ITEA3

OPTIMUM

OPTimised Industrial IoT and Distributed Control Platform
for Manufacturing and Material Handling

Deliverable 2.1
Evaluation of IoT-platforms and Definition of the IIoT-platform Architecture

Deliverable type:	Document
Deliverable reference number:	ITEA 16043 | D2.1
Related Work Package:	WP2
Due date:	2019-02-28
Actual submission date:	2018-04-02
Responsible organisation:	Thorsis
Editor:	Paolo Varutti
Dissemination level:	Public
Revision:	Final | Version 1.0

Abstract:	This documents provides an overview on the current state of the art on IoT technologies and data management systems. These solutions are analyzed w.r.t. their application to smart manufacturing handling such that a description and overall definition of an Industrial IoT-platform and a Data Management System are derived w.r.t. the project specifications. Basic services as well as required interfaces with the other system components are introduced in this document.
Keywords:	IIoT-platform, Data Management System, Services

Table_head	Name 1 (partner)	Name 2 (partner)	Approval date (1 / 2)
Approval at WP level	Ozer Aydemir	Romulus Cheveresan	2019-03-11 / 2019-03-07
Veto Review by ALL partners			2019-04-02

Editor

Paolo Varutti (THORSIS)

Contributors

Peter Danielis, Fabian Hölzke, Hannes Raddatz, Frank Golatowski (URO)
Matthias Riedl (IFAK)
Daesub Yoon, Yang Koo Lee (ETRI)
Paolo Varutti (THORSIS)
Junwook Lee (HANDYSOFT)
Metin Tekkalmaz (ERSTE)
Ozer Aydemir (DIA)
Alexander Nowak (NXP)
Victor De Gouveia (EZERIS)
Romulus Cheveresan (BEIA)

Executive Summary (Thorsis)

This document presents the current state of the art of IoT technologies by providing a general overview on available exiting solutions as well as common problems related to IoT systems. A summary of enabling technologies and standards for the implementation of IoT-systems is presented in Section 3, where several protocols for the IoT-stack are thoroughly investigated. In particular, these are evaluated w.r.t. their application to industrial automation and smart manufacturing handling. Furthermore, solutions for data storage – data management systems – that can be used in an IoT-environment are analyzed in Section 4. Since security is one the most crucial aspects of modern complex interconnected systems, a detailed analysis on possible risks which might arise in (industrial) IoT-environments is presented in Section 5.

The results of the project requirement analysis show that at high abstraction level (enterprise or application level) standard solutions enabling M2M communication are required. To this intend, the standards oneM2M and OPC-UA are selected as possible solutions for the project. At a lower level, the MQTT protocol was pointed out as good candidate for the communication between industrial devices, although this is not able to provide the required deterministic communication which is at times necessary for industrial applications. The use of MQTT as a middle-layer protocol allows not only the use of the oneM2M standard, but also of OPC-UA, although this latest does not require MQTT for its implementation. In fact, TCP/IP infrastructure suffices in this case.

Therefore, it was concluded that the basic necessary technology for the implementation of an IIoT-platform is the availability of a network system providing TCP/IP (or addressing) functionalities. The analysis of the actual underlying physical communication layer was not in the primary scope of the project. Nevertheless, considering the real-time, or deterministic requirements, that often arise in industrial applications, upcoming technologies such LTE-5G, for wireless communication, and TSN, for Ethernet, were pointed out as optimal solutions for IIoT. Unfortunately, both technologies are still in an early stage of specification and testing, which means not only that their specification will be subject to changes, but also that they will not be available in the near future.

Finally, specifications for the Data Management System w.r.t. the project requirements are presented in Section 8, where possible data storage solutions for the OPTIMUM-project are introduced.

Table of Content

Tables

Figures

1 Introduction [Thorsis]

The OPTIMUM-project focusses on the analysis and development of new concepts and technological ideas for the Industry 4.0, in particular for the engineering, optimization and control of processes related to “Digital Manufacturing and Material Handling”.

Solutions for material handling play an extremely important role in logistic and in the all manufacturing processes. At the current state of the art, though, machines – typical machines that find use in material handling applications are cranes and forklifts – are mostly operated manually, and only support limited semi-autonomous functions, e.g. tandem functions when cranes are used in parallel for the transportation of heavy loads. Human operators must constantly supervise the machines and manually control them in order to guarantee safety and security. Moreover, it is up-to-date difficult to analyze in advance the capacity of a plant, for instance at sale or during design phase. Thus, solutions which can support the customers at this early planning stage are more than welcome.

The main goals of OPTIMUM are to elaborate new solutions on the basis of exiting and soon-to-be-available technologies from the Internet-of-Things (IoT) and Machine-to-Machine (M2M) communication fields in order to make the current manufacturing handling and digital manufacturing significantly smarter. The aims of the project are, in particular, to:

- analyze and realize new decentralized control solutions for complex processes;
- develop new ideas and concepts to adapt exiting IoT-solutions and technologies to real industrial applications;
- investigate innovative solutions to include context information such as “machine position”, such that all machines, human operators and devices involved in the industrial processes reach a certain level of context awareness;
- analyze and develop software solutions for the 3D-visualization and virtualization of complex plants as support to the engineering and sale processes.

In this context, a suitable IoT-platform, which allows information exchange at different enterprise levels as well as an efficient real-time M2M-communication on the same layer is fundamental. The main subject of Work Package 2 (WP2) is the investigation of the state of the art of current technological solutions that can be used for the realization of a suitable IoT-platform for industrial applications.

In this document, particular attention is given to OPC-UA[1], an M2M communication protocol specifically developed for industrial automation, which is currently supported and strongly striven by the VDMA (*Verband Deutscher Maschinen- und Anlagenbau*, the German mechanical engineering industry association) as main key technology for the realization of the Industry 4.0. VDMA sees in OPC-UA not only a perfect standard for M2M-communication but also the perfect efficient solution for all industrial engineering processes.

In the following subsections first problem setup and content of WP2 is presented, then, in order to familiarize the reader with basic concepts concerning IoT-systems, an overall description of the general IoT-protocol-stack is provided.

[1] Open Platform Communication – Unified Architecture

1.1 Problem Setup and Content of WP2

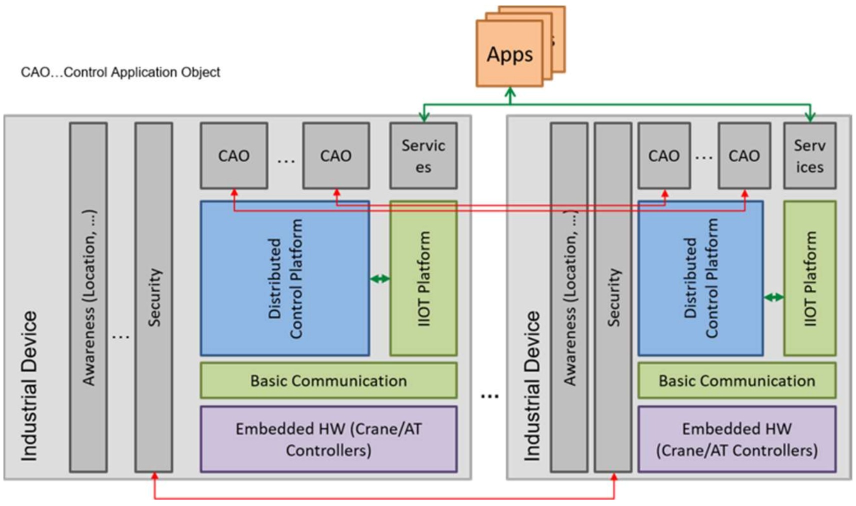

CAO…Control Application Object
Apps
Industrial Device
Awareness (Location, ...)
Security
CAO
CAO
Services
Distributed Control Platform
IIOT Platform
Basic Communication
Embedded HW (Crane/AT Controllers)
Industrial Device
Awareness (Location, ...)
Security
CAO
CAO
Services
Distributed Control Platform
IIOT Platform
Basic Communication
Embedded HW (Crane/AT Controllers)

Figure 1: Overall system architecture and main system components

In Figure 1 the overall system architecture and problem setup for the OPTIMUM-project are presented. The main focus of WP2 is in particular the analysis of possible technologies and IoT-platforms for the realization of an Industrial IoT-platform (IIoT-platform, represented in light green in the above figure), which is a key component for enabling the communication between different subsystems, devices and machines, as well as for the inter-communication at different levels. Additionally, in WP2 also the basic communication technology, i.e. the lower physical communication layer, which allows the actual physical signal and data transfer between system components, is considered. Notice, though, that although this is an important aspect of an IoT-system, the main scope of OPTIMUM is to analyze higher level solutions which enable the realization of complex tasks or applications. To this intent, only negligible attention is paid in WP2 on the actual basic communication. Nevertheless, in Section 2.5.1 a summary of currently available technologies for the basic communication are shortly presented. Additional soon-to-be-ready enabling technologies, with particular attention to the real-time requirements of the distributed control systems, are discussed in Section 3 of D3.1 from WP3.

1.2 IoT-systems and IoT-protocol-stack

The IoT-protocol-stack was developed in order to simplify and facilitate the development of complex networked systems and, thus, strive the pervasive diffusion of the IoT-systems. In Figure 2, the several layers of the IoT-stack are presented and compared to the international standard ISO-OSI-model as well as with the consolidated Internet protocol stack (TCP/IP). As shown in the figure, the main characteristic of this stack is its simplicity. In fact, differently from the ISO-OSI-model, it consists of only four different layers:

- Application Layer
- Application Services Platform
- Network Layer
- And, Device Layer.

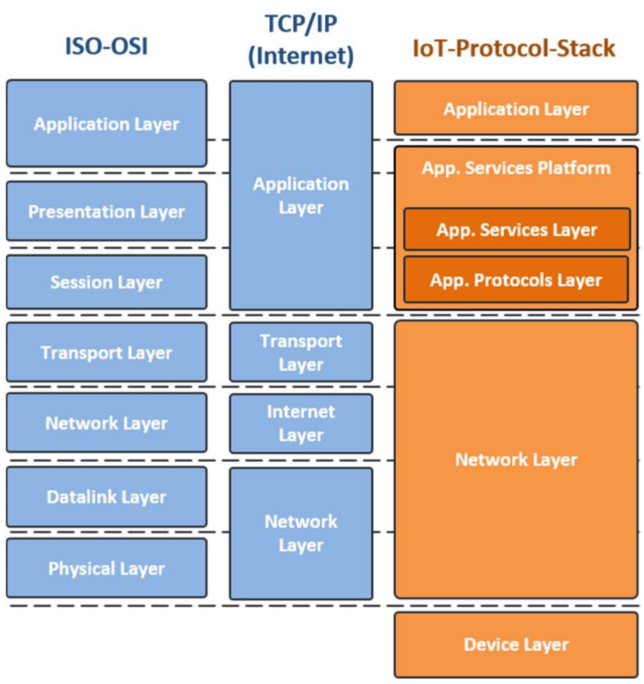


Figure 2: Representation of the IoT-protocol-stack and comparison with the ISO-OSI-model as well as with the consolidated TCP/IP (Internet) one.

With this particular structure, Applications, or Services (at Application Layer), can be developed and implemented independently from the underlying communication technologies (Network Layer). The IoT-platform (Application Services Platform, in the above figure) works as middleware to distribute the different Applications throughout the system components and, at the same time, it guarantees access to the network functionalities and all low level devices, e.g. sensors, actuators (Device Layer). In this way, it is possible to abstract the development and deployment of even complex, distributed, applications from the underlying protocol layers, thus, facilitating the deployment of IoT-systems.

The choice for a suitable IoT-platform is nevertheless application oriented and dependent on the specific system requirements. In fact, in comparison to multimedia or telecommunication systems, where reliable real-time communication does not play a particularly crucial role, industrial control systems can have strict real-time requirements and require deterministic networks.

On the basis of the results of WP1, and in particular of the collected use-cases (D1.1), in WP2 the requirements for industrial control processes, in particular for smart manufacturing and material handling, are analyzed and used as starting point to investigate exiting and soon-to-be-ready technologies that can be used for the implementation of a suitable IIoT-platform.

As stated earlier, particular attention is given to OCP-UA as an enabling technology at application level, since this, as inter alia illustrate in Section 7, seems to able to fulfill most of the project requirements related to IoT communication at enterprise level. Additionally, this technology is extensively supported by the German mechanical engineering industry association, that sees it as the key solution for the fast realization of the so-called Industry 4.0.

The remaining of the document is organized as following. In the next section, current state of the art for today’s IoT-platforms are presented in detail. Next, in Section 3, most relevant enabling technologies, especially on communication domain, for IoT-systems are given. Data storage solutions (Data Management Systems) for IoT are discussed in Section 4, whereas security aspects of (I)IoT-systems are analysed in Section 5. Finally, requirements and specifications for the IIoT-platform and the Data Management System are given respectively in Section 6, 7 and 8.

2 State of the Art for IoT Platforms [ERSTE, Thorsis]

This section aims at providing preliminary information required for designing an IoT platform which fulfils the OPTIMUM requirements. Detailed information on prominent enablers for IoT and more detailed information on security aspect is given later in the document, but in the following subsections, information on IoT platforms is provided at a more abstract level and from a broader perspective. Presented material is mostly obtained from academic literature. Notice that, in the scope of this section, the terms IoT platform, IoT framework and IoT middleware are used interchangeably.

Structure of subsections are as following. First in Section 2.1, definitions of IoT and IoT platform are presented, next in Section 2.2, expectations from IoT platforms are given. Common architectural approaches for IoT platforms are presented in Section 2.3. Then challenges and common issues in IoT platforms are given in Section 2.4. Finally, in Section 5, most prominent and relevant available IoT platforms are listed with some detail.

2.1 Definitions

Idea of Internet of Things (IoT) emerged from RFID community and was first coined by British entrepreneur Kevin Ashton in 1999. Since then, with an increased interest on IoT, it has gained diverse definitions with a broader understanding. Some of the definitions digested in [1] are presented here to have a basis for further discussion on IoT and IoT platforms.

In its special report on Internet of Things issued in March 2014 (IEEE, "Internet of Things," 2014), IEEE described the phrase "Internet of Things" as: "*A network of items—each embedded with sensors—which are connected to the Internet.*"

ITU, which is United Nations specialized agency for information and communication technologies, endorses the definition of IoT as a network that is: "*Available anywhere, anytime, by anything and anyone*" and ITU-T Study Group 13, which leads the work of the ITU on standards for next generation networks (NGN) and future networks (ITU, SERIES Y, 2005), has the following definition: "*A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.*"

The W3C, which is an international community where member organizations, a full-time staff and the public work together to develop Web standards, addresses IoT as part of "Web of Things" and defines it as follows: "*The Web of Things is essentially about the role of Web technologies to facilitate the development of applications and services for the Internet of Things, i.e., physical objects and their virtual representation. This includes sensors and actuators, as well as physical objects tagged with a bar code or NFC. Some relevant Web technologies include HTTP for accessing RESTful services, and for naming objects as a basis for linked data and rich descriptions, and JavaScript APIs for virtual objects acting as proxies for real-world objects.*"

Another general definition of IoT is presented in [2] as "Interconnection of sensing and actuating devices providing the ability to share information across platforms through a unified framework, developing a common operating picture for enabling innovative applications. This is achieved by seamless large scale sensing, data analytics and information representation using cutting edge ubiquitous sensing and cloud computing."

