obtaining user authorization and issuing tokens from the role of the resource server responsible for handling API calls. In contrast, OAuth 1.0 does not distinguish between the roles authorization server and resource server.

This feature is of some relevance for device networks as it simplifies the implementation of the resource server (that actually may be a quite small device) by outsourcing the task of authorization to a separate authorization server (that may be a larger computer).

3.5.5. Relevance of OAuth for BaaS

OAuth is an access control mechanism for clients accessing resources on Web servers that recently gained a lot of attention. The OAuth 2.0 Framework offers alternative flows that open OAuth for new types of scenarios. In particular, client credential flows (2-legged scenario) seem to be applicable for device networks as building automation systems.

While there is some criticism regarding the security level and the complexity of the OAuth 2.0 specification raised by the former OAuth working group leader [109], it is generally considered as a sufficiently secure authorization and access control mechanism. In particular, if combined with the HTTPS protocol, which is increasingly common in today's Web infrastructures, no other cryptographic mechanism beside TLS (upon which HTTPS is based) is required.

For these reasons, several authors recommend use of OAuth 2.0 for the Internet of Things (IoT) and its Web based implementation, the Web of Things (WoT). These works are briefly overviewed below.

- On a Wiki page of the W3C Web of things Community Group titled “General considerations for the Web of Things”, OAuth is considered as a possible mechanism for access control in the WoT [110].
- In two presentations titled “Securing the Internet of Things” and “Federating Access to IoT using OAuth” the use of OAuth in combination with CoAP and MQTT, a very lightweight messaging protocol is investigated [111], [112].

Finally, a proposed addendum to the BACnet standard introducing “BACnet Web services” uses OAuth 2.0 as access control mechanism [113]. Another extension of BACnet, the “BACnet Internet Transport Binding” (ITB) that is currently being specified will also use OAuth 2.0 for secure authorization.

This demonstrates that OAuth is considered to be suitable for the WoT in general and Web-based Building Automation Systems in particular. As OAuth has also been adopted by extensions to the BACnet, it seems to a reasonable choice for the BaaS project as well. This is emphasized by the fact that the combination of OAuth and the CoAP protocol has already been investigated.

3.6. Functional Safety and Reliability in Service Systems

Due to the fact that different devices and components are spread across buildings, it is obvious to think of BAS as distributed systems. A further abstraction or perspective is to assume BAS as service systems where each device and component is represented by services.

Due to BAS controlling the environment where people live respectively work, it is necessary to impose requirements on the behavior and quality of the underlying system. Another point requesting functional safety and reliability in service systems is the apparent complexity arising from applying a service system in the field of building automation, where various services are combined to control respective parts of buildings. Without any requirements on the involved services, it is not possible to assure certain qualities of such a complex service system.

However, hard requirements for functional safety, reliability and predictability, however, are in contrast with the flexibility, dynamic adaptation and dynamic configuration properties, which are typically achieved with service-oriented architectures. Applying service systems for building automation therefore needs approaches which successfully can bridge that gap.

The following subsections give further background about

- metrics
- safety and reliability related standards
- common principles and requirements
- common methods targeting safety and reliability

The last subsection gives a short overview of approaches and ongoing work in the building automation domain.

3.6.1. Metrics

The functional safety and reliability is according to [114], [115] quantifiable by the following parameters:

- **Error Probability:** Probability, that a system running error-free at the beginning becomes erroneous after a certain period of time.
- **Probability of Surviving:** Probability, that a system running error-free at the beginning works without any error until a certain point of time.
- **Mean Time to Failure:** Expected value of time until first occurrence of an error.
- **Failure in Time:** describes the proportion of failing components relating to the number of working components during a certain time interval.
- **Availability:** Probability, that a system is running free of errors to any point of time; this parameter is relevant for systems being considered as guarded by an error treatment transferring the system after occurrence of an error to an error-free state.

3.6.2. Failures and Errors affecting Safety and Reliability

As stated by [116], [117] there are different causes resulting in error-prone systems. Basically the safety and reliability of systems is affected by failure of hard- and software, errors in software and wrong manual user intervention. Failures of hard- and software can as well as errors in software be subdivided into random and systematic [117].

Echtle describes in [114] the causes of failure in detail. Failures emerge either during design time, production or runtime, which can be differentiated according to [114] as follows:

- **Design Time:** Specification, Implementation and Documentation Failures
- **Production:** *no further differentiation*
- **Runtime:** Fault-based, Attrition-based, Random Physical, User Intervention and Maintenance Failures

3.6.3. Safety and Reliability Related Standards and Guidelines

Safety and reliability related standards are mentioned in several publications [117]–[121]. The following standards give several guidelines and procedures how to achieve a certain level of safety and/or reliability.

- **ISO 26262:** This standard has been developed for functional safety in road vehicles. It is the successor of the IEC 61508. The ISO 26262 standard is divided into ten volumes describing: 1. Vocabulary, 2. Management of functional safety, 3. Concept phase, 4. Product development: system level, 5. Product development: hardware level, 6. Product development: software level, 7. Production and operation, 8. Supporting processes, 9. Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses and 10. (Informative) Guidelines on ISO 26262 [118], [121].
  Volumes 4-6 contain common methods for the system development process, which might be adapted and interesting in the development of a complex distributed system as targeted in the BaaS project.
- **IEC 61508:** The IEC 61508 is the standard the ISO 26262 standard is derived from. It is a cross-sector generic guideline for safety-related systems. It is as well as the ISO 26262 divided into several parts. The first four parts are normative and form the actual guideline whereas the last three parts have only an informative character. Part 1 describes common principals which should be followed if no domain specific standards exist. It covers the whole lifecycle of a safety-related system. Part 2 contains guidelines concerning hardware aspects. As part of the overall lifecycle a safety lifecycle for the

hardware level has to be defined. Part 3 comprises techniques and methods how safety-related software should be developed and documented. Moreover are the known safety-integrity levels (SIL) described in this part, which are a method to gain requirements for the development process and the software architecture. The last important part 4 is a glossary including all terms and abbreviations used in the standard [119], [120].

- **DIN EN 61508 (VDE 0803):** This standard is the German version of the IEC 61508 described beforehand [117].
- **ISO 13849:** The ISO 13849 standard "specifies safety requirements and a basic guideline how to develop and integrate safety-related parts of a control system […] including software development following the stages" hazard and risk analysis, safety requirements specification, safety analysis and safety validation [120]. Similar to SIL the "standard specifies 5 performance levels where each level equals a probability of a *dangerous* failure per hour […]. The target level or risk is delivered as a result of the risk analysis carried out according to the requirements of ISO 14121" [120].
- **VDI/VDE 2184:** A more specific standard targeting fieldbus systems is the German guideline VDI/VDE 2184. Due to fieldbus systems being the core infrastructure in building automation systems, this guideline should be considered developing the BaaS platform. It "gives requirements for various life-cycle activities to be met when using fieldbus systems in industrial automation areas demanding regarding safety and timing behavior. The life-cycle model and generally activities relating to functional safety are according to IEC 61508." [120]
- **Common Criteria ISO/IEC 15408:** As described in [122] is the Common Criteria (CC) "a basis for evaluation of security properties of IT products and systems. CC specifies a set of requirements for the security functions of IT products and systems. Additionally, it gives requirements for assurance measures applied to the security functions during security evaluation."
- **EN ISO 14121:** This standard "establishes general principles intended to be used to meet […] risk reduction objectives […]. These principles of risk assessment bring together knowledge and experience of the design, use, incidents, accidents and harm related to machinery in order to assess the risks posed during the relevant phases of the life cycle of a machine." [123]

3.6.4. Common Principles

The respective literature gives several answers on how to achieve functional safety and reliability in service systems. Following the subsequent principles the operation of dependable systems can be supported.

- Error and failure detection through suitable monitoring [124]
- Assurance of an emergency mode in case of errors and failures [124]
- Avoidance of mistakes during all the phases of the system life-cycle [117]
- Prevention, tolerance, removal and forecasting of faults and threats [125]
- Redundancy of important and inalienable system components [117]

In [114], Echtle structures the field of dependability into measures, impairments, procurements and analysis. Dependability is quantifiable through reliability, time to failure and availability. These measures are influenced by errors and faults or failures. To verify the dependability of a system, Echtle states on one side verification and on the other side error forecasting as methods for the analysis of systems. According to [114] there are basically two procurements covering the improvement of system dependability: fault avoidance and fault tolerance. The field of fault tolerance is divided into fault specifications, redundancy, fault diagnosis and error processing. The error processing includes different strategies to handle errors and faults at system runtime to achieve fault tolerant and thus dependable systems. A simplified version of the structure by Echtle is depicted in Figure 13.

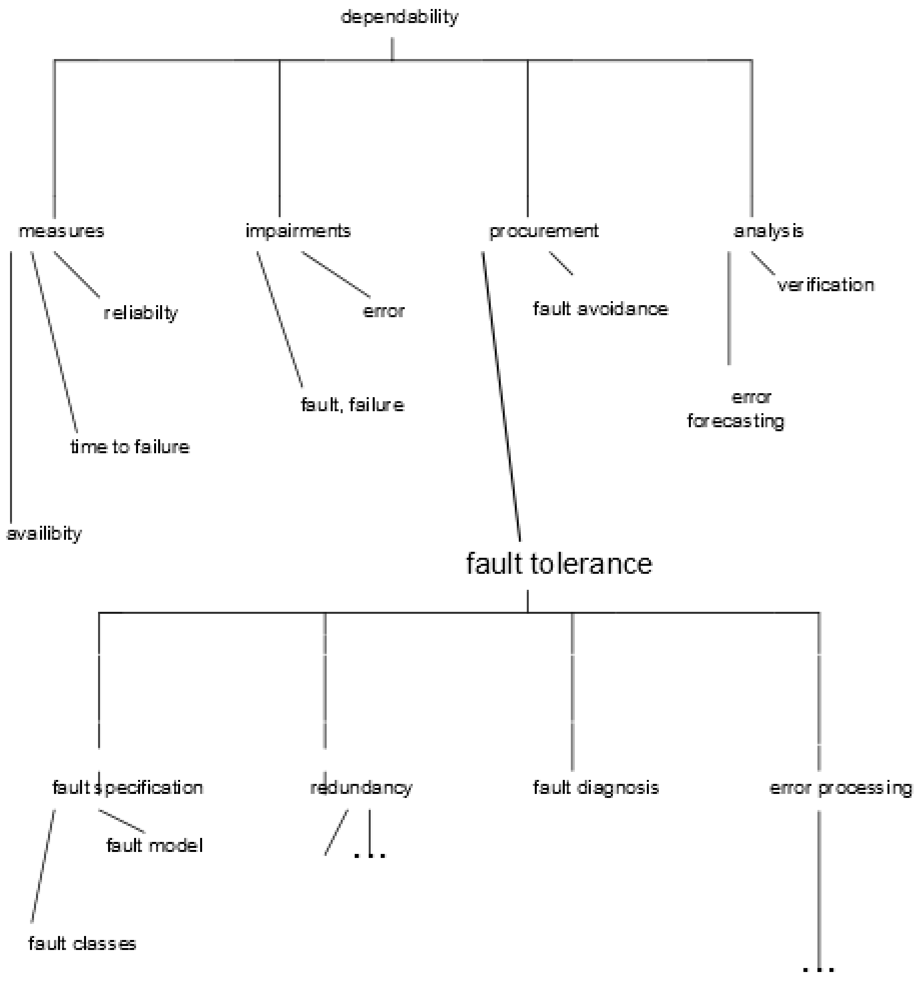


Figure 13: Dependability measures (from [114])

3.6.5. Common Methods Targeting Safety and Reliability

There are many methods approaching and supporting safe and reliable systems. They follow different ideas and thus are not mutually exclusive.

Habermann and Burton [118] propose to ensure safety in the process of system development. This could be achieved through model-based development. Model-based development gives the ability to create a system model with specific constraints like safety and reliability. Due to the automatic derivation of artifacts for the automatic system management there is less probability of errors and faults which can occur through manual development of the management system. A similar idea is stated in [126] by Rodrigues et al. who want “to support reliability design following the principles of the Model Driven Architecture (MDA). By doing this, [they] aim to contribute to the task of consistently addressing dependability concerns from the early to late stages of software engineering”.

Another way of improving the safety and reliability of systems is to use certain guidelines for the development of systems. As described in [117], [120], [122] there are different levels of certain qualities to assure safety of developed systems: Safety Integrity Level (SIL), Evaluation Assurance Level (EAL) and Performance Level (PL). The guidelines should provide a method to achieve a particular level of safety and reliability through the consideration during the development phase.

Elzer describes in [124] that the reliability of systems can be enhanced by using diversity in the system infrastructure. The diversity can be applied to hardware, software, functional aspects and operating conditions. Through this approach and redundancy of components it is less probable, that the whole system stops working properly in case of failing components.

Another approach to prevent errors and faults is to use different analysis techniques as described in [116], [118]. Biegert [116] follows a model-based approach. He proposes to use the system models developed at design time to analyze certain aspects of the resulting system. Habermann and Burton [118] propose a model-based safety analysis as well. They emphasize that the model-based approach allows to automatically analyze dependencies between different levels of the architecture and gives an opportunity to evaluate the system safety or other properties. Another advantage is, that derived actions and measures can directly affect the model and enhance the corresponding system property. This analysis method enables the combination of system design, specification, development and analysis.

There are more reliability and availability prediction methods which are summarized and surveyed in [115]. Immonen and Niemelä define a framework based on Normative Information Model-based System Analysis and Design (NIMSAD) to compare and evaluate different prediction methods. According to their survey there is currently no approach fulfilling all requirements they have considered during evaluation. Their main concern is that "the surveyed approaches […] did not commit themselves to [reliability and availability (R&A)] requirements at any level. Therefore, they failed to define how R&A requirements could be transformed into different architectural decisions and how architectural decisions could be traced back to requirements."

3.6.6. Related Work in the Building Automation Domain

Functional safety in the building automation domain has already been approached among others in [120], [122], [125], [127].

Kastner and Novak survey "general safety standards […] and point out what requirements have to be met by a safety-related building automation technology." Especially they evaluate the possibility of applying safety-related standards like IEC 61508, ISO 13849 and VDI/VDE 2184 to KNX/EIB which is an automation networking technology used in the building automation systems without any safety support [120].

Novak et al. present in [122] and [125] how to engineer a building automation technology taking safety- and security into account. They focus on a life-cycle model which covers all necessary steps from development to operation considering safety and security in all phases. They point out that one main challenge is to harmonize the requirements occurring from safety and security being in contradiction to each other. The requirements derive from hazard, threat and risk analyses which are part of the life-cycle model. As well as in [120], they follow the IEC 61508 standard.

Dongbo et al. follow a different approach to functional safety described in [127]. They emphasize the common structure of building automation and control systems and point out that devices, the communication and controller components in such an infrastructure have to provide and fulfill different functions for a safe and reliable system. They have a more infrastructure-centric approach and argue in terms of reliability of involved components how to achieve a safe building automation and control system.

3.7. OSGi

The focused development of software systems, meaning a focus on the implementation of business logic without the redevelopment of basic functions by reusing already developed is a paradigm that is explored in software engineering for a long time. The component-based software development has influenced this research and provides an approach where comprehensive applications can be assembled of individual software components, provided by various vendors, in a flexible way.

In the early stages, monolithic structured systems dominated the software world, but these systems were then structured more and more fine-grained. Started with the introduction of layered architectures, the division into logically related components continued the trend, until the division into components was achieved.

The idea to make the complexity of software manageable by decomposition into modules was already described by David Parnas in 1972 in [128]. In the component-based software development, however, the aspect of components commercial exploitation is increased. This novel approach has thereby quickly achieved a high level of acceptance and market relevance [129]. Thus, comprehensive component frameworks have been developed in recent years, such as the CORBA Component Model [130], the Component Object Model (COM / DCOM) [131], the Enterprise Java Beans and the OSGi framework.

The OSGi specification [132], realized by the OSGi Alliance, follows the paradigm of component-based software development and defines a dynamic, service-oriented component model for Java. It is a software platform that enables the dynamic integration of independent software components (bundles) and services (services). At runtime, the components can be installed in the Framework, started, stopped and uninstalled without restarting the entire platform. The individual components communicate with each other via services. In this way, complex applications can be easily realized through the composition of these components.

Originally, OSGi was designed for the use in so-called Residential Internet Gateways [133], [134]. The platform serves as a central connection between heterogeneous in-house networks and the Internet in the field of building automation. Nowadays, OSGi is also used for telematics and infotainment systems in the automotive field [135], serves as the basis for the Eclipse platform [136] and is becoming increasingly popular in the field of telemedicine [137].

3.7.1. OSGi Remote Services

At the beginning, the OSGi specification realized only a local component-based platform within the boundaries of a Java VM. Over time, however, more and more applications came into existence, in which communication between OSGi services of different platforms was desirable or necessary. But up to version 4.1 of the OSGi specification this was only realized by research outside the official specification. One of the first distributed OSGi platforms was presented in 2005 within the Newton project, which enabled a distributed communication based on the Service Component Architecture (SCA) [138]. Rellermeyer's R-OSGi [139] is perhaps the most achieved attempt to transparently distribute services on several OSGi platforms. R-OSGi goes further first with a dynamic proxy generation refined with ASM [140] bytecode generation and moreover with a transparent use of local and remote services. The Comoros project took up this point and developed an OSGi middleware based on the DPWS. In addition to the transparent use of local and remote services, legacy services could be distributed without adaptation and, for the first time, native devices and services could be used within an OSGi platform in a transparent manner. Other projects, such as Nyota [141], also allowed a distributed communication, but were changing the core of the OSGi platform and were therefore not completely compatible with the specification.

Parallel to this research the OSGi Alliance developed a specification for a distributed service usage. The RFC 119 (Distributed OSGi) [142] was the first release by the OSGi Alliance. This specification was taken up by the Apache CXF project [143], which has since been regarded as a reference implementation. In version 4.2 of the OSGi Compendium specification the standard was eventually finalized under the name OSGi Remote Services. As a result, existing projects, such as R-OSGi and Comoros have been adapted to this specification. Since the specification leaves much technical space, many different implementations of the specification were developed henceforth. In the Tuscany project an SCA container serves as the implementation of a distribution provider. Within Amdatu, multiple protocols, serializations and discovery mechanisms were implemented. There are variants with HTTP + JSON, HTTP + JavaSerialization for protocols and serialization and with SLP, MulticastDNS and Hazelcast for the discovery. The Eclipse Communication Framework (ECF) [144] focused, apart from the synchronous communication, especially the asynchronous communication and thus developed a Remote Event Admin. Another special type of communication is presented by Ibrahim et al. [145]. Here requests are collected and are transmitted bundled to the client side. In that way, the communication overhead should be reduced in environments of embedded systems. Further implementations of the standard remote services are realized in the Corba-based Service Oriented Framework (SOF), the Karaf project and the Fuse Fabric project.

3.7.2. OSGi Device Integration

In addition to the distributed communication between OSGi platforms, the integration of services and devices of third party technologies is of key importance for the OSGi environment. Due to the original orientation of OSGi as Residential Internet Gateways, the device integration was an integral component of the platform from the beginning. Within the Device Access Specification the handling of devices is specified. The specification describes the discovery of devices, the subsequent linkage with the OSGi framework, as well as the downloading and installation of drivers at runtime to support a hot-plug capability of devices. Currently there are a number of Base Driver implementations for different technologies. The UPnP Base Driver can be considered as the reference implementation of the specification and is now part of the OSGi Compendium specification [146]. Similar technologies are supported with the publication of the DPWS Base Driver [147] and the JINI Base Driver. Within the research project GiraffPlus, part of the AALOA initiative, the ZigBee technology support was implemented which is highly relevant especially in the area of ambient systems. Further Base Driver implementations exist for the technologies Bluetooth, USB and Tmote [148]. Within the OSAmI project, a novel device integration concept was developed by extending the Device Access Specification to encapsulate the services functions as services in terms of a service-oriented architecture [149].

Out of the specification are projects like IoTSys, a work of the University of Vienna, that provides a complete protocol stack for the integration of building automation systems in the Internet-of-Things-World [150], [151]. The Eclipse project SODA [152] is also based on OSGi and addresses the same problem like the OSAmI Device Integration. SODA is not compatible to the Device Access Specification but introduces a layered architecture which allows applying SOA principles, e.g. the composition of services.

3.8. RESTful Web Services

Representational State Transfer (REST) was developed as a way of evaluating architectures of distributed network applications [153] by Roy Fielding for his Ph.D. Thesis [154]. Most of the information on REST in the beginning has been like best practices and has been distributed mainly through informal communication channels such as email lists. Leonard Richardson's and Sam Ruby's book "RESTful Web Services" [153] has been the first comprehensive book on the principles and definitions of Representational State Transfer, "RESTfulness", and its application to Web Services.

In principle, Representational State Transfer (REST) is a general architectural style independent of specific protocols. However, REST is more or less exclusively used in the World Wide Web and the Internet. In some sense, REST is providing a machine-readable web compared to the human-readable web we know from our everyday interaction with the World Wide Web. Consequently, REST is usually connected with HTTP.

3.8.1. Architectural Constraints

REST has the following architectural constraints (collection is based on [155]):

- Client-server communication: The communication according to REST follows a strict client-server model. This leads to a strict separation of functionality or "concerns". For instance, servers are concerned with data storage, clients aren't. Clients are concerned with the user interface, servers aren't. The client-server model improves portability and scalability.
- Statelessness: No client context is stored on the server between requests by the client. This means, that no state about the client is stored on the server after the request of the client has been handled. This requires self-descriptive requests, which allows to distribute multiple, successive requests of a client to different servers with the same functionality. Statelessness improves scalability and reliability.
- Cacheability: Responses are explicitly marked as cacheable or non-cacheable. Cacheable responses can be stored in intermediate devices ("caches") between the server and the client. Cacheability improves scalability and performance.
- Layering: The client connects with the server. The actual communication, however, is transparent to the client. The client may talk to the server directly, or to an intermediary along the path, to a server farm for load balancing, to a cache or to any

other device on the WWW that provides the required server functionality. Layering improves scalability, reliability, and performance.

- Uniform interface: The uniform interface between client and server simplifies the communication between clients and servers. It decouples the two important architectural components – clients and servers – from each other so that both components can evolve independently. There are four guiding principles for the uniform interface:
    - Identification of resources: Individual resources are identified in requests by Uniform Resource Identifiers (URIs). The resources themselves are conceptually separate from their representations in the response to the client. For example, the server does not send its database but some description in a standardized way (e.g. HTML, XML, or JSON).
    - Manipulation of resources through their representations
    - Self-descriptive messages: Every message contains all the information that is necessary to describe the required processing of the message.
    - Hypermedia as the engine of application state: clients make state transitions only through actions that are dynamically identified within hypermedia from the server. A client does not assume availability of any other action for a particular resource beyond those described in representations previously received from the server (plus simple, fixed entry points to the application on the server).
- Code on demand (not mandatory): Servers can temporarily extend or customize the functionality of a client by the transfer of executable code such as Java applets or client-side JavaScripts.

Applications and services conforming to the architectural constraints of REST are called “RESTful”. Or the other way round, an application or service cannot be considered “RESTful” if it violates any of the architectural constraints of REST.

3.8.2. General Principal and Concept

REST can be considered as a well-designed Web application [155]: The user (client) is able to connect to a network of Web pages (on the servers) in order to progress through an application by selecting links leading to the next Web page which is transferred to the user and rendered for his use. The selection of a link corresponds to a state transition of the application, and the next Web page represents the next state of the application.

REST in the WWW uses HTTP for the communication between client and server. However, “RESTfulness” is not a protocol but an architectural concept. Therefore, RESTful architectures can be developed with any set of protocols that is able to fulfill and is following the architectural constraints of REST as given in 3.8.1.

HTTP provides all the necessary means for setting up a RESTful distributed network-based application in the World Wide Web. Moreover, HTTP is a native protocol of the WWW leading to a natural integration of REST into the WWW.

REST is simple and well-defined. Simplicity and lack of unnecessary features are its strength and power [153]. It is using the basic web protocols such as HTTP. This makes RESTful services being part of the Web instead of just being “on” the Web [153] (or running over the Web).

3.8.3. REST applied to Web Services

A Web Service API is called RESTful if it adheres to the REST architectural constraints as given in 3.8.1. A RESTful API is defined according to the following aspects:

- Base URI
- Internet media type such as JSON or XML
- Standard HTTP methods: GET, PUT, POST, DELETE
- Hypertext links to reference
    - state of the application on the server
    - resources

3.8.4. Application Examples

Well-known applications on the WWW, that are RESTful, are Amazon and Google Maps [153].

3.9. Efficient XML Interchange (EXI)

Efficient XML interchange is a binary format for XML. It has the status of a W3C recommendation since 10th of March 2011, is available in the second edition [156] since February 11th 2014 and has been produced by the EXI Working Group. The general approach of EXI is to encode most probable content of the XML documents with less bits, which is similar to the Huffman encoding [157]. The process of generating and parsing EXI is state machine based. The corresponding state machine is called EXI-Grammar and it reflects the XML-schema that is used. In which way this grammar is build depends on the selected mode of EXI. On the one hand there is the schema-informed mode where the grammar is generated out of a XML-Schema document. If the mode is additionally set to strict the EXI-Grammar cannot be changed during runtime. In some cases however it could be useful to handle unexpected elements. Then the non-strict mode is to be used. In this case unknown XML elements are added to the EXI-Grammar when they occur. On the other hand, there is the schema-less mode. In this case the EXI-Grammar is generated only by a set of XML-documents and is still extendable during runtime.

	schema-less	schema-informed strict	schema-informed non-strict
compressed	Byte- aligned	Byte- aligned	Byte- aligned
uncompressed	Bit- and Byte-aligned	Bit- and Byte-aligned	Bit- and Byte-aligned
deviation of the EXI-Grammar	Yes	No	Yes

Table 4: EXI format modes

A concrete EXI format is called EXI-Stream. The structure of an EXI file is only determined by the state changes. The occurrence of state changes or other specific content elements is called EXI-Event. Therefor the EXI-Stream consists of EXI-Events and the related content. In EXI it is possible to send the EXI-Events and the content in separate containers so they don’t have to follow each other directly. This makes it possible to apply additional generic compression algorithms on the stream to reduce the size more effective because of repeated EXI-Events. This mode is called compressed which uses the IETF standardized RFC1951 [158] deflate algorithm. In contrast there is a mode called uncompressed where the encoded EXI-Stream is send without compression.

For confusion, the uncompressed mode does not mean that there is not the possibility to reduce the file size. Most systems using byte as smallest possible storage unit, which means a number from 0 to 256 can be stored. But for most states in an EXI-grammar there are less than 256 possibilities to go on. For this purpose there is a bit-aligned mode which allows EXI to move away from byte-alignment. In this way EXI only uses the number of bit suitable for the number of possible events which, together with the fore mentioned Huffman coding, reduces the size of the stream. In [159], EXI is compared to other generic and XML compression algorithms and shows much better size reduction in DPWS and web-service XML documents.

Beside the compact message sizes of EXI that provokes, e.g., the reduction of network traffic, EXI messages are quite simple and fast to process as well as have a very low memory usage. This is justified by the fact that EXI operates on a set of simple grammar structures which reflect, e.g., a given XML Schema definition. Figure 14 shows an EXI grammar construct that represents the well-known SOAP framework. This grammar is built by the EXI options schema-informed, strict, and bit-aligned.

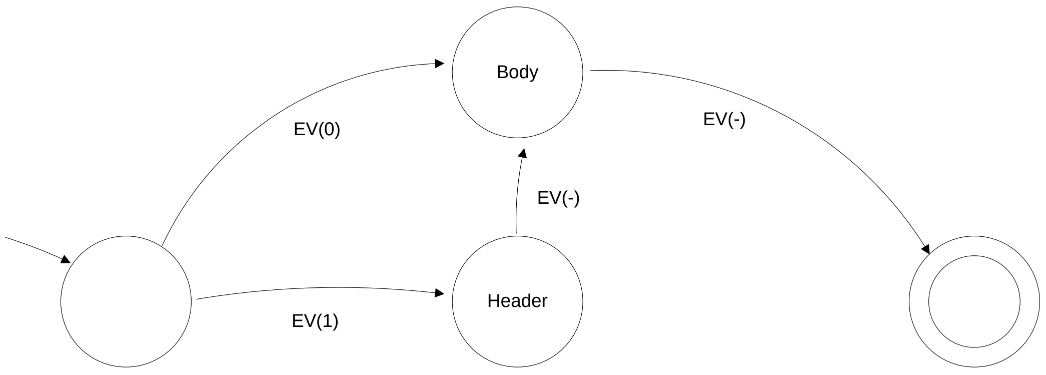


Figure 14 Sample EXI grammar (*Envelope* grammar) of the SOAP-Envelope framework

The start state corresponds to the optional element of the *Header* element in the SOAP XSD definition by the transitions with the event code *EV(1)* (Header element is present) and *EV(0)* (Header element is not present). To illustrate the simple encoding mechanism of EXI let us consider the following message snipped

<Envelope><Header> … </Header><Body> … </Body></Envelope>

EXI would start to apply the *Envelope* root element to a default *root* grammar which typically reflects all global elements defined in the XML Schema. Let assume the transition to the *Envelope* state is assigned with the event code EV(00). Next, we go to the *Envelope* grammar which is shown in Figure 14. Since the *Header* element is present in the snipped, we follow the transition with the event code EV(1). So far, we only spend 3 bits to represent the SOAP framework:

00 1 …

This already shows how efficient the EXI format is. Since EXI is compliant to the XML InfoSet [160], we are also able to operate direct on the EXI stream to retrieve the XML-based date. Furthermore, EXI is a type-aware encoder which enables us to directly use the values in the applications without any type conversations such as *string* to *int* etc. Based on this benefits, EXI is very suitable and feasible in environments which are based on constrained resources such as from microcontrollers [161]. In that context, the W3C EXI Profile [162] can be applied to optimize the memory usage at runtime.

When using EXI it must be noticed that EXI Specification does not define a mandatory mechanism to negotiate or exchange EXI-Grammar or used schema-documents to ensure that the communication partners use the same EXI-Event encoding.

There are several EXI implementations that can be found such as:

- EXIficient [163] is an open source Java implementation for EXI encoding, parsing and Grammar generation
- uEXI [164] is an open source EXI parser written in C aiming at a small footprint as well as exip [165]
- hardexi [166] is a to be published hardware based EXI parser with significant performance increase compared to software implementations.
- openexi [167] is a project to develop open source EXI implementations where the java implementation is currently available and a C# version is in progress.

3.9.1. Relevance of EXI for BaaS

XML is a well-known, platform-independent exchange format with the opportunity to model the data content quite precisely with XML Schema definitions. However, plain-text XML has a negative impact on processing, memory, and bandwidth usage. Since the BaaS project also considers constrained embedded devices, EXI would be a good approach to support an end-to-end XML-based messaging. Furthermore, this would also go in hand in hand with the constrained application protocol (CoAP).

3.10.Constrained Application Protocol (CoAP)

The Constrained application protocol (CoAP) is a protocol of the application layer. It is intended to be used on constrained devices for machine to machine (M2M) communication over IP based networks. Since June 2014 CoAP has been ratified as IETF Standard RFC-7252 [168]. CoAP is already used in several research projects in the area of sensor networks [169][170][171] and the Internet of things (IoT) [172]. CoAP is very similar to the Hypertext Transfer Protocol (HTTP) as sown in Figure 15 but is adapted to resource constrained devices and networks. Therefor it is a RESTful protocol that uses the well-known methods GET to get resources, POST to modify them, PUT to create new resources and DELETE to delete them. The methods are handled in a request response scheme in an asynchronous way. The payload can be any text which includes JSON, XML, EXI and many more. Especially the possibility to use CoAP together with EXI has a good perspective in the M2M communication on constrained networks and devices [173]. In contrast to HTTP, CoAP uses UDP instead of TCP on the transport layer. This reduces the overhead that is made on this layer. Nevertheless CoAP has the possibility to handle reliability, fragmentation and deduplication of messages on its own if needed. Additionally CoAP reduces overhead by coding the message header in a binary way. The efficiency is further enhanced by the possibility to cache CoAP messages on less constrained devices by providing the Max-Age header option. Because of the similarity to HTTP there is the possibility of cross-protocol-proxies between CoAP and HTTP networks to raise interoperability to other networks like the internet.

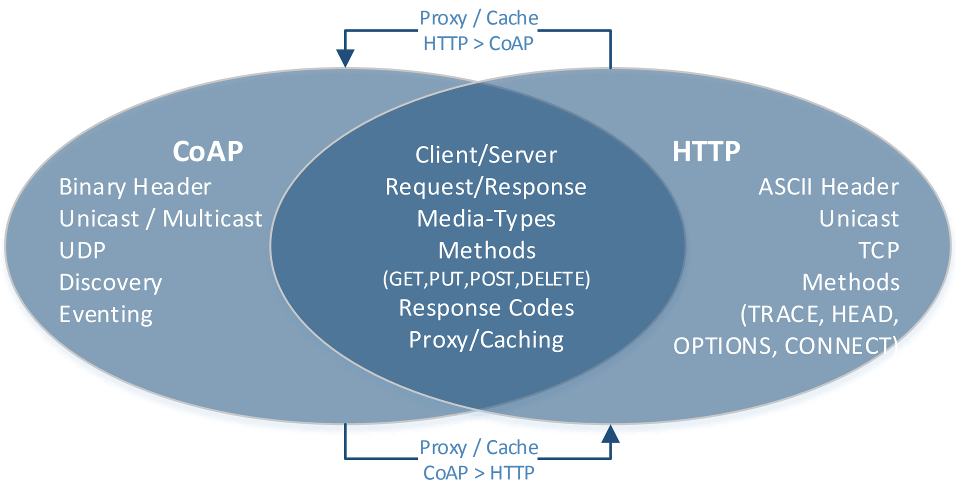


Figure 15: Comparison CoAP and HTTP [159]

In CoAP there are four types of messages that can be used. These are: confirmable (CON), non-confirmable (NC), acknowledgement (ACK) and reset (RST) messages. A CON message contains a message ID. Those messages are resend if the sender does no receive an ACK with the matching ID within an exponentially raising time. Duplicated packets are identified by the ID as well. The response can be attached to the ACK message to reduce bandwidth. If the computing of the response takes too long, it is also possible to send an empty ACK and later on a new CON message with the response. RST messages indicate that the given request cannot be handled by the receiver. The alternative way is to send NC messages. A communication that uses NC messages is connection less so the receiver does not need to send ACK or even react.

Besides RFC 7252 there are some related standards (see Figure 16).

- RFC 6690 “CoRE link format” is a RFC standard already. The CoRE Link Format defines a fixed resource (.well-known/core) on CoAP servers which allows obtaining information about the available resources hosted by a server. This includes size, resource type, path and media type for each individual resource.
- RFC 7641 “CoAP observe” is also an IETF standard, which describes the possibility to observe resources in a publish/subscribe approach by using a observe flag in the CoAP-message-header. When receiving a request with this header option the CoAP server will send notifications of changed resources to the client. Both server and client can cancel the subscription.
- RFC 7390 “CoAP group communication” is for now the lastest related standard. It specifies CoAP communication based on IP multicast. It provides guidance how CoAP should be used in group communication e.g. addressing all devices in a room. Together with the CoRE link format a device discovery can be realized.

Furthermore there are a number of drafts related to CoAP which should be mentioned:

- CoAP block-wise transfer (Draft 20) [174] intends to enable CoAP to transmit larger amounts of data. For this purpose, data is segmented into blocks. CoAP block-wise transfer ensures that these blocks arrive and are handled in the right order.
- CoAP HTTP mapping (Draft 11) [175] specifies how an cross-protocol proxy can translate HTTP queries to CoAP queries and return respective results.
- CoRE resource directory (Draft 07) [176] defines mechanisms to employ entities that host and maintain descriptions of resources held on other servers. This enables resource-discovery in environments where multicast is not allowed or inefficient.
- CoRE formats in JSON & CBOR (Draft 05) [177] tries to represent the the CoRE link format in JSON or CBOR.
- CoAP over TCP & TLS (Draft 05) [178] replaces the UDP layer of CoAP by TCP and thus DTLS by TLS.
- SenML media types (Draft 10) [179] defines the SenML format to represent sensor data and configuration parameters. This format is proposed to be added to the media types supported by CoAP.
- Patch & fetch methods for CoAP (Draft 00) [180] defines mechanisms to partially access resources instead of always read and write complete resources.

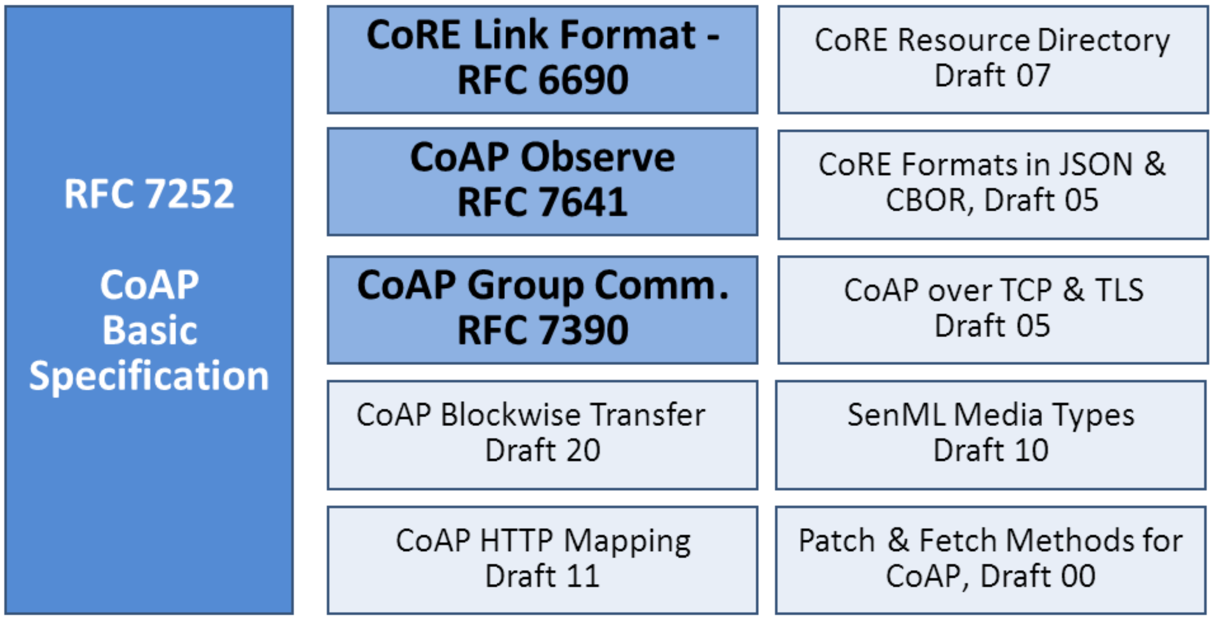

RFC 7252
CoAP Basic Specification
CoRE Link Format - RFC 6690
CoRE Resource Directory Draft 07
CoAP Observe RFC 7641
CoRE Formats in JSON & CBOR, Draft 05
CoAP Group Comm. RFC 7390
CoAP over TCP & TLS Draft 05
CoAP Blockwise Transfer Draft 20
SenML Media Types Draft 10
CoAP HTTP Mapping Draft 11
Patch & Fetch Methods for CoAP, Draft 00

Figure 16: CoAP related standards

There are several implementation of CoAP available. Some of them shall be mentioned here:

- The Swiss Federal Institute of Technology Zurich provides a set of implementations as:
  - Copper [181] is a CoAP user-agent for Firefox and therefore only a CoAP client written in JavaScript and is conform to RFC 7252.
  - Erbium [182] is a REST Engine which is currently used in Contiki and is implemented in C. In contrast to the other implementations this is only conform to the CoAP draft 16.
  - Californium [183] is a CoAP framework in Java which is RFC 7252 conform and currently the most complete implementation of related drafts like CoAP resource discovery and observation.
  - Actinium [184] at last is an Apps server for Californium.

- jCoAp [185] is a CoAP stack for Java which is conform to RFC 7252. It can be used to develop client, server and CoAP<->HTTP-Proxy software. The jCoAP stack is developed by the University of Rostock.
- The University of Bremen provides libCoAP[186] which is a C implementation conform to RFC 7252 and provides the possibility to create clients and servers
- CoAP.net [187] is a C# (.net) implementation based on Californium and is currently conform to the CoAP draft 13

3.10.1. Relevance of CoAP for BaaS

The BaaS project needs communication protocols for different device types from small sensing devices up to enterprise server systems. CoAP matches this requirement by providing a platform-independent transport protocol with reduced overhead for M2M communication while still remaining interoperable with HTTP. Additionally it covers the basics for flexible system configuration including the discovery of servers and resources and it can work hand in hand with the Efficient XML Interchange format (EXI). Furthermore CoAP brings the freedom to choose between a publish/subscribe or push/pull based communication style.

4. Building Automation Data Models

4.1. BACnet

BACnet is one of the predominating standards in the building automation area. The acronym BACnet stands for "Building Automation and Control networking protocol". It defines network stack and application layer for communication in building automation and control systems (BACS). The intention is to allow the communication and integration of BACS equipment from different vendors. The current revision of the standard [188] was done in 2012 and is maintained by the American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE) since 1995 in the Standard Project Committee (SPC) 135. BACnet is adopted as national standard in many countries and was adopted by the International Organization for Standardization (ISO) in 2003, where BACnet is known as ISO 16484-5.

Since the standard [188] does not give the reasoning/explanation for the standard the interested reader is referred to the overview of the protocol and its history in [189].

4.1.1. BACnet Architecture

The BACnet architecture as defined in the BACnet standard is shown in Figure 17. The architecture consists of an application layer and network layer with associated data link and physical layers. A BACnet network is defined to form a local area network, either physically based on data links like MS/TP or logical based e.g. on IP and UDP. Newer directions in BACnet standardization strive to add transport bindings that replace the LAN based network definitions in favor of native IP networking definitions. Such bindings will (among other functionalities) make use of IP routing and thus alleviate from BACnet overlay routing. More details on this may be found in the description of the BACnet IT working group, c.f. Section 4.1.3.1.

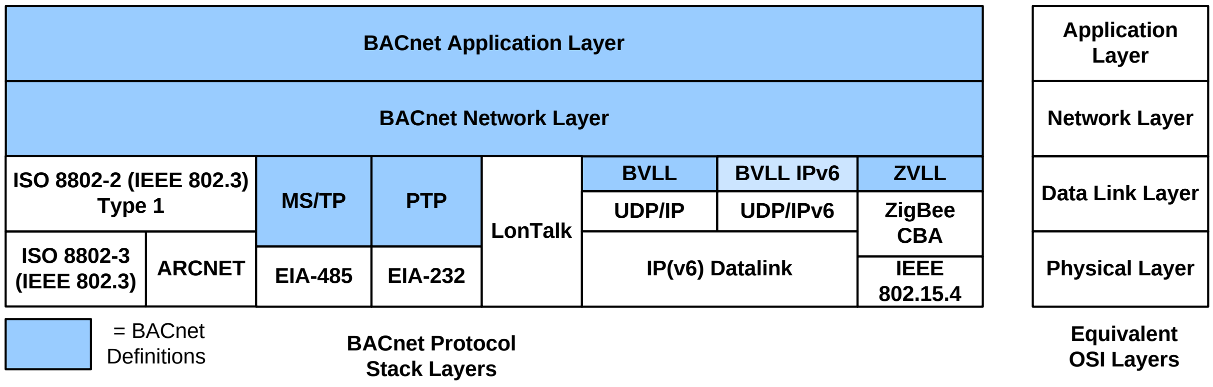


Figure 17: Collapsed BACnet Architecture.

The network layer (and below) definitions are of minor relevance for BaaS. We expect that an IP based gateway will provide the access to data points on legacy devices. Thus, the following sections on BACnet will focus on the BACnet application layer and relevant trends in the BACnet standardization.

4.1.2. BACnet Application Layer

The BACnet application layer is formed of objects and services. Objects represent the data describing the BACS. Services provide a means to communicate between devices.

4.1.2.1. BACnet Objects

BACnet objects describe functions of an automation device in representing a collection of related attributes, which are called properties. A property has a defined data type. Properties might be optional and properties are either readable only or write- and readable. Thus, the BACnet standard defines object types which contain all properties that might be present in an object. A real instance of an object will contain only the properties which are mandatory for an object and the optional properties needed in the implementation of a specific function.

Objects are identified / addressed by a numeric Object_Identifier, which must be unique within a BACnet device. The device has in turn a unique (with regard to the BACnet network) address. This means, each object in a BACnet network can be unambiguously addressed.

Further, each object has an Object_Name, an Object_Type and a Property_List to describe the object and its present properties.

Besides the properties that contain actual values like set points, or sensor readings (Present_Value), many objects implement properties that control the object itself. This applies to starting/disabling an object, reporting on the status or reliability of an object, but also applies to properties around the event/alarm services.