Along with these definitions, it is also worth to mention three visions of IoT elaborated in [3] and [4], since they well capture the perspectives on IoT. These three visions are things oriented vision, internet oriented vision and semantic oriented vision. They are explained by Singh at al. in [4] as following.

The *things-oriented vision* "is supported by the fact that we can track anything using sensors and pervasive technologies using RFID. The basic philosophy is uniquely identifying any object using specifications of Electronic Product Code (EPC). This technique is extended using sensors. It is important to appreciate the fact that future vision will depend upon sensors and its capabilities to fulfill the 'things' oriented vision."

"The *internet-oriented vision* has pressed upon the need to make smart objects which are connected. The objects need to have characteristics of IP protocols as this is one of the major protocols being followed in the world of Internet."

The *semantic-oriented vision* "is powered by the fact that the amount of sensors which will be available at our disposal will be huge and the data that they will collect will be massive in nature. Thus, we will have vast amount of information, possibly redundant, which needs to be processed meaningfully. The raw data needs to be managed, processed and churned out in an understandable manner for better representations and understanding."

These are just a few of the definitions, some by prominent bodies, and different perspectives on IoT. As seen, although there is a common understanding about IoT, there is no commonly accepted definition of it. Similar to its definition, there is no common and standard way of developing IoT systems and there are numerous approaches and competing technologies, each having its own advantages and disadvantages. This diversity, on one hand, means alternatives to choose from according to the specific needs, but on the other hand, makes it difficult to grasp and challenging to develop IoT systems from scratch. IoT platforms (or frameworks or middleware) have emerged to facilitate the development of IoT systems. Here are some definitions and discussions on IoT platforms.

In [5], IoT framework is defined as "a set of guiding principles, protocols and standards which enable the implementation of Internet of Things applications. It can but does not need to be an active participant of the overall IoT system. Frameworks can enhance IoT application development by; rapid implementation, interoperability, maintainability, security and technology flexibility. To achieve rapid implementation many of the 'boiler plate' tasks can be computer aided or removed completely."

In [6], an IoT platform is described as "cloud-based and on premise software packages and related services that enable and support sophisticated IoT services. In some instances, IoT platforms enable application developers to streamline and automate common features that would otherwise require considerable additional time, effort and expense. In other instances, IoT platforms enable enterprises to manage thousands, millions, and even billions of devices and connections across multiple technologies and protocols. Finally, in some cases, IoT software enables developers to combine device and connection data with enterprise-specific customer and ERP data as well as data from third-party sources like social and weather data to create more valuable IoT applications."

In [7], IoT middle is defined with its functionality in an IoT system as follows. "a middleware can offer common services for applications and ease application development by integrating heterogeneous computing and communications devices and supporting interoperability within the diverse applications and services running on these devices."

As the last definition given in this section, in [8], IoT platform is presented using following description "Internet-of-Things (IoT) platform (often referred to as IoT middleware) is a software that enables connecting the machines and devices and then acquisition, processing, transformation, organization and storing machine and sensor data."

2.2 Requirements of IoT Platforms in General

After several definitions of IoT platforms provided in the previous section, this section presents a more detailed information about the expectations from an IoT platform. Please note that, these are not formal lists of requirements but should be treated as general guidelines. Please also note that these are not requirements of OPTIMUM, but functional and non-functional requirements applicable to IoT platforms in general. Depending on the application scenarios one or more of the presented requirements might not be as critical as the others.

Razzaque et al. presents a comprehensive study in [9] on IoT middleware. Authors present different types of requirements of IoT middleware as following.

Middleware Service Requirements:

1. *Resource Discovery*: IoT resources include heterogeneous hardware devices (e.g., RFID tags, sensors, sensor mote, smartphones), devices' power and memory, analogue to digital converter devices (A/D), the communications module available on those devices, and infrastructural or network level information (e.g., network topology, protocols), and the services provided by these devices.
2. *Resource Management*: An acceptable QoS is expected for all applications, and in an environment where resources that impact on QoS are constrained, such as the IoT, it is important that applications are provided with a service that manages those resources. This means that resource usage should be monitored, resources allocated or provisioned in a fair manner, and resource conflicts resolved.
3. *Data Management*: Data is key in IoT applications. In the IoT, data refers mainly to sensed data or any network infrastructure information of interest to applications.
4. *Event Management*: There are potentially a massive number of events generated in IoT applications, which should be managed as an integral part of an IoT middleware.
5. *Code Management*: Deploying code in an IoT environment is challenging, and should be directly supported by the middleware.

Key non-functional requirements:

1. *Scalability*: An IoT middleware needs to be scalable to accommodate growth in the IoT's network and applications/services.
2. *Real-time or Timeliness*: A middleware must provide real time services when the correctness of an operation it supports depends not only on its logical correctness, but also on the time in which it is performed.
3. *Reliability*: A middleware should remain operational for the duration of a mission, even in the presence of failures. The middleware's reliability ultimately helps in achieving system level reliability.
4. *Availability*: A middleware supporting an IoT's applications, especially mission critical ones, must be available, or appear available, at all times. Even if there is a failure somewhere in the system, its recovery time and failure frequency must be small enough to achieve the desired availability.

5. *Security & Privacy*: Security is critical to the operation of IoT. In IoT middleware, security needs to be considered in all the functional and non-functional blocks including the user level application. Context-awareness in middleware may disclose personal information (e.g., the location of an object or a person).
6. *Ease-of deployment*: Since an IoT middleware (or more likely, updates to the middleware) is typically deployed by the user (or owner of the device), deployment should not require expert knowledge or support. Complicated installation and setup procedures must be avoided.

Architectural requirements:

1. *Programming Abstraction*: Providing an API for application developers is an important functional requirement for any middleware. For the application or service developer, high-level programming interfaces need to isolate the development of the applications or services from the operations provided by the underlying, heterogeneous IoT infrastructures. The level of abstraction, the programming paradigm, and the interface type all need to be considered when defining an API. The level of abstraction refers to how the application developer views the system (e.g., individual node/device level, system level). The programming paradigm (e.g., Publish/Subscribe) deals with the model for developing or programming the applications or services. The interface type defines the style of the programming interface.
2. *Inter-operable*: A middleware should work with heterogeneous devices/technologies/applications, without additional effort from the application or service developer. Heterogeneous components must be able to exchange data and services. Interoperability in a middleware can be viewed from network, syntactic, and semantic perspectives, each of which must be catered for in an IoT.
3. *Service-based*: A middleware architecture should be service-based to offer high flexibility when new and advanced functions need to be added to an IoT's middleware.
4. *Adaptive*: A middleware needs to be adaptive so that it can evolve to fit itself into changes in its environment or circumstances. In the IoT, the network and its environment are likely to change frequently.
5. *Context-aware*: Context-awareness is a key requirement in building adaptive systems and also in establishing value from sensed data. The IoT's middleware architecture needs to be aware of the context of users, devices, and the environment and use these for effective and essential services' offerings to users.
6. *Autonomous*: Autonomous means self-governed. Devices/technologies/applications are active participants in the IoT's processes and they should be enabled to interact and communicate among themselves without direct human intervention.
7. *Distributed*: A large-scale IoT system's applications/devices/users (e.g., WSNs, VANETs) exchange information and collaborate with each other.

These requirements presented in [9] mostly deal with the characteristics of IoT systems presented in the same work. Although they are basis for the requirements, they can be taken as a more general guideline in IoT platform design. These characteristics are listed below. For more detailed information on each item [9] can be referred.

1. Characteristics of IoT Infrastructure
    a. Heterogeneous devices
    b. Resource-constrained
    c. Spontaneous interaction
    d. Ultra large-scale network and large number of events

   e. Dynamic network and no infrastructure
   f. Context aware
   g. Intelligence
   h. Location-aware
2. Characteristics of IoT Applications
   a. Diverse applications
   b. Real-time
   c. Everything as a service
   d. Increased security attack surface
   e. Privacy leakage

In [10], Ray presents utility factors of IoT. These are not direct requirements on an IoT platform, but since IoT platforms are used to develop IoT systems, these should be considered in development of IoT platforms. Following IoT utilities are taken as an excerpt from [10].

1. *Dynamic and self-adapting*: IoT devices and systems should have the capability to dynamically adapt with the changing contexts and take actions based on their operating conditions, user's context, or sensed environment.
2. *Self-configuring*: IoT devices may have self-configuring capability, allowing a large number of devices to work together to provide certain functionality (such as weather monitoring). These devices have the ability to configure themselves (in association with IoT infrastructure), setup the networking, and fetch latest software upgrades with minimal manual or user intervention.
3. *Interoperable communication protocols*: IoT devices may support a number of interoperable communication protocols and can communicate with other devices and also with the infrastructure.
4. *Unique identity*: Each of IoT device has a unique identity and unique identifier (such as IP address or URI).
5. *Integrated into information network*: IoT devices are usually integrated into the information network that allows them to communicate and exchange data with other devices and systems. IoT devices can be dynamically discovered in the network, by other devices and/or network, and have the capability to describe themselves (and their characteristics) to other devices or user applications.
6. *Context-awareness*: Based on the sensed information about the physical and environmental parameters, the sensor nodes gain knowledge about the surrounding context.
7. *Intelligent decision making capability*: IoT is multi-hop in nature. In a large area, this feature enhances the energy efficiency of the overall network, and hence, the network lifetime increases. Using this feature, multiple sensor nodes collaborate among themselves, and collectively take the final decision.

Different from previous studies targeting IoT platforms in general, in [11], presents some design considerations for Industrial IoT applications. These design considerations are as following.

1. Energy: How long can an IoT device operate with limited power supply
2. Latency: How much time is needed for message propagation and processing
3. Throughput: What is the maximum amount of data that can be transported through the network
4. Scalability: How many devices are supported

5. Topology: Who must communicate with whom
6. Security and Safety: How secure and safe is the application

2.3 Architectural Approaches to IoT Platforms

Despite IoT being a still emerging field, there is a plethora of IoT platforms available. This section aims to give common features used in IoT systems and classify architecture having similar approaches. So that the architectural design of IoT platform for OPTIMUM can make use of these approaches, depending on the requirements.

Although there are different architectural approaches to IoT platforms, let alone the actual number of different specific architectures, [12] aims to extract a reference architecture to better comprehend and compare IoT platforms. As stated in [12], the "reference architecture does not present new concepts, but provides a more abstract view on the components of IoT platforms and their possible connections." Components of the reference architecture can be omitted in certain IoT platforms.

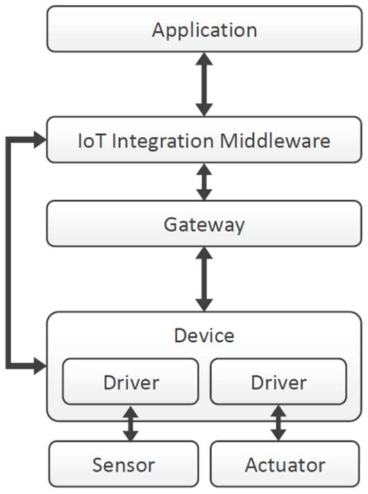


Figure 3: IoT Reference Architecture [12]

Components of the reference architecture is depicted in Figure 3 and described below.

1. *Sensor*: A Sensor is a hardware component, which is used to measure parameters of its physical environment and to translate them into electrical signals. Typically, Sensors are connected to or are integrated into a Device to which the gathered data is sent.
2. *Actuator*: An Actuator is a hardware component, which can act upon, control, or manipulate the physical environment, for example, by giving an optic or acoustic signal. Actuators receive commands from their connected Device. They translate electrical signals into some kind of physical action. Just like Sensors, Actuators are typically connected to or are even integrated into a Device.
3. *Device*: A Device is a hardware component, which is connected to Sensors and/or Actuators via wires or wirelessly or even integrates these components. To process data from Sensors and to control Actuators, typically software in the form of Drivers is required. A Driver in our architecture enables other software on the Device to access Sensors and Actuators. It represents the first possibility to use software to process data produced by Sensors and to control Actuators influencing the physical environment. Thus, Devices are the entry point of the physical environment to the digital world.
4. *Gateway*: Devices are often connected to a Gateway in cases when the Device is not capable of directly connecting to further systems, e.g., if the Device cannot

communicate via a particular protocol or because of other technical limitations. To solve these problems, a Gateway is used to compensate such limitations by providing required technologies and functionalities to translate between different protocols and by forwarding communication between Devices and other systems. A Gateway is, therefore, responsible for supporting the required communication technologies and protocols in both directions and for translating data if necessary.

5. *IoT Integration Middleware*: The IoT Integration Middleware is responsible for receiving data from the connected Devices to process the received data, for example, by evaluating condition-action rules, to provide the received data to connected Applications, and to control Devices in terms of sending commands to be executed by the respective Actuators. A Device can communicate directly with the IoT Integration Middleware if it supports an appropriate communication technology, such as WiFi, a corresponding transport protocol, such as HTTP or MQTT, and a compatible payload format, such as JSON or XML. Otherwise the Device communicates over a Gateway with the IoT Integration Middleware. The IoT Integration Middleware is not limited to the functionality described above. It may comprise all kinds of functionality that is required by a certain cyber-physical system, for instance, a rules engine or graphical dashboards.
6. *Application*: The Application component represents software that uses the IoT Integration Middleware to gain insight into the physical environment by requesting Sensor data or to control physical actions using Actuators.

According to [15], architectures for IoT middleware form three class types: service-based IoT middleware, cloud-based IoT middleware, and actor-based IoT middleware. In the study, each class is explained as following.

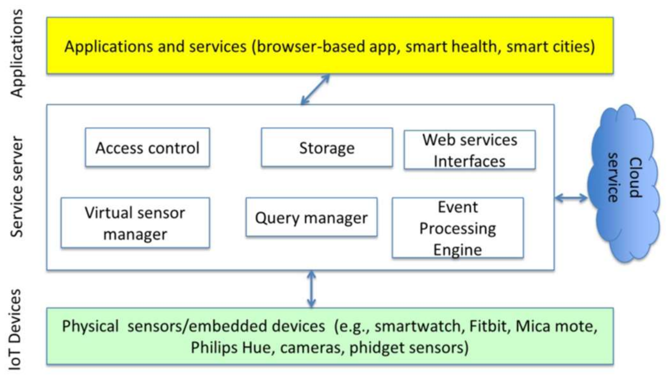

Applications
Applications and services (browser-based app, smart health, smart cities)
Service server
Access control
Storage
Web services Interfaces
Virtual sensor manager
Query manager
Event Processing Engine
Cloud service
IoT Devices
Physical sensors/embedded devices (e.g., smartwatch, Fitbit, Mica mote, Philips Hue, cameras, phidget sensors)

Figure 4: Service-based IoT middleware [15]

The first type, which we refer to it as a *service-based solution*, generally adopts the Service-Oriented Architecture (SOA) and allows developers or users to add or deploy a diverse range of IoT devices as services. Figure 4 depicts a service-based IoT middleware. It is a three-layered architecture adopted by the OpenIoT, a European Union project to standardize IoT platforms. This architecture consists of a Physical Plane (sensors and actuators), a Virtualized Plane (server or cloud infrastructure) and an Application Plane (utility). The main computational units are available in the middle layer or the Virtualized Plane. The generic services available in the middle layer range from access control, storage management and event processing engine. These services support the data collection part of the BAC-like IoT applications, but not the analytic part. The service-based architecture is a high performing heavy weight middleware generally deployed on multiple nodes running in the cloud or on powerful gateways between IoT devices and the applications. It is not designed to be deployed in resource constrained IoT devices (e.g., smart phones) and does not support device to device communication.

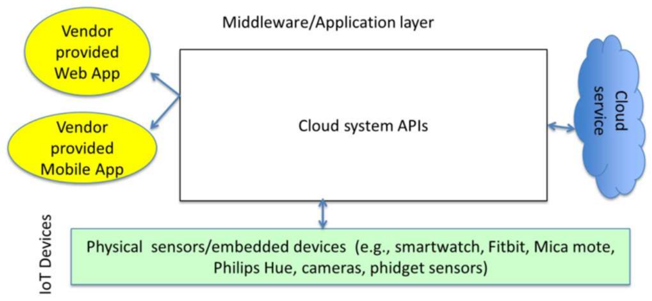

Vendor provided Web App
Vendor provided Mobile App
Middleware/Application layer
Cloud system APIs
Cloud service
IoT Devices
Physical sensors/embedded devices (e.g., smartwatch, Fitbit, Mica mote, Philips Hue, cameras, phidget sensors)

Figure 5: Cloud-based IoT middleware [15]

The second type, which is known as *cloud-based solution*, limits the users on the type and the number of IoT devices that they can deploy, but enables users to connect, collect and interpret the collected data with ease since possible use cases can be determined and programmed a-priori. A cloud-based IoT middleware architecture is shown in Figure 5

The functional components (white box in the diagram) of the middleware are limited to what is available on the cloud and it varies widely among cloud-based platforms. Typically, those functionalities are exposed as a set of APIs. The provided functionalities could be as simple as a very high-performance storage system or a very powerful computation engine with pre-defined monitoring and analysis tools. The services of IoT devices available in the cloud can only be accessed or controlled via either vendor's provided application or cloud supported RESTful APIs.

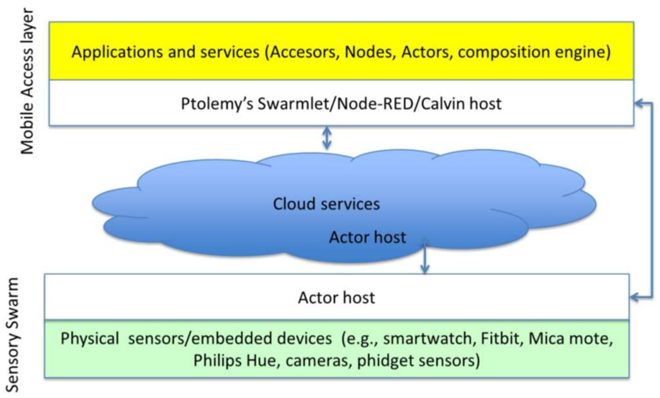

Mobile Access layer
Applications and services (Accesors, Nodes, Actors, composition engine)
Ptolemy's Swarmlet/Node-RED/Calvin host
Cloud services
Actor host
Sensory Swarm
Actor host
Physical sensors/embedded devices (e.g., smartwatch, Fitbit, Mica mote, Philips Hue, cameras, phidget sensors)

Figure 6. Actor-based IoT middleware [15]

The third type is the *actor-based framework* that emphasizes on the open, plug and play IoT architecture. A variety of IoT devices can be exposed as reusable actors and distributed in the network. An actor-based IoT middleware architecture is first presented in Terraswam, a joint research project between universities, government and private companies in USA. A three-concentric circles visual is used to depict the architecture of the actor-based IoT middleware. The outermost circle is the Sensory Swarm (sensors and actuators), the middle circle is the Mobile Access (gateways such as smartphone, Raspberry Pi, Swarmbox, Laptop) and the inner most circle is the Cloud. To facilitate the comparison with other types, we map the three circles into a three layered architectural view.

Figure 6 depicts this architecture. As shown in the figure, the middleware (also named as actor host) is designed to be light-weight that can be embedded in all the layers (sensory layer, mobile access layer and the cloud). The basic middleware computation units are thus distributed in the network (the white box).

In another study [9], IoT middleware design approaches are grouped into 7 as following: event-based, service-oriented, virtual machine-based, agent-based, tuple-spaces, database-oriented, and application-specific. As it can be seen, there are some overlaps with [15]. Descriptions of the classes are given as below in [9].

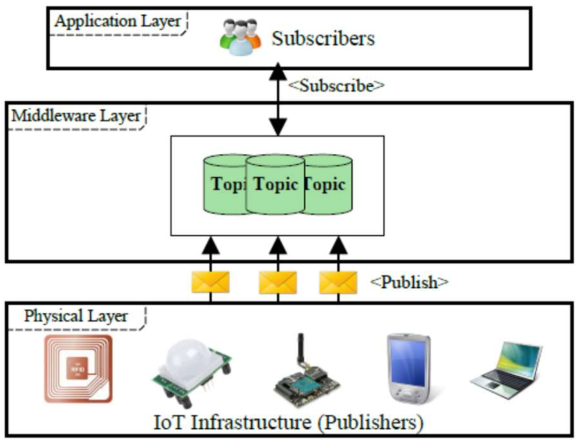


Figure 7: General design model for Event-Based Middleware [9]

In *event-based middleware*, components, applications, and all the other participants interact through events. Each event has a type, as well as a set of typed parameters whose specific values describe the specific change to the producer's state. Events are propagated from the sending application components (producers), to the receiving application components (consumers). An event system (event service), may consist of a potentially large number of application components (entities) that produce and consume events.

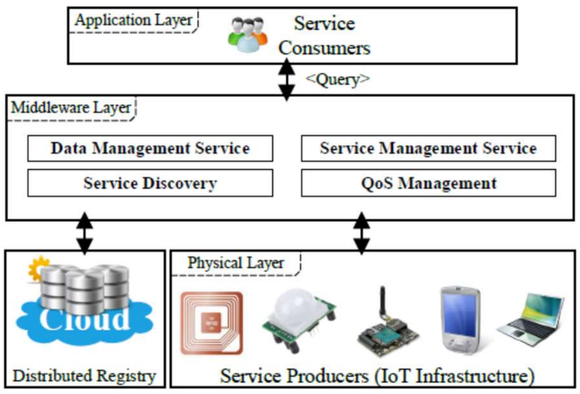


Figure 8: General design model for a Service-Oriented Middleware [9]

The *service-oriented design paradigm* builds software or applications in the form of services. Service-oriented computing (SOC) is based on Service-Oriented Architecture (SOA) approaches and has been traditionally used in corporate IT systems. The characteristics of SOC, such as technology neutrality, loose coupling, service reusability, service composability, service discoverability, are also potentially beneficial to IoT applications. However, IoT's ultra-large-scale network, resource-constrained devices, and mobility characteristics make service discovery and composition challenging. A service-oriented middleware (SOM) has the potential to alleviate these challenges through the provision of appropriate functionalities for deploying, publishing/discovering and accessing services at runtime. SOM also provides support for adaptive service compositions when services are unavailable.

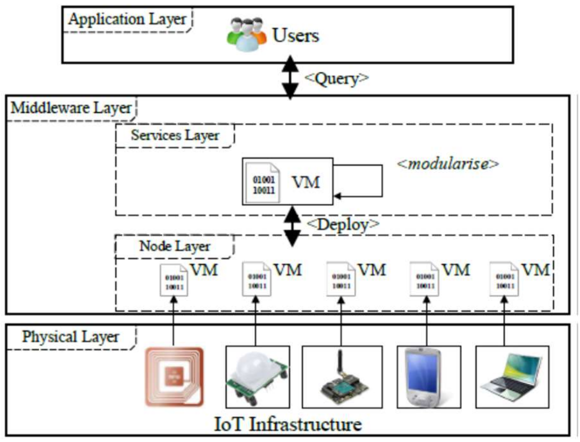


Figure 9: General design model for VM-based middleware [9]

Virtual machine (VM) oriented middleware design provides programming support for a safe execution environment for user applications by virtualizing the infrastructure. The applications are divided into small separate modules, which are injected and distributed throughout the network. Each node in the network holds a VM, which interprets the modules. This approach is commonly used to address a lack of architectural support such as high-level programming abstractions, self-management and adaptivity, while supporting transparency in distributed heterogeneous IoT infrastructures.

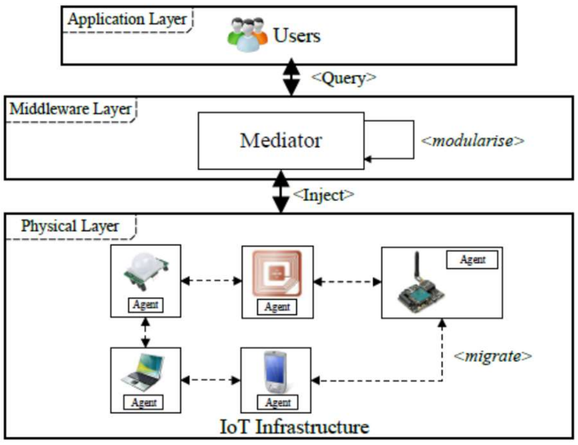


Figure 10: General design model for agent-based middleware [9]

In the *agent-based approach* to middleware, applications are divided into modular programs to facilitate injection and distribution through the network using mobile agents. While migrating from one node to another, agents maintain their execution state. This facilitates the design of decentralized systems capable of tolerating partial failures. Previous research in this area has presented a number of advantages for using mobile agents in generic distributed systems. In the context of the IoT middleware requirements, these are: resource management (network load reduction and network latency reduction), code management (asynchronous and autonomous execution and protocol encapsulation), availability and reliability (robustness and fault tolerance), adaptiveness and heterogeneity. Moreover, an agent can engage in dialogues with other software agents to proactively gather data and update only parts of the application. Additionally, agent-based approaches consider resource constrained devices.

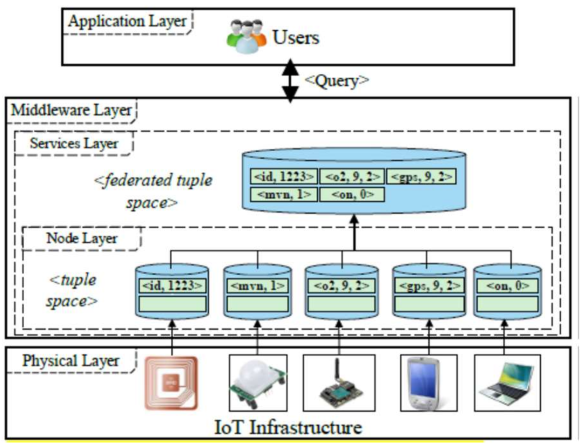


Figure 11: General design model for tuple-space-based middleware [9]

In *tuple-space middleware*, each member of the infrastructure holds a local tuple space structure. A tuple space is a data repository that can be accessed concurrently. All the tuple spaces form a federated tuple space on a gateway (i.e., base station). This approach suits mobile devices in an IoT infrastructure, as they can transiently share data within gateway connectivity constraints. Applications communicate by writing tuples in a federated tuple space, and by reading them through specifying the pattern of the data they are interested in.

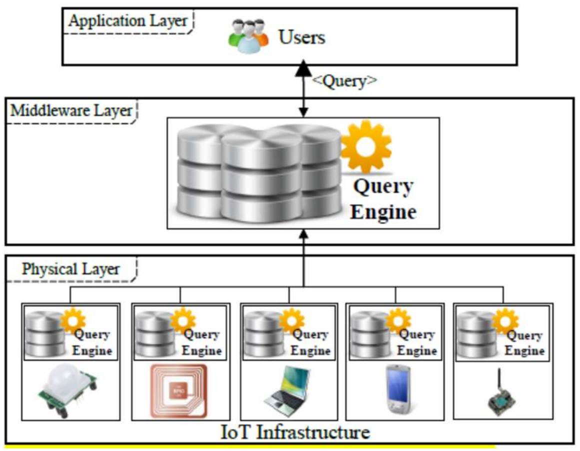


Figure 12: General design model for database-oriented middleware [9]

In *database-oriented middleware*, a sensor network is viewed as a virtual relational database system. An application can query the database using SQL like query language, which enables the formulation of complex queries. Research in this area has been focused on developing a distributed database approach to interoperating systems.

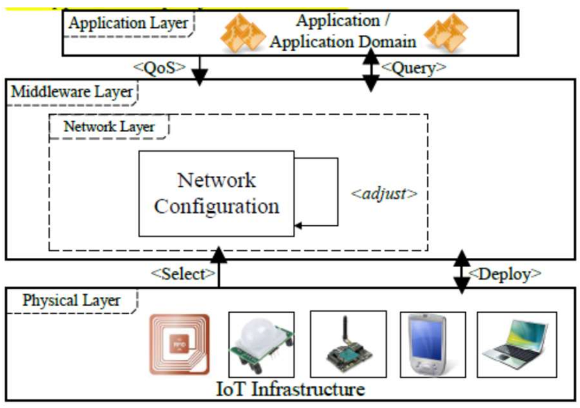


Figure 13: General design model for application-specific middleware [9]

An *application-specific (i.e., application-driven) approach* to middleware focuses on resource management support (i.e., QoS support) for a specific application or application domain by implementing an architecture that fine-tunes the network or infrastructure based on the application or application domain requirements.

Another study dealing with IoT architectural approaches is [5]. This study classifies IoT architectural approaches into three groups as global cloud approach, smart objects approach and local cloud concept. These approaches are explained as following in [5].

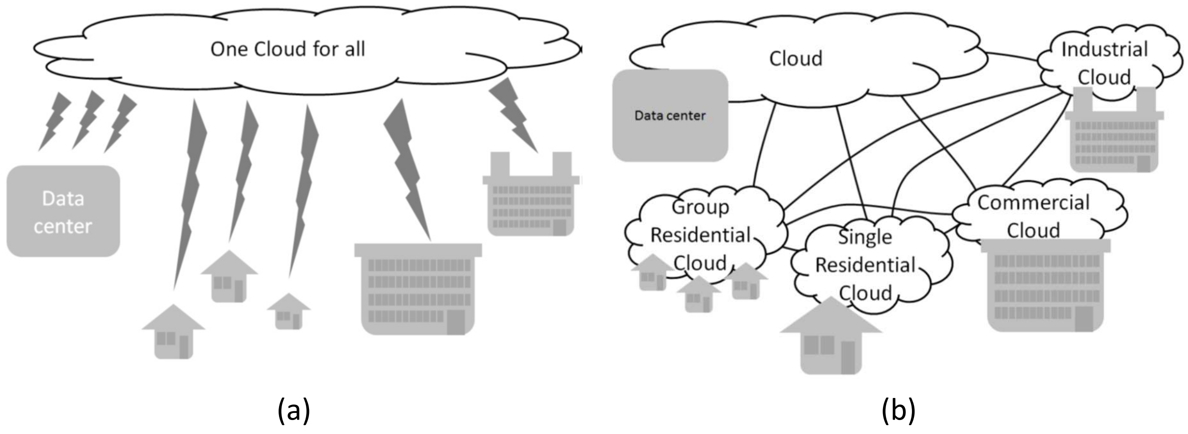


Figure 14: (a) Global, and (b) local cloud concepts [5]

Many frameworks take a data centric or data driven approach. Utilizing a *global cloud*, they focus on enabling collation, visualization and analytics on data. This architecture is well suited for applications such as asset tracking, logistics and predictive maintenance. In some cases, the framework will allow creation of local hosted instances but do not detail a method of interconnecting multiple cloud instances within the framework. This approach is suitable for providing data as a service but will generally leave the implementation specific of the end-points to application developers.