Currently, 54 objects are specified in the BACnet 2012 standard. In [189] these objects are classified in the following categories.

- **Basic Device Object Types:** Device, Analog Input, Analog Output, Analog Value, Binary Input, Binary Output, Binary Value, File
- **Process-related Object Types:** Averaging, Loop, Program
- **Control-related Object Types:** Command, Load Control
- **Meter-related Object Types:** Accumulator, Pulse Converter
- **Presentation-related Object Types:** Group, Global Group, Structured View
- **Schedule-related Object Types:** Calendar, Schedule
- **Notification-related Object Types:** Event Enrollment, Notification Class, Notification Forwarder, Alert Enrolment
- **Logging-Object Types:** Event Log, Trend Log, Trend Log Multiple
- **Life Safety and Security Object Types:** Life Safety Point, Life Safety Zone, Network Security
- **Physical Access Control Object Types:** Access Zone, Access Point, Access Door, Access User, Access Rights, Access Credential, Credential Data Input
- **Simple Value Object Types:** Character String Value, DateTime Value, Large Analog Value, BitString Value, OctetString Value, Time Value, Integer Value, Positive Integer Value, Date Value, Date Time Pattern Value, Time Pattern Value, Date Pattern Value
- **Lighting Control Object Types:** Channel, Lighting Object

4.1.2.2. BACnet Services

While BACnet objects describe the functionality of BAcnet devices, BACnet services are used to communicate between the BA devices. This communication is not restricted to the information contained in the objects and properties (Object Access Services), but also facilitates other services, such as Remote Device Management, Alarm and Event and File Access and Virtual Terminal services.

BACnet supports confirmed and unconfirmed services. Confirmed services are used in direct communication between two BACnet devices and must be acknowledged. The acknowledgment (ACK) may be simple, i.e. just confirm that the request was received and executed but will not contain a response. A complex ACK will also contain a response. Confirmed services can be used for unicast communication patterns. Unconfirmed services are used mainly for broad/multicast communications, but may also be used in unicast communications.

The BACnet standard describes the services by a textual description of the purpose of the service, a table of the structure of the primitives (parameters for request, response, and error and if those parameters are mandatory or user defined), and a textual description of each parameter. Finally the service description contains a description of the service procedure, i.e. how the receiver of a request should process the request.

In the following, we will list the services clustered according the previous mentioned service categories:

- **Alarm and Event Services:** AcknowledgeAlarm, ConfirmedCOVNotification, UnconfirmedCOVNotification, ConfirmedEventNotification, UnconfirmedEventNotification, GetAlarmSummary, GetEnrollmentSummary, GetEventInformation, LifeSafetyOperation, SubscribeCOV, SubscribeCOVProperty
- **File Access Services:** AtomicReadFile, AtomicWriteFile
- **Object Access Services:** AddListElement, RemoveListElement, CreateObject, DeleteObject, ReadProperty, ReadPropertyMultiple, ReadRange, WriteProperty, WritePropertyMultiple, WriteGroup
- **Remote Device Management Services:** DeviceCommunicationControl, ConfirmedPrivateTransfer, UnconfirmedPrivateTransfer, ReinitializeDevice, ConfirmedTextMessage, UnconfirmedTextMessage, TimeSynchronization, UTCTimeSynchronization, Who-Has and I-Have, Who-Is and I-Am
- **Virtual Terminal Services:** VT-Open, VT-Close, VT-Data

4.1.3. Current Trends in BACnet Standardization

In the following we describe some of the more recent directions BACnet is heading, if these might be relevant for BaaS or give indications for BaaS requirements.

4.1.3.1. BACnet IT

The BAcnet IT working group aims on adding an IT/IP based transport binding to the BACnet stack. The first protocol option is adding a HTTP binding to BACnet. This binding is able to make use of IT/IP network mechanisms, such as IP routing. It is also well accepted in IT infrastructures. The following main goals will be achieved with the new binding:

- Replacement of BACnet overlay routing (where possible) and usage of already present IP mechanisms.
- Reduce UDP broadcasts. This includes the utilization of IT registry and discovery mechanisms to replace the BACnet "who-is" and "who-has" broadcasts. Wrong configured BACnet overlay routing led formerly to broadcast storms which caused friction in a shared infrastructure with IT departments.
- Introduce typical and well accepted IT security mechanisms, e.g. HTTPS based on TLS.

4.1.3.2. Extensions for BACnet WS

In the planned addendum 135-2012am the BACnet web services are extended and revised. The revision will include the following features:

- Allow for the exchange of structured data.
- Allow the retrieval of (also non periodic) trend history.
- Support for subscriptions (either by polling or callbacks).
- Move from SOAP to a RESTful approach.

4.1.4. Relevance of BACnet for BaaS

The relevance of BACnet for BaaS is twofold. First, BACnet is one of the predominating protocols in the building automation domain. BaaS is committed to provide legacy integration and BACnet will be a potential candidate for the integration of already existing devices. There are several options to do this; the new BACnet addendum (c.f. Section 4.1.3.2) on web services might reduce implementation effort compared to a native BACnet implementation. It is not clear yet, if implementations of the WS addendum will be available in time for BaaS.

Second, the very limited set of objects and services of BACnet combined with the versatile application of BACnet shows, that building automation tasks can be implemented with such limited object and services. Taking up the hints from BACnet will guide to an efficient and reasonable data model of BaaS.

4.2. oBIX

Open Building Information Exchange (oBIX) [190] is an Organization for the Advancement of Structured Information Standards (OASIS) specification and provides an XML-based data model that is exchanged via Web service interfaces between different building automation components. Thus, the oBIX mechanism provides access to the embedded software systems which sense and control the environment. The current specification version is 1.0 and was accepted as Committee Specification in December 2006. The OASIS oBIX TC is based on members, among others, from Cisco Systems, CABA, IBM, Tridum, and Schneider Electric. Currently, the oBIX TC is working on a new minor version, the oBIX Version 1.1 [191], as well as on encoding and binding variants referred to as Common Encodings Version 1.0 [192], REST Bindings Version 1.0 Public [193], SOAP Bindings Version 1.0 [194], and WebSocket Bindings 1.0 [195]. Furthermore, an oBIX version 2.0 is planned that includes topics such as broadcast, peer-to-peer interactions, and enterprise contracts.

In the following, we give an overview about of the basic technical ideas of oBIX 1.0 as well as the perspectives which are given by the upcoming encoding and binding specifications. We start with basics of a typical oBIX message structure, how it is used in the Web Service context, and which data models are defined as well as the opportunities with the different encoding variants. The subsequent sections provides some insides about oBIX contracts, oBIX Watches, and finally about the work plan of oBIX 2.0.

4.2.1. oBIX Basics

4.2.2. Message Structure

Figure 18 shows a sample oBIX message structure that may be provided by a thermostat. The first element obj, a.k.a. root element, models the entire thermostat. In general, objects are the abstraction used by the oBIX data model (see Section 4.2.4) and each used (sub-) element in message can be mapped to an oBIX object. The attribute href within the obj element is used to identify the Uniform Resource Identifier (URI) for this message. Furthermore, the message contains three nested elements, namely two times real and one bool element. The real elements/objects represent a float value that is given by the val attribute. The name attribute defines the role of the nested elements. Here, the first sub-element represents the space temperature (spaceTemp) and the second sub-element the setpoint. The units attribute is used to assign the values a particular physical unit. The examples show the units assignment of Fahrenheit (obix:units/fahrenheit). The last sub-element/sub-object in this sample message is a bool-based element that represents the furnace state (furnaceOn) which is set to true.

```
<obj href="http://myhome/thermostat">
  <real name="spaceTemp" units="obix:units/fahrenheit" val="67.2"/>
  <real name="setpoint" unit="obix:units/fahrenheit" val="72.0"/>
  <bool name="furnaceOn" val="true"/>
</obj>
```

Figure 18: Sample oBIX message structure

Beside the usage of primitive data types such as real and bool, users are able to define own data structures for their own automation devices. By doing so, contracts are defined and used which is explained in Section 4.2.6.

4.2.3. Web Services

Web services are well known approaches for client-server interactions. oBIX uses Web services for requesting and responding its messages. In general, there are three different kinds of request-response types, as described below.

- **Read**: return the current state of an object at a given URI.
- **Write**: update the state of an existing object at a URI. The new updated state is returned as response message.
- **Invoke**: invoke an operation identified by a given URI. Thereby, the input parameters are transported within the request message and the output result within the response message.

The oBIX standard describes two Web service binding variants which are able to apply these basic types: HTTP/REST and SOAP. In the next two subsections we will explain the usage of these bindings with the defined request-response types above. Furthermore, we will also explain usage of a new binding, CoAP, that is currently described in the draft specification REST Bindings 1.0 [193].

4.2.3.1. HTTP/REST

As aforementioned, REST is an architectural style that is typically used with the HTTP binding for the development of Web services. The following table associates the different HTTP methods to the oBIX types.

oBIX Request Type	HTTP Method	Target
Text	Text	Text
Read	GET	Any object with an href
Write	PUT	Any object with an href and writable=true attribute
Invoke	POST	Any object

Table 5: oBIX type map to HTTP methods

For each HTTP request, the URI addressed within the HTTP header must map to the URI of the object (root element) of the oBIX message. A simple read is initiated by the HTTP GET method and will receive a resulting oBIX message as response. The write and invoke type is initiated by PUT and POST respectively which will also receive the result as an oBIX message.

4.2.3.2. SOAP

SOAP Web services is a well-known approach that is standardized by the W3C. oBIX uses this SOAP binding to transport its messages within the Body element of the SOAP message framework. Each request-response type is reflected by a read, write, and invoke element in the SOAP request message.

Figure 19 depicts a sample SOAP request message that imitate a read of an about object. The corresponding read element is nested in the Body element and contains the URI (http://localhost/obix/about) of the desired object. That means, unlike to the HTTP/REST approach, the URI of the SOAP request is not typically bind to the oBIX object.

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Body>
  <read xmlns="http://obix.org/ns/wsdl/1.0"
        href="http://localhost/obix/about" />
</env:Body>
</env:Envelope>
```

Figure 19: Sample SOAP request message

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
 <env:Body>
  <obj name="about"
       href="http://localhost/obix/about/"
       xmlns="http://obix.org/ns/schema/1.0">
     <str name="obixVersion" val="1.0"/>
     <str name="serverName" val="obix"/>
     <abstime name="serverTime" val="2006-02-08T09:40:55.000+05:00"/>
     <abstime name="serverBootTime" val="2006-02-08T09:33:31.980+05:00"/>
     <str name="vendorName" val="Acme, Inc."/>
     <uri name="vendorUrl" val="http://www.acme.com"/>
     <str name="productName" val="Acme oBIX Server"/>
     <str name="productVersion" val="1.0.3"/>
     <uri name="productUrl" val="http://www.acme.com/obix"/>
  </obj>
 </env:Body>
</env:Envelope>
```

Figure 20: Sample SOAP response message

The corresponding response is shown in Figure 20. It can be seen, the SOAP Body element simple carries the known oBIX message structure.

4.2.3.3. CoAP

As described above, Constrained Application Protocol (CoAP) is an application protocol based on UDP that is specified by the Internet Engineering Task Force (IETF). Since CoAP can be easily transformed to HTTP and mainly consists of the same methods of HTTP (e.g., GET, PUT, PULL) the oBIX types can be identical map to the CoAP methods (compare Table 5 oBIX type map to HTTP methods). In the case of CoAP PUT, POST, or all response messages the MIME type is also same selected as used in HTTP (text/xml). CoAP Observe is a protocol extension for CoAP and can be used to observe changes of states of resources. This mechanism can be used as alternative to the concept of oBIX watches. Doing this, no polling for updates on a resource is required anymore.

4.2.4. Object Model

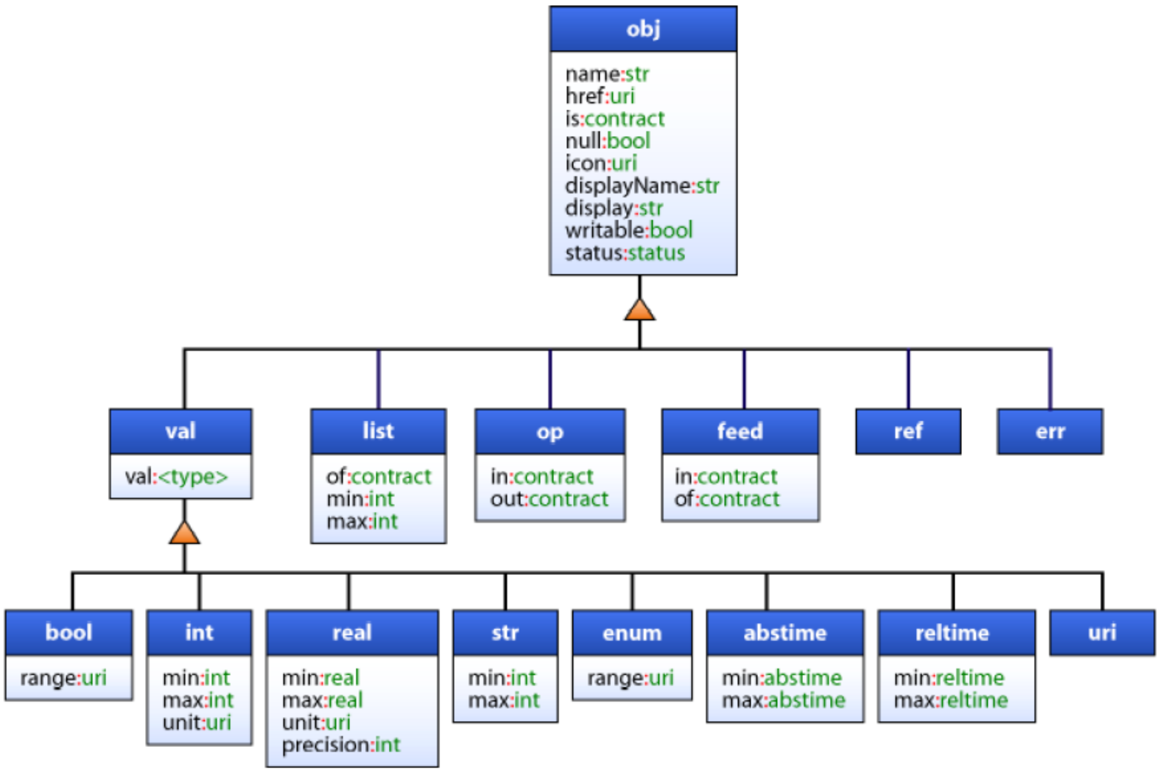


Figure 21: Object Model

The oBIX specification is built on the oBIX object model. Figure 21 depicts this model which consists of a fix set of object types with its supported attributes. As mentioned in Section 4.2.2, each object can be mapped one to one to an XML element type. obj reflects the root abstraction and every XML element in oBIX is a derivation of the obj element. Any obj element or its derivates can have nested other obj elements. The attributes which are available within the obj elements are name (object's purpose), href (URI references for object identification), is (used contracts), null (for null objects), different facets for additional meta-data (icon, displayName, display, writeable, and status), and val (store the actual value).

oBIX supports a set of primitive data type objects: bool, int, real (for floating point number), str (for string), enum, abstime (for absolute point in time), reltime (for relative duration of time), and uri (for URI reference). Some of these objects are able to apply some value restrictions (e.g., min-max ranges) or meta-data (e.g., unit assignments) by the usage of attributes within these object elements.

The list object is used to have a collection of other objects. To define an operation the op object is used. There, the in and out attributes are applied to define the input and return parameter of the operation. The idea of the feed object is to declare a topic for a feed event and it is mainly used with watches (see Section 4.2.7) to subscribe to a stream of events. To create references to another oBIX object, the ref object is applied. Errors are indicated by the err object.

4.2.5. Encoding Variants

The Common Encodings Version 1.0 specification [192] provides three alternative format representations of the oBIX data beside the plain-text XML variant: oBIX Binary, EXI, and JSON.

4.2.5.1. oBIX Binary

The oBIX binary encoding is based strictly on the oBIX data model. Thus, custom namespaces, elements, and attributes are not addressed. Furthermore, oBIX Binary is not compliant with the XML InfoSet requirements [160]. The main encoding idea is based on the assignment a numeric code to each object type and to each facet type (e.g., min and max). Values are also encoded based on their types (bool, int, real, etc.). E.g., Figure 22 shows an example, how data which are boolean typed are mapped into oBIX Binary representation.

```
<bool val="false"/>  =>  08
<bool val="true"/>   =>  09
```

Figure 22: oBIX Binary sample

4.2.5.2. EXI

The W3C Efficient XML Interchange (EXI) [156] format is an opportunity to bring XML-based data into a binary XML representation which also fulfills the XML InfoSet [160] requirements. Thus, binary XML with EXI is able to represent the equivalent content as it given in any plain-text XML representation. The data content can be directly retrieved on the binary XML level without the need of transforming into the plain-text XML representation. There exist a rich number of compression evaluations which consisting compactness ratios where EXI is 100 times smaller than plain-text XML. The main encoding idea of EXI is the usage and traversing of grammars (set of deterministic finite automata (DFA)) that is generic provided by the EXI specification or is constructed if an XML Schema is present. Structure information of XML content is encoded by event codes and data values are encoded type aware. E.g., in the case of boolean-based values, EXI would only use one bit to present true (=1) and false (=0). The Common Encodings Version 1.0 [192] specification provides the EXI options which shall be used when oBIX is used with EXI encoding. This includes the usage of schema-informed coding based on the XML Schema that is provided in oBIX 1.1 [191].

4.2.5.3. JSON

Compared to oBIX Binary and EXI, JavaScript Object Notation (JSON) is a text-based data format developed by Douglas Crockford [196]. JSON uses two structure rules for representing information: A collection of name/value pairs and an ordered list of values. A name/value pair starts with a left brace and ends with a right brace. A colon is used to separate the name and the value. Commas are used to separate multiple name/value pairs. In [192], a grammar is defined that shall be used when oBIX data is represented by JSON. Figure 23 shows an example, how an oBIX XML representation is transformed into a JSON representation.

```
<obj href="/a/">
   <obj name="b" href="b">
          <obj name="c"/>
          <ref name="d" href="d"/>
   </obj>
</obj>
```

```
{
   "obix": "obj",
   "href": "/a",
   "children": [{
          "obix": "obj",
          "name": "b",
          "href": "b",
          "children": [{
                 "obix": "obj",
                 "name": "c",
          }, {
                 "obix": "ref",
                 "name": "d",
                 "href": "d",
          }]
   }]
}
```

Figure 23: Plain-text oBIX XML sample transformed into JSON format

4.2.6. oBIX Contracts

Contracts in oBIX provide the opportunity to build new abstractions upon the core object model which can be collectively agreed to have consistent semantics across vendor implementations. Furthermore, default values can be defined within contracts that avoid the overhead to pass such values over the network for every read. Figure 24 shows an oBIX message sample enriched with more semantics by the usage of contracts (compare Figure 18).

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

  <!-- spaceTemp point -->
  <real name="spaceTemp" is="obix:Point"
        val="-412.0" status="fault"
        units="obix:units/fahrenheit"/>

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

Figure 24: Sample usage of contracts in an oBIX message (sniped)

The two real scalar values are tagged as obix:Points via the is attribute. Thus, clients are able to semantically identify these objects as points.

In general, the oBIX specifications provide a rich set of pre-specified contracts such as Point (reflects a scalar value and its status), History (definition and requesting of time sampled point data), Alarm (defining, routing, and notifying of alarms), Units (physical units), etc.

4.2.7. oBIX Watches

oBIX watches is a polled-based concept for clients to receive access to rapidly changing data. The lifecycle of watches contains following steps (also see Figure 25):

1. With the *make* operation, the client creates a new watch object on the server. The server response a URI to access the new watch.
2. The client registers and/or unregisters objects to watch.
3. Based on the *pollChanges* operation that is periodically polled by the client, the server provides the events which have occurred since last poll
4. By the usage of the *delete* method, a Watch can be freed. Alternatively, the server can free the Watch in the case of the client fails to poll after a predetermined amount of time.

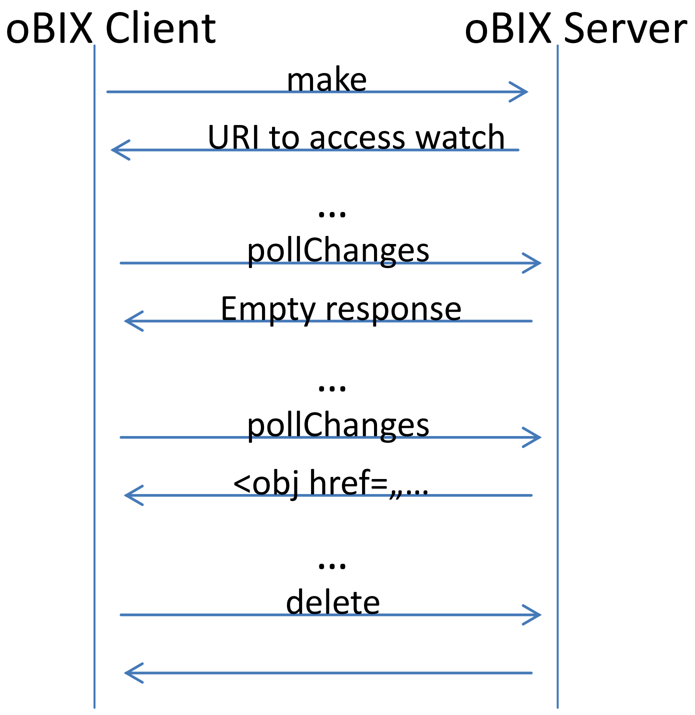


Figure 25: Watch lifecycle

Based on the watches concept with the polling approach, clients do not have to implement a Web service mechanism or have to expose a well-known IP address. On the other hand, this influence the network traffic negatively due to the polling messages which will have an empty response (no changes). As already briefly discussed in Section 4.2.5, CoAP with Observe provides an alternative to oBIX's Watches REST Bindings v1.0 [193]. CoAP Observe provides a subscribe and eventing mechanism without the usage of poll based messages.

4.2.8. oBIX 2.0

As mentioned in the introductory section, the oBIX TC is also currently working on the next major version, the oBIX 2.0. Yet, there is no official working draft available about this new version. However, there exists some articles about the work plan for oBIX 2.0 [197]. The major achievement is the definition of enterprise services by specifying new types of contracts. It is discussed about, among others, patterns for Energy, Advanced Reporting and Aggregation (Historian), Alarm Logic, Building Information Models (BIM), Enterprise Scheduling, and Security Composition purposes:

Energy: oBIX servers may participate in collaborative energy ecosystems

Advanced Reporting and Aggregation: improve the scalability of historians

Alarm Logic: define more complex combinations of events including time dependencies

Building Information Models (BIM): provide definition how oBIX server will make BIM accessible and how to apply BIM as a semantic framework

Enterprise Scheduling: applies semantics of Web Service Calendar to schedule interaction with building systems

Security Composition: provides policy frameworks for secure access to oBIX servers

4.2.9. Relevance of oBIX for BaaS

Open Building Information Exchange (oBIX) is an opportunity to interact with building control systems by the usage of the well-established Web service approach. This provides the advantage of not dealing with the underlying control systems. Instead, we are able to concentrate on more the development and usage of “services” such as value added services (VAS) in BaaS. In addition, oBIX supports different kind of Web service bindings such as HTTP/REST, SOAP, and CoAP to exchange the platform-independent XML-based data model of oBIX. This would also support the different preferences of developers when it comes to the implementations of Web services. Furthermore, oBIX supports binary encodings such as the W3C Efficient XML Interchange (EXI) format which would also open the opportunity the interaction of XML-based data with constrained embedded BaaS devices.

5. Middleware Architecture

5.1. Pervasive Computing Middleware

In 1991, Mark Weiser described a vision of Ubiquitous Computing [198]. He envisioned computing “everywhere”, which became the research direction Mobile Computing, and “in everything”, which became the research direction Pervasive Computing [199].

With smartphones and the App economy[200], Mobile Computing can be considered reality for many people in 2014. For Pervasive Computing this is different. It is not reality yet [201]. The BaaS projected is part of Pervasive Computing as it aims to realize smart spaces.

5.2. Middleware projects

Since 1991 many research projects aimed to implement Pervasive Computing. The maturity of the field becomes visible with the availability of surveys about such middleware designs [201]. Several research projects implemented software for specific devices and use cases [202]. Some projects implement a layer of abstraction, a middleware that aims to provide a usable basis for different scenarios. Following, middleware that was reviewed as highly relevant in surveys [203]–[206] is assessed in detail.

5.2.1. Gaia OS

Gaia OS [207] aims providing a suitable software infrastructure for developing pervasive computing applications. Gaia services are associated with users and move with them through so-called user virtual spaces between the machines a user uses.

Gaia is implemented using CORBA and RPC. Services can communicate with each other using RPC. Mailbox communication and direct communication are supported.

Gaia provides context management over its context file system. Context is accessed over hierarchically structured file names. Gaia does not use context models. Its Metamodel are key-value pairs.

Basic services provided by the Gaia core are a space repository service, an event manager service, a context file system, a presence service, and a context service. Gaia provides local resource management for services and devices.

The base entity of BaaS is a physical space. The base entity of Gaia is a user. Therefore services in BaaS are bound to a space while Gaia services are bound to a user. The different focus makes the mechanisms offered by Gaia only partly usable for BaaS.

5.2.2. Aura

Aura [208] provides functionality for automatic configuring and reconfiguring of software in ubiquitous computing environments according to a user’s needs. It provides a component that manages application adaptation (Odyssey), a file storage backend (Coda), and a remote execution manager (Spectra). Aura uses different communication protocols to operate in heterogeneous environments.

Services communicate over so-called ports. Aura applications typically orchestrate user software such as text editors. A typical use case is a user editing a text with Microsoft Word on one machine then moving to another machine and seamlessly continuing editing with Open Office. The corresponding Aura application manages moving the document and to set up the editor on the target machine accordingly.

Aura service interfaces (ports) are represented in XML. Context management is not in the focus of Aura.

Aura is designed for applications similar to the described use case. Its functionality is too narrow for a use as BaaS middleware.

5.2.3. HomeOS

HomeOS [209] focuses on connecting distributed peripherals to a PC and providing abstract interfaces to them so that they become usable by locally running services.

HomeOS uses a strictly vertical organization where applications do not communicate with each other over the middleware.

HomeOS does not manage context.

With its access rules, the system enables defining and enforcing access policies to devices.

The central architecture and the missing support for context management make HomeOS unsuitable as middleware for BaaS.

5.2.4. CORTEX

CORTEX [210] facilitates the creation of dynamic information exchange overlays between context producers and context consumers. The middleware was designed with car-to-car communication as application scenario in mind.

CORTEX allows service coupling over synchronous context exchange based on locality and interest groups for dissemination. The middleware provides a publish-subscribe event bus.

Exchanged context is defined via XML context models.

With its focus on real-time information exchange in dynamic environments, CORTEX does not match with the targeted environments of BaaS and is therefore not suitable as BaaS middleware.

5.2.5. BOSS

Like BaaS, BOSS [211] targets the automation of professional buildings. BOSS creates an abstract layer on top of the smart devices within a building to enable orchestration service portability.

BOSS provides functionality for context storage, and for transactions. The middleware contains authentication mechanisms. Context is derived in a layered process. The lower layer is the sMap protocol that uses XML to structure context. Types, attributes, relations, and location are used as primary contexts. There is no directory for context models.

BOSS does not provide inter-service communication. The missing of a directory for context models makes it difficult for service developers to reuse context that is provided by locally running services. Both aspects are relevant for BaaS and not sufficiently solved by BOSS.

5.2.6. Context Toolkit

Context Toolkit [212] provides an object-oriented architecture to support rapid prototyping of context-aware applications.

A separation of context and processing logic is introduced that facilitates the creation of services. Such separation structures and facilitates the development of services, and it allows inter-service communication using context that is accessed over fixed interfaces.

Context is managed within services. Services are implemented using a common BaseObject class. This allows all services to communicate over predefined interfaces. Context Toolkit does not use an explicit context model.

Though in principle possible, Context Toolkit does not foster inter-service communication for orchestration services. Inter-service communication is needed for modularization. The inherently object-oriented design may be too limiting for BaaS. The missing of a directory of context models makes the reuse of services difficult.

5.2.7. SOCAM

SOCAM [213] targets the rapid development of context-aware services. The project divides pervasive computing into different context domains such as vehicle, or home. Context travels with users between physical locations.

SOCAM services are implemented as OSGI bundles. Services communicate over context that is provided via a domain-central context database. Service discovery is provided.

SOCAM uses OWL with an RDF representation for context modeling. All context domains share an upper ontology. SOCAM ontologies are maintained by experts.

Maintaining the ontology by experts is likely to be too static for the BaaS context. The missing of security and privacy features is a problem. Using an upper ontology as connecting semantic concept between domains may be too limiting for domain-comprehensive orchestration.

5.2.8. JCAF

JCAF [214] faces the problem that existing middleware typically only lifts the problem of interface heterogeneity to a higher level of abstraction. It aims to provide a generic middleware that is suitable for diverse domains.

JCAF services communicate over RMI. A topic-based publish-subscribe mechanism is provided for coupling between services. The RMI directory is used for service discovery.

Context is stored in services. The JCAF middleware does not provide context management. This functionality must be provided by the service implementations over the fixed JCAF interfaces.

Authentication and authorization via certificates are supported.

The reduction of middleware provided functionality to support various domains is desired in BaaS as well. The missing of context management functionality makes the development of services complicated, which is unsuitable for BaaS.

5.2.9. PACE

PACE [215] targets to facilitate the development of context-aware applications. To provide services with context from smart devices it uses a vertical communication in a layered design.

Communication between orchestration services is supported via context, and events.

The central entity of PACE is the context repository. It contains relations between context objects. The PACE Metamodel is fixed. New context models can be added at run time. A repository for context models is not provided.

The missing of a context repository makes the reuse of context in PACE difficult. Such reuse is intended in BaaS. PACE validates context that is also useful for BaaS.

5.2.10. OPEN

OPEN [216] targets rapid prototyping, sharing, and personalization of context-aware services. Services in OPEN consist of a set of Event-Condition-Action (ECA) rules that are executed by the middleware core on the context database.

OPEN is centralized. Orchestration services are not expected to communicate.

OPEN uses an OWL context model. The entire context is stored and disseminated over the central database the OPEN system runs atop. Context models are maintained by experts.

OPEN supports distributed user-based development that may be interesting for BaaS as well.

With its central design and its limitation of services to ECA rules, OPEN is not suitable as BaaS middleware. However, its concepts are of relevance for the project.

5.2.11. One.World

One.World [217] targets the creation of context-aware services for dynamically changing heterogeneous environments.

Services are coupled over asynchronous events that are exchanged over TCP using a proprietary XML format. A directory service is provided for service discovery.

The service base class provides interfaces for context exchange. Context must be managed within each service. No context model is used.

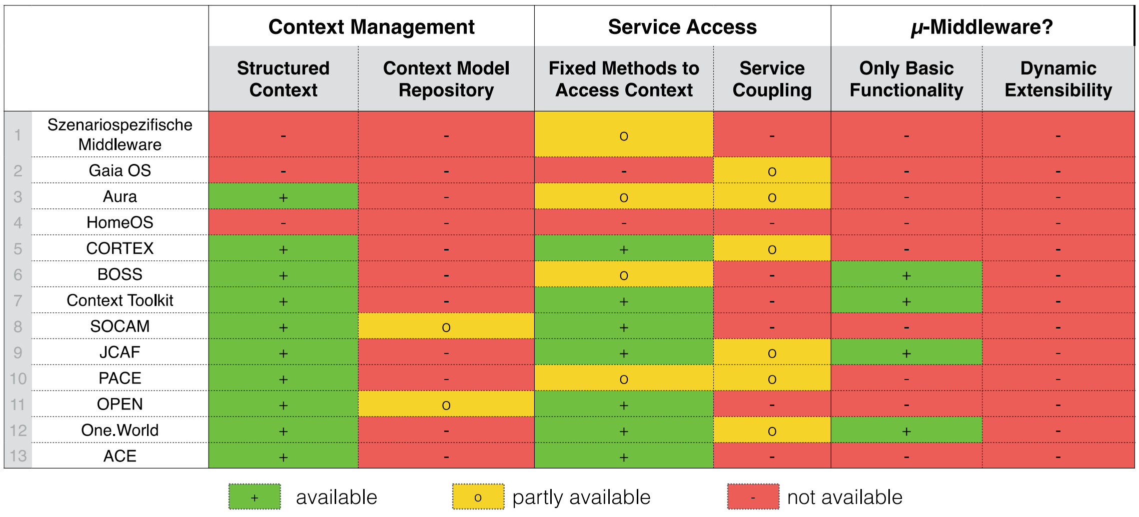
The concept of separating application logic and data is relevant for BaaS. The missing of context management makes service implementations complex. The missing of a context model repository makes the reuse of context difficult. The mechanisms of One.World are relevant for BaaS but the middleware is not directly suitable for the project.

5.2.12. Assessment

Existing middleware provides relevant mechanisms for BaaS. A fundamental problem is that most middleware provides domain-specific functionality such as specific context-reasoning support. Such functionality may be suitable for one target domain but is unsuitable or even inhibitive for implementing scenarios related to other domains, or domain-comprehensive.

Figure 26 summarizes the assessment of state of the art middleware.

The first assessment category concerns context management. As described above most middleware does not use explicit context models. The first column shows that most middleware uses a structure for representing context. The second row shows that a repository for context models is missing which prevents the reuse of context between services.



	Context Management		Service Access		μ-Middleware?	
	Structured Context	Context Model Repository	Fixed Methods to Access Context	Service Coupling	Only Basic Functionality	Dynamic Extensibility
Szenariospezifische Middleware	-	-	o	-	-	-
Gaia OS	-	-	-	o	-	-
Aura	+	-	o	o	-	-
HomeOS	-	-	-	-	-	-
CORTEX	+	-	+	o	-	-
BOSS	+	-	o	-	+	-
Context Toolkit	+	-	+	-	+	-
SOCAM	+	o	+	-	-	-
JCAF	+	-	+	o	+	-
PACE	+	-	o	o	-	-
OPEN	+	o	+	-	-	-
One.World	+	-	+	o	+	-
ACE	+	-	+	-	-	-

+ available   o partly available   - not available

Figure 26: Comparison of Different State of the Art Middleware

The second assessed category is the service access. Many middleware designs use fixed methods for accessing context. This allows in principle to exchange context between services using fixed methods. However, only half of the middleware supports inter-service communication. As indicated with the symbol and the color, the existing approaches only support a subset of possible communication modes. Characteristic properties are referential coupling and temporal coupling.

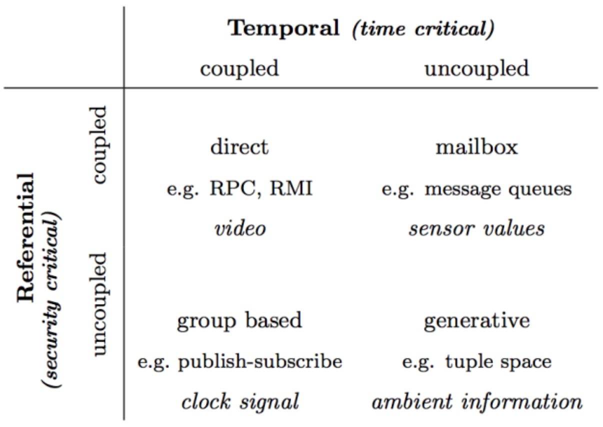


Figure 27: Different coupling modes and their necessity for realizing scenarios.

Figure 27 exemplifies that BaaS services require all shown coupling modes. None of the existing middleware designs supports them.

The last category is the so-called µ-middleware property. It expresses if a middleware is suitable for implementing domain-comprehensive scenarios. The first µ-middleware property covers providing basic, domain-comprehensive functionality only. It is fulfilled by a third of the assessed middleware. The other middleware provides domain-specific functionality resulting in the domain inter-operability problems discussed above.

Offering basic-functionality only makes the development of services complex, as missing support functionality has to be implemented in each service. Therefore, the second µ-middleware property is relevant which is the dynamic extensibility of the middleware core at run time. Such extensions can provide domain-specific functionality by not limiting services that do not need to or cannot use this specific functionality. As Figure 26 shows, none of the existing middleware provides such functionality.

6. Semantic Modeling

6.1. Introduction

Basic entities of BaaS are so-called *smart devices*. A ***smart device*** is a networked device that is able to monitor and control its physical environment via sensors and actuators. The network interface enables remote control. A space that is enriched with such *smart devices* is called ***smart space***. Similar to a conductor who manages an orchestra, software services manage distributed *smart devices* to reach certain goals in a smart space, such as making the inhabitant of a building feel well. The process of software managing *smart devices* is therefore called ***software orchestration***.

Smart space orchestration services require information to reach their goals. Examples for such information are the temperature of a room, the people in the room, or the preferences of those people regarding the room temperature. The information a service needs is called ***context*** [218], [219].

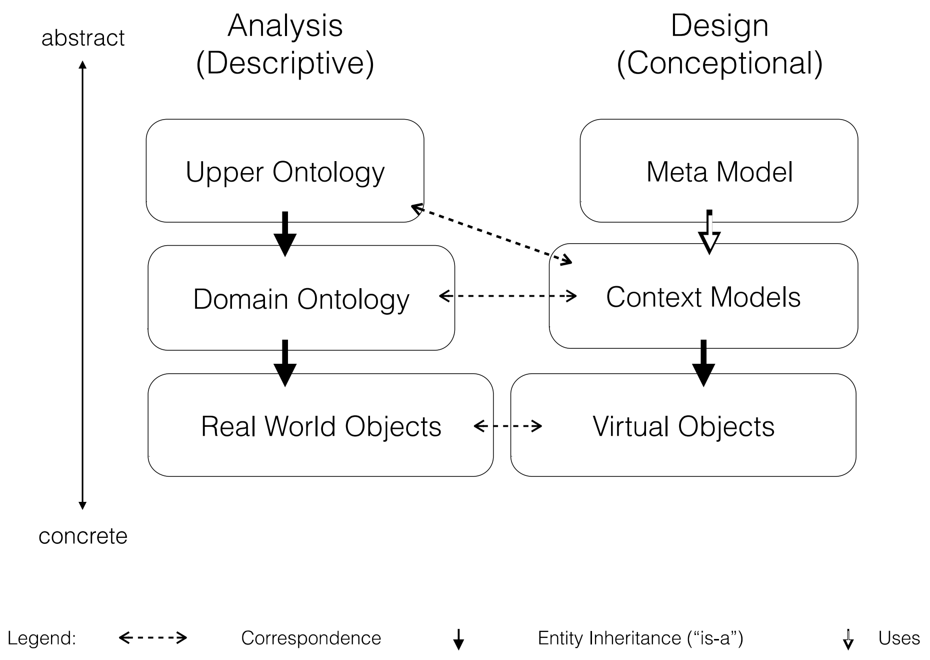
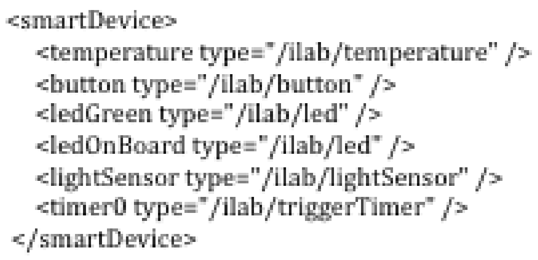


Figure 28: Semantic Modeling Terminology

To make context (re-) usable, it is necessary to structure it. This happens via so-called ***context models***. Figure 28 shows the terminology that is used in the context of semantic modeling. On the bottom left, real world objects that are relevant for software orchestration are shown. Their relevant properties can be defined with so-called ***ontologies***. A *domain-ontology* defines the characteristic properties of real world objects for one particular domain. An *upper ontology* can be used to share knowledge between different domains. It contains the subset of a domain ontology that is shared between different domains. An example is a building that might be characterized by having an entrance and windows in the upper ontology while a domain-ontology of building managers may contain many more maintenance-specific properties such as specific device properties for representing the window state.

The software representation of a real world object is a ***virtual object***. The structure of virtual objects is defined in so-called ***context models*** [218], [220]. An example for a context model is shown in Figure 29. It defines the structure of the information that can be exchanged with the smart device.



```
<smartDevice>
   <temperature type="/ilab/temperature" />
   <button type="/ilab/button" />
   <ledGreen type="/ilab/led" />
   <ledOnBoard type="/ilab/led" />
   <lightSensor type="/ilab/lightSensor" />
   <timer0 type="/ilab/triggerTimer" />
</smartDevice>
```

Figure 29: Context Model of a Smart Device

Different technologies to create context models exist. They are called Metamodels. All have advantages and disadvantages [218], [221].

Key-value pairs are the simplest form of Metamodels. They are simple-to-use and fast to process by machines, as they do not have dependencies to other data structures.

Markup-scheme-based context models use markup languages such as XML as Metamodel. Figure 29 shows an XML markup of a context model. Like the key-value-based models, markup-based models are simple-to-use and fast to process. In addition they allow validation of syntax and some semantic properties. Validation is important for providing secure and reliable software orchestration.

Object-oriented context models typically allow reusing existing context models via inheritance. They represent relationships between context models. This makes them more expressive than the former Metamodels but also slower to process, as some dependencies have to be resolved at run time.

The most expressive Metamodel is an *ontology*. It can represent diverse relations between virtual objects and their properties. Via their semantic definitions, ontologies support syntactic and semantic validation. The diverse relationships between entities reduce the processing performance.

Property	Key-Value Pairs	Markup Scheme	Object Oriented	Ontology Based	VSL Context Model
Simplicity-to-use	++	++	++	+	++
Expressiveness	--	-	+	++	++
Fast processing	++	++	+	--	++
Dynamic Extensibility	--		--	-	++
Collaboration Support	--	-	-	-	++
Validation Support	--	+	+	++	++

Figure 30: Properties of different context modeling techniques

None of the former approaches supports dynamic extensibility and collaborative creation of context models. Both properties are relevant for BaaS as smart spaces can be dynamically enriched with novel smart devices. Figure 30 shows the briefly discussed properties of the different context modeling techniques. A more detailed analysis can be found in [218].

In [218], a novel crowdsourced context modeling approach is presented. It is based on a global directory for context models, the ***context model repository*** (CMR). A CMR can be found at http://cmr.ds2os.org/. It is open to the public, enabling everyone to submit context models that get automatically validated and provided via the directory afterwards.

Collaborative editing potentially leads to divergence in context models. Such a divergence is undesirable as it prevents inter-operability between services that offer similar functionality. As an example, it is desired that all adaptation services that connect lamps with their virtual objects use the same context model.

To foster the convergence of context models, this approach introduces crowdsourced elements. Open statistics measure the popularity of context models, and rank them. This leads to a popularity metric of context models based on different factors such as their use in services.

The statistics that are provided by the CMR are directly correlated to the use of context models in services. Thereby the statistics reflect how well a context model is supported by services. A high rating means that many services support the context model. They either use it as interface towards smart devices (adaptation services), or use it as supported interface for orchestration of smart devices (orchestration services). Service developers aim to write popular services. Popularity is expressed by the use of a service. It is directly related to the use of a certain context model. More popular models are more likely to be used in adaptation or orchestration services as they result in better software support. The described metric leads to a convergence of the context model repository. More details can be found in [218].

6.2. Semantic Web

The Semantic Web Activity [222] is an effort by the World Wide Web Consortium (W3C) to make information available on the Web *understandable* and *processable* by both, humans and machines. Thus tasks set by humans would be unambiguously understandable by machines and more effectively processable by them. The vision of the Semantic Web is to give information well-defined meaning, thereby enabling computers and people to work in cooperation [223].

In the remaining part of this Section we will describe how Semantic Web technologies, as general technologies used on the Web, can be also used for specific problems found in the area of Building Automation. Further on, we will make an overview of Semantic Web technologies (as provided by W3C), and will reference state of the art work in the area of applying Semantic Web technologies in Building Automation.

6.2.1. Role of Semantics in Building Automation Systems

In general, building automation and control systems (BACS) are complex systems. One common source of complexity arises from the need to enable *interoperability* between BACS equipment which originate from different vendors, provides data in different formats, and have different communication facilities. Building Automation protocols, such as for example

BACnet [188], target this problem by standardizing the communication between BACS equipment, and by standardizing objects that can be exchanged during this communication (see Section 4.1). While this is certainly an important step forward, Semantic Technologies aim to bring another level of (semantic) interoperability into the picture. For example, consider interoperability not only between heterogonous BACS devices but also between these devices and services and applications that operate on them. In a dynamic environment where new applications (apps) need to be added without a big effort or functionality of existing services needs to be changed or extended, we need a full visibility and transparency of BACS equipment from the physical layer up to the application layer. We need a machine-processable and unambiguous way to describe devices, their capabilities, relations to other devices, interfaces that expose functionality of devices and so forth. With such a mechanism, we can have a machine support in finding devices with certain functionality, based on their semantic descriptions (no matter at what level, and no matter whether a machine or a human needs to find them).