Figure 14(a) illustrates the concept of a global cloud through which IoT applications connect and communicate.

A *smart objects approach* makes the endpoints active participants within the framework. The end points are included as key aspects of the framework which means the focus of the framework is on interconnecting the end-points. This approach is well suited for distributed automation tasks which require a high level of device independence, such as home and building automation and manufacturing. The data is produced by end points and consumed by end-points, which within this context will usually have some predefine understanding of the end-point pairing.

In both of the approaches mentioned, many features such as end-to-end security and layered interoperability suffer due to ad-hoc development either at the end-points or within the cloud. A third approach becoming more prevalent, is taking into account the need to satisfy real-time automation requirements while not hindering the value of semantic big data and data analytics.

He and Li focus on service-oriented architecture for IoT platforms in [11]. They apply a layered approach, sensing, networking, service and interface being the layers from bottom to up. The architecture is depicted in Figure 15 and its layers are explained as following:

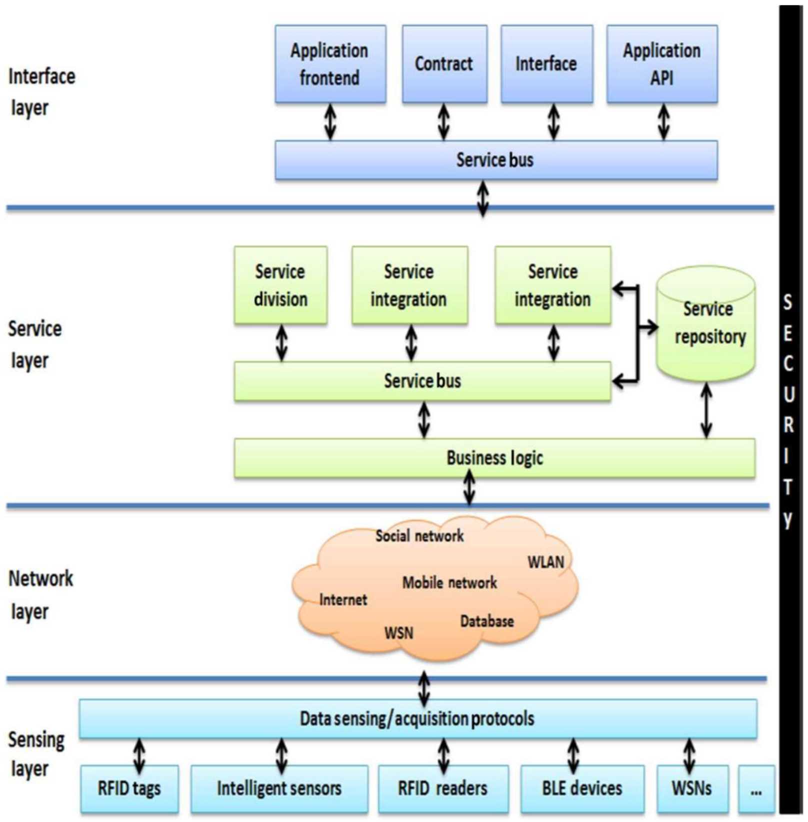


Figure 15: SOA for IoT [11]

1. *Sensing Layer*: In the sensing layer, the wireless smart systems with tags or sensors are now able to automatically sense and exchange information among different devices.
2. *Networking Layer*: The role of networking layer is to connect all things together and allow things to share the information with other connected things. In addition, the networking layer is capable of aggregating information from existing IT infrastructures (e.g., business systems, transportation systems, power grids, healthcare systems, ICT systems, etc.).
3. *Service Layer*: Service layer relies on the middleware technology that provides functionalities to seamlessly integrate services and applications in IoT. The middleware technology provides the IoT with a cost-efficient platform, where the hardware and software platforms can be reused. A main activity in the service layer involves the service specifications for middleware, which are being developed by various organizations. A well-designed service layer will be able to identify common

application requirements and provide APIs and protocols to support required services, applications, and user needs. This layer also processes all service-oriented issues, including information exchange and storage, data management, search engines, and communication. This layer includes the following components.
   a. Service discovery: finding objects that can offer the needed services and information in an efficient way.
   b. Service composition: enabling the interaction and communication among connected things. The discovery phase leverages the relationships among different things to discover the desired service, and the service composition component is to schedule or re-create more suitable services in order to acquire the most reliable services to meet the request.
   c. Trustworthiness management: aiming at determining trust and reputation mechanisms that can evaluate and use the information provided by other services to create a trustworthy system.
   d. Service APIs: supporting the interactions between services required in IoT.

4. *Interface Layer*: In IoT, a large number of devices involved are made by different manufacturers/vendors and they do not always follow the same standards/protocols. As a result of the heterogeneity, there are many interaction problems with information exchange, communication between things, and cooperative event processing among different things. Furthermore, the constant increase of things participating in an IoT makes it harder to dynamically connect, communicate, disconnect, and operate. There is also a necessity for an interface layer to simplify the management and interconnection of things.

As the last work to be presented here, in [16], major building blocks of IoT and main components of IoT platforms are presented and explained in detail. These are explained as below in [16].

Major building blocks of IoT:

1. *Hardware*: This is where data is produced. The hardware layer includes the physical devices with their in-built microprocessors, sensors, actuators and communication hardware.
2. *Communication*: This is where data gets transported. This part of the technology infrastructure ensures the hardware is connected to the network, via proprietary or open-source communication protocols.
3. *Software backend*: This is where data is managed. The software backend manages all connected devices and networks and provides the necessary data integration as well as the interface to other systems (e.g., ERP-system).
4. *Applications*: This is where data is turned into value. In the application layer, IoT use cases get presented to the user (B2C or B2B). Most of the applications run on smartphones, tablets, PCs or other devices/things and “do something valuable” with the data.

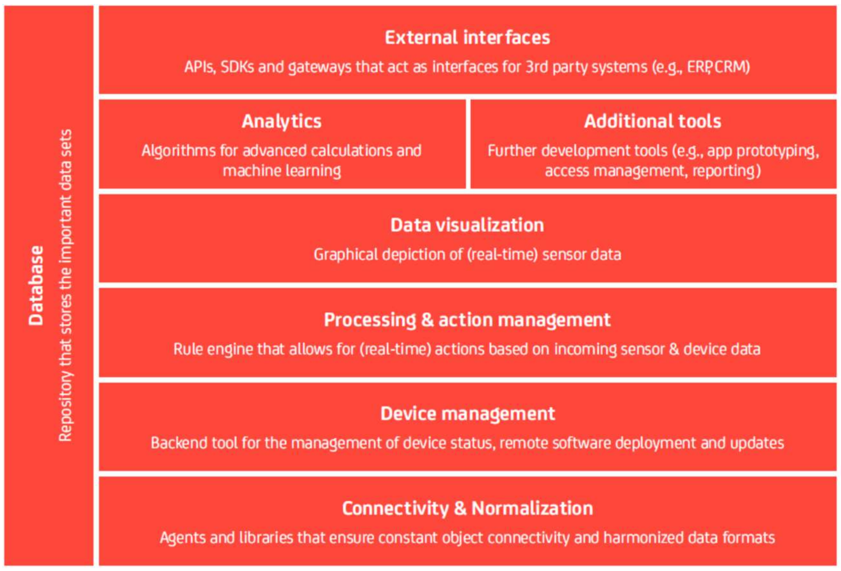

External interfaces
APIs, SDKs and gateways that act as interfaces for 3rd party systems (e.g., ERP, CRM)
Analytics
Algorithms for advanced calculations and machine learning
Additional tools
Further development tools (e.g., app prototyping, access management, reporting)
Data visualization
Graphical depiction of (real-time) sensor data
Processing & action management
Rule engine that allows for (real-time) actions based on incoming sensor & device data
Device management
Backend tool for the management of device status, remote software deployment and updates
Connectivity & Normalization
Agents and libraries that ensure constant object connectivity and harmonized data formats
Database
Repository that stores the important data sets

Figure 16: The eight major building blocks of an IoT platform [16]

Main components of IoT platforms:

1. *Connectivity & Normalization*: Every IoT platform starts with a connectivity layer. It has the function of bringing different protocols and different data formats into one "software" interface. This is necessary in order to ensure all devices can be interacted with and data is read correctly. Having all device data in one place and in one format is the basic necessity to monitor, manage, and analyze IoT devices.
2. *Device Management*: The device management module of an IoT platform ensures the connected objects are working properly and its software and applications are updated and running. Tasks performed in this module include device provisioning, remote configuration, management of firmware/ software updates, and troubleshooting.
3. *Database*: Data storage is a central piece in an IoT platform. An IoT platform therefore usually comes with a cloud based database solution that is distributed across different sensor nodes. It should be scalable for big data and should be able to store both structured (SQL) and unstructured data (NoSQL).
4. *Processing & Action Management*: The data that is captured in the connectivity & normalization module and that is stored in the database gets brought to life in this part of the IoT platform. A rule-based event-action-trigger allows performance of "smart" actions based on specific sensor data.
5. *Additional Tools*: Advanced IoT platforms often offer an additional set of tools for the developer and the manager of the IoT solution. Development tools allow the IoT developer to prototype and test the IoT case. Sometimes even in the form of what-you-see-is-what-you-get-editors (WYSIWYG) that lets you create simple smartphone apps for visualizing and controlling connected devices.
6. *Analytics*: Many IoT use cases go beyond action-management and require complex analytics to get the most out of the IoT data-stream. The analytics engine encompasses all dynamic calculations of sensor data, from basic data clustering to deep machine learning.
7. *Data Visualization*: Sometimes also referred to as "visual analytics," data visualization presents a much-underrated part of the IoT platform. The combination of human eye and brain is still far superior to most analytic and rule-based engines. That is why data visualization is so important: it enables humans to see patterns and observe trends. Visualization comes in the form of line-, stacked-, or pie charts, 2D- or even 3D-models.

The visualization dashboard that is available to the manager of the IoT platform is often also included in the prototyping tools that an advanced IoT platform provides.

8. *External Interfaces*: IoT enabled businesses are rarely built standalone and on a green field. In established companies it is crucial that the Internet of Things integrates with existing ERP systems, management tools, manufacturing execution systems and the rest of the wider IT-ecosystem.

2.4 Challenges in IoT Platforms

IoT systems and IoT platforms are still in the phase of rapid advancement and development and there are still many open and challenging points on technology, standardization, security and privacy that require special attention. This section provides a collection of such issues compiled from different studies.

In [10], Ray provides a long list of technical challenges as following.

- Design of Service oriented Architecture (SoA) for IoT is a big challenge where service-based objects may face problems from performance and cost related issues. SoA needs to handle a large number of devices connected to the system which phrases scalability issues. At this moment, challenges like: data transfer, processing, and management become a matter of burden overheaded by service provisioning.
- IoT is a very complicated heterogeneous network platform. This, in turn enhances the complexity among various types devices through various communication technologies showing the rude behavior of network to be fraudulent, delayed, and non-standardized. The management of connected objects by facilitating through collaborative work between different things e.g., hardware components and/or software services, and administering them after providing addressing, identification, and optimization at the architectural and protocol levels is a serious research challenge.
- If we look from the viewpoint of network services, it seems clear that there is always a lack of a Service Description Language (SDL). Otherwise, it would make the service development, deployment, and resource integration difficult by extending the product dissemination time causing loss in market. Hence, a commonly accepted SDL should be constructed so as the powerful service discovery methods and object naming services be implemented. Novel SDL may be developed to cope with product dissemination after validating the requite SDL specific architecture.
- As of now, IoT is degenerated on a traditional network oriented ICT environment. It is always affected by whatever connected to it. Here, a need of unified information infrastructure is to be sought. Huge number of connected devices shall produce real-time data flow which must be governed by high band width frequency path. Hence, a uniform architectural base is to be created to cater the infrastructure needs sophistically.
- The originated data may be too much large in size that current database management system may not handle in real-time manner. Proper solutions need to be idealized. IoT based data would be generated in a rapid speed. The collected data at receiver’s end shall be stored in efficient way which current RAID technology is incapable of. Here, an IoT based data service centric architecture need to be revised to handle this problem.
- Different devices attached to the IoT will put down data of variety in type, size and formation. These variations should be occupied with the futuristic technology which

may involve multi-varied architectural notion for its ideal indentation. Researcher should come forward with novel Big IoT Data specific design where data can be efficiently handled.

- Data is a raw fact that generally does not conform to non-relevant handouts. Here in case of IoT, data play the massive role in decision making. The value of data is only achievable after filtering process is performed on the pool of data. This meaningful information can only be obtained by orientation of mining, analysis, and understand it. Big data problem is sufficient for handling similar regression. Relevant architectural framework is in evident that can hale data mining, analytics, and hence decision-making services. Big Data approach could be aggregated herewith.
- In addition, industries must seek the challenges of hardware software coexistence around IoT. Variety of devices combined with variety of communication protocols through TCP/IP or advanced software stacks would surely manipulate web services which shall be deployed by various middleware solutions. Particular architecture leveraging the facilitation of heterogeneous protocols shall be devised.
- The IoT is envisaged to include an incredibly high number of nodes. All the attached devices and data shall be retrievable; here in such context, the unique identity is a must for efficient point-to-point network configuration. IPv4 protocol identifies each node through a 4-byte address. As it is well known that the availability of IPv4 numbered addresses is decreasing rapidly by reaching zero in next few years, new addressing policies shall be countered where IPv6 is a strong contender. This is an area where utmost care is needed to pursue device naming and identification capability, where appropriateness of architectural proficiency is a must.
- Standardization is another clot which may precisely be operated for growth of IoT. Standardization in IoT signifies to lower down the initial barriers for the service providers and active users, improvising the interoperability issues between different applications or systems and to perceive better competition among the developed products or services in the application level. Security standards, communication standards and identification standards need to be evolved with the spread of IoT technologies while designing emerging technologies at a horizontal equivalence. In addition, fellow researchers shall document industry-specific guidelines and specify required architectural standards for efficient implementation of IoT.
- From the viewpoint of service, lack of a commonly accepted service description language makes the service development and integration of resources of physical objects into value-added services difficult. The developed services could be incompatible with different communication and implementation environments. In addition, powerful service discovery methods and object naming services need to be developed to spread the IoT technology. Scientists should pave novel architectures to cater with these difficulties.
- The widespread applicability of IoT and associated technologies shall largely depend on the network with information security and data privacy protection. Being highly complex and heterogeneous in nature, IoT always faces severe security and privacy threats.