Traditionally, integration of BACS equipment is achieved using gateways. This requires considerable configuration effort when, for example, applications based on different application models, e.g., BACnet [188], KNX [78], LonWorks [2], ZigBee [224], need to communicate between themselves. Instead of writing mapping rules between each pair of protocols, it is sufficient to map each protocol to an abstract application model (the ontology), which acts as a common base for all protocols [225]. Such an ontology-based application model is easily extensible for new protocols, devices, services or applications, and offers a reasoner support when the integration of a new protocol, device, service or application with an existing model is needed. The employment of an ontology-based model also offers benefits such as a central point for configuration. For instance, changes that are part of a new configuration are automatically converted to meet the respective protocol semantics, and committed to these protocols. Also as stated in [225]: "Additionally, the gateway configuration can be derived automatically using the *reasoning capabilities* of ontologies. Consider, for example, a ZigBee light switch that shall be integrated into a KNX lighting system. To achieve this, the engineer now binds the generic datapoints of these two functions. After having finished the binding of all desired functions, it is possible to automatically generate and export this binding in form of configuration data. This configuration data can then be loaded into the gateways or multi-protocol devices respectively." Moreover semantic reasoning may also be used when implementing a new automation function and searching for devices that should be used for that function. A reasoner can find devices even if their semantic descriptions does not match to an explicit search request. Instead, by exploring various semantic relations in ontology, a semantic reasoner can prove which devices should be a part of the answer. Further on, for a complex Building Automation task, which cannot be fulfilled by a single device, it is possible to automatically find out a set of devices and services, which by partially fulfilling certain functionality can solve the overall task. This is a common requirement when new applications and services are created on top of an existing infrastructure, or when the underlying infrastructure is changed and the change must not affect running applications and services. We will demonstrate the use of semantic reasoners for similar purposes in Building Automation in Section 6.2.8. For further examples of the use of Semantic Technologies, an interested reader is referred, for example, to [226], and research projects: [227] and [228] . Also in [229] the work related to a knowledge-based and context-aware Building Automation System demonstrates the use Semantic Technologies in Building

Automation. The system is grounded on the semantic annotation of both end-user profiles (requirements) and capabilities of BACS equipment, and offers reasoning-based matching of the both (depending on different users' contexts). As stated by the authors of [229]: "Such an approach enables novel resource discovery, matchmaking and decision support features in a BA system. Main benefits are in: (i) determining the most suitable services/functionalities according to implicit and explicit user needs, (ii) allowing device-driven interaction for autonomous adaptation of the environment to context modification."

6.2.2. An Overview of Semantic Web Technologies

In the following we give an overview of Semantic Web technologies provided by the World Wide Consortium (W3C), and review other standardization and research activities that are of relevance for BaaS project. To this end we will briefly describe Semantic Web Layer Cake (see Figure 31) and will give basic information about few frameworks for Semantic Web Services.

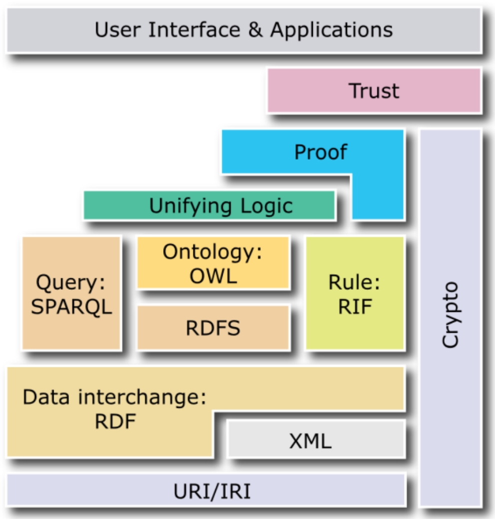


Figure 31: The Semantic Web Stack

6.2.2.1. The Resource Description Framework (RDF)

The Resource Description Framework (RDF) [230] is the core framework. It is a data model that underlines all other Semantic Web technologies above (see Figure 31). The data model enables statements in form of *triples*. A triple is a statement consists of subject, predicate and object. The basic intuition behind a triple is to form a statement where subject and object are connected over a predicate, thereby providing a basic sentence. Subject and object hence constitute a pair of resources that can be considered nodes of a graph connected by an edge that is the predicate. The basic graph can be further connected with other nodes (subjects and objects) thereby creating a larger RDF graph. For example, "device1 is LightActuator" is an example of a triple, where subject "device1" and object "LightActuator" are connected with predicate "is". What is a concept of LightActuator, and how it relates to other important concepts in a domain of interests is defined with RDF Schema (see the next subsection).

It is worth noting that RDF statements can be serialized in different formats such as: XML [231], Notation 3 [232], Turtle [233], JSON [234] and others.

6.2.2.2. RDF Schema (RDFS)

RDF Schema (RDFS) [235] provides a basic vocabulary for creating RDF graphs. Using RDFS it is for example possible to create sets of concepts (classes) important for a certain domain; to establish hierarchies between those classes (e.g., LightActuator is a subclass of class Actuator and Actuator itself is a subclass of class Device); and assign properties to them (e.g., an instance of class Device has property hasDeviceLocation, which relates that instance with an instance of class Location, e.g., "device1 hasDeviceLocation roomArea1"). RDFS hence enables semantic definitions to be introduced over RDF triples, see RDF Semantics [236]. In turn the semantics enables entailment between RDF graphs. For example, for two concepts S and E we can infer that S RDFS-entails E when every RDFS -interpretation which satisfies every member of S also satisfies E. Regarding the expressivity, RDFS is positioned under OWL (see Figure 31). Although RDFS defines only basic constructs to define ontology, in many applications this turns to be sufficiently expressive. Additionally, there exist efficient procedures to conduct reasoning with RDFS ontologies, hence RDFS is a very popular and useful formalism in practice.

6.2.2.3. Web Ontology Language (OWL)

The Web Ontology Language (OWL) [237] is a family of knowledge representation languages for creating ontologies. The languages are characterized by formal semantics and RDF/XML-based serializations for the Semantic Web. OWL extends RDFS with additional constructs and is syntactically embedded into RDF (see Figure 31). The OWL family can be distinguished between OWL and OWL 2 [238]. OWL and OWL 2 are used to refer to the 2004 and 2009 specifications, respectively. The OWL specification includes the definition of three variants: OWL Lite [239]: taxonomies and simple constraints; OWL DL [240]: full description logic support; and OWL Full [240]: maximum expressiveness and syntactic freedom of RDF. In OWL 2, there are three sublanguages of the language: OWL 2 EL [241], OWL 2 QL [242], OWL 2 RL [243]. OWL 2 EL is a fragment that has polynomial time reasoning complexity for all standard reasoning tasks; it is particularly suitable for applications where very large ontologies are needed. The EL acronym reflects the profile's basis in the EL family [244] of description logics (logics that provide existential quantification). OWL 2 QL is particularly suitable for applications where relatively lightweight ontologies are used, but they contain large numbers of individuals and where it is useful or necessary to access the data directly via relational queries (e.g., SQL); OWL 2 RL is particularly suitable for applications where relatively lightweight ontologies are used to organize large numbers of individuals and where it is useful or necessary to operate directly on data in the form of RDF triples. OWL 2 RL enables the implementation of polynomial time reasoning algorithms using rule-extended database technologies.

OWL is the main family of knowledge formalisms for Semantic Web, and so far has been used in academia and in industry.

6.2.2.4. SPARQL Query Language for RDF

SPARQL Query Language for RDF (SPARQL) [245] is an RDF query language, which is a query language capable to retrieve and manipulate data stored in the RDF format. It can be used to query any RDF-based data (i.e., including semantic constructs defined in RDFS and OWL, see Figure 31). A querying language is necessary to retrieve information for Semantic Web applications. While ontology can be defined in RDFS or OWL, in order to use and process data from that ontology we need a language, which can manipulate data – with respect to Semantic Web, this role is precisely fulfilled with SPARQL. SPARQL has been widely used and there exist many implementations of the language in various programming languages, see e.g., [246].

6.2.2.5. Rules

RIF [247] and SWRL [248] bring support of rules. RIF is a W3C Recommendation, and SWRL is a W3C Member Submission. Rules are important for example to allow describing relations that cannot be directly described using Description Logics (which provide the ground logics for OWL), or to enable Closed World Reasoning, see [249].

The top layers of Semantic Web Stack (see Figure 31) contain technologies that are not yet standardized, but these technologies are crucial in realizing the full vision of Semantic Web, see [250]. RIF and SWRL, both belong to this category of the stack. Apart from them, there are also:

- Cryptography: important to ensure and verify that semantic web statements are coming from trusted source. This can be achieved by appropriate digital signature of RDF statements.
- Trust: derived statements will be supported by (a) verifying that the premises come from trusted source and by (b) relying on formal logic during deriving new information.
- User interface: the final layer that will enable humans to use Semantic Web applications.

6.2.2.6. Reasoning

Semantic Web languages are languages defined with a formal semantics. Thanks to this, knowledge defined with such languages may be machine interpretable with *reasoners*. Reasoning is ability to generate non-trivial conclusions from a set of asserted facts or axioms. Semantic Web languages (RDFS and OWL) enable specification of wide range of axioms in form of ontologies and rules. Common reasoning capabilities of RDFS include:

- reasoning about subclass relations;
- reflexivity of subproperty and subclass relations;
- inference through RDFS range and domain constraints;
- transitivity of RDFS subclassOf and subproperty relations.

OWL reasoning capabilities include those from RDFS, and offer (among others) the following:

- enforcing transitivity and symmetric of OWL properties;
- reasoning with OWL inverse relation between properties;
- inheritance of disjointness constraints;

- reasoning with OWL sameAs relationships;
- reasoning with OWL complementOf properties;
- reasoning with OWL caridinality constrains;
- reasoning with OWL value constraints;
- reasoning with OWL intersection and union links between classes.

For more information about reasoning capabilities of Semantic Web languages, an interested reader is referred to, for example, to [251] [252].

6.2.2.7. Stream Reasoning

Semantic reasoners are used today predominantly for reasoning over static or slowly evolving knowledge (ontologies and rules). With emergence of rapidly changing data, e.g., from sensor networks, messages exchanged in web service interactions and ad-hoc business processes, events generated by network diagnosis systems and various devices connected to Web (in context of Web of Things applications), it is important to on-the-fly interpret data and to figure out possible responsive actions. The interpretation of data involves processing of rapidly changing data with respect to static or slowly evolving domain knowledge. The domain knowledge is usually defined as ontology, which represents a domain of interest of an application and help in harnessing the semantics of the overall context. *Stream reasoning* is the task of conjunctively reasoning over streaming events, and static or slowly evolving knowledge. It is, therefore, reasoning that takes streaming events as an input, and by consulting static or slowly evolving knowledge, it continuously derives a streaming output under time constraints [253].

Stream reasoning is a new discipline for which there are no yet standards given by W3C. More recently a W3C Community Group has started to work on RDF Stream Processing (RSP), see [254]. The scope of the group is to give a proposal for querying streaming RDF data. In this regard, the focus is on providing an extension of SPARQL (see Section 6.2.2.7) to handle data streams. As state of the art work in this area we will distinguish following approaches.

Continuous SPARQL (C-SPARQL) [255] is a language for continuous queries over streams of RDF data that extends SPARQL. It extends the SPARQL language by adding support for window and aggregation operations. CQELS-QL [256] is a declarative query language built from SPARQL. Similarly, as C-SPARQL it extends SPARQL with window operators to query RDF streams. CQELS-QL targets the problem of scalable, native and adaptive query processing over Linked Stream Data integrated with Linked Data [257]. Event Processing SPARQL (EP-SPARQL) [258] extends SPARQL language with Event Processing (EP) and reasoning capabilities. Unlike the other two approaches, EP-SPARQL supports both temporal operators (e.g., sequence of events) and strong streaming reasoning capabilities (reasoning about events in context of static knowledge).

6.2.2.8. Semantic Web Services

Semantic Web Services extend web services in a way that data processed by web services is given semantic meaning, and interfaces of web services are semantically described. These extensions aim to enable web services to function more efficiently, e.g., with respect to the task of discovery of relevant web services (e.g., based on their functional and non-functional capabilities), automatic web service composition, imposing various constraints on use of web services and data they process etc.

Since web services are in the scope of BaaS project, in this subsection we will give an overview of various languages and frameworks related to Semantic Web Services.

- OWL-S: Semantic Markup for Web Services - is an OWL ontology for describing Semantic Web Services. The goal of the ontology is to enable web services to be automatically discoverable, invoke-able, compose-able, and monitored, under specified constraints [259]. The main idea of OWL-S was to semantically describe inputs and outputs of web services, as well as relations and constraints between the two (pre-conditions and post-conditions), in order to enable machine-based assessment whether, for example, a web service may fulfill a certain task or whether it is possible to use output of one web service as input of another one. OWL-S also enables a specification of the implementation of the service (also known as grounding), which can be provided in, e.g., in Web Service Definition Language (WSDL), see [260] and [261], although other groundings are possible too, e.g., in SPARQL. With this mechanism, a semantic description of a web service is grounded with a particular web service implementation (e.g., so that a web service, which is selected based on its semantic description, may also be invoked).
- Semantic Annotations for WSDL and XML Schema (SAWSDL) [262] – is a set of semantic extensions for the Web Services Description Language (WSDL) version 2.0 part 1: core language [261]. WSDL was one of first attempts to provide a mechanism for describing web services. However WSDL provides the mechanisms characterize the technical implementation of web services. As stated in [263]: “It does not provide the means to capture the functionality of a service. For example, a service that counts the number of words in a text will be described by WSDL as an interface, which accepts a string and outputs an integer. Clearly, an infinite number of algorithms share those input and output properties, so this information is insufficient to infer any meaning or functionality.” This was the motivation for the work on SAWSDL. It provides semantic annotations for various parts of a WSDL document including: interfaces, operations, inputs and outputs. Apart from that, web services can be added to a central registry of web services (where various categories of web services are semantically described). SAWSDL has the status of the W3C Recommendation.
- Simple Semantic Web Architecture and Protocol (SSWAP) [264] - is a framework that enables creation, discovery and execution of RESTful and other web services. The framework combines concepts from REST, Web Ontology Language (OWL), and web services into a lightweight architecture for web services, and a protocol that enables information sharing in a decentralized fashion. SSWAP is not based on WSDL and SOAP. Instead, it adopts a simple approach based on a RESTful architecture. Similarly as OWL-S, it is an OWL ontology specifically designed to describe web services, i.e., semantic descriptions of their inputs and outputs, and a mapping thereof. The SSWAP ontology further defines a semantic representation that can be used for service search requests and responses, as well as service execution requests and responses. SSWAP employs standard HTTP methods to execute these web services.
- Semantic descriptions for hypermedia APIs (RESTdesc) [263] – is an approach to capture the functionality of hypermedia APIs in a semantic way, so that automated agents can discover what an API does, and invoke it in an automated fashion. RESTdesc expresses the semantics of web services by specifying their pre- and post-conditions in simple logic rules, is based on REST principles, and uses HTTP as a communication protocol. The aim is a versatile description and communication model, enabling fully automated

service discovery and execution, even under changing conditions [263]. This approach is not complex, yet from the semantic expressiveness point of view is very powerful. It is however still an academic approach, for which an open source implementation is available.

Remark: In the state of the art approaches for Semantic Web Service frameworks there exist approaches which are, and are not, compliant with RESTful principles. In the scope of BaaS project we will further consider approaches which are compliant with these principles. However it is worth noting that even RESTful-based approaches are tailored for using HTTP, as a communication protocol. In BaaS project we will deal with resource-constraint devices that are not capable to operate with HTTP. Instead, CoAP protocol is more suitable. But so far we could not find any CoAP-based approach for RESTful Semantic Web Services. Therefore we find interesting further to investigate possibilities to extend the state of the art approaches for RESTful Semantic Web Services also to work with CoAP protocol.

6.2.3. Taxonomies and Ontologies Relevant for the Domain of Building Automation

In applications based on Semantic Web technologies, the main vehicle to give information well-defined meaning is realized with *ontologies*. They unambiguously define meaning of information and establish various relations between it. They are machine-readable. They are defined on commonly accepted understanding of a certain domain (e.g., Building Automation domain), and they are shared. Moreover they enable machines to find relevant information and to reason about it e.g., to derive new properties or relations based on existing knowledge stated in ontology. Therefore in this section we give few state of the art ontologies and taxonomies that we may use further in BaaS project.

6.2.3.1. Semantic Sensor Network (SSN)

The W3C Semantic Sensor Network Incubator group [265] has gathered itself with the goal of applying Semantic Web technologies as a means to enable interoperability for sensors and sensing systems, and to enable high level advanced services over sensor data (e.g., semantic descriptions of different abstraction layers in sensor-based systems, management, integration, interpretation and querying of sensor data in these systems, computer reasoning etc.).

The starting point of the SSN-XG was the work provided by the OGC's Sensor Web Enablement, see [266]. OGC has provided a set of standards to catalogue sensors and understanding the processes by which measurements are reached, as well as limited interoperability and data exchange based on Extensible Markup Language (XML) and standardized tags. However, they do not provide semantic interoperability and do not provide a basis for applying formal reasoning techniques that can ease development of advanced applications.

As a result of its activity, the SSN-XG [265] has produced an OWL 2 ontology which describes sensors in terms of capabilities, measurement processes, as well as observations and deployments. It provides a domain-independent semantic model for sensing applications, thereby distinguishing following different perspectives:

- A sensor perspective, with a focus on what can sense, how it senses, and what is sensed;

- An observation perspective, with a focus on observation data and related metadata;
- A system perspective, with a focus on systems of sensors and deployments;
- A feature and property perspective, focusing on what senses a particular property or what observations have been made about a property.

The prime use of the SSN ontology is to foster semantic interoperability among sensing devices and higher-abstract components that are built on top of sensing devices (e.g., virtual sensors, web services and so forth). For instance, by enabling semantic interoperability it can be prevented that agents (e.g., web services) combine measurements that semantically cannot be combined. Ontologies serve to establish interoperability but they also serve to ensure the interoperability is semantically correct. It has been shown that this is not possible on the syntactic level [267].

Common case categories where SSN Ontology is used are:

- Data Discovery and Linking: Find all observations (from a set of sensing devices) that meet certain criteria, and possibly link these observations to other external data sources. The user may use different criteria to select the spatial area, temporal window, and types of observations to be found, and may relate the obtained results to external data sources.
- Device Discovery and Selection: Find all the (sensing) devices that meet certain criteria. The criteria may include type, certain spatial area, measured phenomenon, range of measurement, availability, owner or responsible party, and manufacturer, or combinations of those.
- Provenance and Diagnosis: Provide additional (meta) information about an instrument to better evaluate and process the sensor data, or to use the instrument in a correct procedure.
- Device Operation Tasking and Programming: Command a device's operation using its description and information on its conditions of use.

6.2.3.2. BACowl - BACnet Ontology

BACnet [188] (see Section 4.1) is a data communication protocol for Building Automation and control networks. BACowl - the BACnet Ontology treats the contents of the BACnet standard as a model. It has its own lexicon, descriptions of both messages between devices and objects, the properties of which represent the various aspects of the hardware, software, and operation of the device. Most of the standard is described using natural language, which does not enable the formal analysis. By translating the descriptions of the objects, properties, and communications messages into a formal language like OWL and applying analysis to the result, BACowl aims to provide a cohesive description of the standard that can be incorporated into other formal models.

The BACowl specification follows the BACnet standard, and uses OWL to formally encode concepts that might not be clearly stated in the standard. For example, constraints such as a structural element cannot be both “context encoded” and “application encoded” at the same time, and that all structural elements must be at least one of these two, have been realized in BACowl. Further on, the BACnet standard sometimes contains very similar terms to describe very different concepts, e.g., “device”, “device object”, “object identifier”, and “device identifier”. It is the goal of BACowl to follow the BACnet standard, but at the same time to give these concepts distinct labels and rules. Using ontology engineering tools, such as for example Protégé [268], it is possible to visualize the graph of the relationship between these terms and to better grasp the conceptual differences between BACnet definitions.

BACnet standard enables BACnet devices to expose their content via web services, which may implement (apart from BACnet) also other communications protocols. BACowl, in its current version, does not model these protocols. But, if for instance a building automation system employs RESTful web services with their own ontology (terminology), then it is possible to extend BACowl with that ontology, or create a mapping between the two.

BACowl is an open source project available at bacowl.sourceforge.net, and is not part of the BACnet standard.

6.2.3.3. Project Haystack

Project Haystack [12] provides tagging conventions and taxonomies for building equipment and operational data. As stated at the project web site: "Haystack defines standardized data models for sites, equipment, and points related to energy, HVAC, lighting, and other environmental systems. A simple REST API is defined to facilitate exchange of Haystack data over HTTP."

Haystack provides a tagging meta-model for building equipment and operational data. The meta-model is a collection of *tags*, i.e., name-value pairs. A tag defines a fact or attribute about an entity. For instance, if one applies the site tag to an entity, then she is declaring that the entity represents a building. If further the geoAddr tag is added, then the street address of the building is declared. A tag-based meta-model is flexible as it enables customization of building equipment and operational data for each new project, and can be extended easily. Haystack tags are essentially a common vocabulary for the Building Automation domain. Once defined, tags may be used to form higher-level abstractions – *entities*. An entity is a meta-model representation of a physical object in the real world. For example, entities include sites, equipment, sensor points, weather stations, etc. An entity is a set of few tags, instantiated for a certain physical object. Haystack defines core entities for a site (building), equipment within a site, point (sensor, actuator or setpoint value for an equipment), and weather (outside weather conditions). Other entities may, of course, be defined depending on project specific needs. Entities are related to each other via specific tags contained by the entities. The tag taxonomy and entities are defined for all important systems in building operations such as: networks (devices, networks, and protocol connections), energy (electricity and flow metering), air handler units and rooftop units, heating, ventilation, and air conditioning (HVAC) zones, lighting systems and many others. The representation of timezones and units of measurement are also defined in the taxonomies. In this way Haystack defines certain semantic structures that represent a common data model, which can be shared among different entities (applications, services etc.). Haystack meta-models and taxonomies are provided in plain text formats, such as comma separated value (CSV), JavaScript object notation (Json) and other specific text formats (e.g., Zinc, Grids etc.). Haystack does not provide its meta-models with a formal semantics, e.g., defined in OWL or RDF(S). Its goal is however to define informal semantics for open models as commonly found in the Building Automation domain. As such this project deserves a credit. However one drawback of the project lays in the fact that we cannot use standard semantic procedures and tools to enable machine interpretation of the semantic models realized with Haystgack (e.g., we cannot use existing reasoning procedures to automatically derive dependences between building equipment represented with these meta-models in cases when they are complex and the dependences are not obvious).

6.2.4. The Smart Appliances REFerence (SAREF) Ontology

The Smart Appliances REFerence (SAREF) ontology [269] is a shared model of consensus that facilitates the matching of existing assets (standards/protocols/datamodels/etc.) in the smart appliances domain. The SAREF ontology provides building blocks that allow separation and recombination of different parts of the ontology depending on specific needs.

About 50 semantic models from the smart appliances and building automation domain have been analyzed and half have been afterwards translated into OWL. Among those are the following semantic assets: ECHONET, EnOcean, KNX, OMA Lightweight M2M, UPnP, W3C SSN, Z-Wave, OSGi DAL, DECT ULE HF, FIEMSER and others. Each of these assets has been translated into OWL as a separate ontology, but 20 reoccurring concepts have been formalized in the SAREF ontology as a common ontology for the domain of smart appliances. With this approach, SAREF aims to become a reference ontology for smart appliances, where translation from the reference ontology to specific assets reduces the effort. Instead of a set of mappings for each pair of assets, we now need just one set of mappings between SAREF and each asset. As already mentioned, for a number of assets in the smart appliances domain SAREF already provides such a mapping. Figure 32 depicts the role of SAREF in the mapping to other assets.

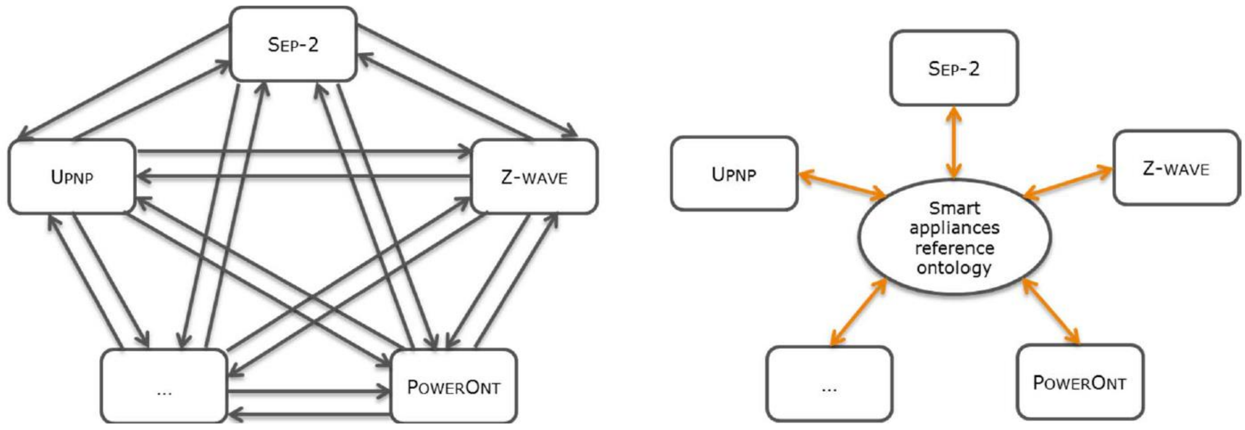


Figure 32: SAREF - a common ontology for the domain of smart appliances

The starting point of SAREF is the concept of Device (e.g., a switch). Devices are tangible objects designed to accomplish one or more functions in households, common public buildings or offices. The SAREF ontology offers a list of basic functions that can be eventually combined in order to have more complex functions in a single device. For example, a switch offers an actuating function of type “switching on/off”. Each function has some associated commands, which can also be picked up as building blocks from a list. For example, the “switching on/off” is associated with the commands “switch on”, “switch off” and “toggle”. Depending on the function(s) it accomplishes, a device can be found in some corresponding states that are also listed as building blocks.

A Device offers a Service, which is a representation of a Function to a network that makes the function discoverable, registerable and remotely controllable by other devices in the network. A Service can represent one or more functions. A Service is offered by a device that wants (a certain set of) its function(s) to be discoverable, registerable, remotely controllable by other devices in the network. A Service must specify the device that is offering the service, the function(s) to be represented, and the (input and output) parameters necessary to operate the service.

The following list shows the basic concepts in SAREF ontology:

- Building Object (Door, Window).
- Building Space.
- Command (e.g. OnCommand, OffCommand, PauseCommand, GetCommand, NotifyCommand,
- SetLEvelCommand).
- Commodity (e.g. Electricity, Gas, Water).
- Device (e.g. Switch, Meter, Sensor, Washing Machine).
- Device Category.
- Duration Description.
- Function (Actuating Function, EventFunction, Metering Function, Sensing Function).
- Function Category.
- Profile.
- Property (Energy, Humidity, Light, Motion, Occupancy, Power, Pressure, Price, Smoke, Temperature, Time).
- Service.
- State.
- Task (e.g. Cleaning, Safety, Entertainment).
- Temporal Entity.
- UnitOfMeasure (e.g. Currency, EnergyUnit, Power Unit, Temperature Unit).

6.2.5. QUDT - Quantities, Units, Dimensions and Data Types Ontologies

QUDT - Quantities, Units, Dimensions and Data Types Ontologies [270] provide a standardized consistent vocabulary for quantities, units, dimensions and data types, focused on terminology used in science and engineering. They have been developed by NASA [271]. The vocabulary is of interest to various users and applications; hence QUDT provides a set of horizontal, cross-domain ontologies. The goal was to specify a machine-interpretable vocabulary that will not change frequently and can be used independently from a variety of encodings, formats, and data definitions. QUDT ontologies are based on the object-oriented principles including encapsulation, inheritance, and polymorphism.

Figure 33 shows a class structure of QUDT Ontologies. A Quantity is an observable property of an object, event or system that can be measured and quantified. For example, the speed of light in a vacuum and the escape velocity from Earth are both quantities. Quantity Kind identifies the observable property quantified, e.g. length, force, frequency. Familiar examples include physical properties such as length, mass, time, force, energy, power, electric charge, etc. Unit is a particular quantity of a given kind that has been chosen as a scale for measuring other quantities of the same kind. For example, the Meter is a quantity of length. A Quantity Value expresses the magnitude and kind of a quantity and is given by the product of a numerical value and a unit of measure. A System of Quantity Kinds is a set of one or more quantity kinds together with a set of zero or more algebraic equations that define relationships between quantity kinds in the set. Examples include Newton’s First Law of Motion, Coulomb’s Law etc. A System of Units is a set of units which are chosen as the reference scales for some set of quantity kinds together with the definitions of each unit.

Units may be defined by experimental observation. Dimension defines a dimension symbol of a quantity kind, and dimension can be defined as a Dimension Vector, e.g. energy has dimension specified as a vector $L^2M^1T^{-2}$ (where L is length [m], M is mass [kg], T is time [s]). For more information on QUDT ontologies, an interested reader is referred to QUDT [270].

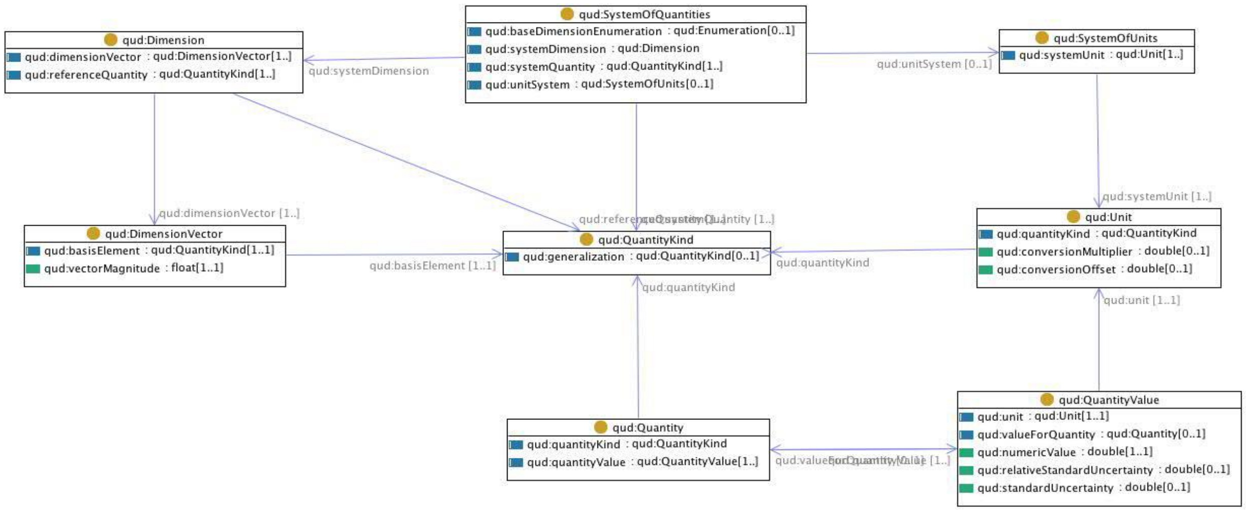

qud:Dimension
qud:dimensionVector : qud:DimensionVector[1..]
qud:referenceQuantity : qud:QuantityKind[1..]
qud:SystemOfQuantities
qud:baseDimensionEnumeration : qud:Enumeration[0..1]
qud:systemDimension : qud:Dimension
qud:systemQuantity : qud:QuantityKind[1..]
qud:unitSystem : qud:SystemOfUnits[0..1]
qud:SystemOfUnits
qud:systemUnit : qud:Unit[1..]
qud:systemDimension
qud:unitSystem [0..1]
qud:dimensionVector [1..]
qud:DimensionVector
qud:basisElement : qud:QuantityKind[1..1]
qud:vectorMagnitude : float[1..1]
qud:basisElement [1..1]
qud:QuantityKind
qud:generalization : qud:QuantityKind[0..1]
qud:quantityKind
qud:systemUnit [1..]
qud:Unit
qud:quantityKind : qud:QuantityKind
qud:conversionMultiplier : double[0..1]
qud:conversionOffset : double[0..1]
qud:unit [1..1]
qud:Quantity
qud:quantityKind : qud:QuantityKind
qud:quantityValue : qud:QuantityValue[1..]
qud:QuantityValue
qud:unit : qud:Unit[1..1]
qud:valueForQuantity : qud:Quantity[0..1]
qud:numericValue : double[1..1]
qud:relativeStandardUncertainty : double[0..1]
qud:standardUncertainty : double[0..1]

Figure 33: Class Structure of QUDT Ontologies

We have used QUDT for modeling quantities and units in BaaS Data Point ontology.

6.2.6. ifcOWL - Industry Foundation Classes Ontology

ifcOWL ontology [272] is an OWL representation of the Industry Foundation Classes (IFC), an ISO standard for a data model that describes building and construction industry data. IFC is an open and neutral data format for Building Information Modeling (BIM). IFC has been developed by buildingSMART [273] (formerly the International Alliance for Interoperability, IAI) to facilitate interoperability in the architecture, engineering and construction (AEC) industry. There are few formats in which IFC is available, e.g., text format, XML format and ZIP-compressed format. Recently buildingSMART has decided to provide an OWL representation of IFC too. This was the reason to form the Linked Data Working Group [274] and the W3C Linked Building Data Community Group [275], which are responsible for building and maintaining a recommended version of an ifcOWL ontology, as an equivalent to the IFC EXPRESS schema. Apart from this, the Linked Data Working Group should propose technical enhancements to the existing standards and align semantic web activities with ongoing efforts in buildingSMART, such as work on the buildingSMART Data Dictionary Browser [276]. There is an ongoing work on the mapping of ifcOWL with SAREF ontology (described also in this section).

6.2.7. Ontology-Based Access Control

Since access control of (data) resources is a topic that is in the scope of BaaS project (see Section 3.5), in this subsection we overview state of the art work in the role-based access control realized with semantic ontologies. The Semantic Web technologies give users and applications a flexible way to access and retrieve decentralized resources on the Web. These concepts and techniques may however be adapted to Building Automation domain too.

The work in [277] provides an interesting access control mechanism, based on the HTTP standard, OAuth Protocol and ontologies. The mechanism can optionally adopt Transport

Layer Security protocol too. OAuth provides access to data stored in a triplestore via a token-based authentication. A logged-in user must obtain a unique token to access the data. After that, the users may access the triplestore[1] without disclosing any identity data. The authorization procedure with OAuth in this approach is pretty much the same as in any other, non-ontology-based approach. The notable difference is however the use of the User Access Ontology (UAO), which is an ontology that describes roles, their permissions, and allowed actions on data (resources). It allows the description of access control lists for users, without assigning them to a single triplestore and/or other databases [277]. UAO defines three groups of permitted actions, i.e., permissions for creating/deleting resources, permissions for modification of existing resources, and permissions for read-only access of resources (querying resources). In total there are nineteen types of actions, which are based on the SPARQL clauses and can be combined with each other. For details about nineteen types of actions an interested reader is referred to Table 2 in [277].

The work presented in [277] is an example of a flexible framework for access control with Semantic Web technologies. It is not the only work on this topic. Ontologies are widely used for the purpose of imposing a flexible access control to resources. This is the case in various environments, including pervasive environments, static and mobile, as well as web and non-web-oriented environments, see also for example [278]. Ontology-based access control of resources is a topic of interest in BaaS project. Hence we will continue further investigation of this topic, and its further adaptations for RESTful web services and the communication based on the CoAP protocol.

6.2.8. Semantic Technologies in Building Automation by Example

In previous sections we have briefly motivated the need for using the semantics, then we made an overview of existing semantic technologies, and finally we provided references about the existing work relevant to BaaS project. In this subsection we will provide few examples in order to more closely show how the so far presented Semantic Technologies may be used in a concrete Building Automation scenario.

Consider a user who wants to create an app (service) that is supposed to control the light at certain location in a smart building from a smart mobile phone. She has just an idea about a desired functionality that she wants to realize, but has no domain knowledge about the underlying infrastructure that is normally required for implementing such functionality (e.g., devices and services available at certain locations, their characteristics, functionally dependences etc.).

Thanks to the BaaS Semantic Model (BSM) she is able to find a service that does what she needs. Further on, she can find out more information about automation function/s, implemented in the service, possible constraints and information to be defined (e.g., about location etc.), required devices, their availability etc. Finally from her mobile phone she can control the light (to invoke the services), thereby going from BSM directly to devices and services that operate on them.

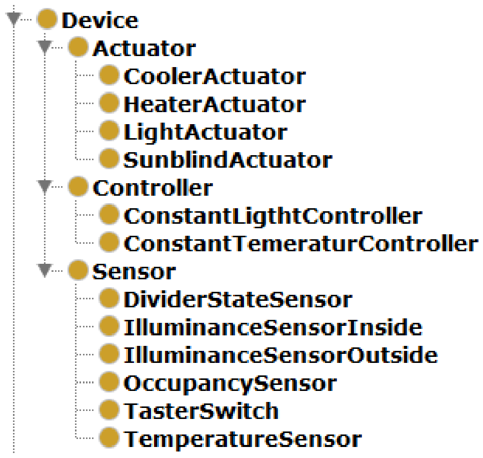
For this example scenario we have developed a small ontology. Figure 34 shows an excerpt of the ontology related to classification of building equipment in the example.

[1] Triplestore is a repository specifically tailored for RDF data.

For example, from Figure 34 we see that Actuator is a class (concept) that is a subclass of class Device. While Figure 34 shows a graphical representation of these two statements, in the ontology file they are actually represented as the following two RDF triples (written in Turtle syntax):

```
@prefix baas: < http://www.baas.org/ontologies/example-ontology-1#> .
baas#Actuator rdf:type :Class ;
baas#Actuator rdfs:subClassOf baas#Device> .
```

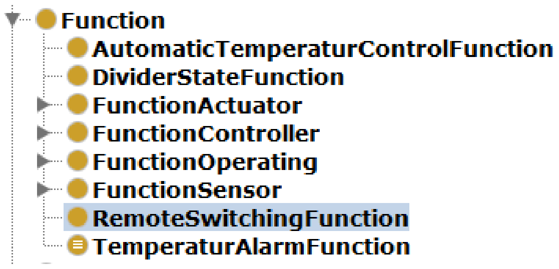
The above two triples are written in *RDF Turtle* syntax, see Section 6.2.2.1. The second triple represents a subclass relation, defined by *RDFS*, see Section 6.2.2.2.



```
Device
  Actuator
    CoolerActuator
    HeaterActuator
    LightActuator
    SunblindActuator
  Controller
    ConstantLigthtController
    ConstantTemeraturController
  Sensor
    DividerStateSensor
    IlluminanceSensorInside
    IlluminanceSensorOutside
    OccupancySensor
    TasterSwitch
    TemperatureSensor
```

Figure 34: Example Classification of Building Equipment

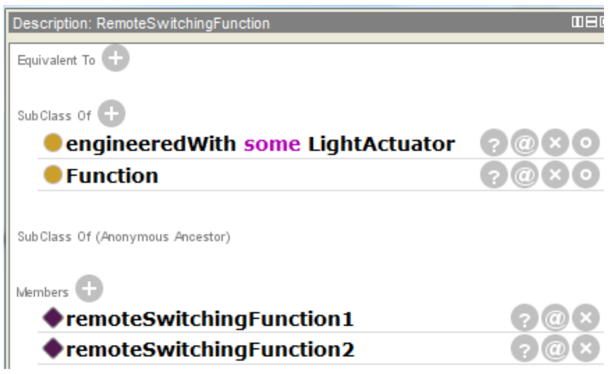
Similarly we can define few automation functions required in our building.



```
Function
  AutomaticTemperaturControlFunction
  DividerStateFunction
  FunctionActuator
  FunctionController
  FunctionOperating
  FunctionSensor
  RemoteSwitchingFunction
  TemperaturAlarmFunction
```

Figure 35: Example Automation Functions

RemoteSwitchingFunction is defined as a function that controls a light from a switch remotely. The function is engineered with *some* device that is a LightActuator (see Figure 36). Some (*someValuesFrom*) is a property restriction defined in *OWL*, see Section 6.2.2.3.



Description: RemoteSwitchingFunction

Equivalent To

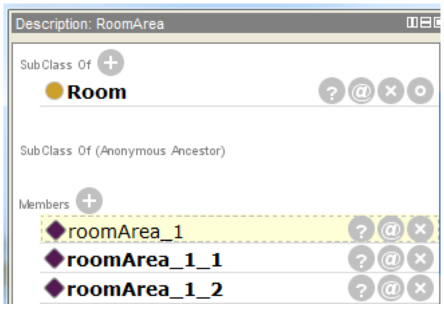
SubClass Of
- engineeredWith some LightActuator
- Function

SubClass Of (Anonymous Ancestor)

Members
- remoteSwitchingFunction1
- remoteSwitchingFunction2

Figure 36: Example Definition of RemoteSwitchingFunction

We have defined two instances of this function, remoteSwitchingFunction1 and remoteSwitchingFunction2. The functions operate at two locations roomArea_1_1 and roomArea_1_2. These two areas are divided by a moving wall, which by changing its state may construct one room area (roomArea_1) out of the two areas, see Figure 37.


Description: RoomArea

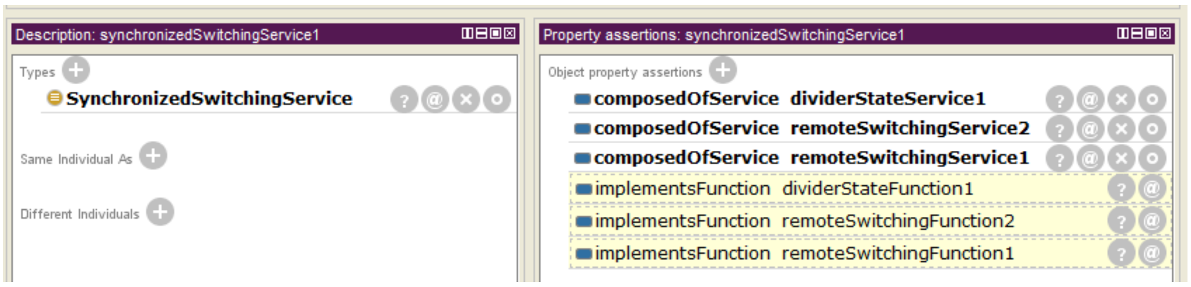
SubClass Of
Room

SubClass Of (Anonymous Ancestor)

Members
roomArea_1
roomArea_1_1
roomArea_1_2

Figure 37: Example Description of various locations in a building

The state change of the moving wall is detected by dividerStateFunction1 (an instance of DividerStateFunction, see Figure 35) and exposed over dividerStateService1 (an instance of Class DividerStateService). Similarly we defined remoteSwitchingService1 and remoteSwitchingService2 as two service instances, exposing remoteSwitchingFunction1 and remoteSwitchingFunction2 respectively. We can now use dividerStateService1 together with two remoteSwitchingService instances to form a SynchronizedSwitchingService. This service controls each remoteSwitchingService separately, or both of them synchronously (if the dividerStateService1 shows the wall is removed). We can define an instance of this service by declaring an instance which is composed of three services: dividerStateService1, remoteSwitchingService1 and remoteSwitchingService2, see Figure 38.


Description: synchronizedSwitchingService1

Types
SynchronizedSwitchingService

Same Individual As

Different Individuals

Property assertions: synchronizedSwitchingService1

Object property assertions
composedOfService dividerStateService1
composedOfService remoteSwitchingService2
composedOfService remoteSwitchingService1
implementsFunction dividerStateFunction1
implementsFunction remoteSwitchingFunction2
implementsFunction remoteSwitchingFunction1

Figure 38: Definition of the SynchronizedSwitchingService

We have modeled each atomic (basic) service to implement certain function. For example, remoteSwitchingService1 implements Function remoteSwitchingFunction1. Additionally, we may want that our composed services inherit functions from their basic services. To do that, the following *SWRL rule* (see also Section 6.2.2.5) may be employed.

```
composedOfService(?z,?x), implementsFunction(?x,?y) ->

   implementsFunction(?z,?y)
```

The rule states that if `z` is a service composed of service `x`, and `x` implements function `y`, then `z` also implements `y`[2]. Figure 38 shows that the user defined synchronizedSwitchingService1 implements functions dividerStateService1, remoteSwitchingFunction1, and remoteSwitchingFunction2 (marked in yellow). This information was derived automatically

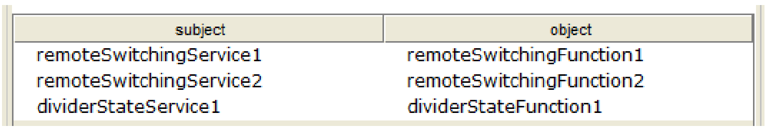
[2] To derive that, apart from the rule, we have also needed to define RDFS domains and ranges for object properties: `composedOfService` and `implementsFunction`.

by the *semantic reasoner*. It shows how the reasoner supports the task of engineering new functionality from an existing one, which may be of great benefit in large systems with many functions, services, and dependences thereof.