Xu et al. in [11] also focus on technological challenges of. They are elaborated as following.

- Design an SOA for IoT is a big challenge, in which service-based things might suffer from performance and cost limitations. In addition, scalability issues often arise as more and more physical objects are connected to the network. When the number of things is large, scalability is problematic at different levels including data transfer and networking, data processing and management, and service provisioning.

- From the viewpoint of network, the IoT is a very complicated heterogeneous network, which includes the connection between various types of networks through various communication technologies. Currently, there is lack of widely accepted common platform that hides the heterogeneity of underlining networks/communication technologies and provides a transparent naming service to various applications. Large amounts of data transmission across the network at the same time can also cause frequent delay, conflict, and communication issues. It is a challenging task to develop networking technologies and standards that can allow data gathered by a large number of devices to move efficiently within IoT networks. Managing connected things in terms of facilitating the collaboration between different entities and the administering devices addressing, identification, and optimization at the architectural and protocol levels is a research challenge.
- From the viewpoint of service, a lack of a commonly accepted service description language makes the service development and integration of resources of physical objects into value-added services difficult. The developed services could be incompatible with different communication and implementation environments. In addition, powerful service discovery methods and object naming services need to be developed to spread the IoT technology.
- As IoT is often developed based on a traditional ICT environment and it is affected by everything connected to the network, it requires a lot of work to integrate IoT with existing IT systems or legacy systems into a unified information infrastructure. Furthermore, with the huge number of things connected to the Internet, a massive amount of real-time data flow will be automatically produced by connected things. The data may not have much meaningful value unless people find an effective way to analyze and understand it. Analyzing or mining massive amounts of data generated from both IoT applications and existing IT systems to derive valuable information requires strong big data analytics skills, which could be challenging for many end users. In addition, integrating IoT devices with external resources such as existing software systems and Web services requires the development of various middleware solutions, since applications vary a lot by industries. Building practical applications in which heterogeneous IoT-related data are combined with traditional data can be a challenging task for a variety of industries.

Ngu et al. [15] also studies some of the key challenges and issues in IoT realm but with a focus on IoT middleware. Following four challenges are discussed in the paper as follows.

- The first challenge is in developing an IoT middleware that must be available in the cloud as well as on the edge (IoT devices and gateways) for supporting all types of IoT applications, for better privacy control and latency. This requires the system to be portable and lightweight. Among the IoT middleware we studied, only Calvin, Node-RED, and Ptolemy Accessor Host are designed to be portable and light-weight. There is a tradeoff between having powerful services such as semantic based discovery, fraud resilient security enforcement, and stream processing versus the ability to deploy an instance of the IoT middleware in constrained devices.

- The second challenge is to empower consumers to create IoT applications targeted to their context. In the cloud-based IoT middleware, no composition engine is provided for consumers. This limits consumers to the pre-programmed IoT applications and prevents consumers from creating their own innovative applications. For the service-based architecture, an SDK tool is provided for crafting an IoT application. This requires low level programming knowledge and does not empower consumers to create their IoT applications targeted to their needs. Currently, only the actor-based IoT middleware and the OpenIoT project provide visual composition tools. The visual tools provided by Node-RED and Ptolemy Accessor Host are early composition tools in this research direction. For example, in Ptolemy Accessor Host, the accessors which are the fundamental elements of the composition tool must be designed and implemented according to a specific programming model. This requires end users to master JavaScript or other scripting languages in order to create an IoT application if the desired accessors have not already been provided. Moreover, an accessor in an IoT application is "designed to fit" a particular usage. Each new usage requires the development of a new accessor. A higher-order accessor or a context-aware accessor needs to be developed for flexible composition of IoT applications. For example, by gathering some contextual information from consumers (location, time, URL, communication protocol), the desired accessor can be configured automatically as a subclass of an existing accessor for the desired IoT application. In addition, currently, no composition tool supports transactional properties. It assumes IoT applications will run from the beginning to the end successfully. There is no provision of rollback or restart from a certain point in a composed IoT application when there is a failure.
- The third challenge is to provide semantic service discovery that goes beyond discovery of IP addresses of the nearby IoT devices. Given that the environment that the IoT application interacts with evolves continuously, new services or devices could come on-line anytime and existing devices might become unavailable. It is essential to be able to discover or query for compatible services at the right time and at the right place both at design time as well as at runtime. For example, in some critical health monitoring IoT application, failed IoT services must be replaced without causing any disruption.
- The fourth challenge is to guarantee the security of IoT applications and also protect the privacy of users. Many applications from a variety of domains ranging from smart healthcare [17] to digital agriculture [18] are utilizing the IoT infrastructure. Critical decisions are going to be made in these applications by analyzing data collected from the IoT devices. This has raised the issues of security, privacy and trustworthiness of IoT generated data [19]. These issues are not limited to data alone, but also the underlying networks and devices. Hence, supporting security, privacy and trust mechanisms within IoT middleware has been recognized as a critical and important issue for the successful deployment of IoT applications, and is deemed as one of the major challenges in both industry and academic communities. Security is generally supported via some kind of authentication and encryption protocols and privacy is addressed by giving the end user the ability to specify different level of access controls without the guarantee of data ownership.

[9] follows a different organizational approach to discuss challenges in IoT middleware. It discusses the possible challenges in relation with the requirements that it provides, which are also presented in Section 2.2. The study presents the challenges as below.

Challenges related to Functional Requirements:

1. *Resource Discovery*: The dynamic and ultra-large-scale nature of the IoT infrastructure invalidates centralized resource registries and discovery approaches. However, deciding between purely distributed and hybrid solutions is complicated. A trade-off is necessary between registry distribution and the number of registries. Fewer registries provide consistent and fast discovery of resources under normal circumstances, but will not scale well when there is a large number of service discovery queries in IoT applications.
2. *Resource Management*: Frequent resource conflicts occur in IoT applications that share resources (e.g., actuators). Conflict resolution will be required to resolve conflicts in resource allocation among multiple concurrent services or applications. This is not considered in most existing middleware solutions
3. *Data Management*: A vast amount of raw data continuously collected needs to be converted into usable knowledge, which implies aggregated and filtered data. Most of the surveyed middleware offer support for data aggregation, but do not consider data filtering.
4. *Event Management*: A large number of events are generated proactively and reactively in IoT. Because of this, it is expected that middleware components may become bottlenecks in the system.
5. *Code Management*: Re-programmability is one of the major challenges not only in IoT, but also in software development. Updates or changes in business logic should be supported by any IoT component.

Challenges related to Non-Functional Requirements:

1. *Scalability*: Since most existing middleware (Table II) are WSNs centric, their network level scalability is also limited to WSNs. They will perform poorly in IoT's ultra-large-scale network.
2. *Real-time*: Applications and services rely on being directly connected to the physical world. Getting real-time information about the state of the real world is still a challenging task. Some middleware approaches are by nature non real-time (e.g., database or tuple-space middleware), while the rest provide at least soft real-time services. Hard real-time can be provided by application-specific middleware approach and a few event based middleware. Current middleware solutions need to consider real-time service composition or self-adaptivity.
3. *Reliability*: Reliability is not addressed in most existing proposals. To achieve middleware reliability, every component or service of a middleware needs to be reliable.
4. *Availability*: Maximizing system availability and fast recovery from failures are challenges that are not specific to IoT, but to any distributed system. In the context of IoT, availability of things and services offered is important. Hardware devices fail periodically and any service they provide will be unavailable when they fail.
5. *Security and Privacy*: All the concerns of security, privacy and trust in all the technologies (e.g., traditional Internet, WSNs, M2M communications, RFID, SCADA, and cloud computing) used in IoT are clearly present in the context of the IoT. Unfortunately, security, privacy and trust are not completely resolved in these technologies.
6. *Ease-of deployment*: Deployment, post-deployment, and re-programmability are important tasks in an IoT middleware lifecycle.

Challenges related to Architectural Requirements:

1. *Programming Abstraction*: Most middleware offers programming abstraction support. However, the new languages and tools that need to be adopted have a steep learning curve for developers and users.
2. *Interoperability*: Network interoperability is well supported by most existing middleware, but many lack support for semantic and syntactical interoperability. Semantic interoperability is very challenging in IoT because of heterogeneity and the lack of standard in ontologies. From all middleware categories, the service-oriented approach offers the best support for semantic interoperability. However, support for syntactic interoperability is limited.
3. *Service-based*: Most of the middleware are service-based. Each service needs to provide a description for service composition or discovery. A standard service description is mandatory to ensure semantic and syntactic interoperability.
4. *Adaptive*: In a number of approaches, adaptation decision making is hard-coded and requires recompiling and redeploying the system or a part of the system. Where adaptation is more dynamic, policies, rules or QoS definitions are used, which can be changed during runtime to create new behaviour. Even though most middleware uses a dynamic approach, the rules, policies and QoS definitions are mostly hard-coded and are not context-aware. In IoT, this approach is not scalable.
5. *Context-awareness and Autonomous behaviour*: Different types of middleware have exploited some level of context awareness. For instance, MUSIC [20] exploits context for self-adaptation to maintain a satisfactory QoS. Popular uses of context (e.g., context-aware resource discovery, context-aware composition, context-aware data management) are missing.

2.5 Available IoT Platforms

2.5.1 Cloud based Platforms

2.5.1.1 Cumulocity

Cumulocity allows customers to create their own service wrappers that means delivering their own services to their customers. It proposes service as a platform technology instead of using complex infrastructure. Cumulocity offers dashboard platform via contextualized data that can create KPI and reports to provide an insight to forecast. In context of architecture and component, sensor node integrated with platform uses methods for access and manipulations to communicate with RESTFULL HTTPS API. This API uses an agent to connect to Cumulocity server where commands are pulled and sent to related client applications. Depending on pull frequency, throughput latency and network delay might occur. This tool is only compatible with RESTful HTTP/S protocol [9]

2.5.1.2 ThingWorx

Thingworx is directly integrated with social media platforms and weather services. Among services, applications and sensors the data will be exchanged via a virtual bus, but this does not support peer to peer communication. The data transfer is provided via network protocols such as MQTT, REST/HTTP and CoAP and Web socket protocols. Mashup builders are correspondent to asset tracking, monitoring and creating dashboards with data retrieved from different agents [9].

2.5.1.3 Xievly

Xievly as two previous models mentioned above will include a bus layer to provide data transfer among the devices via different protocols The message bus integrated with Xively API and web sockets creates data interoperability layer that provides fine grain access for data feeds and data streams. Each device with serial id number is registered to the system and they can retrieve the setup configuration via mutual authentication [9].

2.5.2 Device to Device Platforms

2.5.2.1 IzoT

The architecture will be based on communication stack supporting peer2peer communication. The general use is more compatible with IIOT. Communication stack will help multiple communication for unconstrained devices. The authentication and authorization cases are guaranteed for this platform. IzoT also supports asymmetric and symmetric key encryption and authorization. Private key communication stack degrades Izot performance [9].

2.5.2.2 ThingSquare

The architecture relies on stacks supporting peer2peer communication. The general use is more compatible with IIoT. Communication stack will help to establish multiple communication for unconstrained devices. The authentication and authorization cases are guaranteed for this platform. IzoT also supports asymmetric and symmetric key encryption and authorization. Private communication stack degrades Izot performance [9].

2.5.2.3 Bluemix

Bluemix offers end to end enterprise solution for industry and consumer protocols consisting of Bluemix application providing data management and authorization, Websphere integration middleware supporting MQTT network, Message Sight application supports the network communication among sensors and middleware. MessageSight is able to handle high volume data and can transfer data at high rate speed [9].

2.5.2.4 Open MTC

The architecture consisting of different blocks which are front end block linked to sensors and actuators used for data retrieval and application platform, Back-End block connected to other M2M platform and another application platform covering various and multiple mobile applications such as e-health, smart grid etc. Data transfer between these two blocks will be handled via middleware platform, consisting core features and connectivity segment working as a message converter from front end applications. This part also implements data analysis and rules application via electronic product code segment (EPC). Connectivity component is in charge of device management and core function will provide seamless data communication. It covers protocol adapter (such as Zigbee, Profinet, OPC-UA, Robot OS), apps like load (generator, advanced dashboard, analytics), historical data handling [12].

2.5.2.5 FIWARE

FIWARE allows to connect all devices, to integrate the services and application and to analyze and to reveal the contextual information. By isolating network services and application layers among each other, it reduces network latency and increases scalability and modularity. It encompasses IoTAgents (IoTA), a Context Broker (Orion), Short Term Historic (STH), Connector Framework (Cygnus), Security Component. It contains enriched generic enabler (GE) libraries providing open interfaces for application developer and support interoperability among other GEs. GE is on charge of cloud hosting, data/context management, advanced based user interface. It consists of IoT edge concerned with communication IoT back-end platform and devices and data context broker which establishes intercommunication among application interfaces. This system also includes NGSI devices, interacting with data providers. NGS device, being system entities, helps to transfer the data through IoT broker [12].

2.5.3 ROBOTICS in Industrial Use

Modern robots used in the industry [13] can possibly modify the production mechanisms in the way computers have transformed the office workplace environment. Robots are used for their capabilities to accomplish tasks rapidly, accurately and continuously. They are relevant in various areas in the production industry and brought exceptional value to multiple construction mechanisms. For instance, the petrochemical industry uses robots to increase effectiveness and safety by also reducing the harmful impact towards the environment. In environments where is hazardous for people to work, robots come in handy because of their ability to provide maintenance and repairs. Still, there are some issues regarding trust and responsibility. The robots' purposes within the organizational structure should also be considered. In the end, every shared system presents network layer vulnerabilities. Several robots autonomously function, whereas some may be controlled remotely. Robots have to be sufficiently predictable to function under complicated and active environments with increased confidence levels and still have the capacity to be firmly controlled or instead, to be stopped by the operator, so that the following generation of users and administrators to confide in autonomy. Keeping up this adaptability in the future framework will deliver adequate levels of trust in the activities performed by our robotic correspondents. The people's reaction to extended levels of autonomy has to be considered. Considering that robots do not have enough autonomy, then the administrator will linger on taking care of robots instead of their work. Likewise, another ability should advance for future administrators, in order to repair and preserve robots in their environments. The autonomous capacities primary advantages are to broaden and supplement people's performances, not to fully substitute them. If robots have high autonomy levels, the awareness activities might decrease. Robots can enhance people's speed, perception, ingenuity, resistance to tiredness. Several robots will include the capacity to investigate and carry samples and others will do more refined activities such as repairs and preservation. Mobile robots such as unmanned flying vehicles have been produced to function during a disaster, or just for examinations.

Nanobots represent a branch within robots' field, being widely explored in the security and defense domains nowadays.