In the engineering phase often it is useful to query a data model. In our example scenario, before the user starts creating a new service she may ask the model to retrieve all existing services with corresponding automation functions they implement. This can be done with a following query written in *SPARQL*, see Section 6.2.2.4.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

PREFIX owl: <http://www.w3.org/2002/07/owl#>

PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX baas: <http://www.baas.org/ontologies/example-ontology-1#>
```

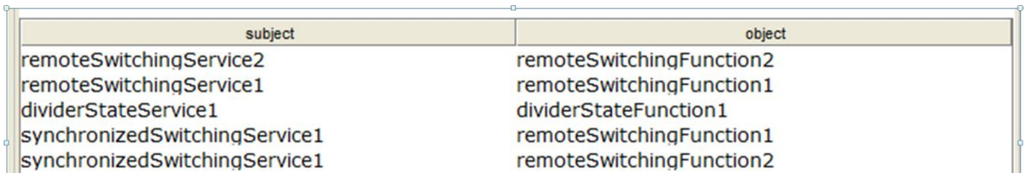
```
SELECT ?subject ?object

      WHERE { ?subject baas:implementsFunction ?object }
```



subject	object
remoteSwitchingService1	remoteSwitchingFunction1
remoteSwitchingService2	remoteSwitchingFunction2
dividerStateService1	dividerStateFunction1

Figure 39: Results from the SPARQL query

The same SPARQL query, after creating the new service synchronizedSwitchingService1, will retrieve results as shown in Figure 40. It is worth noting that the SPARQL engine retrieved, not only explicitly stated information (see Figure 39), but also the results based on evaluation of the SWRL rule for function inheritance.



subject	object
remoteSwitchingService2	remoteSwitchingFunction2
remoteSwitchingService1	remoteSwitchingFunction1
dividerStateService1	dividerStateFunction1
synchronizedSwitchingService1	remoteSwitchingFunction1
synchronizedSwitchingService1	remoteSwitchingFunction2

Figure 40: Updated results from the SPARQL query

So far we have presented a semantic model defined for our example scenario. Having such a model in place, the user may use it to easier develop her idea, i.e., a smart phone app that controls the light at certain location in a smart building. By browsing throughout the model and/or by querying it (as shown in Figure 40), the user can find existing services and functions that are usable for her idea. In this example scenario synchronizedSwitchingService1 appears to be the one, relevant for her smart phone app. Hence she may just want to extend the service with her further ideas and requirements.

In this subsection our intention was to demonstrate how semantic modeling may be helpful in designing Building Automation systems. In particular, in this example the main benefits are related to enabling information search for relevant resources, as well as transparency and visibility of resources including their capabilities, constraints and dependences. If one extended the scenario with a Semantic Web Service model (as given in Section 6.2.2.8), one could also provide further benefits, related for example to interoperability between services or between heterogenic devices.

6.2.9. Conclusion

Modern Building Automation systems will benefit from the use of Semantic Technologies, not only in form of having ontologies as a common conceptualization with the shared vocabulary, but also with concepts and algorithms that enable machine-processable semantics to intelligent services and devices. In this section we have given an overview of semantic standards and state of the art work, relevant for the domain of Building Automation and BaaS project specifically.

Further on, our intention was to demonstrate how Semantic Technologies (in particular RDF, RDFS, OWL, SPARQL, and rules, see Section 6.2.2) may be used in designing Building Automation systems. In particular, we presented an example scenario where main benefits, by deploying Semantic Technologies, are related to enabling information search for relevant resources, as well as transparency and visibility of resources including their capabilities, constraints and dependences. If one extended the scenario further with a Semantic Web Service model as given in Section 6.2.2.8, then one could also provide further benefits, related for example to interoperability between services or between heterogenic devices.

It is also worth noting that Semantic Technologies may provide benefits to different phases in the lifecycle of automation systems. In Section 6.2.8 the focus was mainly on engineering and development phases. But in general, Semantic Technologies may be of great support in operation phases too, as well as in maintenance, and optimization of Building Automation systems. In the context of BaaS project we will therefore continue to work on deployment of Semantic Technologies in order to demonstrate full potential of semantics in Building Automation.

6.3. W3C Web of Things and Thing Description

6.3.1. WoT Overview

In spring 2015, the W3C Web of Things (WoT) [279] interest group was founded. The motivation for initiating this group was based on the fact that the Internet of Things (IoT) suffers from a lack of interoperability across platforms. As a result, developers are faced with data silos, high costs, and limited market potential. This is similar to the situation before the Internet when there were competing non-interoperable technologies. The Internet makes it easy to develop networked applications independent of those technologies. W3C is seeking to do the same for the Internet of Things.

The W3C WoT group is based on more than 160 members and has collaborations and liaisons with IETF/IRTF, oneM2M, OPC Foundation, OCF, IIC, Platform Industry 4.0, and AIOTI. Mainly, the W3C WoT group is working on different use cases, requirements, and makes proposals of the WoT technology building blocks. Currently, four main building blocks are considered: Scripting API, Thing Description, Security, and Discovery. Current working assumptions are evaluated regularly based on different plugfest tasks. Valuable outcomes are used to improve the building blocks. There is a decision that W3C WoT is going to be a standardization group (working group) to standardize the WoT building blocks which will help to overcome domain silos and enable cross domain IoT applications.

In the next section, we will go more into detail of the mentioned Thing Description which enables to describe Things capabilities in a semantic manner.

6.3.2. Thing Description

In different kinds of use cases regarding Things such as in system engineering, application development, identification/discovery etc, specific knowledge is needed like what kind of data and functions does the *Thing* serve, how to access the data and functions, and/or what are the security constraints. The idea of the Thing Description (TD) is to provide a semantic enrichment to answer such kind of questions which enables to identify the semantic meaning by knowing the domain (uni-)specific context and to get all information to set up the corresponding interaction models regarding the supported protocol binding.

Figure 41 shows the basic model, what can be addressed within a TD. The TD relies on the Resource Description Framework (RDF) as an underlying data model. This data model also renders the TD to be extendable, e.g., to involve domain or vendor specific information. As a current serialization format JSON-LD has been proposed in order to benefit from both the widely used JSON-based format and JSON-LD's concept of @context (that provides the mapping from JSON to an RDF model). Since JSON-LD is a text-based representation, a TD may be a burden for resource constrained devices. Based on this, additional binary RDF encoding formats that handle string redundancies well will also be considered in the future.

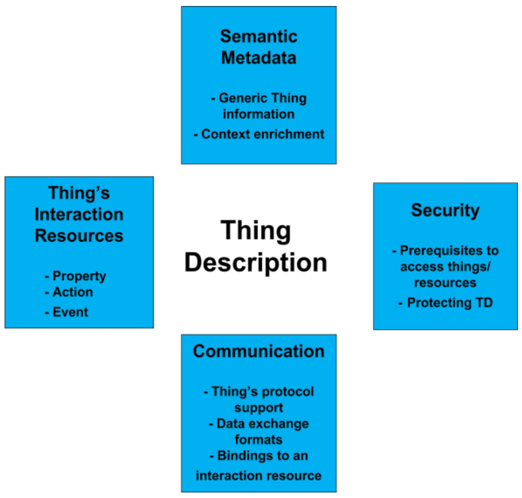
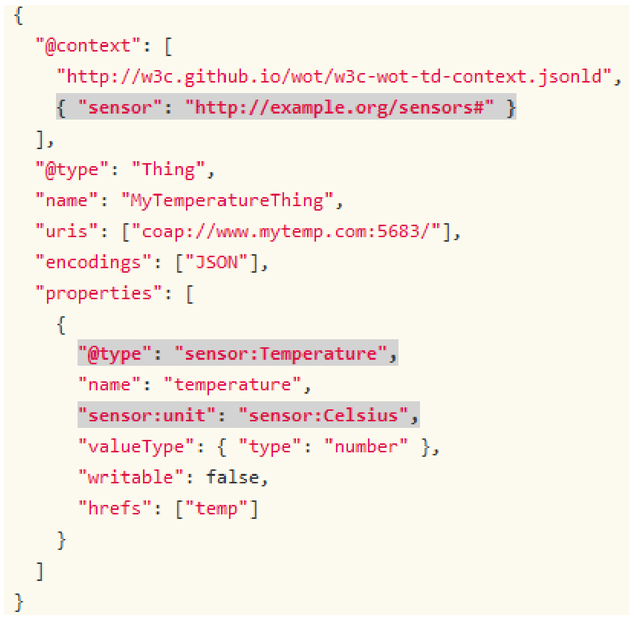

Semantic Metadata
- Generic Thing information
- Context enrichment
Thing's Interaction Resources
- Property
- Action
- Event
Thing Description
Security
- Prerequisites to access things/ resources
- Protecting TD
Communication
- Thing's protocol support
- Data exchange formats
- Bindings to an interaction resource

Figure 41 Concept of the Thing Description



```
{
  "@context": [
    "http://w3c.github.io/wot/w3c-wot-td-context.jsonld",
    { "sensor": "http://example.org/sensors#" }
  ],
  "@type": "Thing",
  "name": "MyTemperatureThing",
  "uris": ["coap://www.mytemp.com:5683/"],
  "encodings": ["JSON"],
  "properties": [
    {
      "@type": "sensor:Temperature",
      "name": "temperature",
      "sensor:unit": "sensor:Celsius",
      "valueType": { "type": "number" },
      "writable": false,
      "hrefs": ["temp"]
    }
  ]
}
```

Figure 42 TD sample based on JSON-LD

Figure 42 shows a simple TD sample of a temperature sensor that supports CoAP binding and JSON as the data exchange format. The temperature value is defined as a Thing property. Based on the predefined implicit mapping to a concrete protocol and the information within the TD we are able to access the temperature resource via a GET on the resource *coap://mytemp.example.com:5683/temp.* The intention of the TD will be to support a rich set of established protocols such as HTTP/CoAP, MQTT, and Web Sockets as well as legacy protocols such as BACnet.

This sample also shows the integration of external context that leads to additional semantic enrichment of a Thing's capabilities (highlight in grey) such as unit (Celsius) and kind of sensor (Temperature). This is one of the strengths of TD that allows the integration of existing domain unspecific ontologies such as SSN and domain specific ontologies from the building domain to increase interoperability.

6.3.3. Conclusion

The W3C WoT shows promising approaches that enables an easy development of application for the IoT in the future. Especially the usage of the Thing Description enables a first entry point to understand Things' capabilities and how to use them (similar to a index.html webpage in the web). Thereby, there are also safety precautions to protect the Thing and its data. In general, such a TD approach can be always offered in IoT applications, even to legacy systems. In the context of building automation it would simplify the engineering, and support plug & play scenarios as well as cross domain applications.

7. Domain Specific Modeling

7.1. Model-Driven Software Development

Nowadays, a software architect uses special software architecture tools with (mostly graphical) editors to create a model, like for example a UML class diagrams or a UML sequence diagrams.

Afterwards, to generate executable software, a developer has to manually create some text files containing source code which meets the formal rules of the used programming language.

This approach is problematic, because it may happen that the same information is kept in two different models or tools which can leads to inconsistencies (media break).

Another problem in this method is that the models created by a software architect often do not have the same level of abstraction the software developer needs for his development work. The software architects wants to describe the software in terms of higher level building blocks, while the developer uses concepts from the underlying programming language like classes for example. The lack of relationships between the different abstractions levels in the models allows a considerable freedom of development, which is not in accordance with a unified reproducible software architecture.

To address the above-mentioned problems in the process of creating executable software, Model-Driven Software Development (MDSD) as a generic term offers techniques and assistance:

"MDSD is a software development methodology, where formal models are used to generate executable software in an automated way" [280, p. 11]

In MDSD formal models play a key role in developing software ("Everything is a model") and are therefore part of the (executable) code. This is a more far-reaching approach as for example existing round-trip engineering tools (RTE) are providing, where changes in the modeling artefacts leads to automatic changes in the source code (e.g. generation of method bodies etc.).

The term "formal" implies that every used model satisfies a formal structure. The model is not meant to be complete but to describe a certain well-defined aspect. Often ***Metamodels*** are used to express such a formal structure.

Not one model but multiple models are used in the application of model-based techniques. As the models describe different aspects at different abstraction levels, ***model-transformations*** are used to integrate them and to generate a high degree of schematic code which is supplemented by manually written code.

Thus, the model-driven software development brings with it many advantages:

- Better handling of complexity
- Improvement of reusability
- Use of automation options
- Increase the software quality in general
- Increased portability through the use of technology-neutral models

There are also alternative approaches to model-driven development. Although they share some common ideas and concepts, they are different in detail:

Generative Programming [281], Software Factories [282], Model-Integrated Computing (MIC) [283], Language Oriented Programming [284], Model-Centric Software Development (MCSD) [285] and the Model Driven Architecture (MDA) [286].

7.2. Metamodeling

A Metamodel is a model of a modeling language. The Metamodel describes the modeling language at a higher level of abstraction, so it must be able to capture all the abilities and characteristics of the model. Thus, it defines the abstract syntax and the static semantics of a modeling language.

Metamodel and model are in a class-instance relationship: Each model is an instance of a Metamodel. A Metamodel can be defined, in principle, by any means, for example, in textual form, to describe model elements and their relationships. Since usually several Metamodels are used in a model-driven software development, it has been found to be advantageous to define meta-models in a unified way. To this end, the concept of Metametamodels is introduced.

The Metametamodel describes the Metamodel. Between the Metametamodel and the Metamodel also a class-instance relationship holds. Although one could introduce any number of meta-levels in this way, one does not go beyond the Metametamodel. Therefore the Metametamodel is described with the same concepts it defines for itself.

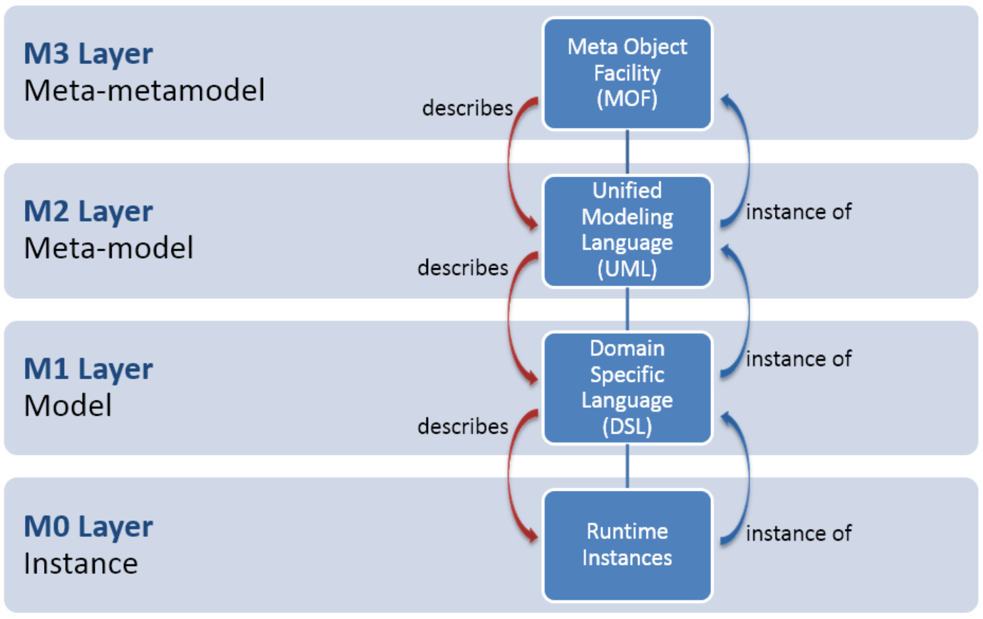


Figure 43: Example of the four-layered MOF Metamodel architecture

The meta-layers and their relationships are described in a meta-model architecture. Thus, the Meta Object Facility (MOF) from the Object Management Group (OMG) [287] defines four layers, that are numbered from M0 (Instances) to M3 (Metametamodel).

Sometimes the term “Meta” in the literature is used relatively between two model levels. In must therefore be considered, to which model the term refers to.

Concrete Metamodels are for example MetaGME [288], Ecore [289], EMOF and CMOF [290]. But also languages like XML Schema [291] or EBNF grammars are suitable for metamodeling, even though they were not originally designed for it.

7.3. Domain Specific Languages

According to the definition from Martin Fowler's book about Domain-Specific-Languages [292] *"A Domain-specific language (DSL) is a computer programming language of limited expressiveness focused on a particular domain."*

The definition of a DSL has multiple aspects:

A DSL is a computer programming language with the goal to do something useful on a computer: One important aspect is that the DSL can be "executed" on a computer. In order to be processed by the computer, a DSL also has a formal structure which is for example described by means of a Metamodel.

In the context of "programing languages" the developer is speaking of "code" which fulfills the formal structure and rules of the language. A little abstracted, this "code" is nothing more than a "model" or an instance of the DSLs Metamodel (see previous chapter). Therefore in this context the terms "code", "model" or "program" are interchangeable, as described in [293, p. 25].

The expressiveness of the language is limited in contrast to general purpose languages (GPLs) whose goal is to be Turing complete. Subsequently GPLs are large and complex and certainly harder to learn in comparison to smaller, more specialized and limited languages which are only applicable to certain domains. This is seen as beneficial because it enables domain experts to define a rich semantic which focuses on their domain only in a very efficient way. A fluent and human-readable DSL syntax is helpful for easy and effective application of the DSL.

Even though it is possible to do MDSD with general purpose modeling languages like UML, "Defining and using DSLs is a flavor of MDSD" because "formal, tool-processable representations of specific aspects of software systems" are created [293, p. 32].

Unfortunately, DSLs do also have some known disadvantages.

An important success factor in the creation of DSL are the tools, even though their functionality is yet remarkable, there is still room for improvement. Currently, most tools come from the research or academia, there are very few commercial tools. Important tool considerations are: How big is the learning curve? How easy is the tool to use? How do the tools support the implementation of new languages? Especially it is advantageous if the tools support modularity and reuse while creating DSLs.

In addition (good) DSLs are hard to design which has various causes. For one, a DSL must be adequately designed to catch up all needed (execution) semantic: in particular the DSL must have a consistent and prescriptive model, which is a model which tells exactly what to do, enabling the computer to generate the target system. It is also difficult to create a very readable syntax.

And then finally it is still necessary to define the model-transformations in case multiple DSLs are used together. This requires a precise description of how a concept in DSL A can be translated into concept or concepts of DSL B. In general, concepts of source DSLs are more abstract than concepts in the target language. Additional levels of abstraction can of course also complicate the development process.

An important distinction has to be made in terms of the involved stakeholders:

- The *DSL software engineer*, whose task is to build new DSLs and to define model-transformations, needs in-depth knowledge about how to design and implement new DSLs.
- The DSL user, who takes an existing DSL and builds target systems with it

The requirements to the stakeholders are of course different. In any case, both stakeholders should have good domain knowledge.

The tooling question is relevant for both the DSL engineer and the DSL user. Fortunately, the last two points (creation of DSL and defining model-transformation among DSLs) need to be developed only once for a certain domain. To conclude the effort to create the DSLs should not be greater than the boost of productivity a DSL user can gain in its domain which brings us to the benefits of DSLs.

Apart from the boost of productivity, DSLs also enables domain experts in taking part in the development process, which is certainly a gain in the overall quality of the software. It is even possible that a domain expert can create software programs on its own in his area of application, where previously trained programmers have been needed.

But also experienced programmers can benefit from DSLs. Since DSLs usually have a higher abstraction in their used concepts, they can be mapped to lower level GPL in a clean and reproducible way.

A good example which is implemented in nearly every DSL are state machines. The DSL syntax allows describing them in a very brief way, while the generated code is much longer.

7.3.1. Categorization of DSLs

7.3.1.1. Internal versus External

There are two main styles of DSLs which are distinguished: Internal and External DSLs.

An internal DSL is a language which is embedded into another language (therefore the term embedded DSL is also found in literature). The host language, usually a dynamically typed general purpose language defines the syntactic frame, because the DSL is always a proper subset of the GPL. As a consequence syntactic constraints are strongly related to the host language. On the other side the DSL implementer is able to reuse the facilities from the host language (parser, type system, etc.) resulting in lower implementation time.

Internal DSL Example with Ruby:

In Ruby there are a number of patterns like method chaining, function sequence or nested functions which are combined to build up internal DSLs. The basic idea is that with the help of these patterns, ruby methods, functions and parameters are defined in the right way so that they reflect the structure of the DSL. After the DSL structure has been set up in the ruby code, the DSL is loaded dynamically with the ruby load function. The DSL code which is of course also a Ruby program is parsed and executed by the Ruby interpreter which then calls the defined methods (and patterns) in the right way.

So for example the following mini-Multiple-Choice DSL which can only describe a list of multiple-choice questions and their answers could be easily implemented in Ruby with the function sequence and chained method patterns: the keywords "*question*", "*is*" and "*correct*" would map to ruby methods while the real question in single quotes is a method parameter in the example.

The Mini-Multiple-Choice DSL:

```
question 'DSL is an acronym for ...'
answer 'Digital Subscriber Line' is correct
answer 'Domain Specific Language' is correct
answer 'I do not care' is wrong
```

The limits of this approach are of course closely related to the host programming language. Furthermore, it would not be possible to implement a complete XML DSL in this way.

If more control over the DSL is needed, ***external DSLs*** are useful. In contrast to internal DSL, external DSLs are running outside the scope of a GPL and can therefore be used stand-alone. The concrete syntax and the semantics can be defined freely, which makes external DSLs very flexible and expressive.

Of course, to create such DSLs tools from the classic compiler construction area like lex/yacc can be used, but that would involve a lot of effort. Today's existing DSLs tools provide better support in implementing and parsing DSLs.

7.3.1.2. Graphical versus Textual

Traditionally programming languages with a textual syntax have prevailed. A textual syntax is precise, concise and also well suited to model behavior oriented applications. By distributing the code into multiple files and directories, big applications also scale well.

During implementation, the developer creates code in a (maybe sophisticated) text editor and if he thinks that his code is valid, the code is saved to disk and the compiler starts his work, beginning to read a stream of bytes from disk and to analyze the code in different phases.

Graphical languages have evolved in a slightly different direction. A graphical editor assists the user in creating a graphical representation, for example a diagram, of the model. Most of these models have been used for 'documentation' purposes and modeling languages like UML are wide-spread. The diagram is usually closely related to the tool and must not be parsed, unlike text files, because the editor ensures that it is valid from the beginning.

The strengths of graphical DSLs is obviously the ability to better visualize relationships among its entities. Graphical DSLs are particularly suitable to represent complex relationships and interdependencies of a limited number of elements.

7.3.2. Abstract versus concrete Syntax

The concrete syntax defines the notion in which developers are writing the code. It is the visual output of the editor the developer can see and interacts with, weather it is textual, graphical, tabular or any combination of the three.

The abstract syntax defines the structure of the DSL and holds semantically relevant information of the language (e.g. in terms of concepts). It represents the core of the model and is important for further processing (model-transformations, execution semantic, etc.).

As the abstract syntax is often parsed into a tree, it is also called the Abstract Syntax Tree (AST). Metamodeling is a suitable technique to define the structure of an AST.

Both abstract and concrete syntax are separated for a clean separation of concern.

7.3.3. Projectional Editing

While classic parser directly work on the bytes of stream containing the source code and parsing against a grammar, projectional editors use a different paradigm and therefore work different.

A projectional editor directly works on the abstract syntax tree of the model and has at least one clearly defined unidirectional projection or mapping to a concrete syntax of the model. All relevant information regarding the model is kept in the abstract syntax, the concrete syntax is used as "view" presenting the content of the model to the user.

The input of the user is first processed by the editor in terms of single keyboard events and gestures. A second mapping maps the user input directly to actions which is manipulating the abstract syntax tree of the model. Afterwards changes in the abstract syntax tree are propagated to the concrete syntax so that the user has some visual feedback regarding the model.

Because concrete and abstract syntax are separated in projectional editors, this allows having multiple mapping from the abstract syntax into a concrete syntax (that is to define multiple views of the model) which creates some useful application cases: For example it is possible to have multiple syntax definitions for the same DSL. This enables also to combine the strength of textual, tabular or graphical representations.

7.4. Model Transformations and Code generation

Model Transformations are an integral part of MDSD and multiple Model Transformation Languages do exist like for example Query/Views/Transformations (QVT) which allows defining a transformation between MOF-based Metamodels.

Model-transformations are necessary and important to integrate multiple models and to close the gap between the different levels of abstraction because the MDSD approach is to have multiple smaller models with higher abstraction levels. To come to executable software the models must be transformed into lower level general purpose languages (e.g. Java or C).

Model-transformation can be categorized into

- Transformation which directly modifies the model:
  The model-transformation uses the model as input and as output model. It is clear that such transformations have to use the same concepts of the model and cannot introduce new concepts from other languages. For example a model-transformation can change existing model entities or add new model entities.
- Model-to-Model transformations:
  In the model-2-model transformation a source model with Metamodel A is transformed into a target model B which has some different Metamodel. The definition of the transformation is only one way. The source model is left unchanged. Only the results of the target models are needed for further processing for example. Usually the source model uses higher abstraction in its concepts than the target model.

- Modell-Weaving (Linking):
  Model-Weaving takes two different models (with different Metamodels) and combines them by “weaving” them together. Connection between the single models elements are called links or references.
- Model-to-Text Transformation / Code generation:
  Model-to-Text transformation focus on the creation of textual artefacts. They are created in the last transformation step. Most often, the textual artefacts are used for further processing in external tools regarding the model-driven-tool chain. A reverse flow back into the model does not occur. To keep the costs of building external DSL low, a defined text transformation maps the model to general purpose source code which is then further processed with conventional compiler tools. Thus, the execution semantics is precisely defined.

7.5. Language Workbench and Tools

The success in the application of model-based techniques depends heavily on the used tools. The term “Language Workbench” has been coined by Martin Fowler in 2005:

“A language workbench is an environment designed to help people create new DSLs, together with high-quality tooling required to use those DSLs effectively.” [292]

Since 2011 the annual Language Workbench Challenge is organized where tool manufacturer and researchers meet to hold a small DSL challenge [294]. The result of this challenge gives a good impression about the current state of the art in the DSL tooling landscape. A comparison of the tools based on a feature model can be found in [295].

The following enumeration of some graphical and textual DSL tools makes no demand in being comprehensive.

7.5.1. Graphical DSLs

In the graphical DSL area a lot of tools and framework do exist.

Microsoft has released some more or less successful products regarding DSLs: Visual Studio DSL Tools [296] and the SQL Server modeling project (formerly known as "Oslo") with the modelling Language M which has support for textual and diagrammatic projectional notions (Quadrant).

One of the few commercial DSL language workbench is the Intentional Domain Workbench by Intentional Software [297] which is a quite sophisticated workbench providing reversible transformations and a projectional editor with support for graphical, tabular and textual syntax.

Another commercial Eclipse EMF based language workbench is „Poseidon for DSL” by the company Gentleware [298].

The company Metacase provides with the product MetaEdit+ also a language workbench and plugins which integrate into Eclipse or Visual Studio IDEs [299].

Of course, the tool vendor of the traditional software architecture UML tools like for example MagicDraw by No Magic provide the ability to define own DSLs [300].

A very well-known open-source graphical modeling framework with a big community is the Eclipse Modeling Framework (EMF) and the Graphical Modeling Framework (GMF). It supports defining models, generating code, creating graphical editors and much more. A comprehensive overview is given in the following book [289].

7.5.2. Textual DSLs

General purpose programming language suitable to define internal textual DSLs are:

- Common Lisp Object System (http://www.dreamsongs.com/CLOS.html)
- Ruby (https://www.ruby-lang.org)
- Compile-time meta-programming (http://convergepl.org/documentation/current/ctmp)
- Clojure (http://clojure.org)
- Scala (http://www.scala-lang.org)

Tools to define text-based external DSLs

- XTEXT - www.eclipse.org/Xtext
- XTEXT is based on the Eclipse EMF ecosystem and is a text parser based system. It supports a separation of programming and modeling. Through its widespread use it can be called as “mainstream” state-of-the-art. ANTLR is used as a parser backend and integrates well with the Eclipse tooling
- Spoofax (http://strategoxt.org/Spoofax/WebHome)
- SDF (Syntax Definition Formalism, http://www.syntax-definition.org)

7.5.3. Case Study - Jetbrains MPS

Jetbrains MPS [301] has gained a lot of reputation in the DSL community because it supports advanced IDE capabilities like code completion, go to definition, real-time type checks, quick fixes or refactoring. The tool is based on IntelliJ IDEA and is developed by Jetbrains (the makers of the well-known IntelliJ IDEA Java IDE).

The IDE is a projectional editor and supports syntax driven editing, thus no parser is used and editor gestures do directly manipulate the abstract syntax tree. The language workbench has its main focus on textual syntax but also supports tabular or (limited) graphical syntax. The graphical capabilities however, can be extended by implementing custom java swing components.

Languages can be combined in very powerful ways, for example language composition can be done with inheritance between languages. This makes it possible to build new languages from available languages.

In comparison to XTEXT, MPS supports with the help of a ‘type system’-DSL a deduction rule based system to implement custom type system within DSLs.

The first step in writing a new DSL is to define the structure of the DSL. This is supported with an own ‘structure’-DSL which defines for example ‘concepts’ that correspond to a metaclass.

In the second step projection rules for the editors are defined. They define how a particular abstract concept is to be represented visually. Reusable building blocks are provided to implement the editor visualization.

Model-to-Model and Model-to-text transformations are well supported.

8. Supporting Technologies

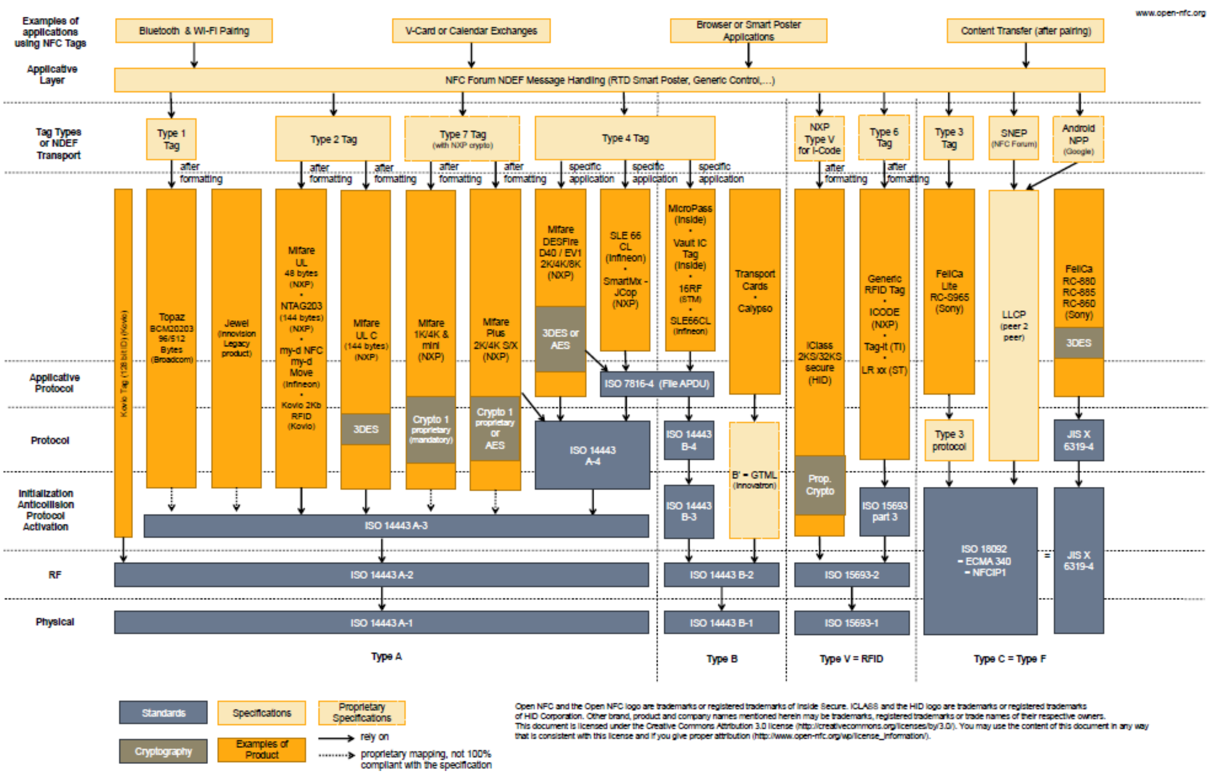
8.1. NFC

8.1.1. Introduction

Near Field Communication (NFC) [302] is a standards-based short-range wireless connectivity technology that makes life easier and more convenient for consumers around the world by making it simpler to make transactions, exchange digital content, and connect electronic devices with a touch. NFC is compatible with hundreds of millions of contactless cards and readers already deployed worldwide.

Present and anticipated applications include contactless transactions, data exchange, and simplified setup of more complex communications such as Wi-Fi. Communication is also possible between an NFC device and an unpowered NFC chip, called a "tag".

NFC standards cover communications protocols and data exchange formats, and are based on existing radio-frequency identification (RFID) standards including ISO/IEC 14443 and FeliCa. The standards include ISO/IEC 18092 and those defined by the NFC Forum, which was founded in 2004 by Nokia, Philips Semiconductors (became NXP Semiconductors since 2006) and Sony, and now has more than 160 members. The Forum also promotes NFC and certifies device compliance and if it fits the criteria for being considered a personal area network.



www.open-nfc.org
Examples of applications using NFC Tags
Bluetooth & Wi-Fi Pairing
V-Card or Calendar Exchanges
Browser or Smart Poster Applications
Content Transfer (after pairing)
Applicative Layer
NFC Forum NDEF Message Handling (RTD Smart Poster, Generic Control,...)
Tag Types or NDEF Transport
Type 1 Tag
Type 2 Tag
Type 7 Tag (with NXP crypto)
Type 4 Tag
NXP Type V for I-Code
Type 6 Tag
Type 3 Tag
SNEP (NFC Forum)
Android NPP (Google)
after formatting
specific application
Kovio Tag (128 bit ID) (Kovio)
Topaz BCM20203 96/512 Bytes (Broadcom)
Jewel (Innovision Legacy product)
Mifare UL 48 bytes (NXP) • NTAG203 (144 bytes) (NXP) • my-d NFC my-d Move (Infineon) • Kovio 2Kb RFID (Kovio)
Mifare UL C (144 bytes) (NXP)
3DES
Mifare 1K/4K & mini (NXP)
Crypto 1 proprietary (mandatory)
Mifare Plus 2K/4K S/X (NXP)
Crypto 1 proprietary or AES
Mifare DESFire D40 / EV1 2K/4K/8K (NXP)
3DES or AES
SLE 66 CL (Infineon) • SmartMx - JCop (NXP)
MicroPass (Inside) • Vault IC Tag (Inside) • 16RF (STM) • SLE66CL (Infineon)
Transport Cards • Calypso
IClass 2KS/32KS secure (HID)
Prop. Crypto
Generic RFID Tag • ICODE (NXP) • Tag-it (TI) • LR xx (ST)
FeliCa Lite RC-S965 (Sony)
LLCP (peer 2 peer)
FeliCa RC-880 RC-885 RC-860 (Sony)
3DES
Applicative Protocol
ISO 7816-4 (File APDU)
Protocol
ISO 14443 A-4
ISO 14443 B-4
B' - GTML (Innovatron)
Type 3 protocol
JIS X 6319-4
Initialization Anticollision Protocol Activation
ISO 14443 A-3
ISO 14443 B-3
ISO 15693 part 3
ISO 18092 - ECMA 340 - NFCIP1
=
JIS X 6319-4
RF
ISO 14443 A-2
ISO 14443 B-2
ISO 15693-2
Physical
ISO 14443 A-1
ISO 14443 B-1
ISO 15693-1
Type A
Type B
Type V = RFID
Type C = Type F
Standards
Specifications
Proprietary Specifications
Cryptography
Examples of Product
rely on
proprietary mapping, not 100% compliant with the specification

Figure 44: NFC architecture

8.1.2. Features

NFC is a set of short-range wireless technologies [303], typically requiring a distance of 10 cm or less. NFC operates at 13.56 MHz on ISO/IEC 18000-3 air interface and at rates ranging from 106 Kbit/s to 424 Kbit/s. NFC always involves an initiator and a target; the initiator actively generates an RF field that can power a passive target. This enables NFC targets to take very simple form factors such as tags, stickers, key fobs, or cards that do not require batteries. NFC peer-to-peer communication is possible, provided both devices are powered. A patent licensing program for NFC is currently under deployment by France Brevets, a patent fund created in 2011. The program under development by Via Licensing Corporation, an independent subsidiary of Dolby Laboratories, terminated in May 2012. A public, platform-independent NFC library is released under the free GNU Lesser General Public License by the name libnfc.

NFC tags contain data and are typically read-only, but may be rewriteable. They can be custom-encoded by their manufacturers or use the specifications provided by the NFC Forum, an industry association charged with promoting the technology and setting key standards. The tags can securely store personal data such as debit and credit card information, loyalty program data, PINs and networking contacts, among other information. The NFC Forum defines four types of tags that provide different communication speeds and capabilities in terms of configurability, memory, security, data retention and write endurance. Tags currently offer between 96 and 4,096 bytes of memory.

Two communication modes:

Passive communication mode: The initiator device provides a carrier field and the target device answers by modulating the existing field. In this mode, the target device may draw its operating power from the initiator-provided electromagnetic field, thus making the target device a transponder.

Active communication mode: Both initiator and target device communicate by alternately generating their own fields. A device deactivates its RF field while it is waiting for data. In this mode, both devices typically have power supplies.

ISO/IEC 18000-3 [304] is an international standard for passive RFID item level identification and describes the parameters for air interface communications at 13,56 MHz The target markets for MODE 2 are in tagging systems for manufacturing, logistics, retail, transport and airline baggage. MODE 2 is especially suitable for high speed bulk conveyor fed applications.

MODE 2 RFID tags are passive deriving their power from the interrogating signal generated by an RFID interrogator. Power is transferred from the interrogator to the tag by a high frequency magnetic field using coupled antennae coils in the reader and the tag. The powering field frequency is 13,56 MHz ± 7 kHz.

Dialogue between the interrogator and the tag is conducted on an Interrogator-Talks-First (ITF) basis. Following activation of the tag by the interrogator's interrogating signal the tag waits silently for a valid command. After receiving a valid command the tag transmits a reply in response to the command. The air interface operates as a full duplex communication link. The interrogator operates with full duplex transmissions being able to transmit commands whilst simultaneously receiving multiple tag replies. Tags operate with half duplex transmissions.

Commands are transmitted from the interrogator to the tag by Phase Jitter Modulation (PJM) of the powering field. PJM transmits data as very small phase changes in the powering field. There is no reduction in the transfer of power to the tag during PJM and the bandwidth of PJM is no wider than the original double-sided spectrum of the data. The PJM sideband levels and data rates are decoupled allowing the sideband levels to be set at any arbitrary level without affecting the data rate. The command data rate is 423,75 Kbit/s encoded using Modified Frequency Modulation (MFM).

Tags reply to the interrogator by inductive coupling whereby the voltage across the tag antenna coil is modulated by a subcarrier. The subcarrier is derived from division of the powering field's frequency. Tags can select from one of eight subcarrier frequencies between 969 kHz and 3013 kHz. The reply data rate is 105,9375 Kbit/s encoded using MFM and modulated onto the subcarrier as Binary Phase Shift Keying (BPSK). To ensure that tags replying on different channels are simultaneously received tag replies are band limited to reduce data and subcarrier harmonic levels.

Multiple tag identification is performed using a combination of Frequency Division Multiple Access and Time Division Multiple Access (FTDMA). There are eight reply channels available for tags to use. In response to a valid command each tag randomly selects a channel on which to transmit its reply. The reply is transmitted once using the selected channel. Upon receiving the next valid command each tag randomly selects a new channel and transmits the reply using the newly selected channel. This method of reply frequency hopping using random channel selection is repeated for each subsequent valid command. The interrogator can selectively mute identified tags to remove them from the identification process. When a tag is muted the tag will not transmit any replies. In addition to random channel selection the tags can randomly mute individual replies. When a reply is muted the tag will not transmit that reply. Random muting is necessary when identifying very large populations of tags during singulation. All FTDMA frequency and time parameters are defined by the command.

All commands are time stamped and tags store the first time stamp received after entering an interrogator. The stored time stamp defines precisely when the tag first entered the interrogator and provides a high resolution method of determining tag order which is decoupled from the speed of identification. Tag temporary settings, such as the time stamp, are stored in Temporary Random Access Memory (TRAM) that retains data contents during power outages caused by switching of the powering field in orientation insensitive interrogators.

8.1.3. Architecture

By utilizing the key elements in existing and recognized standards like ISO/IEC 18092 and ISO/IEC 14443-2,3,4, as well as JIS X6319-4 [305], the NFC Forum Specifications form a technology standard that harmonizes and extends existing contactless standards unlocking the full capabilities of NFC technology across the different contactless operating modes, peer-to-peer mode, reader/writer mode, card emulation mode.

New versions of nine technical specifications, comprising an integrated and streamlined set designed to be used together, have been published. The new versions deliver greater interoperability, faster read and write performance, mediated handover, and lower power consumption, as well as additional functionality for products incorporating NFC technology. The new set of specifications supports compatibility among NFC devices implemented with previously released versions. The set's components are marked as NEW in the list below.

NFC Logical Link Control Protocol (LLCP) Technical Specification

LLCP defines an OSI layer-2 protocol to support peer-to-peer communication between two NFC-enabled devices, which is essential for any NFC applications that involve bi-directional communications. The specification defines two service types, connectionless and connection-oriented, organized into three link service classes: connectionless service only; connection-oriented service only; and both connectionless and connection-oriented service. The connectionless service offers minimal setup with no reliability or flow-control guarantees (deferring these issues to applications and to the reliability guarantees offered by ISO/IEC 18092 and ISO/IEC 14443 MAC layers). The connection-oriented service adds in-order, reliable delivery, flow-control, and session-based service layer multiplexing.

LLCP is a compact protocol, based on the industry standard IEEE 802.2, designed to support either small applications with limited data transport requirements, such as minor file transfers, or network protocols, such as OBEX and TCP/IP, which in turn provide a more robust service environment for applications. The NFC LLCP thus delivers a solid foundation for peer-to-peer applications, enhancing the basic functionality offered by ISO/IEC 18092, but without affecting the interoperability of legacy NFC applications or chipsets.

NFC Digital Protocol Technical Specification

This specification addresses the digital protocol for NFC-enabled device communication, providing an implementation specification on top of the ISO/IEC 18092 and ISO/IEC 14443 standards. It harmonizes the integrated technologies, specifies implementation options and limits the interpretation of the standards; in essence, showing developers how to use NFC, ISO/IEC 14443 and JIS X6319-4 standards together to ensure global interoperability between different NFC devices, and between NFC devices and existing contactless infrastructure.

The specification defines the common feature set that can be used consistently and without further modification for major NFC applications in areas such as financial services and public transport. The specification covers the digital interface and the half-duplex transmission protocol of the NFC-enabled device in its four roles as Initiator, Target, Reader/Writer and Card Emulator. It includes bit level coding, bit rates, frame formats, protocols, and command sets, which are used by NFC-enabled devices to exchange data and bind to the LLCP protocol.

NFC Activity Technical Specification

The specification [306] explains how the NFC Digital Protocol Specification can be used to set up the communication protocol with another NFC device or NFC Forum tag. It describes the building blocks, called Activities, for setting up the communication protocol. These Activities can be used as defined in this specification or can be modified to define other ways of setting up the communication protocol, covering the same or different use cases. Activities are combined in Profiles. Each Profile has specific Configuration Parameters and covers a particular use case. This document defines Profiles polling for an NFC device and establishment of Peer to Peer communication, polling for and reading NFC Data Exchange Format (NDEF) data from an NFC Forum tag, and polling for a NFC tag or NFC device in combination.

The combination of Activities and Profiles define a predictable behavior for an NFC Forum device. This does not limit NFC Forum devices from implementing other building blocks or defining other Profiles – for other use cases – on top of the existing ones.

NFC Simple NDEF Exchange Protocol (SNEP) Specification

The Simple NDEF Exchange Protocol (SNEP) allows an application on an NFC-enabled device to exchange NFC Data Exchange Format (NDEF) messages with another NFC Forum device when operating in NFC Forum peer-to-peer mode. The protocol makes use of the Logical Link Control Protocol (LLCP) connection-oriented transport mode to provide a reliable data exchange.

NFC Analog Technical Specification

This specification addresses the analogue characteristics of the RF interface of the NFC-Enabled Device. The purpose of the specification is to characterize and specify the externally observable signals for an NFC-Enabled Device without specifying the design of the antenna of an NFC-Enabled Device. This includes power requirements (determining operating volume), transmission requirements, receiver requirements, and signal forms (time/frequency/modulation characteristics).

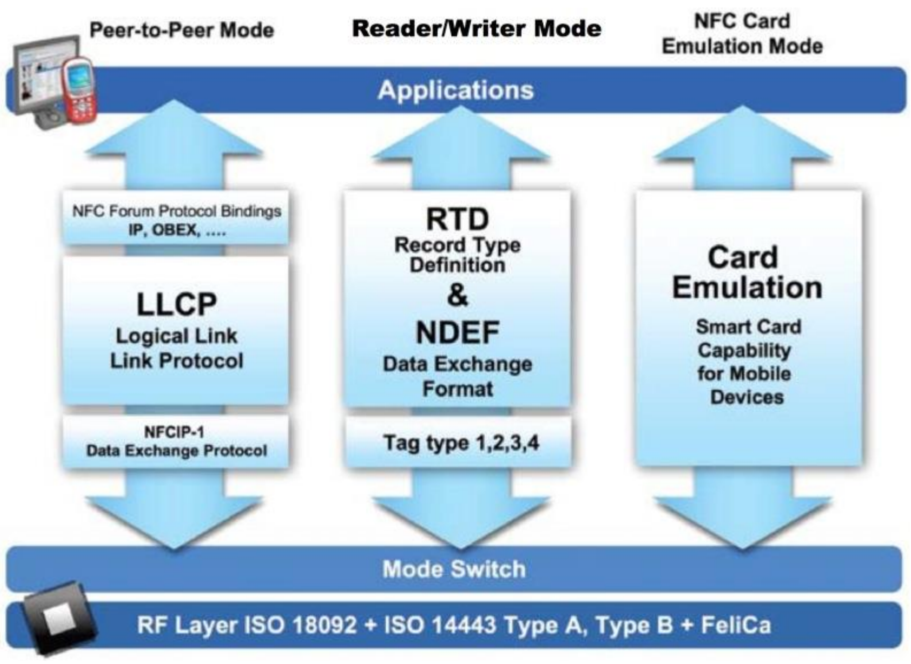
This document is intended for use by manufacturers wanting to implement an NFC-Enabled Device. The scope of this document covers the analog interface of the NFC-Enabled Device in its four roles (Peer Mode Initiator, Peer Mode Target, Reader/Writer Mode and Card Emulation Mode) for all three technologies (NFC-A, NFC-B, and NFC-F) and for all the different bit rates (106kbps, 212kbps, and 424kbps).

NFC Controller Interface (NCI) Technical Specification

The NCI specification defines a standard interface within an NFC device between an NFC controller and the device’s main application processor. The NCI makes it easier for device manufacturers to integrate chipsets from different chip manufacturers, and it defines a common level of functionality and interoperability among the components within an NFC-enabled device. With the availability of the NCI, manufacturers will have access to a standard interface they can use for whatever kind of NFC-enabled device they build – including mobile phones, PCs, tablets, printers, consumer electronics, and appliances. This will enable manufacturers to bring new NFC-enabled devices to market faster. The NCI provides users a logical interface that can be used with different physical transports, such as UART, SPI, and I2C.

8.1.4. Standards

ISO/IEC 18092:2013 [307] defines communication modes for Near Field Communication Interface and Protocol (NFCIP 1) using inductive coupled devices operating at the center frequency of 13,56 MHz for interconnection of computer peripherals. It also defines both the Active and the Passive communication modes of Near Field Communication Interface and Protocol (NFCIP-1) to realize a communication network using Near Field Communication devices for networked products and also for consumer equipment. ISO/IEC 18092:2013 specifies, in particular, modulation schemes, coding, transfer speeds, and frame format of the RF interface, as well as initialization schemes and conditions required for data collision control during initialization. Furthermore, ISO/IEC 18092:2013 define a transport protocol including protocol activation and data exchange methods.



Peer-to-Peer Mode
Reader/Writer Mode
NFC Card Emulation Mode
Applications
NFC Forum Protocol Bindings IP, OBEX, ...
LLCP Logical Link Link Protocol
NFCIP-1 Data Exchange Protocol
RTD Record Type Definition & NDEF Data Exchange Format
Tag type 1,2,3,4
Card Emulation Smart Card Capability for Mobile Devices
Mode Switch
RF Layer ISO 18092 + ISO 14443 Type A, Type B + FeliCa

Figure 45: NFC Standard

ISO/IEC 14443 [308] is a four-part international standard for Contactless Smart Cards operating at 13.56 MHz in close proximity with a reader antenna. Proximity Integrated Circuit Cards (PICC) are intended to operate within approximately 10cm of the reader antenna.

Part 1 [ISO/IEC 14443-1:2000(E)] defines the size and physical characteristics of the card. It also lists several environmental stresses that the card must be capable of withstanding without permanent damage to the functionality. These tests are intended to be performed at the card level and are dependent on the construction of the card and on the antenna design; most of the requirements cannot be readily translated to the die level. The operating temperature range of the card is specified in Part 1 as an ambient temperature range of 0°C to 50°C.

Part 2 [ISO/IEC 14443-2:2001(E)] defines the RF power and signal interface. Two signaling schemes, Type A and Type B, are defined in part 2. Both communication schemes are half duplex with a 106 Kbit per second data rate in each direction. Data transmitted by the card is load modulated with a 847.5 kHz subcarrier. The card is powered by the RF field and no battery is required.

Part 3 [ISO/IEC 14443-3:2001(E)] defines the initialization and anti-collision protocols for Type A and Type B. The anti-collision commands, responses, data frame, and timing are defined in Part 3. The initialization and anti-collision scheme is designed to permit the construction of multi-protocol readers capable of communication with both Type A and Type B cards. Both card types wait silently in the field for a polling command. A multi-protocol reader would poll one type of card, complete any transactions with cards responding and then poll for the other type of card and transact with them.

Part 4 [ISO/IEC 14443-4:2001(E)] defines the high-level data transmission protocols for Type A and Type B. The protocols described in Part 4 are optional elements of the ISO/IEC 14443 standard; proximity cards may be designed with or without support for Part 4 protocols. The PICC reports to the reader if it supports the Part 4 commands in the response to the polling command (as defined in Part 3). The protocol defined in Part 4 is also capable of transferring application protocol data units as defined in ISO/IEC 7816-4 and of application selection as defined in ISO/IEC 7816-5. Note that ISO/IEC 7816 is a Contacted Integrated Circuit Card standard.

This application note is intended to summarize the requirements of ISO/IEC 14443 that apply to Type B integrated circuits. It is not intended to describe all possible interpretations of these requirements. The requirements in Part 1 and for Type A cards will not be discussed in detail. Part 4 requirements are not discussed in detail. Recent amendments to the ISO/IEC 14443 standards are beyond the scope.

8.1.5. Why is it necessary to BaaS?

Mainly, the NFC technology is a cheap technology that is increasingly being used more today, especially on mobile devices. With this technology, you can take it all in a single device, and in one easy action can: log in (COMPANY IDENTIFICATION users who have booked the room), do feedback ...

All this thanks to one of its main features, High speed setting, virtually instantaneously without pairing. Just by bringing the devices

As for safety, have a great high degree of safety because of its short range.

8.2. Zigbee

8.2.1. Introduction

ZigBee [309] is a specification for a suite of high level communication protocols used to create personal area networks built from small, low-power digital radios. ZigBee is based on an IEEE 802.15 standard. Though low-powered, ZigBee devices can transmit data over long distances by passing data through intermediate devices to reach more distant ones, creating a mesh network; i.e., a network with no centralized control or high-power transmitter/receiver able to reach all of the networked devices. The decentralized nature of such wireless ad hoc networks makes them suitable for applications where a central node can't be relied upon.

ZigBee is used in applications that require only a low data rate, long battery life, and secure networking. ZigBee has a defined rate of 250 Kbit/s, best suited for periodic or intermittent data or a single signal transmission from a sensor or input device. Applications include wireless light switches, electrical meters with in-home-displays, traffic management systems, and other consumer and industrial equipment that requires short-range wireless transfer of data at relatively low rates. The technology defined by the ZigBee specification is intended to be simpler and less expensive than other WPANs, such as Bluetooth or Wi-Fi.

ZigBee networks are secured by 128 bit symmetric encryption keys. In home automation applications, transmission distances range from 10 to 100 meters line-of-sight, depending on power output and environmental characteristics.

ZigBee was conceived in 1998, standardized in 2003, and revised in 2006. The name refers to the waggle dance of honey bees after their return to the beehive.

8.2.2. Features

Basic Software Architecture

ZigBee [310] offers high-level functionality concerned with network structure, message routing and security. This functionality is provided by the ZigBee software layer. This layer sits above another layer concerned with low-level network operation such as addressing and message transmission/reception - this is referred to as the Physical/Data Link level. The application is the highest level software, sitting above the ZigBee layer.

This basic architecture is illustrated in the diagram on the right.

The Physical/Data Link level is based on the **IEEE 802.15.4** wireless network standard, described below.

IEEE 802.15.4 Foundation

As indicated above, the ZigBee software sits on top of the Physical/Data Link level [311], which is provided by the IEEE 802.15.4 standard. This standard brings many of the fundamental principles of ZigBee networks, including:

- Ultra-low power consumption
- Use of unlicensed radio bands
- Easy installation
- Low cost

ZigBee builds on the IEEE 802.15.4 functionality by adding capabilities for more flexible network topologies, intelligent message routing and enhanced security measures.

The Physical/Data Link level comprises two IEEE 802.15.4 layers:

- Media Access Control (MAC) sub-layer
- Physical layer (PHY)

These are illustrated in the diagram on the right and are described later in the course.

Frequency Bands

IEEE 802.15.4 (and therefore ZigBee) was designed to operate in unlicensed radio frequency (RF) bands. The unlicensed RF bands are not the same in all territories of the world, but IEEE 802.15.4 employs three possible bands, at least one of which should be available in a given territory. The three bands are centered on the following frequencies: **868**, **915** and **2400 MHz**

The characteristics and geographical applicability of these RF bands are shown in the diagram on the right, and explained below:

- Each RF band is divided into a number of channels
- There are 27 channels across the three RF bands, numbered 0 to 26
- The three RF bands have different data rates: 20, 40 and 250 kbps (with increasing frequency)

The 868- and 915-MHz frequency bands offer certain advantages such as fewer users, less interference, and less absorption and reflection. However, the 2400-MHz band is far more widely adopted for a number of reasons, including:

- Worldwide availability for unlicensed use
- Higher data rate (250 kpbs) and more channels

- Lower power (transmit/receive are on for a shorter time due to higher data rate)

Channel Selection

Energy detection functionality is included that can be used by higher software layers to avoid interference between radio communications. The best frequency channel can be selected at initialization.

Range of Transmission

The range of a radio transmission is the distance the radio communication can travel before it becomes undetectable. This is dependent on the operating environment - for example, inside or outside a building. The best results are obtained in an open area. Inside a building, the range is reduced due to absorption, reflection, diffraction and standing wave effects caused by walls and other solid objects.

With a standard device (around 0 dBm output power):

- In an open area, a range of over 200 meters can typically be achieved (Jennic has measured in excess of 450 meters)
- In a building, a range of 30 meters can typically be achieved

However, high-power modules (greater than 15 dBm output power) can achieve a range of 5 times greater than a standard module.

Network Topologies

The way that a message is routed from one network node to another depends on the network topology. This page provides a brief description of the possible topologies of a ZigBee network. The ZigBee topologies are described in more detail later in this course.

A ZigBee network can adopt one of the three topologies: **Star**, **Tree**, **and Mesh**. These are illustrated on the right and briefly described below.

Star Topology

A Star network has a central node, which is linked to all other nodes in the network. All messages travel via the central node.

Tree Topology

A Tree network has a top node with a branch/leaf structure below. To reach its destination, a message travels up the tree (as far as necessary) and then down the tree.

Mesh Topology

A Mesh network has a tree-like structure in which some leaves are directly linked. Messages can travel across the tree, when a suitable route is available.

8.2.3. Architecture

The ZigBee RF4CE architecture is defined in terms of a number of blocks or layers in order to simplify the specification. Each layer is responsible for one part of the specification and offers services to the next higher layer and utilizes services from the next lower layer. The interfaces between the layers serve to define the logical links that are described in this specification. The layout of the layers is based on the open systems interconnection (OSI) seven-layer model. The ZigBee RF4CE specification is designed to be built onto the IEEE 802.15.4 standard MAC and PHY layers and provides networking functionality and public application profiles which can interface to the end user application. Manufacturer specific extensions to public application profiles can be defined by sending vendor specific data frames within the profile. In addition, manufacturer specific profiles can also be defined [312].

Functional

Interworking between ZigBee networks and IP is a key for many applications, and the ZigBee Gateway specification must meet many different needs. Therefore the ZigBee Gateway does not define a single protocol; rather it defines a two-layered API:

- A set of abstract (protocol independent) functions:
    - The Gateway specification defines a Remote Procedure Call-based (RPC) API to ZigBee functionality and the management of the IP gateway itself:
        - Support for complete Application
        - Support Layer (APS), ZigBee Device Object (ZDO), and security services (SEC) commissioning both into and out of ZigBee networks
        - It allows interactions between IP applications and the profile applications on ZigBee devices
        - By exposing the application interface in a standard way remote IP applications can be interoperable with Gateways from multiple vendors
        - The set of mandatory functions is reduced at a minimum, allowing differentiation of products based on the different levels of implementation
    - An extensible set of RPC protocols (i.e. bindings) specifying how to expose the API using a specific protocol:
        - The different bindings provide scalability for cost versus feature tradeoffs.
        - We do not have to re-invent interfaces for every system since the platform specification is not bound to a specific profile but gives the tools to IPHAs to support the profiles needed
        - Release 1 of the Gateway specification features SOAP, REST and GRIP bindings
            - SOAP provides higher level web services oriented access to the Gateway API
            - REST provides a lightweight web-based API
            - GRIP is the protocol of choice for simplest ZigBee Gateway Devices, given its tiny footprint

To address diverse applications in an efficient manner, a compliant gateway does not need to implement all the RPC bindings; it is enough to implement only one of them, or possibly more than one, if desired. Application-specific gateways that build on the general ZigBee Gateway specification can indicate which binding must be implemented, depending on the specific use case and/or the typical scenario. This way, a profile could mandate one of the bindings for specific use cases or allow the vendor to decide which binding to implement (or define a fourth binding not described in the specification).

The two-tiered API is matched by a two-layered functional architecture:

- A northbound “interface” implementing at least one of the three bindings
- A protocol-agnostic layer that implements each sub-segment of the overall API:
    - APS, ZDP, ZigBee Cluster Library (ZCL) and ZigBee Network Layer (NWK) expose the different layers of the ZigBee stack.

- Gateway Management Object (GMO) provides access to low-level ZigBee stack functions as well as high-level “macro” functions. These coarse-grained functions reduce complexity on IPHA and optimized IP network traffic
- ZigBee Gateway Device (ZGD) specification defines its own information base (GIB) and cluster to advertise the “Gateway service” to ZigBee nodes

8.2.4. Standards

ZigBee is based on an IEEE 802.15 standard.

The new IEEE standard, 802.15.4 [312], defines the physical layer (PHY) and medium access control sub layer (MAC) specifications for low data rate wireless connectivity among relatively simple devices that consume minimal power and typically operate in the Personal Operating Space (POS) of 10 meters or less. An 802.15.4 net- work can simply be a one-hop star, or, when lines of communication exceed 10 meters, a self-configuring, multi-hop network. A device in an 802.15.4 network can use either a 64-bit IEEE address or a 16-bit short address assigned during the association procedure, and a single 802.15.4 network can accommodate up to 64k (216) devices. Wireless links under 802.15.4 can operate in three license free industrial scientific medical (ISM) frequency bands. These accommodate over air data rates of 250 kb/sec in the 2.4 GHz band, 40 kb/sec in the 915 MHz band, and 20 kb/sec in the 868 MHz Total 27 channels are allocated in 802.15.4, with 16 channels in the 2.4 GHz band, 10 channels in the 915 MHz band, and 1 channel in the 868 MHz band. Wireless communications are inherently susceptible to interception and interference. Some security research has been done for WLANs and wireless sensor net-works, but pursuing security in wireless networks remains a challenging task. 802.15.4 employs a fully handshake protocol for data transfer re-liability and embeds the Advanced Encryption Standard (AES) for secure data transfer. In the following subsections, we give a brief overview of the PHY layer, MAC sub layer and some general functions of 802.15.4. Detailed information can be found in.

<u>The PHY layer</u>

The PHY layer provides an interface between the MAC sub layer and the physical radio channel. It provides two services, accessed through two service access points (SAPs). These are the PHY data service and the PHY management service. The PHY layer is responsible for the following tasks:

- Activation and deactivation of the radio transceiver: Turn the radio transceiver into one of the three states, that is, transmitting, receiving, or off (sleeping) according to the request from MAC sub layer. The turnaround time from transmitting to receiving, or vice versa, should be no more than 12 symbol periods.
- Energy detection (ED) within the current channel: It is an estimate of the received signal power within the bandwidth of an IEEE 802.15.4 channel. No attempt is made to identify or decode signals on the channel in this procedure. The energy detection time shall be equal to 8 symbol periods. The result from energy detection can be used by a network layer as part of a channel selection algorithm, or for the purpose of clear channel assessment (CCA) (alone or combined with carrier sense).
- Link quality indication (LQI) for received packets: Link quality indication measurement is performed for each received packet. The PHY layer uses receiver energy detection (ED), a signal-to-noise ratio, or a combination of these to measure the strength and/or quality of a link from which a packet is received. However, the use of LQI result by the network or application layers is not specified in the standard.

- Clear channel assessment (CCA) for carrier sense multiple access with collision avoidance (CSMA- CA): The PHY layer is required to perform CCA using energy detection, carrier sense, or a combination of these two. In energy detection mode, the medium is considered busy if any energy above a predefined energy threshold is detected. In carrier sense mode, the medium is considered busy if a signal with the modulation and spreading characteristics of IEEE 802.15.4 is detected. And in the combined mode, both conditions aforementioned need to be met in order to conclude that the medium is busy.
- Channel frequency selection: Wireless links under 802.15.4 can operate in 27 different channels (but a specific network can choose to support part of the channels). Hence the PHY layer should be able to tune its transceiver into a certain channel upon receiving the request from MAC sub layer.
- Data transmission and reception: This is the essential task of the PHY layer. Modulation and spreading techniques are used in this part. The 2.4 GHz PHY employs a 16-ary quasi-orthogonal modulation technique, in which each four information bits are mapped into a 32-chip pseudo-random noise (PN) sequence. The PN sequences for successive data symbols are then concatenated and modulated onto the carrier using offset quadrature phase shift keying (O-QPSK). The 868/915 MHz PHY employs direct sequence spread spectrum (DSSS) with binary phase shift keying (BPSK) used for chip modulation and differential encoding used for data symbol encoding. Each data symbol is mapped into a 15-chip PN sequence and the concatenated PN sequences are then modulated onto the carrier using BPSK with raised cosine pulse shaping.

The MAC sub layer

The MAC sub layer provides an interface between the service specific convergence sub layer (SSCS) and the PHY layer. Like the PHY layer, the MAC sub layer also provides two services, namely, the MAC data service and the MAC management service. The MAC sub layer is responsible for the following tasks:

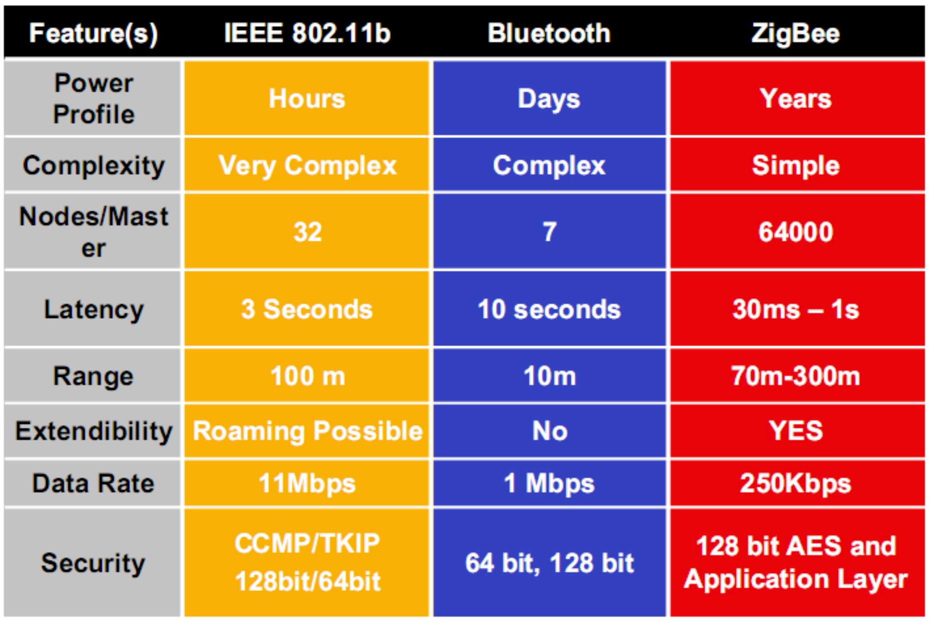
- Generating network beacons if the device is coordinator: A coordinator can determine whether to work in a beacon enabled mode, in which a super frame structure is used. The super frame is bounded by network beacons and divided into aNumSuperframeSlots (default value 16) equally sized slots. A coordinator sends out beacons periodically to synchronize the attached devices and for other purposes (see subsection II-C).
- Synchronizing to the beacons: A device attached to a coordinator operating in a beacon enabled mode can track the beacons to synchronize with the coordinator. This synchronization is important for data polling, energy saving, and detection of orphaning.
- Supporting personal area network (PAN) as-association and disassociation: To support self-configuration, 802.15.4 embeds association and dis-association functions in its MAC sub layer. This not only enables a star to be setup automatically, but also allows for the creation of a self-configuring, peer-to-peer network.
- Employing the carrier sense multiple access with collision avoidance (CSMA-CA) mechanism for channel access: Like most other protocols designed for wireless networks, 802.15.4 uses CSMA-CA mechanism for channel access. However, the new standard does not include the request-to-send (RTS) and clear-to-send (CTS) mechanism, in consideration of the low data rate used in LR-WPANs.

- Handling and maintaining the guaranteed time slot (GTS) mechanism: When working in a beacon enabled mode, a coordinator can allocate portions of the active super frame to a device. These portions are called GTSs, and comprise the contention free period (CFP) of the super frame.
- Providing a reliable link between two peer MAC entities: The MAC sub layer employs various mechanisms to enhance the reliability of the link between two peers, among them are the frame acknowledgement and retransmission, data verification by using a 16-bit CRC, as well as CSMA-CA.

General Functions

The standard gives detailed specifications of the following items: type of device, frame structure, super frame structure, data transfer model, robustness, power consumption considerations, and security. In this subsection, we give a short description of those items closely related to our performance study, including type of device, super frame structure, data transfer model, and power consumption considerations.

8.2.5. Why is it necessary to BaaS?



Feature(s)	IEEE 802.11b	Bluetooth	ZigBee
Power Profile	Hours	Days	Years
Complexity	Very Complex	Complex	Simple
Nodes/Master	32	7	64000
Latency	3 Seconds	10 seconds	30ms – 1s
Range	100 m	10m	70m-300m
Extendibility	Roaming Possible	No	YES
Data Rate	11Mbps	1 Mbps	250Kbps
Security	CCMP/TKIP 128bit/64bit	64 bit, 128 bit	128 bit AES and Application Layer

Figure 46: Comparing Communication Features

Ultra Low Power Consumption (Smart Energy BaaS)

An important aim of ZigBee and the IEEE 802.15.4 standard is the provision for building autonomous, low-powered devices. These devices are powered from an internal source, such as a battery pack or solar cells, and therefore need no external power supply or power cabling. There are many wireless applications that require this type of device, from light-switches, active tags and security detectors to remote industrial control and monitoring.

From a user perspective, autonomously powered devices have the following advantages:

- **Easy and low-cost installation of devices**: No need to connect to a separate power supply
- **Flexible location of devices**: Can be installed in difficult places where there is no power supply, and can even be used as mobile devices

- **Easily modified network**: Devices can easily be added or removed, on a temporary or permanent basis

Autonomous network devices are generally battery-powered or solar powered:

- **Battery-powered devices**: Since these devices are generally small, they use low-capacity batteries. Infrequent device. maintenance is often another requirement, meaning long periods between battery replacement and the need for long-life batteries
- **Solar-powered devices**: Solar power is an example of “energy harvesting” in which the device absorbs and stores energy from its surroundings. In this case, the energy is collected in the form of light and the device must be located in a well-lit environment.

The power consumption of these devices must be carefully managed to make optimum use of very limited power resources over an extended period of time. These devices therefore present significant technical challenges to keep power consumption low. The methods employed to minimize power consumption include the following:

- **Low duty cycle**: Most of the power consumption of a wireless network device corresponds to the times when the device is active - that is, transmitting or receiving. The active time as a proportion of the time interval between activities is called the duty cycle. Power use is optimized by using extremely low duty cycles, so that the device is active for a very small fraction of the time. This is helped by making the transmission/reception time short and the time interval between transmission/reception long.
- **Sleep mode**: When not transmitting or receiving, the device should revert to a sleep mode during which the power consumption is minimal.
- **Modulation**: The modulation schemes used to transmit data (BPSK for 868/915 MHz, O-QPSK for 2400 MHz) minimize power consumption by using a peak-to-average power ratio of one.

Easy Installation

- **Minimal Cabling:** A ZigBee network can be installed with a minimum of cables:
- **No network cables**: as a wireless network, there are any cables between network nodes for communication and control purposes.
- **Fewer power cables**: Where autonomous (battery-powered or solar-powered) devices are used, there is no need for power cabling to connect the device to an external power source.

Therefore, ZigBee avoids much of the wiring and construction work required when installing cable-based networks.

- Configuration Options

A ZigBee network can be installed with any one of three configuration options to suit the end-user, location or application. The three options are described below.

Option	Installer	Description
Pre-configured	Manufacturer	A system in which all parameters are configured by the manufacturer. The system is used as delivered and cannot be readily modified or extended. Examples: vending machine, patient monitoring unit.

Option	Installer	Description
Self-configuring	End-user	A system that is installed and configured by the end-user. The network is initially configured by sending “discovery” messages between devices. Some initial user intervention is required to set up the devices - for example, by pressing one or more buttons. Once installed, the system can be easily modified or extended without any re-configuration by the user - the system detects when a device has been added, removed or simply moved, and automatically adjusts the system settings. Examples: A Do-It-Yourself home security or home lighting system.
Custom	System-Integrator	A system that is developed for a specific application/location. It is designed and installed by a system integrator using off-the-shelf network devices. The system is usually configured using a software tool.

Table 6: Zigbee Network Configuration Option

8.3. Bluetooth

Bluetooth technology is a global wireless standard that enables short-range wireless data communication. It has been proposed by major companies in the communication industry in order to create an alternative data transmission standard that is not based on cable connections. Bluetooth technology is widely used because usage cost is not expensive and it is compatible with almost all devices. Bluetooth technology is very suitable to IoT applications with its advantages like power-efficient and low-cost.

2.4 GHz industrial, scientific and medical (ISM) radio band is the operating frequency band of the Bluetooth technology. In most countries, 80 MHz band is allocated to this band. Some radio front-end features are [313] [314]:

- Frequency hopping is utilized with the rate of 1600 hops per second in order to combat interferences in the band.
- By using Gaussian Frequency Shift Keying (GFSK) modulation, 1 Mbps link speed is obtained. Usage of GFSK brings low complexity. Phase Shift Keying (PSK) is another modulation technique used in Bluetooth technology.
- Time Division Duplex (TDD) scheme is included for full-duplex transmission.
- Minimum recommended receiver sensitivity is determined as -70 dBm. For proper interference performance, desired signal power is 3 dB over the reference sensitivity level.
- Center frequency must be within the range of +75 kHz or -75 kHz from the carrier frequency.

8.4. RFID Reader and Tags

8.4.1. Radio Frequency Identification

Nowadays, mobility and thus wireless communication is a primary concern in consumer computing. Radio frequency identification (RFID) is one of the most widely used wireless technology for automatic identification and data capture applications. Due to its lightweight, low power consumption, cost-effectiveness and non-line-of-sight readability RFID offers practical technology for context-aware computing like indoor localization systems [315].

An RFID module uses electromagnetic waves and employs a chip and an antenna for two-way transfer of the data. RFID systems can be classified as tag and reader, where the reader reads data generated by the tag as the names suggest. Nowadays, various types of applications such as asset tracking, supply chain management and payment systems (electronic tickets) includes RFID systems [316].

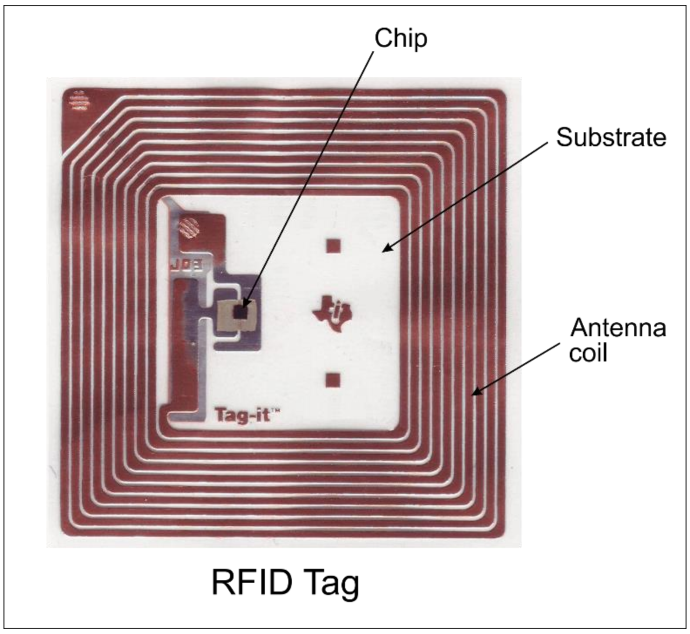


Figure 47: RFID tag architecture [317]

The most distinctive property of RFID from earlier barcode technology is that it does not require a line of sight. RFID provides identification from a distance varying according to the type of the module employed. Based on their power consumption, RFID tags can be categorized as active tags, passive tags and semi-passive (battery-assisted) tags. Active tags employ internal power sources and can continuously communicate with the reader for long ranges, whereas passive tags collect power from interrogating waves and backscatter the radio signal to the nearby reader [316]. The main advantages of passive RFID systems over active RFID systems are the lower costs, smaller sizes, and longer life spans, however they are not always practical since the power constraint results in lack of functionality. Another categorization can be made by frequency range since RFID systems operate at a variety of radio frequencies from 120Hz to 10Ghz. Depending on the working frequency RFID systems are named as low frequency, high frequency, and ultra-high frequency. Generally, a system operating at a higher frequency facilitates faster data transfer rates and longer read ranges than lower frequency systems [318].

8.4.2. RFID Tags

For object identification, tags are used in radio-frequency identification systems. Depending on the battery property, tags can be categorized into three types which are passive, active and battery-assisted passive. A passive tag has no battery and exploits the radio energy that is received from a reader. A battery-assisted tag has a small battery and becomes active in discrete instants. An active tag has a battery and thus more flexibility in terms of energy usage.

Two parts are the specific main components of an RFID tag. These parts are an integrated circuit and an antenna. Processes like information storage and processing, RF signal modulation and demodulation are controlled within the integrated circuit. Transmission and reception of signals are realized via antennas.

8.4.3. RFID Readers

An RFID reader is a device that is used in RFID systems in order to connect tags and system software [319]. Readers connect with tags, which are located in the area of readers and start to perform some operations such as classifying tags in terms of meeting a criteria and encoding of tags.

Readers also contain antennas for signal transmission with tags. Received data are sent to a computer for processing. Readers can be used in a stationary position or can be used with a microprocessor or mobile unit.

8.5. Machine to Machine (M2M)

8.5.1. Introduction

The term is used to refer to machine to machine (M2M) communication, i.e., automated data exchange between machines. Viewed from the perspective of its functions and potential uses, M2M [320] is causing an entire “Internet of things”, or internet of intelligent objects, to emerge.

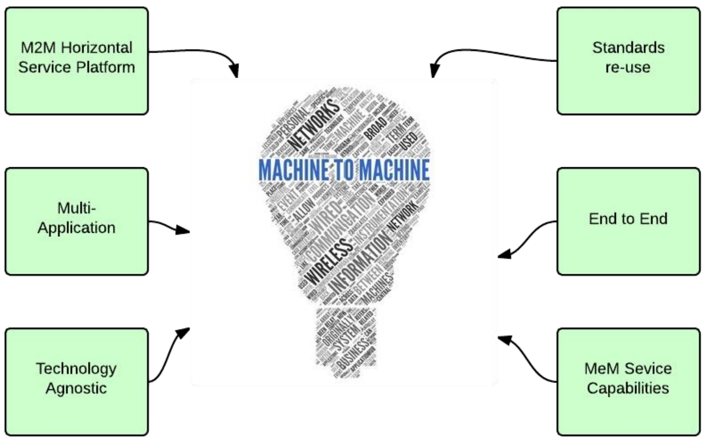


Figure 48: M2M

In our everyday lives we are using more and more automated and intelligent machines. Such machines operating remotely can provide important information. They also need to be managed and monitored. Increasing numbers of machines are able to do this automatically and without the control of their owners. Machines are designed to initiate communication in order to provide vital information, save time and money.

This fast-growing sector has the potential to connect up to 50 billion machines today, and even more in the near future.

IoT (Internet of the Things)

The continuous progress in microelectronics and networking techniques make it now possible to envision networks formed by the interconnection of smart 'network enabled' objects and secure and efficient deployment of services on top of them. This is the vision of the Internet of the things.

We now see the deployment of a new generation of networked objects with communication, sensory and action capabilities for numerous applications. The interconnection of objects having advanced processing and connection capabilities is expected to lead to a revolution in terms of service creation and availability and will profoundly change the way we interact with the environment. The physical world will merge with the digital/virtual world [321].

8.5.2. Benefits

What is the main benefit of such automatic communication? The owner of the machines don't need to visit them personally to verify their operations or interpret a machines status manually. Such machines can automatically send the information to the owner or to another machine or software application that processes the data further. Such Machine to Machine (M2M) communication is cheaper, faster and brings new possibilities.

8.5.3. Application

Although some industry players classify M2M applications [322] according to category or type, it may be more useful for an operator to look at them from a different perspective: applications that are not at all related to person-to-person communications versus applications that relate in some manner to mobile applications used by people. Examples of the latter include:

- Security – vehicle security and anti-theft
- Transport and logistics – navigation information
- Metering – consumption of electricity, gas and water
- Smart living/entertainment – remote controls, synchronization and smart appliance

8.5.4. Issues

IPv6 Support for Network Address Availability

The Number Resource Organization, the entity tasked with protecting the allocated pool of remaining IPv4 internet addresses, issued a statement indicating than less that 10% of IPv4 addresses remain un-allocated. Clearly, if millions to hundreds of millions of new devices are going to be networked in an "Internet of Things" in the coming years, this shortage of IPv4 addresses poses a new challenge.

Security

With the 30 billion to 50 billion devices predicted for 2020, a large part of the problem will be the management of each individual end point, and the complexity that comes with that.

If it has an IP address, regardless of whether it's fixed or mobile or a device, it needs a security protocol, and that security policy should be in line with the fully blown policy that the enterprise has.

Clearly, security is a critical requirement due to the remote controlled and usually unsupervised nature of M2M systems. As the wireless sensors are the primary drivers of M2M communication, and they are being deployed in many applications like vehicle-to-vehicle communication systems, health monitoring and alarm systems, smart metering, industrial automation and safety monitoring. M2M terminals are typically required to be small, inexpensive, able to operate unattended by humans for extended periods of time, and to communicate over the wireless networks. These practicality requirements are making it harder to deploy heavy security precautions on these systems, while security is a critical aspect for them.

8.5.5. Standards

Seven Standards Development Organizations (SDOs) are set to create a cooperative M2M standards activity [321].

In doing this, they intend to pull together multiple industries to create specifications that will improve the return on investment on standards work and create a worldwide potential for M2M applications.

The global initiative has currently gathered representatives from: the Association of Radio Industries and Businesses (ARIB) and the Telecommunication Technology Committee (TTC) of Japan; the Alliance for Telecommunications Industry Solutions (ATIS) and the Telecommunications Industry Association (TIA) of the USA; the China Communications Standards Association (CCSA); the European Telecommunications Standards Institute (ETSI); and the Telecommunications Technology Association (TTA) of Korea.

ETSI is the bigger promoter of this activity, so we are going to delve a little more into it and the standards they promote.

ETSI

M2M is part of ETSI, one of the world’s leading standards development organizations for Information and Communication Technologies (ICTT).

ETSI has created a dedicated Technical Committee with the mission to develop standards for M2M Communications. The group will provide an end-to-end view of M2M standardization.

ETSI's broad representation from the global telecoms and ICT industry enables us to take a ‘big picture' view. This committee includes European, American and Asian experts from telecoms network operators, equipment vendors, administrations, research bodies, and of course M2M specialist companies.

Standards:

TR 102 996	Interworking between the M2M Architecture and M2M Area Network Technologies

TS 102 921	mIa, dIa and mId interfaces
TR 101 584	Study on Semantic support for M2M Data
TS 102 690	Functional Architecture
TR 102 732	Use Cases for M2M applications for eHealth
TR 102 732	Use Cases for M2M applications for Connected Consumer
TS 102 689	M2M service requirement
TR 102 725	Definitions
TS 103 104	Interoperability Test Specification for CoAP Binding of ETSI M2M Primitive
TR 102 898	Use cases of Automotive Applications in M2M capable networks
TS 103 093	BBF TR-069 compatible Management Objects for ETSI M2M
TR 102 935	Applicability of M2M architecture to Smart Grid Networks; Impact of Smart Grids on M2M platform
TS 103 167	Threat analysis and counter measures to M2M service layer.
TR 102 691	Smart Metering Use Cases

8.6. Web GIS Services

8.6.1. Introduction

One of the goals of BaaS is to have multi-layered perspectives to visualize physical assets of a set of buildings, BaaS indicators such as heat-maps of occupancy, inactivity period, lease expirations, total costs per square foot, costs of deferred maintenance, efficiency, and other key performance indicators, and anomalies, incidents and defects detected on the building infrastructure geolocated (in real time) through a web application for final users and industries for a better decision making. Therefore the management of geospatial information together with CEP services plays an important role in this project.

8.6.2. Spatial Data Infrastructure

In the last decade the use of geospatial information has experienced a large increase due to advances in software development in Geographic Information Systems (GIS), progress in data standardization and the implementation of Spatial Data Infrastructures (SDI) through Geoportals (web mapping applications).

A Spatial Data Infrastructures (SDI) is a basic set of technologies, polices and institutional arrangements to facilitate the availability and access to geographic information (maps, ortophotos, satellite images, etc.) via the Internet. From the technological point of view, a SDI can be defined as a decentralized network of servers that provide spatial data, metadata, search and visualization methods and standard interfaces to access to the geographic data.

The goals of a SDI are:

- To facilitate the access and the integration of the spatial information allowing the knowledge diffusion and the optimization of the decision making.
- To promote the generation of metadata for the spatial information by using standards; avoiding the duplicity of efforts and cutting off costs.

- To encourage the cooperation between agents, promoting the exchange of information

To get these goals it is essential that a SDI is composed of the following components:

- An institutional framework that encourages its creation and maintenance (INSPIRE)
- A data police that promotes the generation and the access to reference data
- Technology needed for system operation (Web GIS Technologies)
- Standards to improve the interoperability of spatial data allowing the exchange of information between different agents (OGC)

8.6.3. Institutional framework: INSPIRE Directive

Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE) was published in the official Journal on the 25th April 2007 [323].

The INSPIRE Directive is based on the infrastructures for spatial information established and operated by the 27 Member States of the European Union. The Directive addresses 34 spatial data themes needed for environmental applications, with key components specified through technical implementing rules. This makes INSPIRE a unique example of a legislative "regional" approach.

The INSPIRE directive came into force on 15 May 2007 and will be implemented in various stages, with full implementation required by 2019.

The INSPIRE directive aims to create a European Union (EU) spatial data infrastructure. This will enable the sharing of environmental spatial information among public sector organizations and better facilitate public access to spatial information across Europe.

A European Spatial Data Infrastructure will assist in policy-making across boundaries. Therefore the spatial information considered under the directive is extensive and includes a great variety of topical and technical themes.

INSPIRE is based on a number of common principles:

- Data should be collected only once and kept where it can be maintained most effectively.
- It should be possible to combine seamless spatial information from different sources across Europe and share it with many users and applications.
- It should be possible for information collected at one level/scale to be shared with all levels/scales; detailed for thorough investigations, general for strategic purposes.
- Geographic information needed for good governance at all levels should be readily and transparently available.
- Easy to find what geographic information is available, how it can be used to meet a particular need, and under which conditions it can be acquired and used.

8.6.4. Web GIS Technologies

Nowadays, there is a wide range of private and open source technologies for the implementation of Spatial Data Infrastructures (SDI) through Geoportals (web mapping applications).

Recently, the technology and architecture that could service and deployment models offer are a key area of research and development for GIS technology. The GIS technologies in the cloud are presented as SaaS (Software as a Service) that provide cloud-based clients and applications that solve complex business problems using GIS tools.

The software and the data are hosted on the servers of the company that supplies these services that are usually accessed through a web interface.

Some examples of GIS products (privative and open source software) are the following ones:

- **GIS Cloud** [324]: It is a service that allows the store, edit, publish, share geographic information in the cloud through a web interface. It allows different formats of spatial data importation, the connection with the data base PostGIS, with WMS severs, with tiles servers and the publication of maps from ArcMap Software. It offers two APIs, one Rest and other JavaScript, for the creation of client applications. But due to these applications do not support standards, the integration of these services and data in other applications is complicate.
- **OpenGeo Suite** [325]: There is a version of the OpenGeo Suite for the cloud. The suite is composed of the main GIS open source products such as: PostGIS (database), Geoserver and Geowebcache (maps and tiles servers), OpenLayers (map web client) and y Tomcat (application's server). Therefore, it offers applications to edit, print, style edit and share information allocated in PostGIS and published by Geoserver.
- **CartoDB** [326]: CartoDB is a product of the Spanish company Vizzuality that allows importing, designing and publishing maps. Besides it offers spatial analyze, advanced style through CartoCSS and APIs for the development of applications based on Leaflet and Google Maps. This products is mainly focused to the store of vector date in the database PostGIS.
- **ArcGIS Server** [327]: ESRI offers GIS services in the cloud of its product ArcGIS through Amazon EC2 and its own infrastructure.
- **IkiMap** [328]: ikiMap is a service created by the Spanish company Sixtema that allows the generation, publishing and especially the sharing of maps in the cloud. The main feature of the service is that it is intended to non-professional users through a friendly web interface that can be used by non-expert in GIS issues. IkiMap infrastructure uses some open source GIS products such as PostGIS, OpenLayers and gvSIG Mini.
- **MapBox** [329]: It offers publish and maps design service in the cloud, mainly oriented to the tiled services. It offers tools for the advanced design of maps such as TileMill, for layers composition, and options for traffic analyze in the tile server, etc.

The following frameworks are used for the development of Web GIS applications:

- **OpenLayers** [330]: It is an open-source library for the creation of Web GIS applications. Currently is the most robust and mature framework for the generation of web map applications based on the standards implementation with the support of the most formats, services and GIS protocols. It has an optimized and reduced version for mobile devices and the version 3 of the framework is being developed. This new version will offer an easier API and WebGL support.
- **LeafletJS** [331]: It is an open-source JavaScript library based on HTML5. Its main feature is that is very light and its optimization for mobile devices. Its API has less features and it is simpler than OpenLayers, but is very useful for applications requiring only the visualization of maps.
- **Google Maps** [332]: Google offers a JavaScript API for the development of web applications that access to its Google Maps services (tiling, routing, geocodification).

- **PolyMaps** [333]: It is a JavaScript library to render vector and raster maps by using the standard SVG.
- **ModestMaps** [334]: It is the open-source JavaScript library of MapBox. As LeafletJS and the PolyMaps frameworks, it is oriented to the visualization and interaction with easy maps. Its main feature is its lightness (10 KB compressed) and its optimization for mobile devices.
- **MapQuery** [335]: MapQuery is the open-source JavaScript framework that joins jQuery with OpenLayers. It is oriented to the development of interface user components. Its main drawback is that there are no many active developers.
- **jQuery Geo** [336]: It is an open-source JavaScript framework similar to MapQuery.
- **OpenGeo Suite SDK** [337] : It is the open-source JavaScript framework that joins ExtJS and GeoExt with OpenLayers. It allows the quick configuration ot web GIS applications through a set of plugins and widgets.
- **GeoExt** [338]: It is an open-source JavaScript framework that joins ExtJS with OpenLayers, allowing the generation of interface user components.
- **GeoExt Mobile** [338]: It is an attempt of port de GeoExt for using Sencha, the ExtJS version for mobile devices.

Regarding to the development of the Web GIS applications there are two types of components. Firstly, the JavaScript frameworks that allow to locate a map on the web, add layers, apply styles, interact with the maps (navigation, object selection, draw geometries, etc.), etc. and on the other hand, the developments frameworks, which allow to configure Web Mapping applications, add components of user interface, buttons to active controls, display alphanumeric information associated to the map, modify the characteristics of the display (connect to new servers, add new layers, advanced editing , queries, ...).

Among the JavaScript libraries, the most optimal and robust library to develop Web GIS application, taking into account only the API) is OpenLayers. The problem of this library is that most of its features do not work on mobile devices, mainly because it was not designed for this type of devices and therefore the version for mobile devices is simply a reducer version.

Apart from OpenLayers, over the last years it has been appeared other open-source libraries, based on HTML5, optimized for mobile devices, both in library size and optimized for mobile, both in library size, and quantity of KB transferred, memory consumption, etc. However these libraries have a lack of advanced features such as: editing, theme support multiple formats, protocols, etc.

Regarding to the frameworks, all the aforementioned are based on OpenLayers. This fact is due OpenLayers is the most used and the unique that has been adapted for mobile devices is GeoExt Mobile, although it offers only some services (navigation, selection, etc.)

8.6.5. WEB GIS Standards (OGC)

The Open Geospatial Consortium OGC is an organization composed of companies, universities and public administrations that has developed over recent years a number of protocols and standards that provide the technology framework for interoperability in GIS and Spatial Data Infrastructure [339].

The OGC Standards are developed within the OGC Standards Program. Here the OGC Technical Committee and OGC Planning Committee work in a formal consensus process to arrive at approved (or "adopted") OGC® standards.

The most common approved standards used in Web mapping applications for publishing geographic data, using the criteria of OGC, are:

- **WMS y WMTS (Web Map Service and Web Map Tile Service):**

The Web Map Service Interface Standard (WMS) provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases. A WMS request defines the geographic layer(s) and area of interest to be processed. The response to the request is one or more geo-registered map images (returned as JPEG, PNG, etc.) that can be displayed in a browser application. The interface also supports the ability to specify whether the returned images should be transparent so that layers from multiple servers can be combined or not.

A WMTS enabled server application can serve map tiles of spatially referenced data using tile images with predefined content, extent, and resolution.

- **WFS (Web Feature Service)**

The Web Feature Service (WFS) represents a change in the way geographic information is created, modified and exchanged on the Internet. Rather than sharing geographic information at the file level using File Transfer Protocol (FTP), for example, the WFS offers direct fine-grained access to geographic information at the feature and feature property level. Web feature services allow clients to only retrieve or modify the data they are seeking, rather than retrieving a file that contains the data they are seeking and possibly much more. That data can then be used for a wide variety of purposes, including purposes other than their producers' intended ones.

In the taxonomy of services defined in ISO 19119, the WFS is primarily a feature access service but also includes elements of a feature type service, a coordinate conversion/transformation service and geographic format conversion service.

- **WCS and WCTS (Web Coverage Service and Web Coverage Tile Service)**

The OGC Web Coverage Service (WCS) supports electronic retrieval of geospatial data as "coverages" – that is, digital geospatial information representing space/time-varying phenomena. A WCS provides access to coverage data in forms that are useful for client-side rendering, as input into scientific models, and for other clients. The WCS may be compared to the OGC Web Feature Service (WFS) and the Web Map Service (WMS). As WMS and WFS service instances, a WCS allows clients to choose portions of a server's information holdings based on spatial constraints and other query criteria.

A WCTS enabled server application can serve coverage tiles of spatially referenced data using tile images with predefined content, extent, and resolution making faster the access and the updating of geographic information.

- **Catalogue Web Services (CSW)**

The OGC Catalogue Web Services allows searching for and retrieve any type of information object (data, service instance, service type, schema, style description, etc.) based on its properties described in “metadata.”

- **WPS (Web Processing Service)**

The Web Processing Service (WPS) Interface Standard provides rules for standardizing how inputs and outputs (requests and responses) for geospatial processing services, such as polygon overlay. The standard also defines how a client can request the execution of a process, and how the output from the process is handled. It defines an interface that facilitates the publishing of geospatial processes and clients' discovery of and binding to those processes. The data required by the WPS can be delivered across a network or they can be available at the server.

- **NetCDF**

The OGC netCDF encoding supports electronic encoding of geospatial data, specifically digital geospatial information representing space and time-varying phenomena. NetCDF is a data model for array-oriented scientific data. A freely distributed collection of access libraries implementing support for that data model, and a machine-independent format are available. Together, the interfaces, libraries, and format support the creation, access, and sharing of multi-dimensional scientific data.

- **WFS Gazetteer**

WFS Gazetteer is a Gazetteer Service profile of the OGC Web Feature Service Standard. The OGC Gazetteer Service allows a client to search and retrieve elements of a georeferenced vocabulary of well-known place-names.

- **SWE, Sensor Web Enablement**

The SWE standards standardize the use of sensors in Web. SWE allows the discovery, exchange and processing of sensor observations. So its aim is the same as the standards HTTP and HTML that enable the exchange of different types of information

8.6.6. Conclusions

The features related to Web GIS Technologies fit with BaaS requirements and JavaScript frameworks as OpenLayers. If the solution has to capture events in real-time, there will be a need to use new non GIS Technologies, like Node.JS or socket.io, to complements standard GIS solution (GeoExt, etc.).

8.7. Embedded Systems

8.7.1. Overview of embedded systems architectures

The term Embedded system refers to an electronic device programmed to control a specific appliance. Due to single-purpose design, it is possible to optimize it for specific applications and thus reduce the final product price. This category also covers devices that are expandable in terms of software and hardware and can be widely used as personal computers which is, of course, used with the advantage.

As an example of an embedded system, Intel platform iMote2 based on Intel processor PXA271 Xscale is shown in Figure 49. It is obvious that the system consists of identical (similar) components as a personal computer - processor, RAM and flash, IO interfaces, power, real-time clock, signal processor and radio module of standard 802.15.4 (ZigBee). These days most of these components are advantageously combined in one integrated circuit - a system on a chip and the PCB (printed circuit board) is only a kind of base for mounting other components and for accessing chip interfaces.

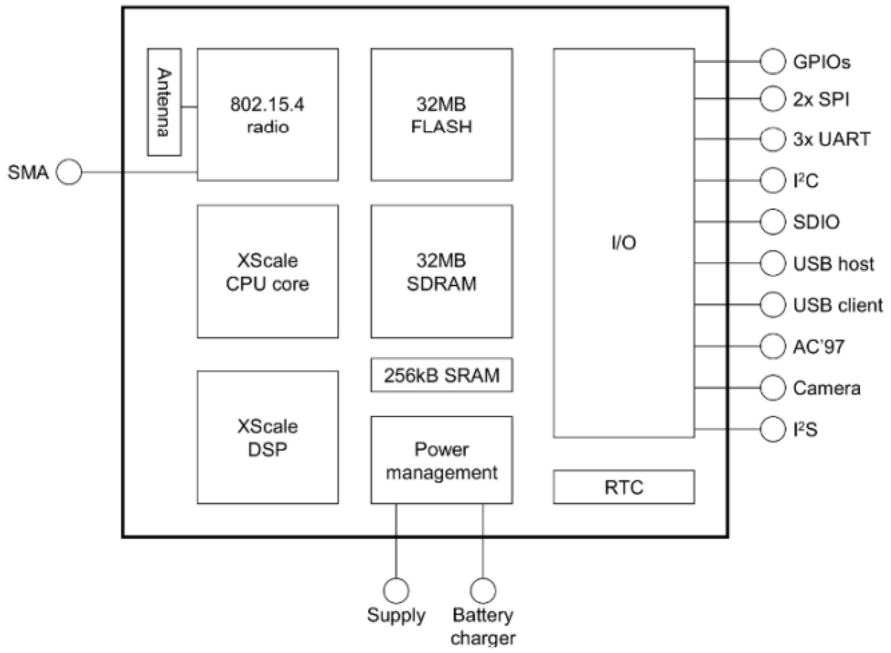


Figure 49: Block diagram of embedded computer Imote2

Integrating of all computer components into one package of integrated circuit create System-on-Chip (SoC). These components are microprocessor, combination of memories (RAM, ROM and EEPROM or flash), sources of timing signals, interfaces (especially USB, Ethernet, SPI, I2C, GPIO, and others), voltage regulators and power management circuits. These individual blocks are interconnected usually by an internal bus AMBA (Advanced Microcontroller Bus Architecture - developed by company ARM). Block diagram of this circuit is shown in Figure 50.

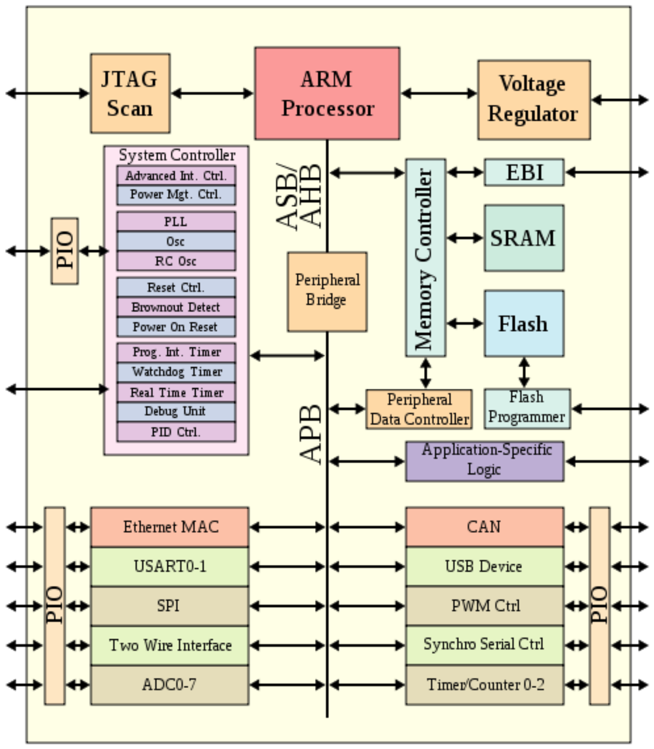


Figure 50: Block diagram of SoC

8.7.2. Operating systems

An operating system is an essential computer software that is loaded into the computer memory during system start and remains in operation until its shutdown. It provides core and auxiliary system services and tools. Its main task is to enable users to control the computer (run programs, transfer their inputs and outputs ...), and create stable application interface abstracting hardware control using easy-to-use features (Application Program Interface - API) thus enabling resources management - to allocate system resources to processes. Regarding the resource allocation to processes, an operating system can be either single tasking or multitasking. Although, a processor core can run only a single execution at a time, the illusion of simultaneity is created by the scheduler of the multitasking operating system, accomplished by rapidly switching between processes. For use in embedded systems, it is possible to consider GNU Linux, which supports many processor architectures (e.g. PowerPC, ARM, x86, SPARC, MIPS). It is also appropriate to mention TinyOS system, which is used in the sensor networks. However, for the mission critical applications in which reliability is an important concern, a real time operating system (e.g. Zephyr, VxWorks, FreeRTOS), which has a scheduler providing minimal interrupt latency and minimal thread switching latency, should be considered.

TinyOS – open, component-based operating system designed specifically for sensor networks. This is an event-driven system when the corresponding code (user application that was compiled together with the operating system) is called at an event occurrence. TinyOS is the de facto standard platform for academic research in the field of sensor networks.

PikeOS – an operating system for security critical applications developed specially for easy separation or verification of the individual parts of an electronic system using so-called Partitions. In these Partitions, applications or whole operating systems can run without affecting each other. Guest operating systems must be adapted to PikeOS interface. It runs on processor architectures like PowerPC, x86, ARM, MIPS, SPARC and SuperH

RTX – an extension of the Windows operating system using a collection of libraries providing access to the subsystem via the RTX API. Above all it adds an independent thread scheduler and shortens the distinguishable time unit from 5ms to 20 microseconds. Supported processor architecture is the same as for Windows NT/2000/XP.

RTLinux – a commercial Linux real-time modification which in addition to core adjustment also brings its own set of tools and modules that sufficiently extend the capabilities of the core. It tries to approach as much as possible to the standard POSIX 1003.1. It was designed to run on cheaper and less powerful computers based on x86. RTLinux can be considered as a complete operating system with predictable operations in real time, without the interface for standard none real-time Linux.

Riot – a free, partially POSIX compliant, open source operating system. It runs on several platforms and supports multiple architectures (e.g. MSP430, ARM7, Cortex-M0, x86).

Zephyr – a scalable real-time operating system with a small-footprint kernel. Zephyr is designed for use on resource-constrained systems (e.g. simple embedded environmental sensors, LED wearables, sophisticated smart watches, IoT wireless gateways), and supports multiple architectures (e.g. ARM Cortex-M, Intel x86, and ARC).

VxWorks – the most widespread commercial real-time operation system though it is not POSIX compatible. Supported platforms are x86, MIPS, PowerPC, and ARM.

8.7.3. Processors

According to internal architecture, processors can be divided into two groups – with a reduced instruction set (RISC) and with a large set of machine instructions (CISC).

8.7.3.1. Architecture with a complex instruction set – CISC

The reason for the appearance of the architecture with a complex instruction set was primarily the price and capacity of memories. There were the need to minimize the amount of the memory that the program needs for its activities and the amount of memory that the program needs for its storage. This led to the creation of processors that perform complex operations. However, since complex operations require more complex hardware (which is also more expensive), an interlayer was created – microprogram which is an interpreter, recoding complex instructions to simple instructions.