2.5.4 Nanobots

Nanobots represent a type of a little robot [14]. A nano-robot represents an artificial robot with the dimension of few micrometers or less, comprised from nanoscopic elements with distinctive sizes in the interval from 1 to 100 nm. A nanobot displayed the ability to travel freely through the hole human body circulatory system. In the future, these nanobots could be utilized in production systems, as in the provisioning of a microscopic view towards the situations crucial to specific bio-pharmaceutical or nuclear facilities. The idea of studying the state of fluid suspension with entire groups of nanobots could be explained in the bloodstream. A nanobot in a vein has shown the ability to feel the metabolic sequence of the family of cells fed by the vein itself, therefore examining the cells restrained within a given length of the tube. Each nanobot is a self-propelled device, taking energy from the environment, being able to identify and anchor to the elements within their process. They can sense membranes and consequently recognize the state of health of its environment. They also may be applied to collect the information, to transmit it to the central unit, and ultimately, take actions that might affect the overall process situations. Within an entire group of nanobots, each bot holds particular chemicals to be released for detection by other nanobots. This could also be used in a production facility to carry a message from one location to the other.

It is a difficult matter to ensure that nanobots and nanobot groups are working securely. Nanobots are by definition tiny and are therefore very challenging to monitor for unique malicious behaviors, especially if a massive group of nanobots is transmitted. If nanobots alone are programmed with software, how may someone scan the nanobot's operating code for malware? If nanobot groups are programmed with chemical means, would there be a means to assure that the use and control of the swarm not be taken over by a malicious entity, in the same manner, that bacteria and viruses affect the biological receptors? How the nanobot group will be supported and monitored? How nanobots are disposed, when they reach their end of life? As with other features of innovative IoT devices, nanobot systems provide great utility but have not been yet analyzed or developed for security and safety.

3 IoT-framework Standards and Enabling Technologies for IoT-systems

In this chapter a summary of the most relevant available standards and enabling technologies for IoT-Systems is presented. In particular, protocols that operate at the different levels of the IoT-stack are introduced by following the structure of the IoT-stack itself from a top-down perspective, i.e. from the Application Services Layer till the Device Layer. These standards were proposed over the past decade to meet IoT current and future needs, but not all of them are necessarily suitable for industrial applications.

In the following, the protocols OPC-UA, oneM2M and DDS for Application Services Layer are presented. Then, in Section 3.2 protocols for the Application Protocol Layer such as MQTT, CoAP, XMPP and AMQP are discussed. Although the analysis and implementation of the Network Layer is not the primary goal of the project, a review on exiting protocols is presented in Section 3.3. Finally, the Device Layer is discussed in Section 3.4.

3.1 Application Services Layer

3.1.1 OPC Unified Architecture (OPC-UA) [ifak]

OPC-UA is an M2M communication protocol specifically developed by the OPC foundation for industrial automation with the goal of combining all previous and/or existing protocols to a common (unified) data model in order to facilitate interoperability at different levels.

The protocol is specified by the IEC 62541 norm. Its main feature is to enable the exchange of information models of any complexity – both instances and types (metadata) – thus, realizing interoperability between machines at semantic level. It was designed to support a wide range of systems, ranging from PLC’s in production to enterprise servers, which are characterized by their diversity in terms of size, performance, platforms and functional capabilities. To this aim, OPC-UA was specifically developed considering explicitly following features:

- Being independent from the underlying Physical Layer, i.e. solutions like CAN, Profinet IO or industrial WiFi can be used for the actual data transfer.
- Being independent from the functionalities of the available operative system.
- Considering explicitly security communication aspects such that data manipulation can be prevented.
- Describing data semantically such that complex data types rather than simple bits or bytes can be easily exchanged.

The protocol supports two different type of communication paradigms: client/server as well as publish and subscribe, where the OCP-UA-Server is configured such that specific information is automatically delivered to OPC-UA-Clients that are interested in receiving certain information. Both solutions are also independent from the underlying Transport Layer and, depending on the application and performance that shall be realized, can be easily implemented over TCP, HTTP, HTTPs as well as UDP, AMQP and MQTT, by implementing different transport profiles.

Figure 17 gives an overview of the implementation of the OPC-UA-Stack in an industrial environment. As described in the figure, the standard specifies mainly two different paradigms: OPC-UA for Services (client/server) and OPC-UA for Message Model (public and subscribe). As the name suggests, the first paradigm is specifically thought for the realization of (web)services, where information is exchanged in XML- or JSON-format. This particular encoding makes the exchanged data easy to read and to process, but it can be poorly performing for industrial application that have restricted available resources.

The second paradigm, on the other hand, is specifically conceived for industrial automation systems. In this case, data are represented in a binary way such that the exchanged messages require less overhead and less resources in order to achieve higher system performance.

The OCP-UA-standard is nowadays still in evolution and subject to current and future standardization, in particular for industrial applications. Further information about it and its implementation can be found for instance in [21].

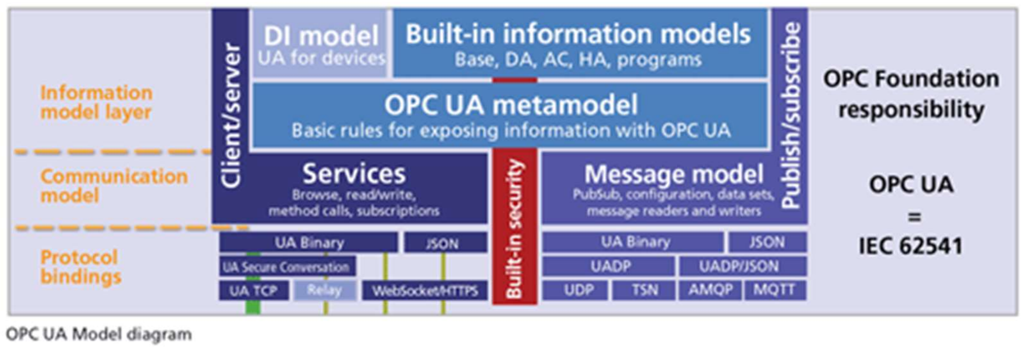


Figure 17: OPC-UA transport profiles.

OPC-UA over Time Sensitive Networks (TSN) [URO]

Although OPC-UA-Binary can provide higher performance than OPC-UA-XML, it does not directly take tight time constraints of real-time system into account. Therefore, it cannot guarantee network determinism for those industrial application that might require it.

To solve this issues, recent research has focus on the idea of implementing OCP-UA over Time Sensitive Networks (TSNs) or Time Sensitive Networking (TSN), [22]. TSN is the IEEE802.1Q defined standard technology to provide deterministic messaging on standard Ethernet networks. TSN is managed centrally and can deliver guarantees of delivery as well as minimize jitter by using time scheduling for real-time applications that require determinism.

The three main characteristic of TSN can be summarizes as follows:

- Time synchronization: all devices requiring real-time communication need to have a common understanding of time.
- Scheduling and traffic shaping: all devices share common rules w.r.t. how communication packets are processed and forward.
- Selection of communication paths, path reservation and fault-tolerance: all devices share common rules in selecting communication paths and in reserving time-slots, possibly providing path-redundancy to ensure fault-tolerance.

The IEEE802.1Q standard works at Data Link Layer (level 2 protocol) and it can be used in any kind of environment, i.e. it is not limited to applications that work over the Internet, but it can essentially also be used for industrial applications. Therefore, it possible to implement the OPC-UA protocol over TSN in order to guarantee determinism for those industrial applications that requires it.

Further information about OPC-UA over TSN can be found for instance in [23],[24] and [25]. Notice that the TSN standard is quite new and practical applications as well as commercial devices implementing this standard are still limited. This is also due to the fact that at the current state of the art, the components for TSN are highly expensive and the deployment of even small networks can sum up to a few thousand euros. Additionally, for the moment being, the standard is limited to cabled networks and no solution is currently provided for wireless ones. This partially limits the spreading of TSN-solutions, in particular for IoT applications in which wireless connectivity plays in many cases a crucial role.

3.1.2 oneM2M for the Industry Domain [ETRI]

oneM2M Overview

oneM2M is the global standards initiative that covers requirements, architecture, API specifications, security solutions and interoperability for Machine-to-Machine and IoT technologies. It has been formed in 2012 and consists of eight of the world's preeminent standards development organizations: ARIB (Japan), ATIS (U.S.), CCSA (China), ETSI (Europe), TIA (U.S.), TSDSI (India), TTA (Korea), and TTC (Japan).

The goal of oneM2M is to develop technical specifications which address the need for a common M2M Service Layer that can be readily embedded within various hardware and software, and relied upon to connect the myriad of devices in the field with M2M application servers worldwide. It also provides a framework to support applications and services such as the smart grid, connected car, home automation, public safety, and health. oneM2M actively encourages industry associations and forums with specific application requirements to participate in oneM2M, in order to ensure that the solutions developed support their specific needs.

oneM2M Architecture

oneM2M functional architecture consists of three types of entities (AE, CSE, NSE) and interfaces between each entity. AE means the “Application Entity”, which is a SW application operated by devices, gateways or user terminals. CSE refer to the “Common Service Entity”, which represents an instantiation of a set of "common service functions" of the M2M environments. NSE refer to the “Network Service Entity”, which provides services from the underlying network to the CSEs.

Each CSE can be located on an infrastructure node or a middle node. Typically, the infrastructure node (IN) is a cloud or remote server, and the middle node (MN) is the site gateway. ADN, which refers to Application Dedicated Node, is generally a device that cannot load CSE but can load only AE function. ASN, which refers to Application Service Node, is a device that can be loaded with CSE and has a service function to AE. The interface between CSE and AE is referred to as Mca. The interface between CSEs is referred to as Mcc.

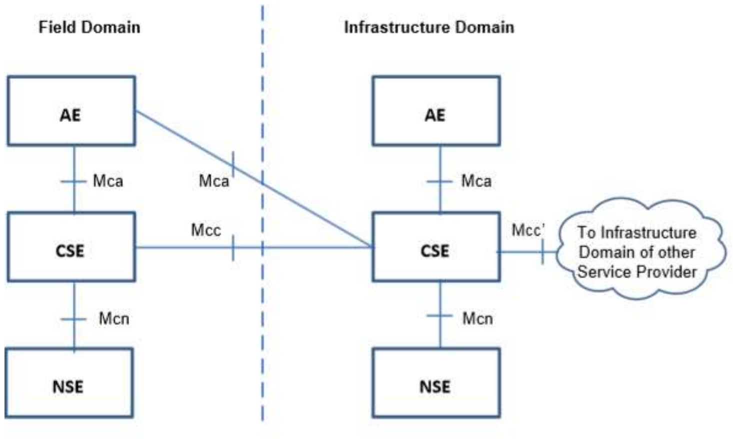


Figure 18: oneM2M functional architecture.

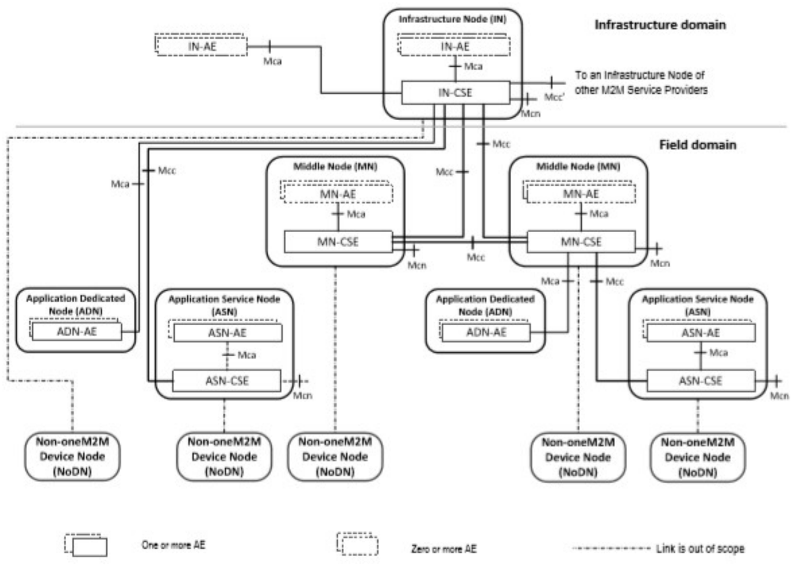


Figure 19: Configuration supported by oneM2M architecture.

Common Service Function (CSF)

CSF is referred to the services provided by the Common Services Layer in the M2M System and reside within a CSE. It provides services to the AEs via the Mca reference point and to other CSEs via the Mcc reference point.

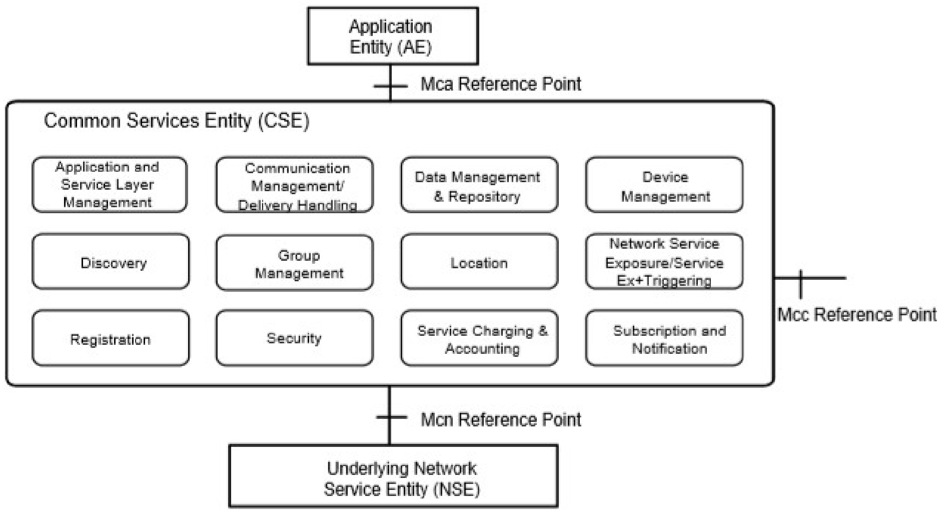

Application Entity (AE)
Mca Reference Point
Common Services Entity (CSE)
Application and Service Layer Management
Communication Management/ Delivery Handling
Data Management & Repository
Device Management
Discovery
Group Management
Location
Network Service Exposure/Service Ex+Triggering
Registration
Security
Service Charging & Accounting
Subscription and Notification
Mcc Reference Point
Mcn Reference Point
Underlying Network Service Entity (NSE)

Figure 20: Common Service Function (CSF)

Industrial domain architecture using oneM2M

oneM2M technology can also be applied to industry domains. oneM2M technology is considered to achieve the communication and interaction from machine-to-machine without human support. This technology provides opportunities to achieve synchronous, continuous and effective information exchange in manufacturing scenarios.

In an industrial domain architecture, a variety of devices within the plant can provide each other's capabilities via M2M communications. Each manufacturing services connect with factories via the M2M system. In addition, the complex service can be sent to several factories synchronously, to enable effective collaboration between factories

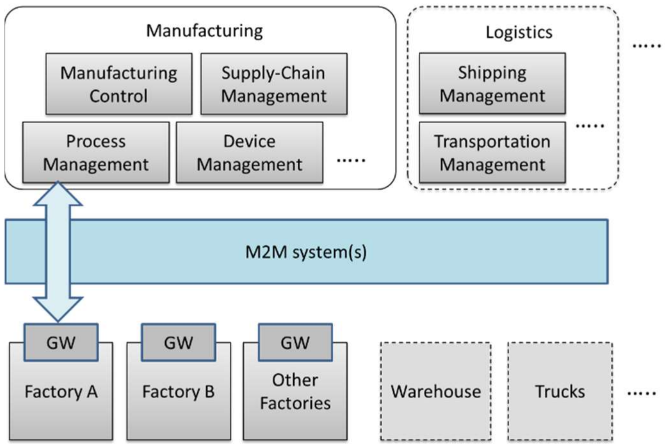


Figure 21: Industrial Domain Architecture.

In this document, we present three use cases in an industry environment using oneM2M.

oneM2M technology can also be applied to industry domains. The first case is “On-demand Data Collection for Factories”. In order to collect every real time data, oneM2M gateway need to be interfaced with industrial bus system (Real-time Ethernet; IEC TC 65). But, it is difficult to gather all data. oneM2M gateway can handle the necessary data depending on user policy (rule based, pre-processing/filtering); data catalogues. Figure 22 shows the structure applied to the data collection of factory environment by applying oneM2M.

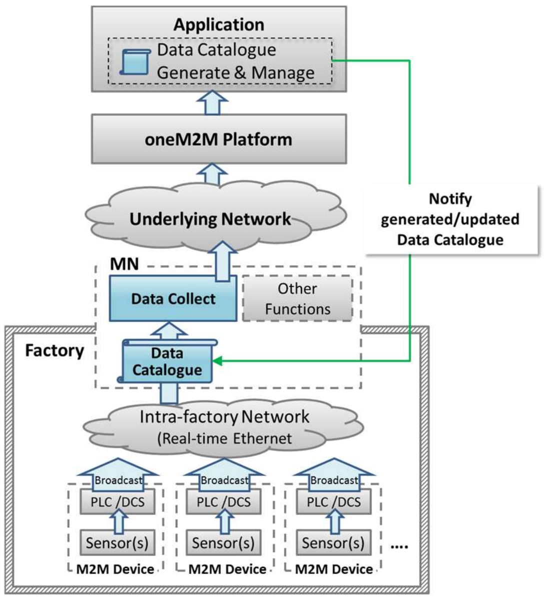


Figure 22: High-level Illustration of On-demand Data Collection for Factories

The second use case is “Data Process for Inter-Factory Manufacturing”. To achieve remote manufacturing, a significant amount of data is generated for monitoring purposes and broadcast via an intra-factory network (e.g. real-time Ethernet) through PLCs or Distributed Control Systems (DCS), etc. For monitoring product lines efficiently and effectively, Middle Nodes (MNs) (which means the gateway) will selectively collect necessary data from an intra-factory network and then send this data to the oneM2M services platform for use by manufacturing control applications. Figure 23 shows the structure in which data collected in each plant environment is processed at the MN level and forwarded to the server.

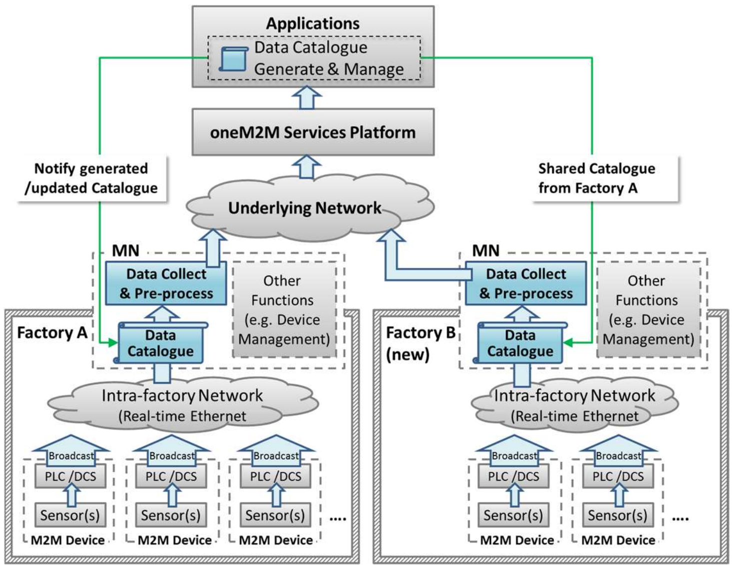


Figure 23: High-level Illustration of Data Process for Inter-factory Manufacturing

The last use case is “Realtime Data Collection”. To achieve adequate control, real-time Ethernet, with which sensors and devices are connected through controllers, are required to provide real-time transmission and a high level of reliability. The oneM2M MN shall be able to transmit data according to priority in preparation for temporal performance degradation of underlying network, and for temporal increase of the amount of information data. Figure 24 shows the structure for real-time data collection. For real-time data processing, a plurality of MNs are required, and each MN needs to include a buffering function for data processing.

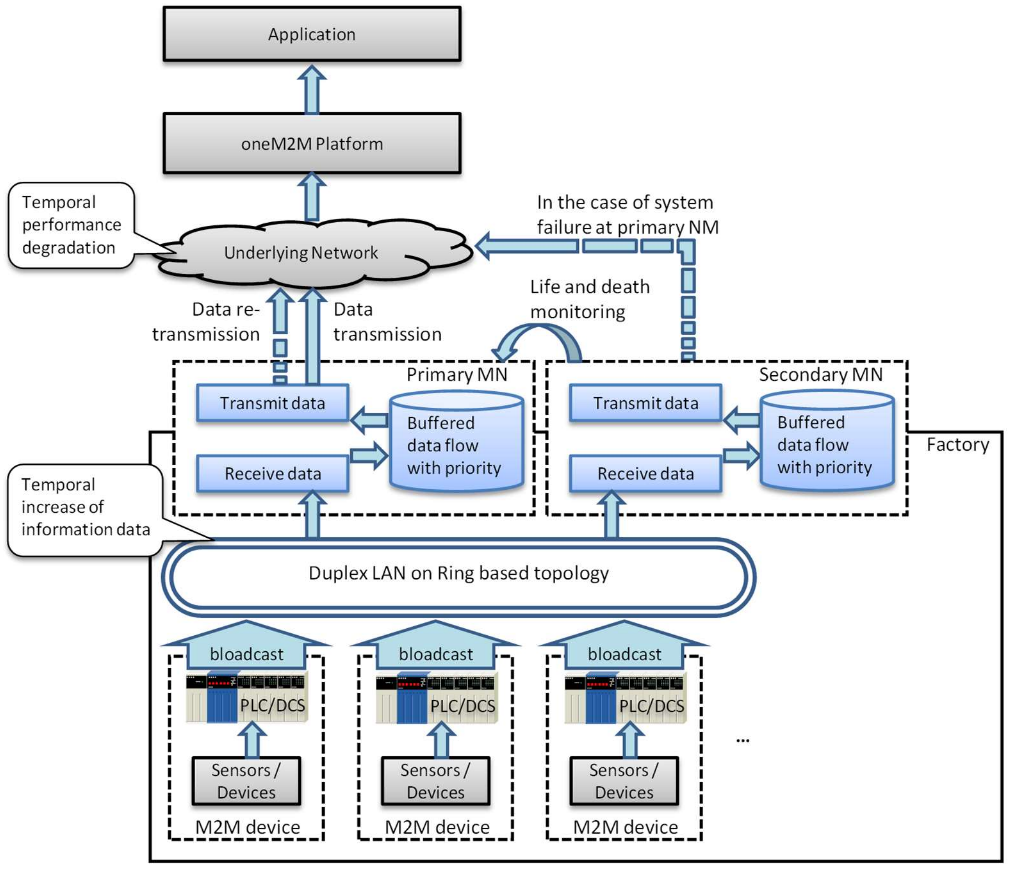


Figure 24: High-level Illustration of Real Time Data Collection

3.1.3 Data Distribution Service (DDS) [URO]

DDS is a content-based publish-subscribe communication protocol for Machine-to-Machine (M2M) applications. It is a decentralized implementation of the publish/subscribe concept that uses, among others, multicasts to distribute the messages instead of a centralized broker. For this reason, DDS is referred to as real-time capable and suits well for one-to-many as well as many-to-one applications. However, the standard also defines the client-server pattern as an alternative. Furthermore, it focusses on Quality of Service (QoS) and reliability by introducing several parameters, e.g., for data availability, delivery, timeliness and security. The protocol offers built-in security mechanisms like data confidentiality and integrity, authentication and authorization of publishers (data writer) and subscribers (data reader). DDS is a data-centric approach and introduces the concept of a virtual global data space (see Figure 25). Using this concept, a device can access information of all devices in the DDS network comparable to getting data from a local data storage. In this way, the data exchange looks from an application perspective similar to accessing local system memory. The DDS API translates these virtual data read and write commands into real-world actions because the data is stored on all devices across the network. Due to the content-based approach, complex filter queries are possible to receive only the required data.

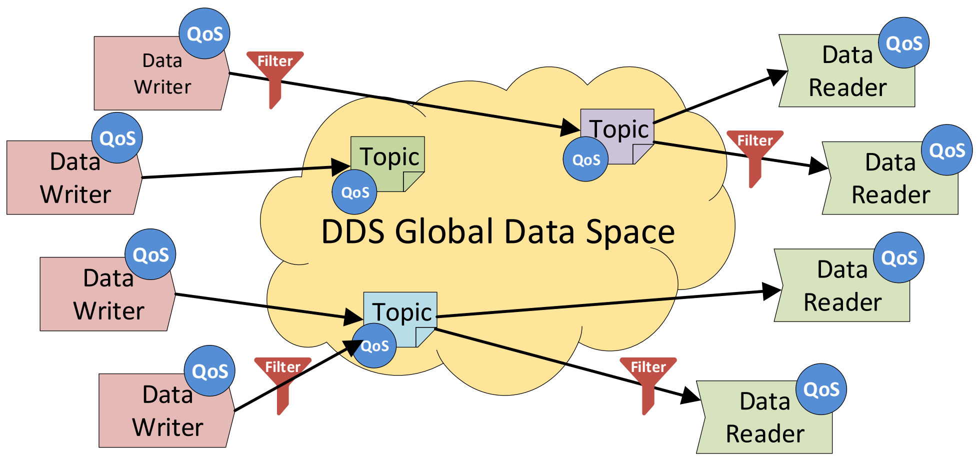


Figure 25: DDS Global Data Space Concept, derived from portals.omg.org

Furthermore, DDS offers an interoperability layer that makes it independent of the underlying layers (DDSI-RTPS, see Figure 26). In this way, it supports communication over, e.g., TCP and UDP as well as shared memory. Currently, solutions that use DDS as the communication layer for OPC UA are also subject of research.

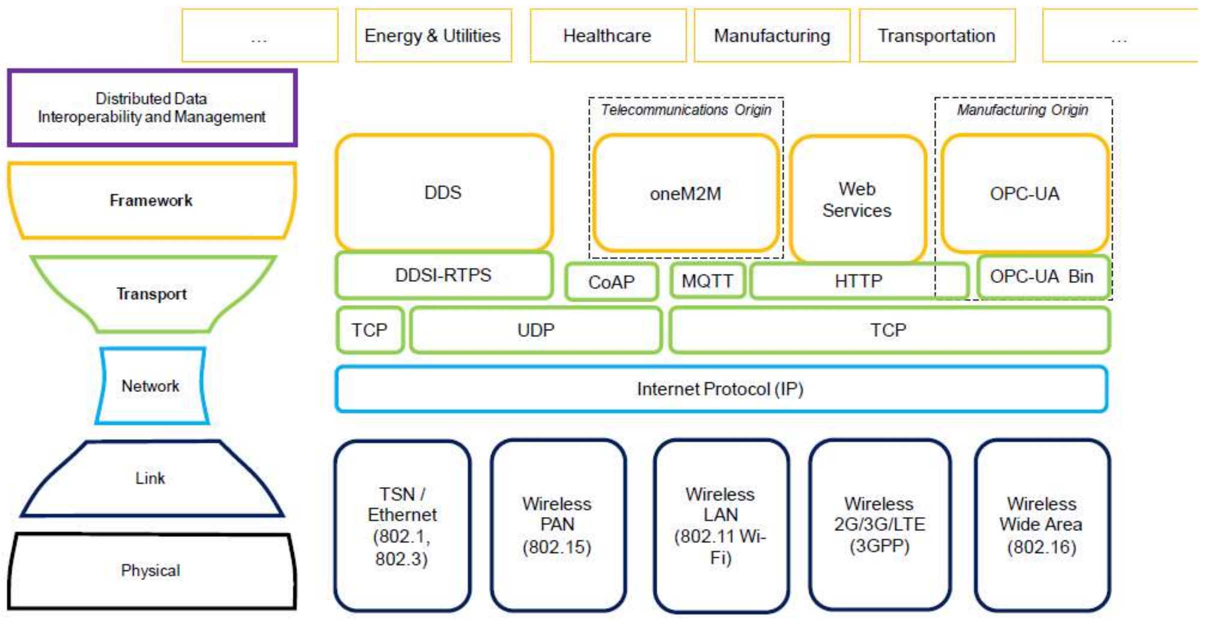


Figure 26: DDS and other Protocols in the Network Stack [27]

Further information about DDS can be found inter alia in [26] and [27].

3.2 Application Protocol Layer

3.2.1 Message Queue Telemetry Transport (MQTT) [URO]

MQTT is a lightweight machine-to-machine (M2M) protocol for message transport. It is one of the currently most popular publish-subscribe (pub/sub) solutions and was initially developed for (many-to-one) data collection.

This protocol is a centralized pub/sub implementation, standardized by the Organization for the Advancement of Structured Information Standards (OASIS) as a protocol for the IoT, with topic-based filtering and a dedicated broker (Figure 27). Furthermore, it uses the connection-oriented transport protocol TCP and TLS/SSL for secured communication. Independent of reliability mechanisms in TCP, MQTT offers three Quality of Service (QoS) levels listed in Table 1 to ensure the successful transmission of messages. These levels can be set individually for each topic, publisher-broker and broker-subscriber relation.

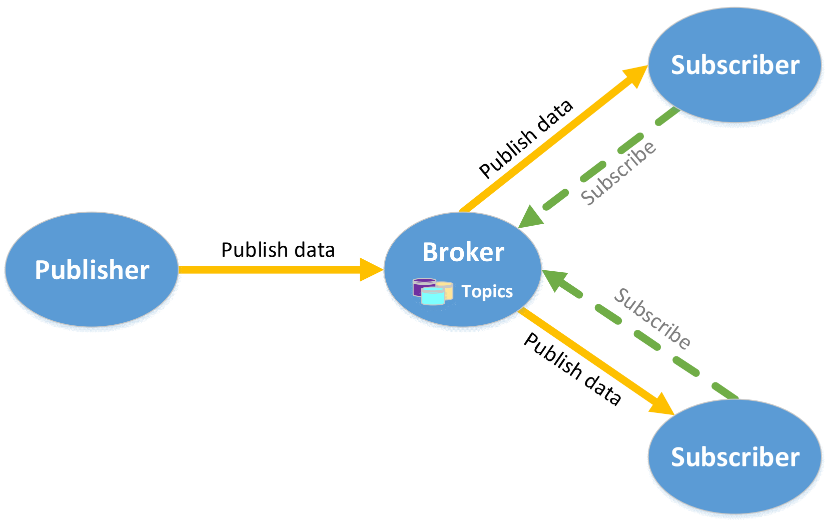


Figure 27: MQTT Publish-Subscribe Pattern

An advantage of MQTT is the centering of complex logic in the broker while the publishers and subscribers can be simple and lightweight applications running on resource-constrained devices.