As an example, it is possible to specify the i386 processor series, which uses one part of the processor (microprogram) to interpret a complex instruction to simple ones, and subsequently the second part of the processor process them, using instruction parallelism, speculative evaluation, and other advanced methods. This way, one CISC instruction is performed as several RISC microinstructions. The main advantage of this approach is that the microprogram can be changed without intervention in hardware

The main disadvantage results from the different lengths of instructions: different length complicates decoding and planning, thus it is not possible (or very difficult) to implement advanced methods mentioned above.

8.7.3.2. Architecture with a reduced instruction set – RISC

Unlike architecture with a comprehensive instruction set, where designers try to eliminate 'semantic gap' between higher programming languages and the native language of processor comprising complex instructions, the architecture with a reduced instruction set focuses on simplifying its own instruction set. The reason for this is the fact that nowadays high-level programming language compilers rarely implement the same functionality in the same way, sometimes they even bypass complex instructions - so compilers use simpler instructions to implement the code.

Architecture with a reduced instruction set also uses fewer data types, trying to work effectively with simple data types by which it is possible to create more complex structures (which are not used so often). Thanks to it, the number of addressing modes is reduced, which leads to a reduction of flexibility, but there are no problems arising from different access times to operands located in different memory locations and especially the problems with decoding and execution planning of the instruction that vary in length.

Memory access is also reduced by using a large number of registers (IA-32 architecture has only 8 registers for general use, unlike the Itanium where it is possible to use 128 registers).

In the following text we consider only RISC architecture because it is unnecessary to use emulated architecture, which has higher demands on power consumption, or which shows various transient effects, increases power dissipation and other effects that can be ignored at lower frequencies.

The most popular 32-bit RISC processors used today are ARC, ARM, Atmel AVR, ColdFire, MIPS, PowerPC, SPARC, and SuperH. Of these listed, it makes sense to further consider only

ARM, MIPS and PowerPC because only these offer "PC-like" features, small packaging (hence the possibility of implementation in embedded devices) and low.

MIPS (Microprocessor without Interlocked Pipeline Stages) is not real processor, but fully synthesized core that manufacturers can integrate into their products. Probably the best known is the use in SGI computers (e.g. R10000 processor), famous for its performance when working with three-dimensional scenes and rendering. MIPS processors found huge application in widespread consoles Nintendo 64, PlayStation 2 and other devices such as set-top boxes, mobile phones, printers and so on.

Acronym MIPS means a processor without automatically organized pipeline. Pipeline idea is based on the fact that one instruction does not necessarily use all CPU resources. The processor can then process multiple instructions simultaneously, but only on condition that these instructions do not use the same processor resources.

PowerPC is microprocessor architecture created by the alliance Apple-IBM-Motorola in 1991. Originally designed for use in home computers, but they also became popular in embedded devices as well as among high-performance processors. The greatest architecture success was in Apple's personal computers in years 1994-2006 (after then Apple switched to Intel platform). Nowadays, the PowerPC core is used in the Cell processor, which is used in Sony game console Playstation 3.

ARM (Advanced RISC Machine) is a microprocessor architecture developed by ARM Limited, which is used in many embedded systems. With its energy saving properties, ARM processors are mainly used in mobile industry of consumer electronics, where low power consumption is very important. Today, the ARM family of processors include 75% of all 32-bit RISC processors in embedded devices. It makes ARM the most widely used architecture in the world. ARM processors can be found in all sectors of consumer electronics from PDAs, mobile phones, multimedia players, portable game consoles and calculators to computer peripherals.

ARM technology offers many advantages from the perspective of developers. Because it is only a processor core, which is already used by a variety of manufacturers, is very easy to migrate between entirely different types of circuits. Another significant advantage is the availability of efficient development environment (IDE) and variously equipped development kits. 32-bit microprocessors based on ARM core are often well equipped with a variety of peripherals that offer much more options than the basic versions of the common microprocessors.

8.7.4. Selection of suitable architecture

In the previous chapters, the analysis of individual criteria for choosing the appropriate platform was made.

From the point of view of the platform (processor architecture), use of the platform based on ARM processor appears to be the best option. Processors of the ARM Cortex A7 and Cortex A15 series include support of virtualization instructions and allow current running multiple operating systems. Of course, it depends on the specific implementation – selected processor can have sufficient performance, but the board on which it is mounted, may not be designed appropriately for the use of the performance. Likewise, it is not possible to compare individual platforms according to the energy demands (e.g. Atmel AT91SAM9G20 has according the manufacturer stated consumption 80mW at full load, but the specific implementation - board Calao Systems USB-A9G20-C01 has the stated input power of 2.5 W, because it is necessary to supply other circuits as well), it is also necessary compare particular implementations.

8.7.4.1. Compact, lightweight

The size and weight have a general rule - less is better. We can choose from many standards, but not all manufacturers follow these sizes. Standard board sizes are listed in Table 7, including the sizes of the most common boards of embedded systems. In addition to systems mentioned in this table there are still many interesting models that use different dimensions of the PCB.

Form factor	PCB size [mm]
WTX	356x425
AT	350x305
Baby-AT	330x216
BTX	325x266
ATX	305x244
Extended ATX	305x330
LPX	330x229
microBTX	264x267
NLX	254x228
microATX	244x244
DTX	244x203
FlexATX	229x191
Mini-DTX	203x170
EBX	203x146
Mini-ITX	170x170
EPIC	165x115
ESM	149x71
Nano-ITX	120x120
COM Express	125x95
ESMExpress	125x95
ETX/XTX	114x95
Pico-ITX	100x72
PC/104	96x90
ESMini	95x55

Form factor	PCB size [mm]
BeagleBoard	76x76
SODIMM/MXM	66x50
mobile-ITX	60x60
CoreExpress	58x65

Table 7: Sizes of printed circuit boards

8.7.4.2. Sufficient functionality and performance

These parameters are based both on the hardware equipment and on running operating system that must be capable of using all these features effectively. Performance can be expressed by using Dhrystone benchmark, which is a test program measuring efficiency of performance of typical set of integer calculations. Dhrystone does not perform calculations in floating point arithmetic, these operations can be measured by Whetston test. The results are reported in Dhrystones per second. Another way of measuring performance is CoreMark that in each iteration focuses on the operation with index (sorting, searching), matrix operations and others.

8.7.4.3. Connectivity, modularity

Connectivity means the ability to connect peripheral devices such as keyboard, monitor, camera ... using standard interfaces such as PS/2, USB, SDIO, and others. Modularity means the "stacking" feature, where it is possible to connect more expansion boards (e.g. accelerator, auxiliary batteries, hard disk), usually via a proprietary interface (e.g. Beagle Board1 offers 28-pin connector enabling connection via I2C, I2S, SPI and MMC/SD).

8.7.5. Overview of analyzed modules

8.7.5.1. Gumstix Verdex Pro XL6P

Gumstix Verdex Computers are on the market for some time. Model Verdex Pro XL6P is currently the most powerful of them. Technical parameters of this model are summarized in Table 8. For the model Verdex exist expansion modules, especially module with Ethernet 10/100 Mb/s. The board Verdex XL6P and Ethernet module Netpro-VX can be seen in Figure 51.

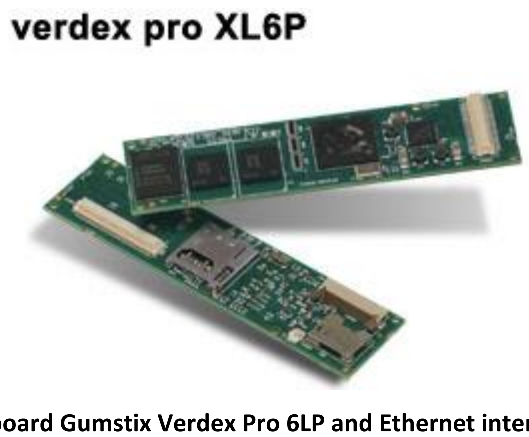


Figure 51: Mainboard Gumstix Verdex Pro 6LP and Ethernet interface board Netpro-vx

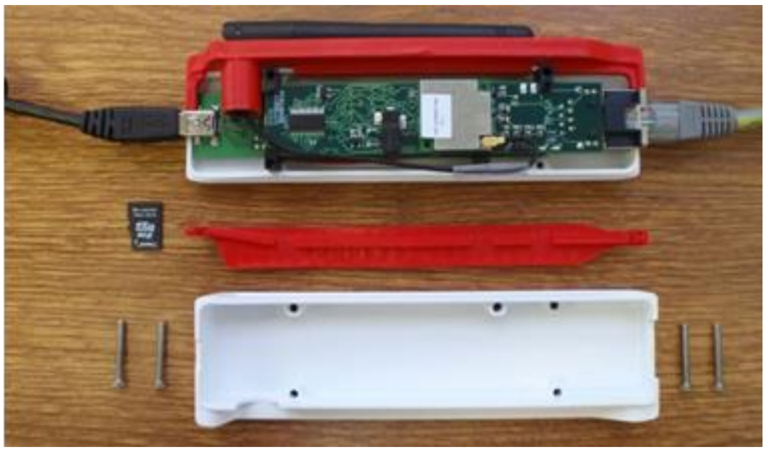


Figure 52: Uncovered firewall prototype microcomputer-based on Gumstix Verdex developed at Masaryk University

Processor	MarvellTM PXA270 with XScaleTM 600MHz
Memory RAM	128MB
Memory FLASH	32MB
Interfaces	60-pin Hirose, 80-pin Hirose, 24-pin flex ribbon
Size	80mm x 20mm
Price	169.00 USD

Table 8: Computer Gumstix Verdex Pro 6LP specification

8.7.5.2. Gumstix Overo Earth

Overo model series is the successor of Verdex series. It is smaller in size, has a powerful processor, but unfortunately there isn't a sufficiently large range of expansion modules. At the beginning of the project, this model seemed very promising, development of additional modules, however, did not continue according to our expectations. Technical parameters of this model are summarized in Table 9: Gumstix Overo Earth Computer Specifications and the module is in the Figure 53.

Figure 53: Computer Gumstix Overo Earth

Processor	OMAP 3503 with ARM Cortex-A8 CPU 600MHz
Memory RAM	256MB
Memory FLASH	256MB
Interfaces	70-pin AVX 5602 series, 27-pin flex ribbon
Size	58mm x 17mm x 4,2mm
Price	149.00 USD

Table 9: Gumstix Overo Earth Computer Specifications

8.7.5.3. Calao Systems USB-A9G20-C01

This model probably most closely resembles our original idea of size and shape. The single board integrates computer, Ethernet interface and power from the USB port, which was necessary to develop for computer Verdex. The main disadvantage of this computer is a relatively high weight (30g), which is concentrated in the opposite side of the board than the USB connector, which this computer connects to the host PC. This causes considerable mechanical stress, and breaking. Technical parameters of this model are summarized in the Table 10 and the module is in the Figure 54.

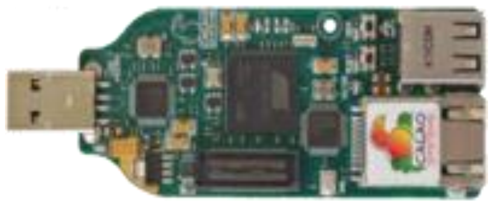


Figure 54: ***One-board computer CALAO USB-A9G20-C01***

Processor	ATMEL AT91SAM9G20 400MHz
Memory RAM	64MB
Memory FLASH	256MB
Interfaces	50-pin
Size	85mm x 36mm
Price	149.00 EUR

Table 10: Calao Systems USB-A9G20-C01 computer specification

8.7.5.4. DIL/NetPC DNP/9200

The computer is primarily intended for development and laboratory environments. Its great advantage is pulling all external interfaces onto socket DIL-64. This greatly facilitates the development of prototypes of other devices, because it is possible to use the Universal PCB. In the course of development, we can easily change the connection without requiring any need to develop a new PCB. This model provides all the basic types of interface and is very suitable for sensor management. Technical parameters are given in Table 11 and the computer is in the Figure 55.

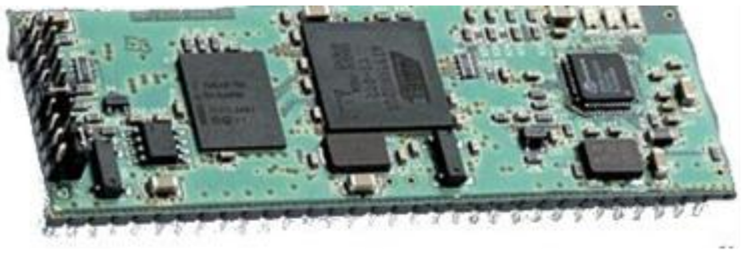


Figure 55: ***Computer DNP/9200***

Processor	Atmel AT91RM9200 32-bit ARM9 180 MHz
Memory RAM	32MB
Memory FLASH	16MB
Interfaces	DIL-64
Size	82mm x 28mm
Price	149.00 USD

Table 11: DNP/9200 computer specification

8.7.5.5. BeagleBone Black

This single board computer is the newest member of the BeagleBoard family. It is an open hardware project of single-board computers providing a broad software support: Android, Gentoo, ArchLinux, Minix, LinuxCNC and others. PCB with dimensions 86.36 mm x 53.34 mm x 4,76 mm includes a direct interfaces 10/100 Ethernet, USB, HDMI with a maximum resolution of 1280x1024 @ 60Hz, microSD. There is a number of additional modules that are compatible with BB Black. Microcomputer is possible to extend with the LCD module, CANbus, relay outputs, and opto-coupler isolated inputs, etc.

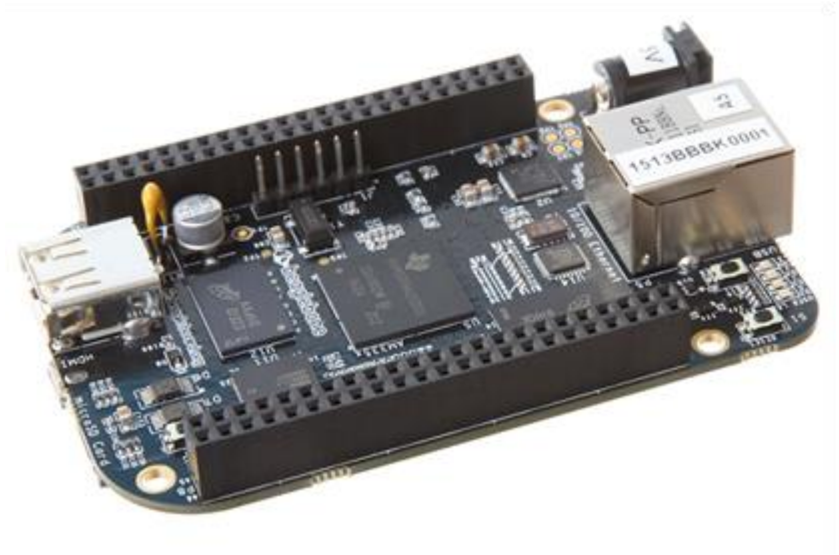


Figure 56: ***Single-board BeagleBone Black computer***

Processor	Sitara AM3359AZCZ100, 1GHz
Memory RAM	512MB

Memory FLASH	2GB
Size	3,4“ x 2,1“
Interfaces	USB, HDMI, Ethernet, UART, SPI, I2C,GPIO

Table 12: Technical parameters of BeagleBone Black computer

8.7.5.6. cubieboard2

Cubieboard2 single board computer is equipped with SoC Allwinner A20 based on Processor Dual core ARM Cortex-A7. The computer allows to connect a SATA drive through expansion board DVK521 that connects to two 48-pin connector. It provides connection of other external boards for the development of systems based on cubieboard2. The computer allows to work under Android, Ubuntu and other Linux distributions.

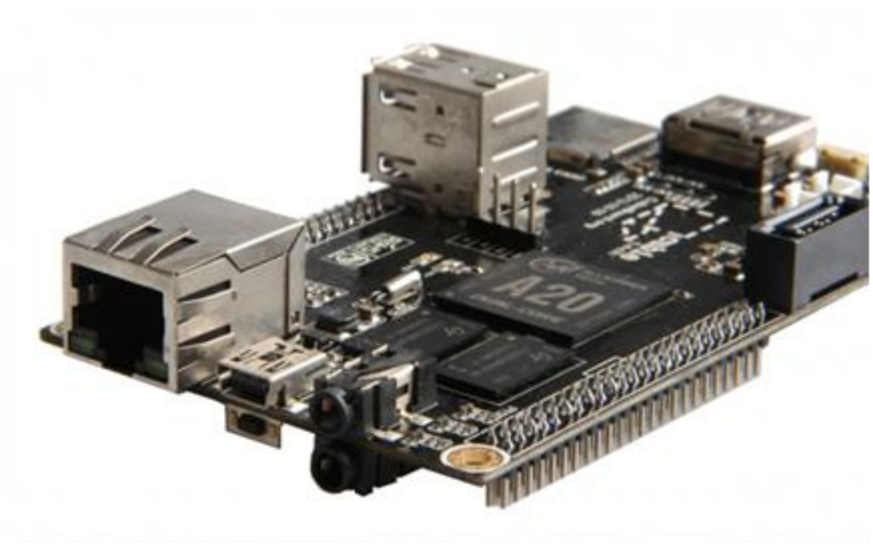


Figure 57: ***Singe-board computer Cubieboard2***

Processor	SoC A20, Dual core ARM cortex-A7 Dual-Core ARM® Mali400 GPU
Memory RAM	RAM 1GB DDR3 @480MHz, 3.4GB internal NAND flash, up to 64GB on SD slot, up to 2T on 2.5 SATA disk
Interface	HDMI 1080p Output, 10/ 100 Mbps Ethernet, 2 USB Host, 1 micro SD slot, 1 SATA, 1 IR 96 extend pin including I2C, SPI, …
Size	100 mm x 60 mm
Price	59.00 USD

Table 13: Parameters of cubieboard2 computer

8.7.5.7. phyCORE-OMAP5430

phyCORE-OMAP5430 System on Module (SOM) is one in a serie of modules developed by phyCORE Phytec. It is based on the processor TI OMAP5430. Architecture OMAP5430 processor includes dual-core ARM ® Cortex ™-A15 MPCore completed with several graphics accelerators. Processor directly provides advanced imaging and high performance peripheral interface.

phyCORE-OMAP5430 SOM is also available as a development kit, with optional supplement LCD and WiFi module.

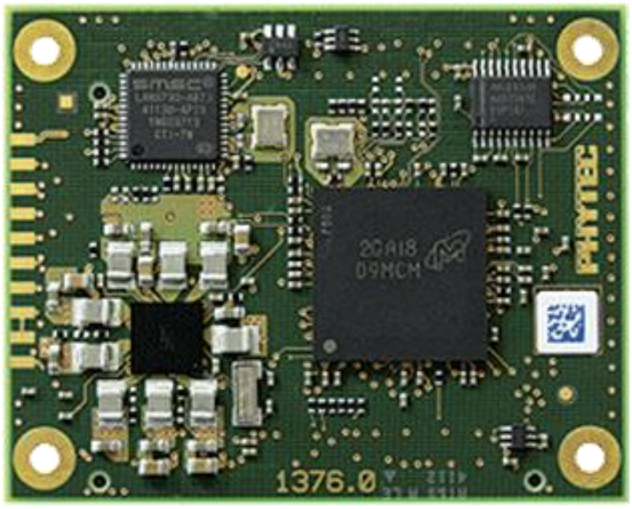


Figure 58: ***System on Module phyCORE-OMAP5430***

Processor	TI's OMAP5430 processor, 2x ARM Cortex™-A15 cores, 2x ARM Cortex™-M4 processors Multi-core POWERVR™ SGX544-MPx 3D graphics accelerator IVA-HD multimedia accelerator
Memory	up to 4 GB LPDDR2 RAM 64+ GB eMMC Flash Flash Expansion - 2x SDIO/MMC
Interface	10/100 Mbit/s Ethernet 1x USB 3.0 SuperSpeed Dual-Role-Device 1x USB 2.0 High-Speed Host, 1 x HSIC 4x UARTS (one RS-232) 3x I2C / 1x 1-Wire / 1x MCBSP / 2x MCSPI
Size	55 x 45 mm

Table 14: Parameters SOM phyCORE-OMAP5430

8.7.5.8. EPP-Pico-OMAP5430

Single board computer of industry standard Pico-ITX is based on the OMAP ™ 5 platform from Texas Instruments. It provides a wide range of interfaces, including Ethernet, DVI, and wireless connectivity based on the module Murata WiLink ™ 8.0. Software support includes Linux and Android 4.1 BSP and provides software compatibility with open source platform PandaBoard.

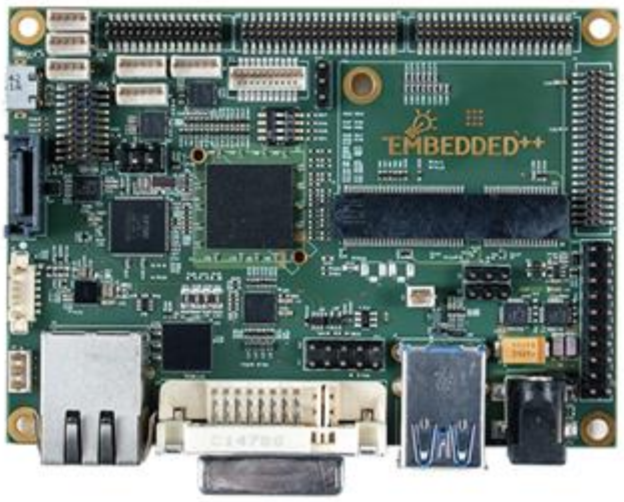


Figure 59: ***Singleboard computer EPP-Pico-OMAP5430***

Processor	Texas Instruments OMAP5430 Processor, dual core ARM Cortex-A15 up to 2 GHz, 2x ARM Cortex™-M4 processors DSP TMS320DM64 32-bit
Memory	LP-DDR2 RAM 2GB, eMMC storage up to 16 GB up to 64GB on SD slot
Interface	1x USB 3.0, 3x USB 2.0 SATA port 100 Mbit Ethernet 3x RS-232, SPI master port, GPIO DVI video out, LVDS dual channel Serial camera input port Micro SD card socket Wireless connectivity - Wi-Fi, GNSS, Bluetooth, FM
Size	100 x 72 x 19.48 mm

Table 15: Parameters EPP-Pico-OMAP5430

8.7.5.9. OMAP5432 EVM

OMAP5432 EVM of SVTRONICS and Texas Instruments is a development module based on OMAP5432 multimedia processor. It runs at a frequency of 1.5 GHz and has a double-controller ARM® Cortex™-A15 MPCore™ and dual-core SGX544 graphics processor. It provides full power for software development utilizing of the OMAP5432. Through the expansion connectors it provides support for the development of additional functionality and options.

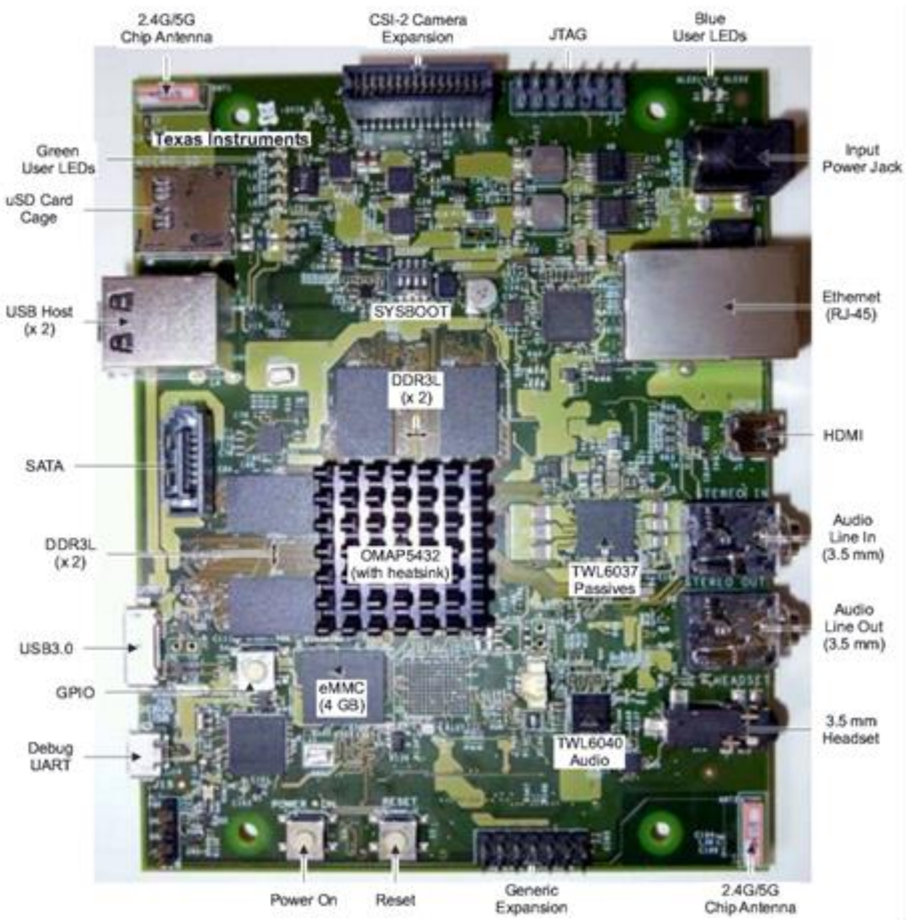


Figure 60: *Singleboard computer OMAP5432 EVM*

Processor	Texas Instruments OMAP5432
Memory	DDR3L RAM 2GB
Interface	1x USB 3.0, 2x USB 2.0 SATA port 100 Mbit Ethernet 3x RS-232, SPI master port, GPIO DVI video out, LVDS dual channel Serial camera input port Micro SD card socket Wireless connectivity - Wi-Fi, GNSS, Bluetooth, FM
Size	127 x 100 mm

Table 16: Parameters OMAP5432 EVM

8.7.5.10. RFM22 Module

RFM22 module is used for RFID reader and tag operation. RFM22 is a low cost RF transceiver module [340]. This wireless communication device can be used both as a reader and a tag. This module is used in various applications such as remote control, sensor networks, industrial control, and home automation. It supports 433 MHz operating frequency and has sensitivity of -121 dBm. Moreover, the module has a 64-byte transmit and receive FIFO and features like automatic packet control, low battery detector. Three modulation types are supported which are Frequency Shift Keying, Gaussian Frequency Shift Keying, and On-Off Keying. The module also supports digital Received Signal Strength Indicator (RSSI). The module has proper design characteristics for usage with a microcontroller or a microprocessor. It is also capable of direct connection with a microcontroller or microprocessor due to ADC on the module. As a crucial feature, the module can be controlled via software that is included in the microprocessor.

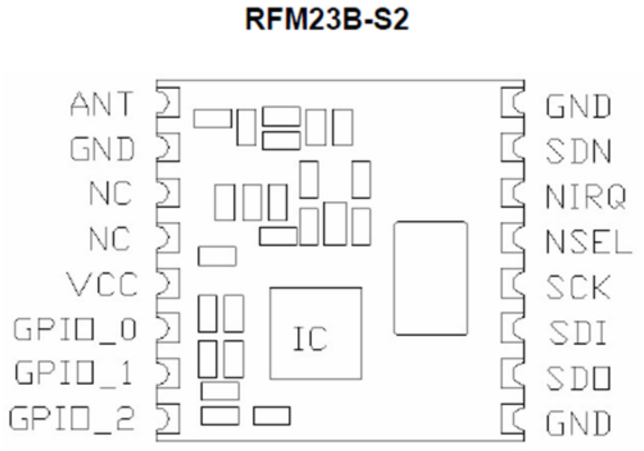


Figure 61: RFM22 RF module connection [340]

An RFM22 module communicates with a microprocessor by SPI communication protocol. As shown in Figure 61, connections between SDI and MOSI, SDO and MISO, NSEL and SS, SCK and SCK must be provided for SPI communication. The communication protocol is used for writing to registers and reading from registers on the module. SPI data transmission of this module is done with a 16-bit array where bits are allocated as follows: one bit for read-write selection, 7 bits for address area, and 8 bits for data.

8.7.5.11. Arduino Uno

Arduino is a microcontroller board based on the ATMEL microcontroller series [341]. Arduino UNO is the simplest and the cheapest board among the Arduino board family. Other than being a microcontroller, the board consists of 14 digital input/output pins, 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an (In-Circuit Serial Programming (ICSP) header, and a reset button. The Arduino has a specific software called Arduino Suite to program any Arduino compatible board. With a growing number of product-specific libraries Arduino offers useful tools for designs related with embedded systems.

Technical specifications of Arduino UNO areas follows [341]:

- Microcontroller: ATmega328
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Input Voltage (limits): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- Analog Input Pins: 6
- DC Current per I/O Pin: 40 mA
- DC Current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB of which 0.5 KB used by boot loader

- SRAM: 2 KB
- EEPROM: 1 KB
- Clock Speed; 16 MHz

A front view and the main components of the Arduino UNO board can be seen in Figure 62.

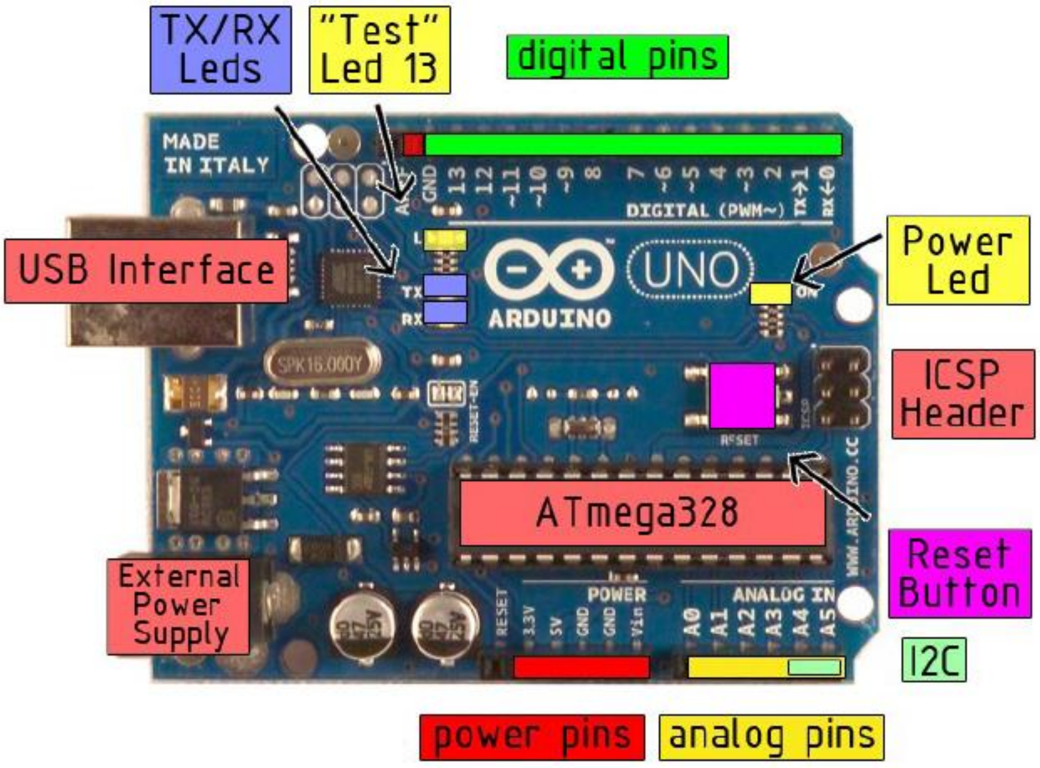


Figure 62: Arduino UNO board [341]

Descriptions of pin connections are:

- ANT: Antenna is connected to this pin
- GND: Represents ground pin
- NC: Not connected which means that these are empty pins
- VCC: Power supply is connected to this pin
- GPIO_0/1/2: Represents the general input-output pins
- SDN: Shutdown pin and grounded in this project
- NIRQ: General microcontroller interrupt status control pin and grounded in this project
- NSEL: Serial interface select input and connected to slave select pin of Arduino
- SCK: Serial clock input and connected to serial clock output of Arduino
- SDI: Serial data input and connected to master (Arduino) output
- SDO: Serial data output and connected to master (Arduino) input

8.7.5.12. Intel Galileo

Intel Galileo is a microcontroller board that includes an Intel Quark SoC X1000 Application Processor and a 32-bit Intel Pentium-class system on a chip [342]. It has hardware and software compatibility with shields specific for Arduino Uno R3. In other words, the Galileo board has Arduino 1.0 pinout structure. This board has flexible usage that is operational with various operating systems. Moreover, a Galileo board can also be programmed with Arduino integrated development environment (IDE) software. Due to this compatibility, a Galileo board can be used in various scenarios. A physical view of the board can be seen in Figure 63.

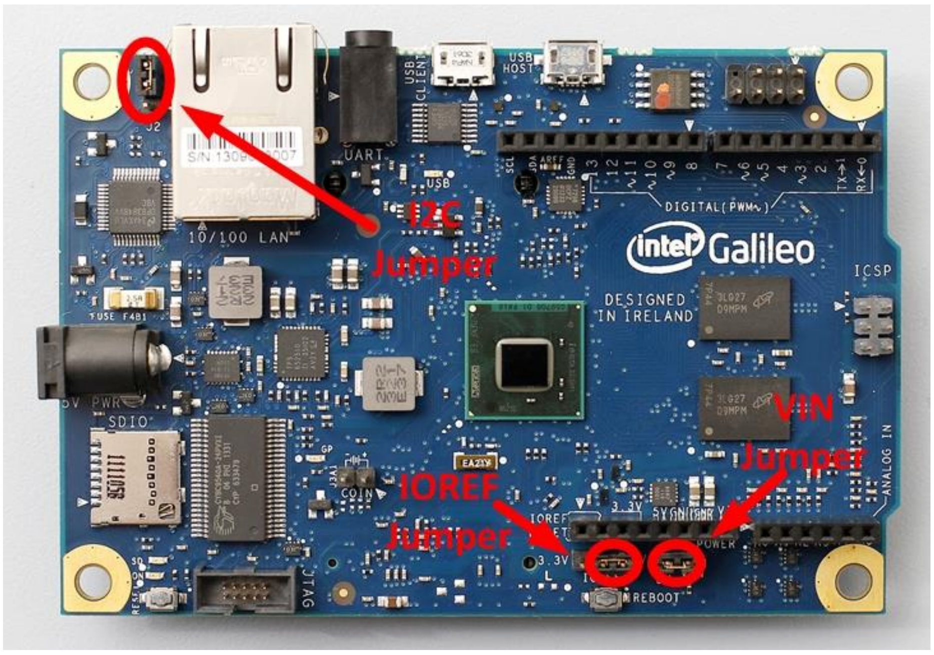


Figure 63: Intel Galileo board [342]

The initial structure of the board is shown in Figure 64.

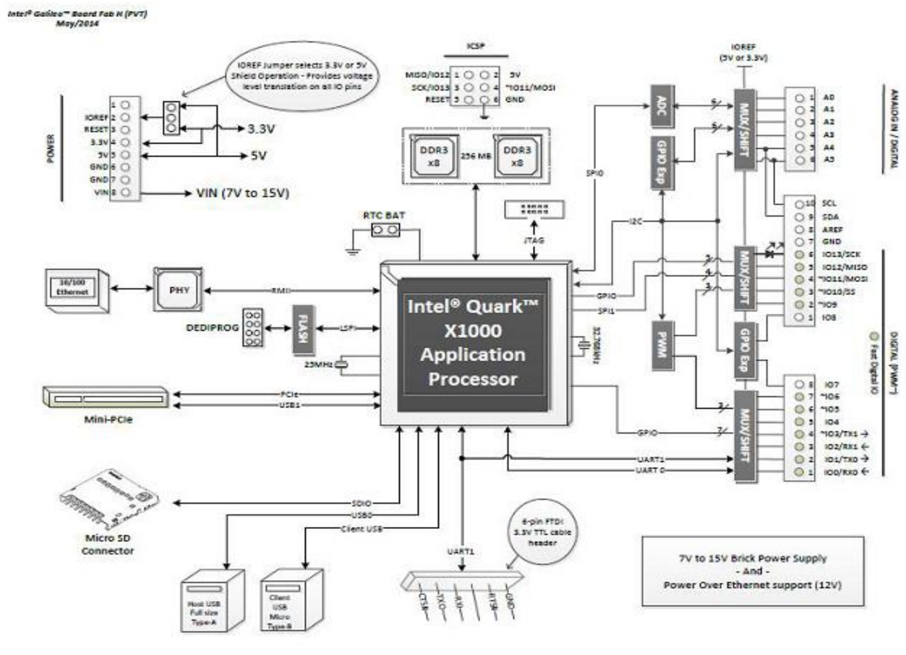

Intel® Quark™ X1000 Application Processor
IOREF Jumper selects 3.3V or 5V Shield Operation - Provides voltage level translation on all IO pins
3.3V
5V
VIN (7V to 15V)
RTC BAT
PHY
10/100 Ethernet
DEDIPROG
FLASH
Mini-PCIe
Micro SD Connector
DDR3 x8
256 MB
ICSP
JTAG
ADC
GPIO Exp
MUX/SHIFT
PWM
7V to 15V Brick Power Supply - And - Power Over Ethernet support (12V)
6-pin FTDI 3.3V TTL cable header

Figure 64: Board structure [342]

The processor and storage capabilities of the board are listed below [342]:

- Instruction set architecture (ISA)-compatible 32-bit Intel® Pentium® processor
- 16 Kbytes L1 cache
- 512 Kbytes of on-die embedded SRAM
- Simple to program: single thread, single core, constant speed

- ACPI-compatible CPU sleep states supported
- Integrated real-time clock (RTC) with optional 3V “coin cell” battery for operation between turn on cycles
- 400 MHz clock speed
- 8 Mbyte Legacy SPI Flash to store firmware (bootloader) and the latest sketch
- Between 256 Kbytes and 512 Kbytes dedicated for sketch storage
- 512 Kbytes embedded SRAM
- 256 Mbytes DRAM
- Optional micro SD card offers up to 32 Gbytes of storage
- USB storage works with any USB 2.0 compatible drive
- 11 Kbytes EEPROM programmed via the EEPROM library

8.7.5.13. Temperature Sensor

The LMT84 is a precision analog output CMOS integrated-circuit temperature sensor [343]. Analog output of the sensor changes linearly and inversely proportional to temperature. The sensor has a physical view as in Figure 65.

Figure 65: LMT84 temperature sensor [343]

Some features of the sensor are [343]:

- Low 1.5 V Operation
- Very Accurate: ±0.4°C typical
- Wide Temperature Range of –50°C to 150°C
- Low 5.4µA Quiescent Current
- Average Sensor Gain of -5.5 mV/°C
- Output is Short-Circuit Protected
- Push-Pull Output with ±50 µA Drive Capability
- Cost-effective Alternative to Thermistors

8.7.5.14. HC-05 Bluetooth RF Transceiver Module

HC-05 Bluetooth Module is a well-known Bluetooth serial module. In order to convert serial port to Bluetooth, serial modules which have two modes as master and slave, are used [344]. In master mode, HC-05 module can both search and pair with a Bluetooth device automatically. The module is a 2.4 GHz radio transceiver and compatible with Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation. A physical view of the module is shown in Figure 66.

Figure 66: HC-05 Bluetooth Module [344]

Technical details of the supported hardware and software features are [345]:

- Typical -80dBm sensitivity
- Up to +4dBm RF transmit power
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- PIO control
- UART interface with programmable baud rate
- With integrated antenna
- With edge connector
- Default Baud rate: 38400, Data bits:8, Stop bit:1,Parity: No parity, Data control: has supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Given a rising pulse in PIO0, device will be disconnected.
- Status instruction port PIO1: low-disconnected, high-connected;
- PIO10 and PIO11 can be connected to red and blue led separately. When master and slave are paired, red and blue led blinks 1time/2s in interval, while disconnected only blue led blinks 2times/s.
- Auto-connect to the last device on power as default.
- Permit pairing device to connect as default.
- Auto-pairing PINCODE:"0000" as default
- Auto-reconnect in 30 min when disconnected as a result of beyond the range of connection.

8.7.5.15. Conclusion

As best platforms for further development were selected modules based on ARM Cortex A7 or ARM Cortex A15 CPU. Further development will take place on the modules cubieboard2 and EPP-Pico-OMAP5430.

8.8. Pixage - Digital Signage

Pixage is a digital signage application developed by KoçSistem where content management is conducted centrally and end players broadcast by receiving contents from the center. Unlike traditional info-screens, Pixage facilitates dynamic content display using internet infrastructure.

Pixage features conducting training programs, broadcasting ads, sending multiple corporate communications messages with specific contents.

8.8.1. **Features**

- Easy content management
- Messages Specific to The Target Group
- Multiple Broadcasts on a Single Screen
- Integration with Third Parties

- Ease of Manageability
- Facility to Broadcast Different Contents
- Right Point, Right Time, Right Message
- Multi-language
- Remote or centric software updates
- Player box maintenance jobs
- Centric license control module
- HTTP/HTTPS file transfer

Limitless content groups can be organized on the Pixage application using desired quantity of installed contents.

By grouping the Pixage players, a more manageable system that directs the messages to the appropriate target group can be created. Analyzing purchasing decisions results in regrouping the target regions and creating separate broadcast channels for each group. Any desired content list can be streamed on each channel.

The time and duration of the broadcasts are set by Pixage, so that a system in a public store can be customized to stream different broadcasts for the customers who visit the stores in the morning and for the ones who visit in the evening.

More attractive messages can be created using Pixage. Any number of contents can be simultaneously played on the screen. For example, by dividing the screen into two parts, the stock exchange data can be displayed at the same time with the company's corporate video.

Pixage can integrate any kind of third party contents. For example, corporate applications and the latest sales rates can be displayed together. Pixage enables broadcasting a variety of contents from different data sources such as weather, match scores, financial data and news.

Full HD video, PowerPoint, RSS, photographs, 3D graphics, text and on-video scrolling texts are only a few of the broadcast types that are available in Pixage to be presented on the players.

8.6.2. Contents

- Video
- Image/Graphic
- Flash-URL Feed
- Web Pages-URL Feed
- Power Point Presentation (full screen with PPT viewer)
- Special Contents
    - Digital clock
    - Displaying from TV card
    - URL Feed
    - RSS Feed
    - SQL Feed
    - XML Feed
    - MMS
    - Rich Text
    - Ticker Text

- News
- Weather Forecast

8.6.3. Architecture

Pixage implements the relations between major applications to satisfy all kinds of needs quickly and effectively without any problem (Figure 67). Processes are made at the same time, so synchronization is achieved.

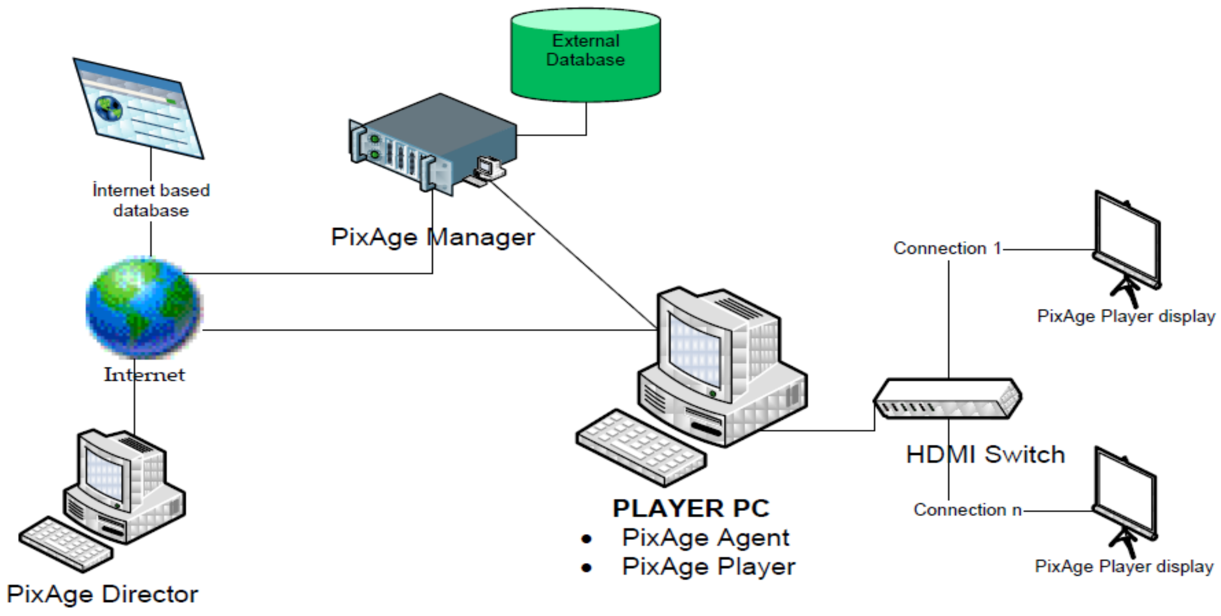


Figure 67: Data Flow

Pixage Manager is responsible for transferring the contents and broadcasts to the players. Pixage Director which has been set up on the manager PC is responsible for the creation of contents and broadcasts. Pixage Agent that is set up on the player connects with Pixage Manager, configures, installs content, logs, records and updates software. Pixage Player displays scheduled or designed broadcast on different screens as monitor, LCD screen, and plasma by tracking play list.

Security is an important subject in a digital signage application. There are some precautions for the data security in Pixage. Pixage contains access limits for applications. Authorized users can only access the system based on their roles. Also data communication is done in an entirely secure platform by high level security algorithms.

Pixage has a reliable network control structure (Figure 68). For this reason, the player status can be tracked by the web-based report tools. If there is a problem in any application the problem's details are sent to the related users by e-mail.

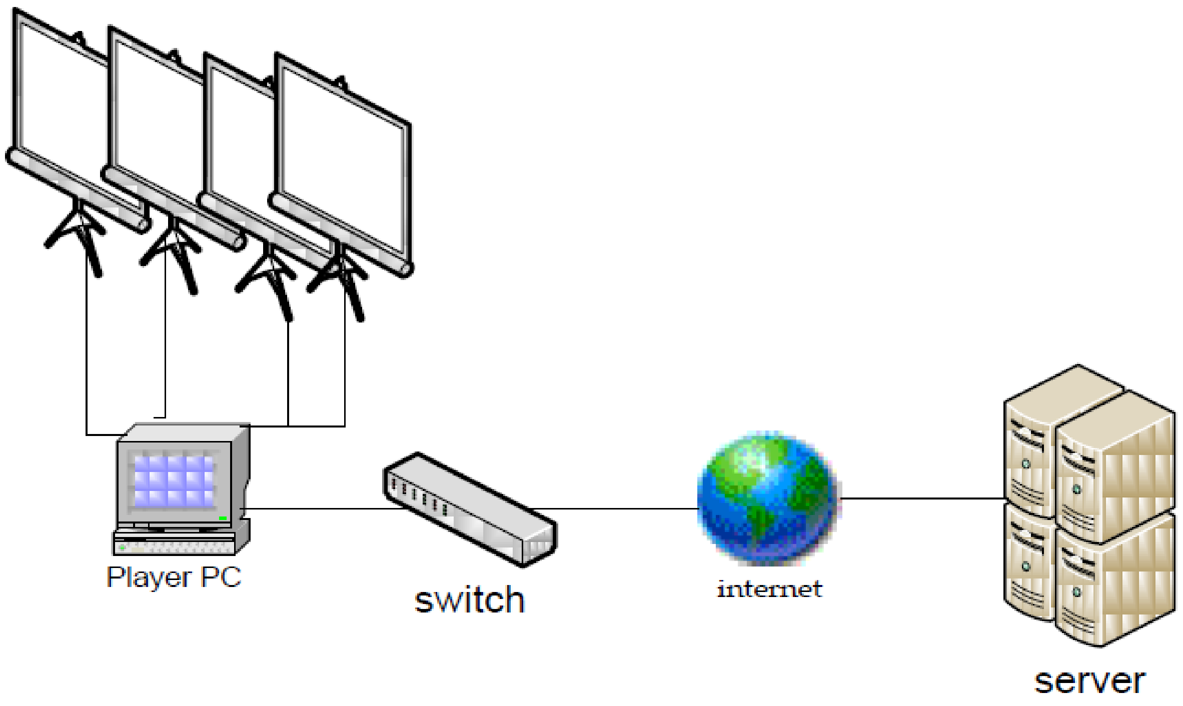


Figure 68: System architecture & Server – Player connection

8.6.4. Why is it necessary to BaaS?

Nowadays many buildings like stadiums, corporate buildings, hotels, hospitals and shopping malls are covered with digital screens that display info or broadcast ads. Digital screen usage cannot be ignored in a building's infrastructure.

Integration of Pixage via Baas Services adds the advantages of digital signage systems into the Baas framework. The Baas framework will allow an easy way of communication between the digital screens and the other devices and systems installed in a building.

In the evacuation scenario/use-case, screens will quickly guide people to the safe zones when an alert is generated from a fire/smoke detector.

8.9. Keycloak - IDM

"Keycloak is an SSO solution for web apps, mobile and RESTful web services. It is an authentication server where users can centrally login, logout, register, and manage their user accounts. The Keycloak admin UI can manage roles and role mappings for any application secured by Keycloak. The Keycloak Server can also be used to perform social logins via the user's favorite social media site i.e. Google, Facebook, Twitter etc. [346]

8.9.1. Features [346]

- SSO and Single Log Out for browser applications
- Social Login. Enable Google, GitHub, Facebook, Twitter social login with no code required.
- LDAP and Active Directory support.
- Optional User Registration
- Password and TOTP support (via Google Authenticator). Client cert auth coming soon.

- Forgot password support. User can have an email sent to them
- Reset password/totp. Admin can force a password reset, or set up a temporary password.
- Not-before revocation policies per realm, application, or user.
- User session management. Admin can view user sessions and what applications/clients have an access token. Sessions can be invalidated per realm or per user.
- Pluggable theme and style support for user facing screens. Login, grant pages, account mgmt, and admin console all can be styled, branded, and tailored to your application and organizational needs.
- Integrated Browser App to REST Service token propagation
- OAuth Bearer token auth for REST Services
- OAuth 2.0 Grant requests
- OpenID Connect Support.
- SAML Support.
- CORS Support
- CORS Web Origin management and validation
- Completely centrally managed user and role mapping metadata. Minimal configuration at the application side
- Admin Console for managing users, roles, role mappings, clients, user sessions and allowed CORS web origins.
- Account Management console that allows users to manage their own account, view their open sessions, reset passwords, etc.
- Deployable as a WAR, appliance, or on Openshift. Completely clusterable.
- Multitenancy support. You can host and manage multiple realms for multiple organizations. In the same auth server and even within the same deployed application.
- Identity brokering/chaining. You can make the Keycloak server a child IDP to another SAML 2.0 or OpenID Connect IDP.
- Token claim, assertion, and attribute mappings. You can map user attributes, roles, and role names however you want into a OIDC ID Token, Access Token, SAML attribute statements, etc. This feature allows you to basically tailor however you want auth responses to look.
- Supports JBoss AS7, EAP 6.x, Wildfly, Tomcat 7, Tomcat 8, Jetty 9.1.x, Jetty 9.2.x, Jetty 8.1.x, and Pure JavaScript applications. Plans to support Node.js, RAILS, GRAILS, and other non-Java deployments

8.9.2. Architecture [347]

In a traditional multi-tiered architecture like the one shown in the Figure 69, a server-side web tier deals with authenticating the user by calling out to a relational database or an LDAP server. An HTTP session is then created containing the required authentication and user details. The security context is propagated between the tiers within the application server so there is no need to re-authenticate the user.

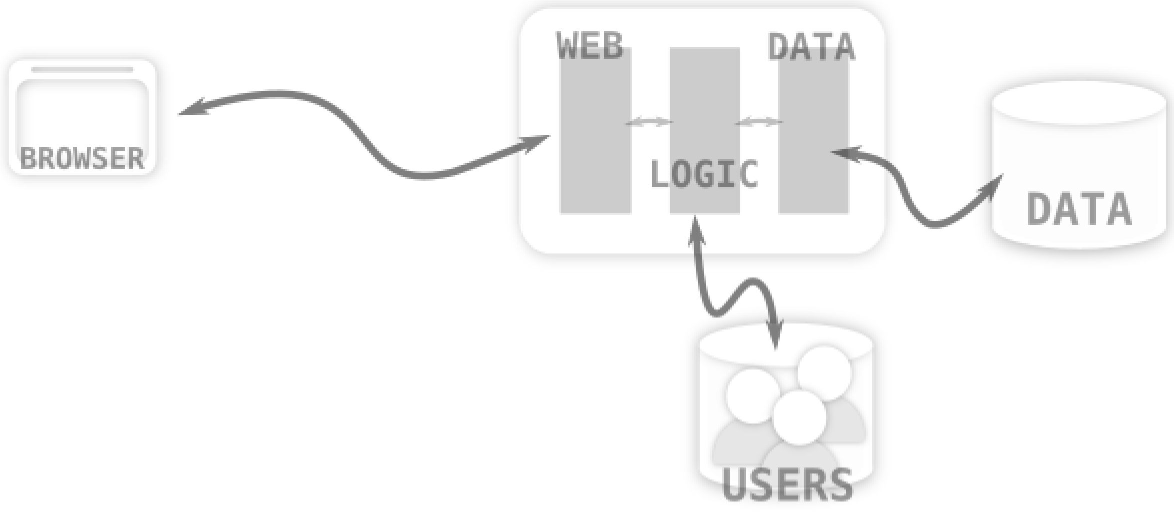


Figure 69: Traditional multi-tiered architecture

With a microservice architecture (Figure 70), the server-side web application is gone; instead, we have HTML5 and mobile applications on the client side. The applications invoke multiple services, which may in turn call other services. In this architecture, there is no longer a single layer that can deal with authentication and usually it is stateless as well, so there is no HTTP session.

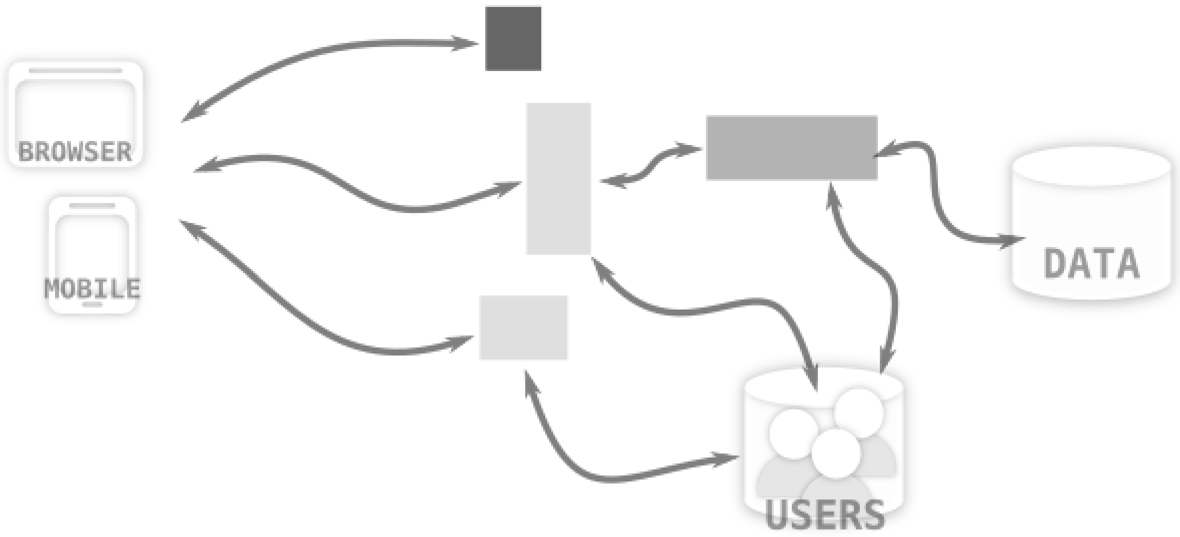


Figure 70: Microservice architecture

As microservices are all about having many smaller services each that deal with one distinct task the obvious solution to security is an authentication and authorization service. This is where Keycloak and OpenID Connect comes to the rescue. Keycloak provides the service you need to secure microservices.

The first step to securing microservices is authenticating the user. This is done by adding the Keycloak JavaScript adapter to your HTML5 application. For mobile applications there's a Keycloak Cordova adapter, but there's also native support though the AeroGear project [348].

In an HTML5 application you just need to add a login button and that is pretty much it. When the user clicks the login button, the user’s browser is redirected to the login screen on the Keycloak server. The user then authenticates with the Keycloak server. As the authentication is done by the Keycloak server and not your application, it is easy to add support for multi-factor authentication or social logins without having to change anything in your application.

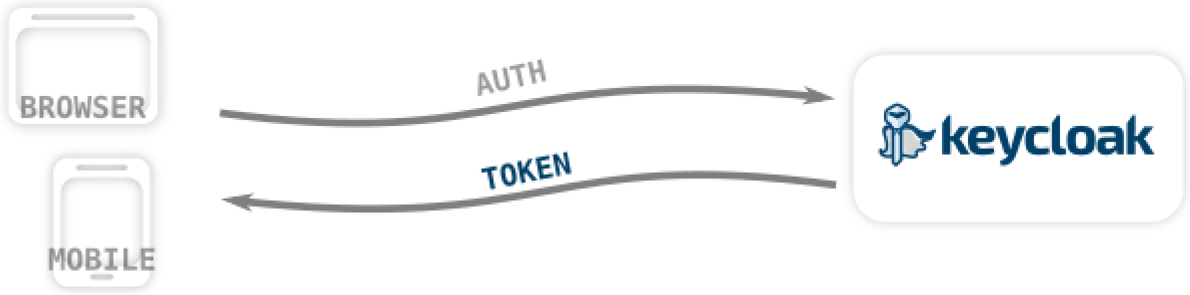


Figure 71: User authentication with Keycloak

Once the user is authenticated, Keycloak gives the application a token (Figure 71). The token contains details about the user as well as permissions the user has.

The second step is to secure the microservices. Again, with Keycloak this is easy to do with our adapters. We have JavaEE adapters, NodeJS adapters and we are planning to add more in the future. If we do not have an adapter, it is also relatively easy to verify the tokens yourself. A token is basically just a signed JSON document and can be verified by the services itself or by invoking the Keycloak server.

If a service needs to invoke another service it can pass on the token it received, which will invoke the other service with the users permissions (Figure 72). Soon we will add support for services to authenticate directly with Keycloak to be able to invoke other services with their own permissions, not just on behalf of users.

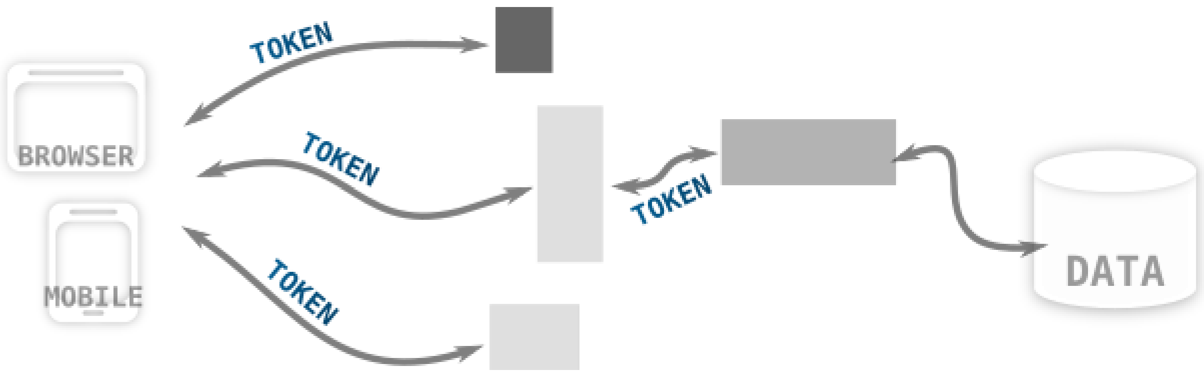


Figure 72: Secure service invocation with tokens

For more details about OpenID Connect you can look at the OpenID Connect website [349], but the nice thing is with Keycloak you don't really need to know the low level details so it's completely optional. As an alternative just go straight to the Keycloak website [350], download the server and adapters, and check out our documentation and many examples.

8.9.3. Why is it necessary to BaaS?

Until recently, most of the internet applications are designed to run on single server model and to use password authentication to allow access to applications meaning that users need to enter credentials to gain access to the specified system. This also means that users need to remember their credentials for each web application that they want to access. Recent developments now allow central management for users and their credentials. This architecture is designed for any project that targets to provide a platform for various applications running on cloud environments. Although these applications run independently, for such platforms it is important to provide a trust relationship between applications and the platform, allowing users to sign-on and gain access without being further prompted for credentials for each application.

9. List of Figures

10. List of Tables

11. References

[1] American Society of Heating, Refrigerating and Air-Conditioning Engineers (ASHRAE), “BACnet - A Data Communication Protocol for Building Automation and Control Networks,” 2009. [Online]. Available: http://www.bacnet.org/Bibliography/EC-9-97/EC-9-97.html.

[2] “LonWorks,” *Echelon*, 2013. [Online]. Available: http://www.echelon.com/technology/lonworks/.

[3] R. Zimmer, “North America - Slow and Cautious With NextGen Projects,” 27-Jun-2005.

[4] S. Soucek and G. Zucker, “Current developments and challenges in building automation,” *e & i Elektrotechnik und Informationstechnik*, vol. 129, no. 4, pp. 278–285, Jun. 2012.

[5] “Directive 2010/31/EU of the European Parliament and of the Council,” *Official Journal of the European Union*, May 2010.

[6] “ISO 16484-2:2004 Building automation and control systems (BACS) -- Part 2: Hardware.” ISO/TC 205.

[7] H. Merz, T. Hansemann, and C. Hübner, *Building Automation - Communication Systems with EIB/KNX, LON, and BACnet*. Springer, 2009.

[8] “System topologies - Building Technologies - Siemens.” [Online]. Available: http://www.buildingtechnologies.siemens.com/bt/global/en/buildingautomation-hvac/building-automation/building-automation-and-control-system-europe-desigo/system/topologies/small-to-extensive-systems/Pages/from-small-to-complex-system-topologies.aspx#. [Accessed: 27-Apr-2014].

[9] S. Szucsich, “Web Services in Building Automation with focus on BACnet/WS,” 2010.

[10] “Communication (KNX, BACnet) - Building Technologies - Siemens.” [Online]. Available: http://www.buildingtechnologies.siemens.com/bt/global/en/buildingautomation-hvac/building-automation/building-automation-and-control-system-europe-desigo/system/communication/Pages/communication.aspx. [Accessed: 27-Apr-2014].

[11] M. Jung, J. Weidinger, W. Kastner, and A. Olivieri, “Building Automation and Smart Cities: An Integration Approach Based on a Service-Oriented Architecture,” 2013, pp. 1361–1367.

[12] “Project Haystack.” [Online]. Available: http://project-haystack.org/.

[13] “Niagara AX.” [Online]. Available: http://www.niagaraax.com/galleries/brochures/Niagara_Brochure.pdf.

[14] S. Azhar, M. Hein, and B. Sketo, “Building Information Modeling (BIM): Benefits, Risks and Challenges.”

[15] “ELASSTIC Project.” [Online]. Available: http://www.elasstic.eu/.

[16] L. F. Cranor, “A Framework for Reasoning About the Human in the Loop.,” in *UPSEC*, 2009.

[17] C. Müller-Schloer, H. Schmeck, and T. Ungerer, Eds., *Organic Computing - A Paradigm Shift for Complex Systems*. Birkhäuser, 2011.

[18] J. O. Kephart and D. M. Chess, “The Vision of Autonomic Computing,” in *Computer*, 2003, vol. 36, num. 1, pp. 41–50.

[19] D. L. Tennenhouse, “Proactive Computing,” *Commun. ACM*, vol. 43, no. 5, pp. 43–50, 2000.

[20] H. Schmeck, C. Müller-Schloer, E. Çakar, M. Mnif, and U. Richter, “Adaptivity and Self-organization in Organic Computing Systems,” *ACM Trans. Auton. Adapt. Syst.*, vol. 5, no. 3, p. 10:1–10:32, Sep. 2010.

[21] R. Laddaga, “Self-Adaptive Software,” DARPA Broad Agency Announcement, 1997.

[22] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, A. L. Wolf, and E. L. Wolf, “An Architecture-Based Approach to Self-Adaptive Software,” *IEEE Intelligent Systems*, vol. 14, pp. 54–62, 1999.

[23] K. J. Lieberherr and C. Xiao, “Customizing Adaptive Software to Object-Oriented Software Using Grammars.,” *Int. J. Found. Comput. Sci.*, vol. 5, no. 2, pp. 179–208, 1994.

[24] K. J. Lieberherr, I. Silva-Lepe, and C. Xiao, “Adaptive Object-Oriented Programming Using Graph-Based Customization.,” *Commun. ACM*, vol. 37, no. 5, pp. 94–101, 1994.

[25] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel, “Towards a Taxonomy of Software Change: Research Articles,” *J. Softw. Maint. Evol.*, vol. 17, no. 5, pp. 309–332, Sep. 2005.

[26] M. Salehie, S. Li, R. Asadollahi, and L. Tahvildari, “Change Support in Adaptive Software: A Case Study for Fine-Grained Adaptation,” in *Proceedings of the 2009 Sixth IEEE Conference and Workshops on Engineering of Autonomic and Autonomous Systems*, Washington, DC, USA, 2009, pp. 35–44.

[27] M. C. Huebscher and J. A. McCann, “A Survey of Autonomic Computing - Degrees, Models, and Applications,” *ACM Comput. Surv.*, vol. 40, no. 3, pp. 1–28, Aug. 2008.

[28] IBM, “An Architectural Blueprint for Autonomic Computing,” IBM, Jun. 2005.

[29] J. P. Thompson, “Web-Based Enterprise Management Architecture,” *Communications Magazine, IEEE*, vol. 36, no. 3, pp. 80–86, 1998.

[30] *Common Information Model (CIM) Specification*. Distributed Management Task Force, 1999.

[31] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, “Design and Evaluation of a Wide-Area Event Notification Service,” *ACM Transactions on Computer Systems*, vol. 19, pp. 332–383.

[32] T. Smolarek, *Effective Device Management: Using Open Mobile Alliance Device Management*. Germany: LAP Lambert Academic Publishing, 2011.

[33] J. D. Turner, D. A. Bacigalupo, S. A. Jarvis, and D. N. Dillenberger, “Application Response Measurement of Distributed Web Services,” *Journal of Computer Resource Measurement*, vol. 108, pp. 45–55, 2002.

[34] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin, *Simple Network Management Protocol (SNMP)*. IETF, 1990.

[35] H. Kreger, W. Harold, and L. Williamson, *Java and JMX: Building Manageable Systems*. Boston, MA: Addison-Wesley, 2003.

[36] M. G. Hinchey and R. Sterritt, “Self-Managing Software,” *Computer*, vol. 39, no. 2, pp. 107–109, Feb. 2006.

[37] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

[38] D. Alur, D. Malks, and J. Crupi, *Core J2EE Patterns: Best Practices and Design Strategies*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[39] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester, UK: Wiley, 1996.

[40] J. O. Kephart, “Research Challenges of Autonomic Computing,” in *ICSE*, 2006, pp. 15–22.

[41] E. Gat, “Three-layer Architectures,” in *Artificial Intelligence and Mobile Robots*, Cambridge: MIT/AAAI Press, 1997, pp. 195–210.

[42] S. Dobson, S. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, “A Survey of Autonomic Communications,” *ACM Trans. Auton. Adapt. Syst.*, vol. 1, no. 2, pp. 223–259, Dec. 2006.

[43] C. Müller-Schloer and H. Schmeck, “Organic Computing: A Grand Challenge for Mastering Complex Systems,” *it - Information Technology*, vol. 52, no. 3, pp. 135–141, 2010.

[44] H. Prothmann, *Organic traffic control.* Karlsruhe Institute of Technology, 2011.

[45] Universität Paderborn, “Collaborative Research Centre 614,” 2014. [Online]. Available: http://www.sfb614.de/.

[46] O. Oberschelp, T. Hestermeyer, B. Kleinjohann, and L. Kleinjohann, “Design of self-optimizing agent-based controllers.,” in *Proceedings of the Workshop 2002*, 2002.

[47] D. Pavlovic, “Towards Semantics of Self-Adaptive Software.,” in *IWSAS*, 2002, vol. 1936, pp. 65–74.

[48] G. Karsai and J. Sztipanovits, “A Model-Based Approach to Self-Adaptive Software,” *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 46–53, May 1999.

[49] P. Clements, “A Survey of Architecture Description Languages,” in *Proc. Int’l Workshop on Software Specification and Design*, 1996, pp. 16–25.

[50] J. S. Bradbury, J. R. Cordy, J. Dingel, and M. Wermelinger, “A Survey of Self-Management in Dynamic Software Architecture Specifications,” in *Proc. ACM SIGSOFT workshop on Self-managed systems (WOSS*, Newport Beach, California, 2004, pp. 28–33.

[51] D. Garlan, R. T. Monroe, and D. Wile, “Acme: Architectural Description of Component-Based Systems,” in *Foundations of Component-Based Systems*, G. T. Leavens and M. Sitaraman, Eds. Cambridge University Press, 2000, pp. 47–68.

[52] M. Sloman, “Policy Driven Management For Distributed Systems,” *Journal of Network and Systems Management*, vol. 2, pp. 333–360, 1994.

[53] R. Yavatkar, D. Pendarakis, and R. Guerin, *A Framework for Policy-based Admission Control*. IETF, 2000.

[54] J. O. Kephart and W. E. Walsh, “An Artificial Intelligence Perspective on Autonomic Computing Policies.,” in *POLICY*, 2004, pp. 3–12.

[55] D. Hutchison, G. Coulson, A. Campbell, and G. S. Blair, “Quality of Service Management in Distributed Systems,” in *Department of Computing, Lancaster University, Lancaster*, 1994, pp. 1–4.

[56] H. Lutfiyya, G. Molenkamp, M. Katchabaw, and M. A. Bauer, “Issues in Managing Soft QoS Requirements in Distributed Systems Using a Policy-Based Framework.,” in *POLICY*, 2001, vol. 1995, pp. 185–201.

[57] M. Woodside and D. A. Menasc?, “Guest Editors’ Introduction: Application-Level QoS,” *IEEE Internet Computing*, vol. 10, no. 3, pp. 13–15, 2006.

[58] D. A. Menascé and M. N. Bennani, “Autonomic Virtualized Environments,” in *IN PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON AUTONOMIC AND AUTONOMOUS SYSTEMS (ICAS 2006*, 2006.

[59] Y. Agarwal, B. Balaji, R. Gupta, J. Lyles, M. Wei, and T. Weng, “Occupancy-Driven Energy Management for Smart Building Automation,” in *Proceedings of the 2nd ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Building*, New York, NY, USA, 2010, pp. 1–6.

[60] J. Howard and W. Hoff, “Forecasting building occupancy using sensor network data,” in *Proceedings of the 2nd International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, New York, NY, USA, 2013, pp. 87–94.

[61] P. Alpar and J. Niedereichholz, *Data Mining Im Praktischen Einsatz: Verfahren und Anwendungsfälle Für Marketing, Vertrieb, Controlling und Kundenunterstützung.* Braunschweig, Wiesbade n: Vieweg+Teubner Verlag, 2000.

[62] U. Fayyad, G. Piatetsky.shapiro, and P. Smyth, “From Data Mining to Knowledge Discovery in Databases,” *AI Magazine*, vol. 17, pp. 37–54, 1996.

[63] Y. Brun, G. D. Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw, “Engineering Self-Adaptive Systems through Feedback Loops,” in *Self-Adaptive Systems*, Springer-Verlag Berlin Heidelberg, 2009.

[64] B. Park, Y. J. Won, H. Yu, J. W.-K. Hong, H.-S. Noh, and J. J. Lee, “Fault Detection in IP-based Process Control Networks using Data Mining,” in *Integrated Network Management, 2009.*, Long Island, NY, USA, 2009, pp. 211–217.

[65] S. Piramuthu, “Evaluating feature selection methods for learning in data mining applications.,” *European Journal of Operational Research*, vol. 156, no. 2, pp. 483–494, 2004.

[66] F. Beekmann and P. Chamoni, “Verfahren des Data Mining,” in *Analytische Informationssysteme*, Berlin Heidelberg: Springer, 2006, pp. 263–282.

[67] Y. Gao, E. Tumwesigye, B. Cahill, and K. Menzel, “Using data mining in optimisation of building energy consumption and thermal comfort management,” in *2nd International Conference on Software Engineering and Data Mining (SEDM), 2010*, Chengdu, China, 2010, pp. 434–439.

[68] M. W. Craven and J. W. Shavlik, “Using Neural Networks for Data Mining,” *Future Generation Computer Systems*, vol. 13, pp. 211–229, 1997.

[69] C. Böhm and F. Krebs, “High performance data mining using the nearest neighbor join,” in *Proceedings. 2002 IEEE International Conference on Data Mining, 2002.*, 2002, pp. 43–50.

[70] Microsoft Corporation, “Service Oriented Architecture (SOA),” 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/bb833022.aspx.

[71] Oracle, “Remote Method Invocation introduction,” 2014. [Online]. Available: http://www.bacnet.org/Bibliography/EC-9-97/EC-9-97.html.

[72] R. T. Fielding and R. N. Taylor, “Principled design of the modern Web architecture,” *ACM Transactions on Internet Technology*, vol. 2, no. 2, pp. 115–150, May 2002.

[73] D. Barisic, M. Krogmann, G. Stromberg, and P. Schramm, “Making Embedded Software Development More Efficient with SOA,” 2007, pp. 941–946.

[74] F. Jammes and H. Smit, “Service-oriented architectures for devices - the SIRENA view,” 2005, pp. 140–147.

[75] OASIS, “Devices Profile for Web Services (DPWS),” 2009. [Online]. Available: http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01.

[76] Contributing Members of the UPnP Forum, “UPnP Device Architecture 1.1,” 2008. [Online]. Available: http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf.

[77] Bluetooth SIG, “Bluetooth,” 2014. [Online]. Available: http://www.bluetooth.com/Pages/Fast-Facts.aspx.

[78] “KNX,” *KNX Association*, 2012. [Online]. Available: http://www.knx.org/knx-en/index.php.

[79] H. Bohn, A. Bobek, and F. Golatowski, “SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains,” 2006, pp. 43–43.

[80] H. Newman, “BACnet Explained, Part One‘ ’BACnet Today,” *Supplement to ASHRAE Journal*, vol. Vol. 55, no. No. 11, pp. B2–B7, Nov. 2013.

[81] LonMark International, “LonWorks Terminology,” 2014. [Online]. Available: http://www.lonmark.org/technical_resources/terminology_a-c.

[82] ServiceWave 2008, *Towards a service-based internet: first European conference, ServiceWave 2008, Madrid, Spain, December 10-13, 2008: proceedings*. Berlin: New York : Springer, 2008.

[83] R.-T. Lin, Chin-Shun Hsu, Tee Yuen Chun, and Sheng-Tzong Cheng, “OSGi-Based Smart Home Architecture for heterogeneous network,” 2008, pp. 527–532.

[84] J. Bai, H. Xiao, X. Yang, and G. Zhang, “Study on integration technologies of building automation systems based on web services,” 2009, pp. 262–266.

[85] H.-J. Yim, Y.-Y. Hwang, and K.-C. Lee, “A self-organizing two-way DPWS adaptor for adaptive interoperability of multiple heterogeneous services,” 2010, p. 1.

[86] openHAB UG, “openHAB,” 2014. [Online]. Available: http://www.openhab.org/.

[87] The Eclipse Foundation, “eclipse smarthome,” 2014. [Online]. Available: http://eclipse.org/smarthome/.

[88] R. Anderson, *Security engineering*. John Wiley & Sons, 2008.

[89] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-Policy Attribute-Based Encryption,” in *IEEE Symposium on Security and Privacy, 2007. SP ’07*, 2007, pp. 321–334.

[90] J.-H. Hoepman, “Privacy Design Strategies,” in *ICT Systems Security and Privacy Protection*, N. Cuppens-Boulahia, F. Cuppens, S. Jajodia, A. A. E. Kalam, and T. Sans, Eds. Springer Berlin Heidelberg, 2014, pp. 446–459.

[91] J. Jenkov, “OAuth 2.0 Tutorial,” *jenkov.com*, Apr-2014. [Online]. Available: http://tutorials.jenkov.com/oauth2/index.html.

[92] “OAuth,” *Wikipedia*, Apr-2014. [Online]. Available: http://en.wikipedia.org/wiki/OAuth.

[93] A. Parecki, “OAuth Community,” *OAuth Community Site*, Apr-2014. [Online]. Available: http://oauth.net/.

[94] E. Hammer-Lahav, “Introduction to OAuth,” *OAuth Community Site*, Sep-2007. [Online]. Available: http://oauth.net/about/.

[95] B. Shaffer, R. Weaver, and L. Pelham, “OAuth2 in 8 Steps Tutorial,” *KnpUniversity*, Apr-2014. [Online]. Available: http://knpuniversity.com/screencast/oauth/intro.

[96] R. Boyd, *Getting Started with OAuth 2.0*. O’Reilly Media, 2012.

[97] A. Parecki, *OAuth 2.0: The Definitive Guide*. O’Reilly Media, 2014.

[98] G. Brail and S. Ramji, *OAuth - The Big Picture*, Kindle Edition. apigee, 2012.

[99] S. Haiges, *OAuth 2.0 - Client & Server*, Kindle Edition. entwickler.press, 2013.

[100] M. Spasovski, *OAuth 2.0 Identity and Access Management Patterns*, Kindle Edition. Packt Publishing, 2013.

[101] “OpenID,” *Wikipedia*, Apr-2014. [Online]. Available: http://en.wikipedia.org/wiki/OpenID.

[102] “OAuth Core 1.0,” Dec-2007. [Online]. Available: http://oauth.net/core/1.0/.

[103] D. Atkins and H. Tschofenig, “OAuth Status Pages,” *Internet Engineering Taskforce*, Apr-2014. [Online]. Available: http://tools.ietf.org/wg/oauth/.

[104] E. Hammer-Lahav, “RFC 5849: The OAuth 1.0 Protocol,” *Internet Engineering Taskforce*, Apr-2010. [Online]. Available: http://tools.ietf.org/html/rfc5849.

[105] D. Hardt, “RFC 6749: The OAuth 2.0 Authorization Framework,” *Internet Engineering Taskforce*, Oct-2012. [Online]. Available: http://tools.ietf.org/html/rfc6749.

[106] D. Hardt and M. B. Jones, “RFC 6750: The OAuth 2.0 Authorization Framework: Bearer Token Usage,” *Internet Engineering Taskforce*, Oct-2012. [Online]. Available: http://tools.ietf.org/html/rfc6750.

[107] T. Lodderstedt, M. McGloin, and P. Hunt, “RFC 6819: OAuth 2.0 Threat Model and Security Considerations,” *Internet Enginerring Task Force*, Jan-2013. [Online]. Available: http://tools.ietf.org/html/rfc6819.

[108] E. Hammer-Lahav, “OAuth Bearer Tokens are a Terrible Idea,” *hueinverse*, Sep-2010. [Online]. Available: http://hueniverse.com/2010/09/oauth-bearer-tokens-are-a-terrible-idea/.

[109] E. Hammer-Lahav, “OAuth 2.0 and the Road to Hell,” *hueniverse*, Jul-2012. [Online]. Available: http://hueniverse.com/2012/07/oauth-2-0-and-the-road-to-hell/.

[110] “General considerations for the Web of Things,” *W3C*, Apr-2014. [Online]. Available: http://www.w3.org/community/wot/wiki/General_considerations_for_the_Web_of_Things.

[111] P. Fremantle and P. Madsen, “Securing the Internet of Things,” *slideshare*, Mar-2014. [Online]. Available: http://de.slideshare.net/pizak/securing-the-internet-of-things.

[112] P. Fremantle, “Federating Access to IoT using OAuth,” *FOSDEM 2014*, Feb-2014. [Online]. Available: https://fosdem.org/2014/schedule/event/deviot05/.

[113] “Proposed Addendum am to Standard 135-2012, BACnet®- A Data Communication Protocol for Building Automation and Control Networks,” *ASHRAE*, Apr-2014. [Online]. Available: https://osr.ashrae.org/sitepages/showdoc2.aspx/ListName/Public%20Review%20Draft%20Standards/ItemID/1102/IsAttachment/N/Add-135-2012am-PPR1-draft-26_chair_approved.pdf.

[114] K. Echtle, *Fehlertoleranzverfahren*. Berlin; Heidelberg; New York; London; Paris; Tokyo; Hong Kong; Barcelona: Springer, 1990.

[115] A. Immonen and E. Niemelä, “Survey of reliability and availability prediction methods from the viewpoint of software architecture,” *Software & Systems Modeling*, vol. 7, no. 1, pp. 49–65, Nov. 2007.

[116] U. Biegert, “Ganzheitliche modellbasierte Sicherheitsanalyse,” *atp - Automatisierungstechnische Praxis 45*, 2003.

[117] I. Rolle, “Funktionale Sicherheit programmierbarer elektronischer Systeme,” in *Funktionale Sicherheit*, W. A. Halang, Ed. Berlin Heidelberg: Springer-Verlag, 2013, pp. 1–6.

[118] A. Habermann and S. Burton, “Safety systematisch verankern,” *Automobil Elektronik*, Feb-2012.

[119] O. C. Winne and M. Hafner, “Leitfaden für die Norm IEC 61508,” *Elektronik Praxis*, 05-Apr-2009.

[120] W. Kastner and T. Novak, “Functional safety in building automation,” 2009, pp. 1–8.

[121] J. Sauler, S. Kriso, and M. Hafner, “ISO 26262 - Die zukünftige Norm zur funktionalen Sicherheit von Straßenfahrzeugen,” *Elektronik Praxis*, 31-Aug-2011.

[122] T. Novak, A. Treytl, and P. Palensky, “Common approach to functional safety and system security in building automation and control systems,” 2007, pp. 1141–1148.

[123] “ISO 12100:2010 Safety of machinery -- General principles for design -- Risk assessment and risk reduction.” [Online]. Available: http://www.iso.org/iso/catalogue_detail?csnumber=38479.

[124] P. F. Elzer, “Fehlertolerante verteilte Systeme aus Standardkomponenten,” in *Funktionale Sicherheit*, W. A. Halang, Ed. Berlin Heidelberg: Springer-Verlag, 2013, pp. 69–76.

[125] T. Novak and A. Gerstinger, “Safety- and Security-Critical Services in Building Automation and Control Systems,” *IEEE Transactions on Industrial Electronics*, vol. 57, no. 11, pp. 3614–3621, Nov. 2010.

[126] G. N. Rodrigues, G. Roberts, and W. Emmerich, “Reliability Support for the Model Driven Architecture,” in *Architecting Dependable Systems II*, Springer Berlin Heidelberg, 2004, pp. 79–98.

[127] P. Dongbo, L. Feng, Z. Xuelian, and L. Tao, “Functional safety in building automation and control systems,” in *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, 2008, pp. 467–470.

[128] D. L. Parnas, “On the Criteria to Be Used in Decomposing Systems into Modules,” *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, Dec. 1972.

[129] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, Revised. Addison Wesley Pub Co Inc, 2011.

[130] Object Management Group, “CORBA Component Model 4.0 Specification,” Object Management Group, Specification Version 4.0, Apr. 2006.

[131] D. Thompson, C. Exton, L. Garrett, A. S. M. Sajeev, and D. Watkins, *Distributed Component Object Model (DCOM)*. 1997.

[132] The OSGi Alliance, *OSGi Core Release 5*. The OSGi Alliance, 2012.

[133] D. Marples and P. Kriens, “The Open Services Gateway Initiative: an introductory overview,” *Communications Magazine, IEEE*, vol. 39, no. 12, pp. 110–114, Dec. 2001.

[134] K. Hofrichter, “The residential gateway as service platform,” in *Consumer Electronics, 2001. ICCE. International Conference on*, 2001, pp. 304–305.

[135] A. Saad, “Java-based Functionality and Data Management in the automobile—Prototyping at BMW Car IT GmbH,” *JavaSPEKTRUM. SIGS Datacom*, vol. 2, pp. 49–53, 2003.

[136] D. Gruber, B. J. Hargrave, J. McAffer, P. Rapicault, and T. Watson, “The Eclipse 3.0 platform: Adopting OSGi technology,” *IBM Systems Journal*, vol. 44, no. 2, pp. 289–299, 2005.

[137] A. Hein, M. Eichelberg, O. Nee, A. Schulz, A. Helmer, and M. Lipprandt, “A Service Oriented Platform for Health Services and Ambient Assisted Living,” in *Advanced Information Networking and Applications Workshops, 2009. WAINA ’09. International Conference on*, 2009, pp. 531–537.

[138] M. Beisiegel, D. Booz, A. Colyer, H. Hildebrand, J. Marino, and K. Tam, *SCA Service Component Architecture*. 2007.

[139] J. S. Rellermeyer, G. Alonso, and T. Roscoe, “R-OSGi: Distributed Applications Through Software Modularization,” in *Proceedings of the ACM/IFIP/USENIX 2007 International Conference on Middleware*, New York, NY, USA, 2007, pp. 1–20.

[140] E. Bruneton and others, “ASM: A Code Manipulation Tool to Implement Adaptable Systems,” in *Adaptable and Extensible Component Systems*, 2002.

[141] *Nyota Online Documentation*. 2007.

[142] E. Newcomer and others, *RFP 119 - Distributed OSGi*. OSGi Alliance, 2008.

[143] C. Apache, “An Open Source Service Framework,” *See: http://cxf. apache. org*, p. 111, 2009.

[144] S. Lewis and C. ANISCZYK, “Eclipse Communication Framework,” *Internet Homepage, April*, 2005.

[145] A. Ibrahim, Y. Jiao, E. Tilevich, and W. Cook, “Remote Batch Invocation for Compositional Object Services,” in *ECOOP 2009 – Object-Oriented Programming*, vol. 5653, S. Drossopoulou, Ed. Springer Berlin Heidelberg, 2009, pp. 595–617.

[146] The OSGi Alliance, *OSGi Compendium Release 5*. The OSGi Alliance, 2012.

[147] A. Bottaro, E. Simon, S. Seyvoz, and A. Gerodolle, “Dynamic Web Services on a Home Service Platform,” in *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, 2008, pp. 378–385.

[148] S.-T. Cheng, C.-H. Wang, and G.-J. Horng, “OSGi-based smart home architecture for heterogeneous network,” *Expert Systems with Applications*, vol. 39, no. 16, pp. 12418–12429, 2012.

[149] T. Gorath, M. Eichelberg, A. Hein, E. Zeeb, and D. Timmermann, “Technologieunabhängige geräteintegration des osami-projekts,” *Ambient Assisted Living-AAL*, 2010.

[150] M. Jung, J. Weidinger, C. Reinisch, W. Kastner, C. Crettaz, A. Olivieri, and Y. Bocchi, “A Transparent IPv6 Multi-protocol Gateway to Integrate Building Automation Systems in the Internet of Things,” in *Green Computing and Communications (GreenCom), 2012 IEEE International Conference on*, 2012, pp. 225–233.

[151] M. Jung, C. Reinisch, and W. Kastner, “Integrating Building Automation Systems and IPv6 in the Internet of Things,” 2012, pp. 683–688.

[152] S. de Deugd, R. Carroll, K. E. Kelly, B. Millett, and J. Ricker, “SODA: Service Oriented Device Architecture,” *Pervasive Computing, IEEE*, vol. 5, no. 3, pp. 94–96, Jul. 2006.

[153] L. Richardson and S. Ruby, *RESTful Web Services*. O’Reilly, 2007.

[154] R. T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” University of California Irvine, 2000.

[155] “Representational state transfer,” *Wikipedia (English)*. 23-Apr-2014.

[156] J. Schneider, T. Kamiya, D. Peintner, and R. Kyusakov, “Efficient XML Interchange (EXI) Format 1.0 (Second Edition),” 11-Feb-2014. [Online]. Available: http://www.w3.org/TR/2014/REC-exi-20140211/. [Accessed: 24-Apr-2014].

[157] D. A. Huffman, “A Method for the Construction of Minimum-Redundancy Codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[158] P. Deutsch, “DEFLATE Compressed Data Format Specification version 1.3,” May-1996. [Online]. Available: http://www.rfc-editor.org/rfc/rfc1951.txt. [Accessed: 25-Apr-2014].

[159] G. Moritz, “Web Services in stark ressourcenlimitierten Umgebungen,” Dissertation, University of Rostock, Rostock, 2012.

[160] J. Cowan and R. Tobin, “XML Information Set (Second Edition),” 04-Feb-2004. [Online]. Available: http://www.w3.org/TR/xml-infoset/.

[161] S. Käbisch, D. Peintner, J. Heuer, and H. Kosch, “Efficient and Flexible XML-based Data-Exchange in Microcontroller-based Sensor Actor Networks,” in *SOCNE 2010*, 2010.

[162] “Efficient XML Interchange (EXI) Profile for limiting usage of dynamic memory.” [Online]. Available: http://www.w3.org/TR/2014/PR-exi-profile-20140506/. [Accessed: 03-Jun-2014].

[163] “EXIficient - open source implementation of the W3C Efficient XML Interchange (EXI) format.” [Online]. Available: http://exificient.sourceforge.net/. [Accessed: 05-May-2014].

[164] “ws4d-uexi - EXI parser for highly resource constrained devices - Google Project Hosting.” [Online]. Available: https://code.google.com/p/ws4d-uexi/. [Accessed: 05-May-2014].

[165] “EXIP.” [Online]. Available: http://exip.sourceforge.net/. [Accessed: 05-May-2014].

[166] V. Altmann, J. Skodzik, P. Danielis, F. Golatowski, and D. Timmermann, “Real-Time Capable Hardware-based Parser for Efficient XML Interchange,” To be published.

[167] “Welcome to OpenEXI !” [Online]. Available: http://openexi.sourceforge.net/. [Accessed: 05-May-2014].

[168] Z. Shelby, K. Hartke, and C. Bormann, “The Constrained Application Protocol (CoAP).” [Online]. Available: https://tools.ietf.org/html/rfc7252. [Accessed: 24-Jun-2016].

[169] S. Bandyopadhyay and A. Bhattacharyya, “Lightweight Internet protocols for web enablement of sensors using constrained gateway devices,” in *2013 International Conference on Computing, Networking and Communications (ICNC)*, 2013, pp. 334–340.

[170] F. Gramegna, S. Ieva, G. Loseto, and A. Pinto, “Semantic-enhanced resource discovery for CoAP-based sensor networks,” in *2013 5th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, 2013, pp. 233–238.

[171] H. A. Khattak, M. Ruta, and E. Di Sciascio, “CoAP-based healthcare sensor networks: A survey,” in *2014 11th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, 2014, pp. 499–503.

[172] C. Bormann, A. P. Castellani, and Z. Shelby, “CoAP: An Application Protocol for Billions of Tiny Internet Nodes,” *IEEE Internet Computing*, vol. 16, no. 2, pp. 62–67, Mar. 2012.

[173] A. P. Castellani, M. Gheda, N. Bui, M. Rossi, and M. Zorzi, “Web Services for the Internet of Things through CoAP and EXI,” in *2011 IEEE International Conference on Communications Workshops (ICC)*, 2011, pp. 1–6.

[174] “draft-ietf-core-block-20.” [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-block/. [Accessed: 24-Jun-2016].

[175] “draft-ietf-core-http-mapping-11.” [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-http-mapping/. [Accessed: 24-Jun-2016].

[176] “draft-ietf-core-resource-directory-07.” [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-resource-directory/. [Accessed: 24-Jun-2016].

[177] “draft-ietf-core-links-json-05.” [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-core-links-json/. [Accessed: 24-Jun-2016].

[178] “draft-tschofenig-core-coap-tcp-tls-05.” [Online]. Available: https://datatracker.ietf.org/doc/draft-tschofenig-core-coap-tcp-tls/. [Accessed: 24-Jun-2016].

[179] “draft-jennings-senml-10.” [Online]. Available: https://datatracker.ietf.org/doc/draft-jennings-senml/. [Accessed: 24-Jun-2016].

[180] “draft-vanderstok-core-etch-00.” [Online]. Available: https://datatracker.ietf.org/doc/draft-vanderstok-core-etch/. [Accessed: 24-Jun-2016].

[181] “Copper (Cu) CoAP user-agent - JavaScript CoAP Implementation.” [Online]. Available: http://people.inf.ethz.ch/mkovatsc/copper.php. [Accessed: 22-May-2014].

[182] “Erbium (Er) REST Engine - C CoAP Implementation.” [Online]. Available: http://people.inf.ethz.ch/mkovatsc/erbium.php. [Accessed: 22-May-2014].

[183] “Californium (Cf) CoAP framework - Java CoAP Implementation.” [Online]. Available: http://people.inf.ethz.ch/mkovatsc/californium.php. [Accessed: 22-May-2014].

[184] “Actinium (Ac) App-server for Californium - CoAP Web Scripting.” [Online]. Available: http://people.inf.ethz.ch/mkovatsc/actinium.php. [Accessed: 22-May-2014].

[185] “jcoap - jCoAP is a Java Library implementing the Constrained Application Protocol (CoAP).” [Online]. Available: https://code.google.com/p/jcoap/. [Accessed: 29-May-2015].

[186] “Olaf Bergmann.” [Online]. Available: http://www.informatik.uni-bremen.de/~bergmann/. [Accessed: 20-May-2014].

[187] “CoAP.NET,” *SourceForge*. [Online]. Available: http://sourceforge.net/projects/coapnet/. [Accessed: 20-May-2014].

[188] ASHRAE, *Standard 135-2012 - BACnet® - A Data Communication Protocol for Building Automation and Control Networks (ANSI Approved)*. ANSI/ASHRAE, 2012.

[189] M. Newman, *BACnet: The Global Standard for Building Automation and Control Networks*. New York, USA: Momentum Press, 2013.

[190] F. Brian, “oBIX 1.0,” 05-Dec-2006. .

[191] C. Gemmill, “oBIX 1.1 (Committee Specification Draft 02),” 19-Dec-2013. [Online]. Available: http://docs.oasis-open.org/obix/obix/v1.1/csprd02/obix-v1.1-csprd02.html. [Accessed: 25-Apr-2014].

[192] M. Jung, “Encodings for OBIX: Common Encodings Version 1.0 (Committee Specification Draft 02),” 19-Dec-2013. [Online]. Available: http://docs.oasis-open.org/obix/obix-encodings/v1.0/csprd02/obix-encodings-v1.0-csprd02.html.

[193] C. Gemmill and M. Jung, “Bindings for OBIX: REST Bindings Version 1.0 (Committee Specification Draft 02),” 19-Dec-2013. [Online]. Available: http://docs.oasis-

open.org/obix/obix-rest/v1.0/csprd02/obix-rest-v1.0-csprd02.html. [Accessed: 25-Apr-2014].

[194] M. Jung, “Bindings for OBIX: SOAP Bindings Version 1.0 (Committee Specification Draft 02),” 19-Dec-2013. [Online]. Available: http://docs.oasis-open.org/obix/obix-soap/v1.0/csprd02/obix-soap-v1.0-csprd02.html. [Accessed: 28-Apr-2014].

[195] M. Hub, “Bindings for OBIX: WebSocket Bindings Version 1.0 (Committee Specification Draft 01),” 19-Dec-2013. [Online]. Available: http://docs.oasis-open.org/obix/obix-websocket/v1.0/csprd01/obix-websocket-v1.0-csprd01.html. [Accessed: 28-Apr-2014].

[196] D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON),” Jul-2006. [Online]. Available: http://tools.ietf.org/html/rfc4627. [Accessed: 25-Apr-2014].

[197] T. Considine, “Work Plan for oBIX 2.0,” 26-Mar-2013. [Online]. Available: http://www.newdaedalus.com/articles/2013/3/26/work-plan-for-obix-20.html.

[198] M. Weiser, “The Computer for the 21st Century,” *Scientific American*, Sep. 1991.

[199] K. Lyytinen and Y. Yoo, “Issues and Challenges in Ubiquitous Computing,” *Communications of the ACM*, vol. 45, no. 12, Dec. 2002.

[200] B. Bergvall-Kåareborn and D. Howcroft, “The Apple business model: Crowdsourcing mobile applications,” *Accounting Forum*, vol. 37, no. 4, pp. 280–289, Dec. 2013.

[201] G. D. Abowd, “What next, ubicomp?: celebrating an intellectual disappearing act,” in *UbiComp ’12: Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, 2012.

[202] T. Kindberg, B. Schiele, A. Schmidt, and K. Rehman, “What makes for good application- led research in ubiquitous computing?,” *Pervasive 2005 Workshop*, May 2005.

[203] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, “Middleware for distributed context-aware systems,” in *OTM’05: Proceedings of the 2005 Confederated international conference on On the Move to Meaningful Internet Systems*, 2005.

[204] M. Knappmeyer, S. L. Kiani, E. S. Reetz, N. Baker, and R. Tonjes, “Survey of Context Provisioning Middleware,” *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 3, pp. 1492–1519, 2013.

[205] V. Raychoudhury, J. Cao, M. Kumar, and D. Zhang, “Middleware for pervasive computing: A survey,” *Pervasive and Mobile Computing*, Sep. 2012.

[206] D. J. Cook and S. K. Das, “Pervasive computing at scale: Transforming the state of the art,” *Pervasive and Mobile Computing*, vol. 8, no. 1, pp. 22–35, Feb. 2012.

[207] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt, “Gaia,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 6, no. 4, pp. 65–67, Oct. 2002.

[208] J. P. Sousa and D. Garlan, “The Aura Software Architecture: an Infrastructure for Ubiquitous Computing,” Aug. 2003.

[209] C. Dixon, R. Mahajan, S. Agarwal, A. J. B. Brush, B. Lee, S. Saroiu, and P. Bahl, “An operating system for the home,” in *NSDI’12: Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.

[210] C.-F. Sørensen, M. Wu, T. Sivaharan, G. S. Blair, P. Okanda, A. Friday, and H. Duran-Limon, “A context-aware middleware for applications in mobile Ad Hoc environments,” in *MPAC ’04: Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing*, 2004.

[211] J. Taneja, A. Krioukov, S. Dawson-Haggerty, and D. E. Culler, “Enabling Advanced Environmental Conditioning with a Building Application Stack,” *eecs.berkeley.edu*, 2013.

[212] A. K. Dey, D. Salber, M. Futakawa, and G. D. Abowd, “An Architecture to Support Context-Aware Applications,” *UIST’99*, 1999.

[213] T. Gu, H. K. Pung, and D. Q. Zhang, “A service-oriented middleware for building context-aware services,” *Journal of Network and Computer Applications*, vol. 28, no. 1, pp. 1–18, Jan. 2005.

[214] J. E. Bardram, “The Java Context Awareness Framework (JCAF) – A Service Infrastructure and Programming Framework for Context-Aware Applications,” in *Pervasive Computing*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 98–115.

[215] K. Henricksen and J. Indulska, “A software engineering framework for context-aware pervasive computing,” in *Pervasive Computing and Communications, 2004. PerCom 2004. Proceedings of the Second IEEE Annual Conference on*, 2004, pp. 77–86.

[216] B. Guo, D. Zhang, and M. Imai, “Toward a cooperative programming framework for context-aware applications,” *Personal and ubiquitous computing*, vol. 15, no. 3, pp. 221–233, Aug. 2010.

[217] R. Grimm, J. Davis, E. Lemar, and A. Macbeth, “System support for pervasive applications,” *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 421–468, 2004.

[218] M.-O. Pahl and G. Carle, “Crowdsourced Context-Modeling as Key to Future Smart Spaces,” in *Network Operations and Management Symposium 2014 (NOMS 2014)*, 2014.

[219] A. K. Dey, “Understanding and Using Context,” *Personal and ubiquitous computing*, vol. 5, no. 1, pp. 4–7, 2001.

[220] U. Aßmann, S. Zschaler, and G. Wagner, “Ontologies, Meta-models, and the Model-Driven Paradigm,” in *Ontologies for Software Engineering and Technology*, Springer, 2006, pp. 249–273.

[221] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni, “A survey of context modelling and reasoning techniques,” *Pervasive and Mobile Computing*, vol. 6, no. 2, pp. 161–180, Apr. 2010.

[222] “W3C Data Activity,” *The W3C Semantic Web Site*, 2013. [Online]. Available: http://www.w3.org/2013/data/.

[223] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American*, vol. 284, no. 5, pp. 34–43, May 2001.

[224] “ZigBee Building Automation Standard Overview.” [Online]. Available: https://www.zigbee.org/Standards/ZigBeeBuildingAutomation/Overview.aspx. [Accessed: 22-May-2014].

[225] C. Reinisch, W. Granzer, F. Praus, and W. Kastner, “Integration of heterogeneous building automation systems using ontologies,” Orlando, FL, USA, 2008, pp. 2736–2741.

[226] S. Runde, A. Fay, and W.-O. Wutzke, “Knowledge-based Requirement-Engineering of building automation systems by means of Semantic Web technologies,” in *Knowledge-based Requirement-Engineering of building automation systems by means of Semantic Web technologies*, Cardiff, Wales, 2009, pp. 267–272.

[227] “Automated Design for Building Automation (AUTEG),” 2009. [Online]. Available: http://iis807.inf.tu-dresden.de/~auteg/en/index.html.

[228] “Automated Installation of Wireless Systems for Building Automation (AUDRAGA),” 2011. [Online]. Available: http://iis807.inf.tu-dresden.de/~auteg/en/index.html.

[229] M. Ruta, F. Scioscia, E. D. Sciascio, and G. Loseto, “Semantic-Based Enhancement of ISO/IEC 14543-3 EIB/KNX Standard for Building Automation,” *IEEE Trans. Industrial Informatics*, pp. 731–739, 2011.

[230] G. Klyne and J. J. Carroll, Eds., *Resource Description Framework (RDF): Concepts and Abstract Syntax*. W3C Recommendation, 2004.

[231] “RDF 1.1 XML Syntax,” *W3C*, 25-Feb-2014. [Online]. Available: http://www.w3.org/TR/rdf-syntax-grammar/.

[232] “Notation3 (N3): A readable RDF syntax,” *W3C*, 28-Mar-2011. [Online]. Available: http://www.w3.org/TeamSubmission/n3/.

[233] “Turtle - Terse RDF Triple Language,” *W3C*, 28-Mar-2011. [Online]. Available: http://www.w3.org/TeamSubmission/turtle/.

[234] “The JSON Data Interchange Format.” ecma International, Oct-2013.

[235] D. Brickley, R. V. Guha, and B. McBride, Eds., *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, 2004.

[236] P. Hayes, “RDF Semantics,” World Wide Web Consortium, Recommendation, Feb. 2004.

[237] D. L. McGuinness and F. van Harmelen, “OWL Web Ontology Language Overview,” W3C, REC-owl-features-20040210, 2004.

[238] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, “OWL 2 Web Ontology Language Primer,” World Wide Web Consortium, W3C Recommendation, Oct. 2009.

[239] “OWL Lite,” *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s2.1.

[240] “OWL DL & OWL Full,” *W3C*, 2004. [Online]. Available: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s4.

[241] “OWL 2 EL,” *W3C*, 2012. [Online]. Available: http://www.w3.org/TR/owl2-profiles/#OWL_2_EL.

[242] “OWL 2 QL,” *W3C*, 2012. [Online]. Available: http://www.w3.org/TR/owl2-profiles/#OWL_2_QL.

[243] "OWL 2 RL," *W3C*, 2012. [Online]. Available: http://www.w3.org/TR/owl2-profiles/#OWL_2_RL.

[244] "EL++," *W3C*, 2008. [Online]. Available: http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/#ref-ELpp.

[245] W. The W3C SPARQL Working Group, Ed., *SPARQL Query Language for RDF*. 2013.

[246] "SPARQL Implementations," *W3C*, 2014. [Online]. Available: http://www.w3.org/wiki/SparqlImplementations.

[247] "Rule Interchange Format," *W3C*, 2013. [Online]. Available: http://www.w3.org/standards/techs/rif#w3c_all.

[248] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, and M. Dean, "Semantic Web Rule Language," *W3C*, 2004. .

[249] "Closed World Assumption," *Wikipedia*, 2014. [Online]. Available: http://en.wikipedia.org/wiki/Closed_world_assumption.

[250] "Unrealized Semantic Web technologies," *Wikipedia*, 2013. .

[251] P. Hitzler, M. Krötzsch, and S. Rudolph, *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.

[252] "RDFS/OWL Reasoning Capabilities," *RDFS/OWL Reasoning Capabilities*. [Online]. Available: http://www.ksl.stanford.edu/software/jtp/doc/owl-reasoning.html.

[253] D. Anicic, "Event Processing and Stream Reasoning with ETALIS," Dissertation/Ph.D. thesis, The Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany, 2011.

[254] "RDF Stream Processing," *RDF Stream Processing Community Group*, 2013. [Online]. Available: http://www.w3.org/community/rsp/.

[255] D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus, "An execution environment for C-SPARQL queries," in *Proceedings of the 13th International Conference on Extending Database Technology*, 2010, pp. 441–452.

[256] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth, "A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data," in *Proceedings of the 10th International Conference on The Semantic Web - Volume Part I*, Berlin, Heidelberg, 2011, pp. 370–388.

[257] "Linked Data - Connect Distributed Data across the Web," 2007. [Online]. Available: http://linkeddata.org/.

[258] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, "EP-SPARQL: a unified language for event processing and stream reasoning," in *Proceedings of the 20th International Conference on World Wide Web*, New York, NY, USA, 2011, pp. 635–644.

[259] D. Martin, M. Burstein, J. Hobbs, O. Lassila, and D. McDermott, "OWL-S: semantic markup for Web services," *W3C*, 2004. .

[260] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web Services Description Language (WSDL) 1.1," *Web Services Description Language (WSDL) 1.1*, 2001. [Online]. Available: http://www.w3.org/TR/wsdl.

[261] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, “Web Services Description Language (WSDL) Version 2.0,” *W3C*, 2007. [Online]. Available: http://www.w3.org/TR/wsdl20/.

[262] J. Farrell and H. Lausen, “Semantic Annotations for WSDL and XML Schema,” *W3C*, 2007. [Online]. Available: http://www.w3.org/TR/sawsdl/.

[263] R. Verborgh, T. Steiner, D. Van Deursen, J. De Roo, R. Van de Walle, and J. Gabarró Vallés, “Capturing the functionality of Web services with functional descriptions,” *Multimedia Tools and Applications*, vol. 64, no. 2, pp. 365–387, May 2013.

[264] D. Gessler, G. Schiltz, G. May, S. Avraham, C. Town, D. Grant, and R. Nelson, “SSWAP: A Simple Semantic Web Architecture and Protocol for semantic web services,” *BMC Bioinformatics*, vol. 10, no. 1, p. 309, 2009.

[265] “SSN-XG,” *W3C*, 2011. [Online]. Available: http://www.w3.org/2005/Incubator/ssn/.

[266] “SWE,” *OGC*. [Online]. Available: http://www.opengeospatial.org/ogc/markets-technologies/swe.

[267] F. Probst and M. Lutz, “Giving Meaning to GI Web Service Descriptions.,” in *WSMAI*, 2004, pp. 23–35.

[268] “Protégé,” *Protege Stanford University*, 2014. [Online]. Available: http://protege.stanford.edu/.

[269] “Smart Appliances Project.” [Online]. Available: https://sites.google.com/site/smartappliancesproject/home.

[270] “QUDT - Quantities, Units, Dimensions and Types,” 15-Aug-2016. [Online]. Available: http://qudt.org/.

[271] “NASA,” *NASA*. [Online]. Available: http://www.nasa.gov/index.html.

[272] Pieter Pauwels, “ifcOWL ontology file added for IFC4_ADD1,” 12-Dec-2014. [Online]. Available: https://www.w3.org/community/lbd/2014/12/12/ifcowl-ontology-file-added-for-ifc4_add1/.

[273] “buildingSMART,” *International home of openBIM*. [Online]. Available: http://buildingsmart.org/.

[274] “Linked Data Working Group,” *buildingSMART*. [Online]. Available: http://buildingsmart.org/standards/standards-organization/groups/linked-data-working-group/.

[275] “Linked Building Data Community Group.” .

[276] “buildingSMART Data Dictionary Browser.” [Online]. Available: http://bsdd.buildingsmart.org/.

[277] D. Tomaszuk and H. Rybinski, “OAuth+UAO: A Distributed Identification Mechanism for Triplestores,” in *ICCCI (1)*, 2011, vol. 6922, pp. 275–284.

[278] M. Bourimi, S. Scerri, M. Planaguma, M. Heupel, F. Karatas, and P. Schwarte, “A two-level approach to ontology-based access control in pervasive personal servers,” 2011.

[279] “Web of Things at W3C.” [Online]. Available: https://www.w3.org/WoT/.

[280] T. Stahl, *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. Heidelberg: Dpunkt-Verl., 2007.

[281] K. Czarnecki, *Generative programming: methods, tools, and applications*. Boston: Addison Wesley, 2000.

[282] J. Greenfield, *Software factories: assembling applications with patterns, models, frameworks, and tools*. Indianapolis, IN: Wiley Pub, 2004.

[283] "GME: Generic Modeling Environment | Institute for Software Integrated Systems," *GME: Generic Modeling Environment | Institute for Software Integrated Systems*. [Online]. Available: http://www.isis.vanderbilt.edu/projects/gme. [Accessed: 26-Apr-2014].

[284] S. Dmitriev, "Language Oriented Programming: The Next Programming Paradigm." [Online]. Available: http://www.jetbrains.com/mps/docs/Language_Oriented_Programming.pdf.

[285] D. Waddington and P. Lardieri, "Model-Centric Software Development," *Published by the IEEE Computer Society*, pp. 28–29, 2006.

[286] "OMG Model Driven Architecture." [Online]. Available: http://www.omg.org/mda/. [Accessed: 26-Apr-2014].

[287] "OMG's MetaObject Facility (MOF) Home Page." [Online]. Available: http://www.omg.org/mof/. [Accessed: 26-Apr-2014].

[288] "MetaGME 2000: Metamodeling Environment." [Online]. Available: http://w3.isis.vanderbilt.edu/projects/gme/meta.html. [Accessed: 26-Apr-2014].

[289] D. Steinberg, Ed., *EMF: Eclipse Modeling Framework*, 2nd ed., Rev. and updated. Upper Saddle River, NJ: Addison-Wesley, 2009.

[290] "OMG Meta Object Facility (MOF) Core Specification Version 2.4.1." OMG, Jun-2013.

[291] "XML Schema Part 0: Primer Second Edition." [Online]. Available: http://www.w3.org/TR/xmlschema-0/. [Accessed: 26-Apr-2014].

[292] M. Fowler, *Domain-specific languages*. Upper Saddle River, NJ: Addison-Wesley, 2011.

[293] S. Benz and M. Völter, *DSL engineering: designing, implementing and using domain-specific languages*. [S.l.]: CreateSpace Independent Publishing Platform, 2013.

[294] "Language Workbench Challenge | Comparing Tools of the Trade." [Online]. Available: http://www.languageworkbenches.net/. [Accessed: 28-Apr-2014].

[295] S. Erdweg, T. Storm, M. Völter, M. Boersma, R. Bosman, W. Cook, A. Gerritsen, A. Hulshout, S. Kelly, A. Loh, G. P. Konat, P. Molina, M. Palatnik, R. Pohjonen, E. Schindler, K. Schindler, R. Solmi, V. Vergu, E. Visser, K. Vlist, G. Wachsmuth, and J. Woning, "The State of the Art in Language Workbenches," in *Software Language Engineering*, vol. 8225, M. Erwig, R. Paige, and E. Wyk, Eds. Springer International Publishing, 2013, pp. 197–217.

[296] S. Cook, Ed., *Domain-specific development with Visual Studio DSL tools*. Upper Saddle River, NJ: Addison-Wesley, 2007.

[297] “Overview of our Technology - Intentional Software.” [Online]. Available: http://www.intentsoft.com/intentional-technology/. [Accessed: 28-Apr-2014].

[298] “Gentleware - model to business: poseidon for dsls.” [Online]. Available: http://www.gentleware.com/poseidon-for-dsls.html. [Accessed: 28-Apr-2014].

[299] “MetaCase - Domain-Specific Modeling with MetaEdit+.” [Online]. Available: http://www.metacase.com/de/. [Accessed: 28-Apr-2014].

[300] “MagicDraw UML Profiling And DSL.” [Online]. Available: http://www.magicdraw.com/files/manuals/beta/MagicDraw%20UMLProfiling&DSL%20UserGuide.pdf.

[301] “JetBrains :: Meta Programming System — Language Oriented Programming environment and DSL creation tool.” [Online]. Available: http://www.jetbrains.com/mps/. [Accessed: 28-Apr-2014].

[302] “Near field communication,” *Wikipedia, the free encyclopedia*. 15-Jun-2014.

[303] “Near Field Communication Technology Standards.” [Online]. Available: http://www.nearfieldcommunication.org/technology.html. [Accessed: 16-Jun-2014].

[304] “NFC Forum Technical Specifications | NFC Forum.” [Online]. Available: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/nfc-forum-technical-specifications/. [Accessed: 21-May-2014].

[305] “NFC Forum Specification Architecture | NFC Forum.” [Online]. Available: http://nfc-forum.org/our-work/specifications-and-application-documents/specifications/. [Accessed: 12-Jun-2014].

[306] “NFC Standards, Products and Specifications.” [Online]. Available: http://open-nfc.org/documents/PRE_NFC_0804-250%20NFC%20Standards.pdf. [Accessed: 16-Jun-2014].

[307] “ISO/IEC 18092:2004.” [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=38578. [Accessed: 16-Jun-2014].

[308] “ISO/IEC 14443.” [Online]. Available: http://www.atmel.com/images/doc2056.pdf. [Accessed: 16-Jun-2014].

[309] “ZigBee,” *Wikipedia, the free encyclopedia*. 25-May-2014.

[310] “ZigBee Building Automation Features.” [Online]. Available: https://www.zigbee.org/Standards/ZigBeeBuildingAutomation/Features.aspx. [Accessed: 22-May-2014].

[311] “Wireless Networks.” [Online]. Available: http://www.jennic.com/elearning/zigbee/files/html/module1/module1-2.htm. [Accessed: 29-May-2014].

[312] “IEEE Standard Association - IEEE Get Program.” [Online]. Available: http://standards.ieee.org/getieee802/download/802.15.4d-2009.pdf. [Accessed: 05-Jun-2014].

[313] P. Bhagwat, “Bluetooth: technology for short-range wireless apps,” *IEEE Internet Computing*, vol. 5, no. 3, pp. 96–103, Mai 2001.

[314] “Bluetooth Radio Architecture.” [Online]. Available: https://developer.bluetooth.org/TechnologyOverview/Pages/Radio.aspx.

[315] Bob Violino, “RFID Business Applications,” *RFID Journal*, 16-Jan-2005. [Online]. Available: http://www.rfidjournal.com/articles/view?1334.

[316] S. A. Weis, “RFID (radio frequency identification): Principles and applications,” *System*, vol. 2, no. 3, 2007.

[317] Lim Siong Boon, “NFC (Near Field Communication), RFID,” 03-Mar-2012. [Online]. Available: http://www.siongboon.com/projects/2012-03-03_rfid/index.html.

[318] R. Want, “An Introduction to RFID Technology,” *IEEE Pervasive Computing*, vol. 5, no. 1, pp. 25–33, 2006.

[319] “How Do RFID Systems Work,” *Impinj*. [Online]. Available: http://www.impinj.com/resources/about-rfid/how-do-rfid-systems-work/.

[320] “M2M,” *Wikipedia, la enciclopedia libre*. 31-May-2014.

[321] “ETSI - Machine to Machine comunications,” *ETSI - World Class Standards*. [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/m2m. [Accessed: 18-Jun-2014].

[322] Jack Brown, “Machine-2-Machine Internet of Things Real World Internet,” 24-Aug-2011. [Online]. Available: http://www.slideshare.net/jackebrown/machine-2-machine-internet-of-things-real-world-internet-8996257. [Accessed: 18-Jun-2014].

[323] “INSPIRE > WELCOME TO INSPIRE.” [Online]. Available: http://inspire.ec.europa.eu/. [Accessed: 18-Jun-2014].

[324] “GIS Cloud :: It’s about the Apps, not the Maps!” [Online]. Available: http://www.giscloud.com/. [Accessed: 18-Jun-2014].

[325] “OpenGeo Suite Cloud Edition - Boundless.” [Online]. Available: http://boundlessgeo.com/2010/08/opengeo-suite-cloud-edition/. [Accessed: 18-Jun-2014].

[326] “Create amazing maps with your data — CartoDB.” [Online]. Available: http://cartodb.com/. [Accessed: 18-Jun-2014].

[327] “En la Nube: Cloud GIS | Esri España.” [Online]. Available: http://www.esri.es/es/productos/arcgis/arcgis/en-la-nube--cloud-gis/. [Accessed: 18-Jun-2014].

[328] “IkiMap - Wikipedia, la enciclopedia libre.” [Online]. Available: http://es.wikipedia.org/wiki/IkiMap. [Accessed: 18-Jun-2014].

[329] “Tour | Mapbox.” [Online]. Available: https://www.mapbox.com/tour/. [Accessed: 18-Jun-2014].

[330] “OpenLayers: Home.” [Online]. Available: http://www.openlayers.org/. [Accessed: 18-Jun-2014].

[331] “Leaflet - a JavaScript library for mobile-friendly maps.” [Online]. Available: http://leafletjs.com/. [Accessed: 18-Jun-2014].

[332] “Google Maps JavaScript API v3 — Google Developers.” [Online]. Available: https://developers.google.com/maps/documentation/javascript/. [Accessed: 18-Jun-2014].

[333] “Polymaps.” [Online]. Available: http://polymaps.org/. [Accessed: 18-Jun-2014].

[334] “Modest Maps | Home.” [Online]. Available: http://modestmaps.com/. [Accessed: 18-Jun-2014].

[335] “MapQuery.” [Online]. Available: http://mapquery.org/. [Accessed: 18-Jun-2014].

[336] “jQuery Geo - JavaScript mapping API.” [Online]. Available: http://jquerygeo.com/. [Accessed: 18-Jun-2014].

[337] “Other Software - Boundless.” [Online]. Available: http://boundlessgeo.com/solutions/solutions-software/software/. [Accessed: 18-Jun-2014].

[338] “JavaScript Toolkit for Rich Web Mapping Applications — GeoExt v1.1.” [Online]. Available: http://geoext.org/. [Accessed: 18-Jun-2014].

[339] “Open Geospatial Consortium | OGC(R).” [Online]. Available: http://www.opengeospatial.org/. [Accessed: 18-Jun-2014].

[340] “RFM22B/23B ISM TRANSCEIVER MODULE,” *HopeRF*. [Online]. Available: https://www.sparkfun.com/datasheets/Wireless/General/RFM22B.pdf.

[341] “Arduino Uno,” *Arduino*. [Online]. Available: https://www.arduino.cc/en/Main/ArduinoBoardUno.

[342] “Intel® Galileo Gen 2 Development Board,” *Intel*. [Online]. Available: http://www.intel.com/content/www/us/en/embedded/products/galileo/galileo-overview.html.

[343] “LMT84 Analog Temperature Sensor,” *Texas Instruments*. [Online]. Available: http://www.ti.com/product/LMT84.

[344] “HC Serial Bluetooth Products User Instructional Manual.” [Online]. Available: http://www.tec.reutlingen-university.de/uploads/media/DatenblattHC-05_BT-Modul.pdf.

[345] “HC-05-Bluetooth to Serial Port Module.” [Online]. Available: http://www.thaieasyelec.com/downloads/EFDV390/EFDV390_Datasheet.pdf.

[346] “Keycloak Overview.” [Online]. Available: http://keycloak.github.io/docs/userguide/keycloak-server/html/Overview.html.

[347] S. Thorgersen, “Keycloak: Securing Microservices,” 21-May-2015. [Online]. Available: http://blog.keycloak.org/2015/05/securing-microservices.html.

[348] “AeroGear - Open Source Libraries for Mobile Connectivity.” [Online]. Available: https://www.aerogear.org/.

[349] “OpenID Connect.” [Online]. Available: http://openid.net/connect/.

[350] “Keycloak.” [Online]. Available: http://www.keycloak.org/.