Table 1: MQTT QoS Levels

QoS Level	Description
0 (at most once/ fire and forget)	Best-effort delivery, no acknowledgement (ACK) from receiver, no re-transmission on application layer, only reliability mechanisms of TCP used
1 (at least once)	A message is at least once delivered to a receiver, receiver sends ACK message, multiple message delivery possible
2 (exactly once)	Message is delivered exactly once by using a four-part handshake, slowest but safest level, e. g., necessary if duplicate messages can disrupt receiver program flow

There exists a derivate of MQTT called MQTT-SN for sensor networks. While MQTT requires a lossless connection via TCP/IP, MQTT-SN was developed to be agnostic of the underlying network protocols. Usually, this derivate is used on top of the connection-less protocol UDP that causes less communication overhead at transport layer due to missing reliability mechanisms. For resource-constrained embedded devices, e. g., nodes in sensor networks, the additional energy cost of a reliable connection-based transmission (TCP) can have a major impact on battery life. One of several reasons for this problem is the communication pattern of sensor nodes. In general, such a device remains for a long time in a low power state (sleep mode) and wakes up only by interrupt of an external trigger or timer. After interrupt, the device sends only one or a few messages and goes back to sleep again. When using a TCP-based communication, additional management messages need to be sent besides the data messages due to connection establishment, maintenance and teardown. After each sleep phase, the connection is again established with the same communication overhead (see section 3.2.2 for more details). In comparison, using connectionless protocols, only the necessary data messages are transmitted. Besides the change in transport layer protocol, a major difference to normal MQTT is the requirement of a gateway between publisher/subscriber and broker (Figure 28). The devices running a publisher or subscriber use a connection-less protocol like UDP. However, the broker is still based on TCP communication. Therefore, a gateway needs to translate between the communication partners. Furthermore, an additional QoS level (-1) is introduced to simplify the publishing process of data for resource-constrained devices.

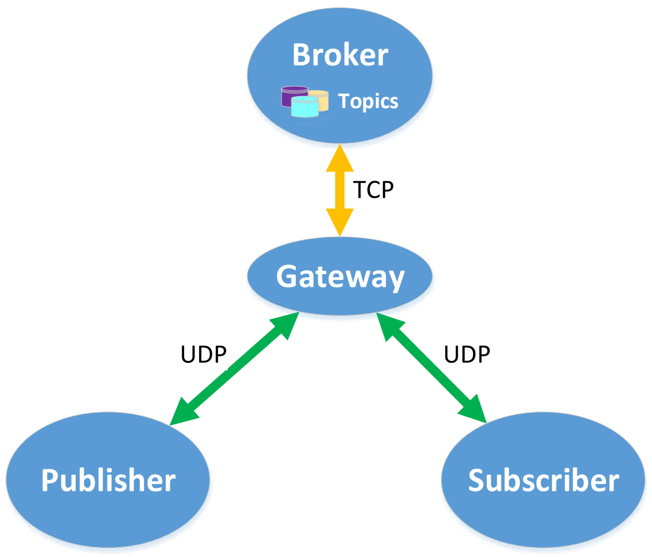


Figure 28: MQTT-SN Publish-Subscribe Pattern

Further detailed information about the MQTT protocol can be found in [28] and [29].

3.2.2 Constrained Application Protocol (CoAP) [URO]

CoAP is a representative of the Representational State Transfer (RESTful) concept suitable for distributed embedded systems due to its decentralized architecture. REST protocols apply the client-server pattern. Clients establish connections to servers. A connection initiation by the server is not provided. Furthermore, connections between servers and clients are stateless. In this way, requests from a client are processed on server side independently of the previous communication. Thus, individual connections are independent of each other. In order to keep a communication history, it is possible for each client to cache the data. The RESTful architecture enables the independent development of individual services, which together form a large architecture. Another feature of such protocols is the ability to run as multi-layered systems. In this case, a client sends a request to a server. If the server needs further information to answer this client message, it sends a request to other servers. The transmission of executable code from server to client is also supported (code-on-demand). To assign addresses to services in a RESTful architecture, the Uniform Resource Locator (URL) is used. A service can offer multiple resources. To address the individual resources, the Uniform Resource Identifier (URI) is used.

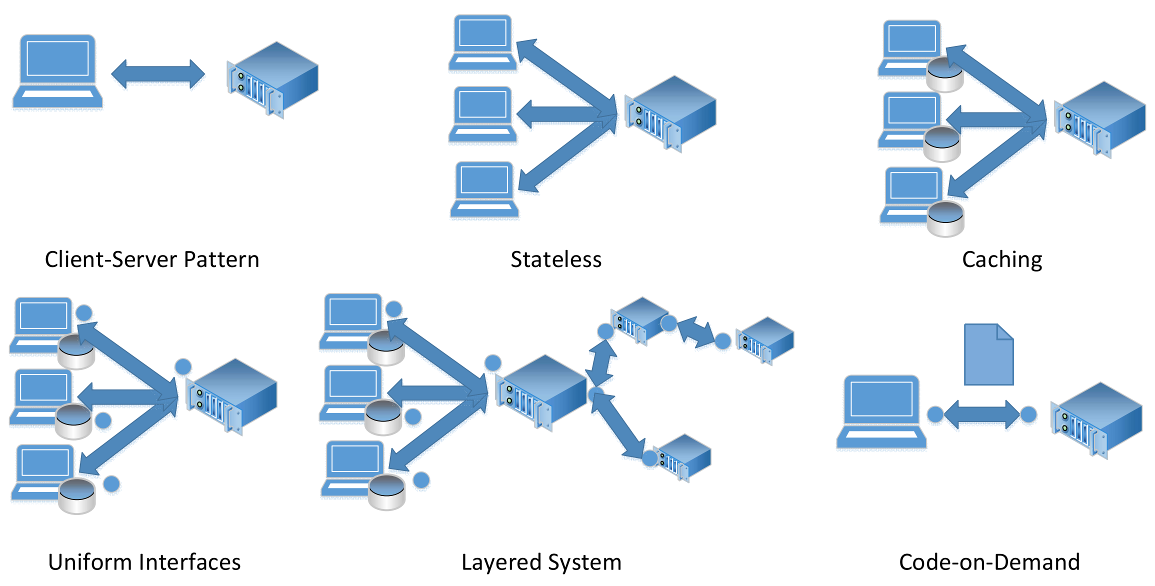


Figure 29: Features of a RESTful Architecture

CoAP is very similar to Hyper Text Transfer Protocol (HTTP). However, it has a smaller header than HTTP because it uses binary encoding. In addition, as shown in Figure 30, CoAP is based on UDP instead of TCP (HTTP). If an HTTP client wants to send a request to an HTTP server, a connection must first be established via TCP. The TCP handshake needs three messages before the application can exchange messages over HTTP. Each TCP connection is terminated by another three-way handshake. With UDP, however, no connection is established. The data is immediately sent to the server via UDP. Thus, CoAP is particularly suitable for micro devices, since fewer and shorter messages are sent in comparison with HTTP.

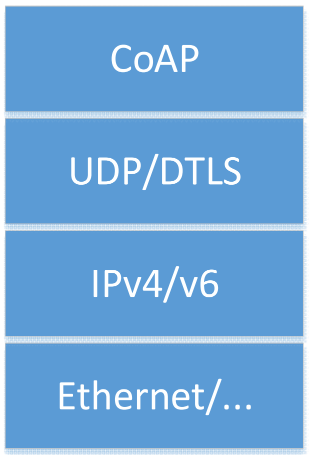


Figure 30: CoAP Protocol Stack

Each CoAP server provides resources to clients accessing the resources through CRUD operations. CRUD operations are Create, Read, Update, and Delete. These operations are also supported by HTTP. Due to the strong similarity of CoAP and HTTP, proxies can be implemented that translate between both protocols. CoAP defines a mandatory resource (well-known/core) that every server has to offer. After requesting this resource, the client gets an overview of all resources of the server. Another property of CoAP is the asynchronous eventing mechanism. Clients subscribe to a resource on a server. The server stores all subscribers and sends notification of a resource change.

CoAP messages are based on UDP and therefore have no security features such as encryption and integrity. The CoAP standard defines the Datagram Transport Layer Security (DTLS) protocol for the aspect of communication security. DTLS encrypts the data, so the confidentiality of the information is ensured. In addition, data secured with DTLS can be checked for change during transmission (message integrity). DTLS also ensures message authenticity. The sender of a message can be identified by, e. g., the use of symmetric keys and certificates. A certificate includes a public key and meta information about the device. Server and client certificates ensure bidirectional authenticity.

On the network layer, CoAP supports IPv4 and IPv6 as well as the 6LoWPAN set of underlying layers (IPv6, 6LoWPAN Adaption Layer, IEEE 802.15.4) for constrained environments.

More information about the CoAP standard can be found for instance in [30].

3.2.3 Extensible Messaging and Presence Protocol (XMPP) [URO]

XMPP is a simple instant messaging standard that was developed for near real-time text, audio and video chatting. It uses TCP and is a partial decentralized protocol that supports both communication patterns, client-server and publish-subscribe. Looking at the communication structure, it is located between MQTT (centralized) and CoAP (decentralized). Clients need always a server as a relay to exchange messages with each other, but multiple servers can be interconnected to form a decentralized network. XMPP uses the human readable Extensible Markup Language (XML) as message format. Due to the large message payload caused by XML encoding, the extension Efficient XML Interchange (EXI) is currently under development for XMPP to reduce this communication overhead. Furthermore, it supports extensions to customize the protocol for application requirements. XMPP can be combined with TLS and Simple Authentication and Security Layer (SASL) to support security features, but end-to-end security was no goal during design time.

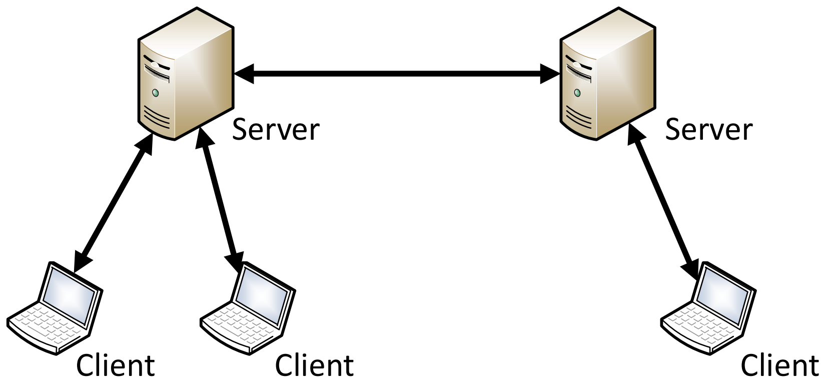


Figure 31: Distributed Architecture of XMPP

Additional information about the XMPP Core Standard are available in [31].

3.2.4 Advanced Message Queuing Protocol (AMQP) [URO]

AMQP is a binary communication protocol using a centralized publish-subscribe pattern with broker instances. Initially developed in the context of financial services, the protocol became in 2014 an open standard (ISO/IEC 19464) and is currently used in a wide range of application domains. It supports self-describing data encoding and reliable message exchange using QoS primitives similar to MQTT (see 3.2.1). To realize such delivery guarantees, a reliable transport layer like TCP is required. The self-description concept in AMQP can be extended with additional meta-data during runtime. This allows use case specific type interpretations and also the extension of messages with additional information during transmission through a network, e. g., by gateways and brokers.

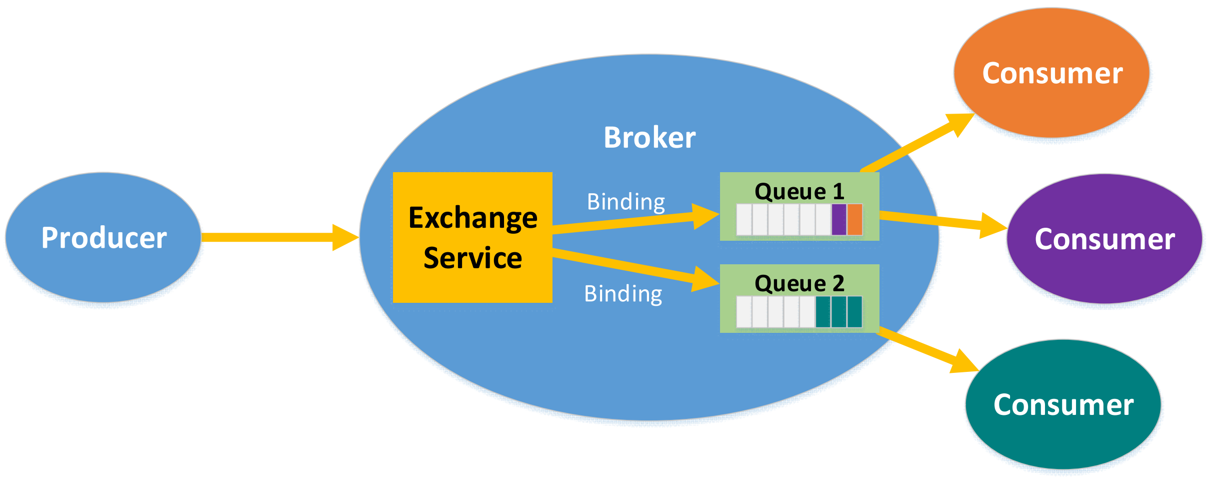


Figure 32: AMQP Communication Model

Similar to MQTT, producers send messages to a broker and these packets are delivered to consumers. The difference to MQTT is the internal structure of the broker. It consists of an exchange module and message queues. Published messages are directly sent to the exchange service and based on bindings (predefined rules), the packets are distributed to specified FIFO-queues. All consumers associated with these queues can obtain the stored messages on request or in a subscription relation. To interconnect producer and consumer, the name of an exchange service is necessary. Several brokers/exchange services are possible in a network. The way of how to forward the published messages to the message queues can be specified by exchange types that are listed in Table 2.

Table 2: Exchange Types of AMQP

Exchange Type	Description
Direct	Content of message header field 'routing key' needs to match exactly the 'binding key' of a message queue. The message is forwarded to this queue.
Topic	A message is distributed to all queues that have a complete or matching key compared to the message key. Realized by wildcard symbols: #, *
Fanout	Keys are ignored, messages are delivered to all queues attached to the exchange service.
Header	Keys are ignored, header fields are used instead. Therefore, key-value pairs are used to find matching queues.

In a case where multiple consumers use the same message queue, as shown in Figure 32, the messages are only delivered once to one of the consumers. In this way, e. g., tasks can be distributed among several clients with the guarantee that a message is only processed by one client. AMQP supports security by using TLS and Simple Authentication and Security Layer (SASL). Further information about the protocol and its specifications can be found in [32], [33] and [34].

3.3 Network Layer [Thorsis]

In this section, some of the currently available network technologies for communication are presented, with particular focus on Datalink, Routing and Encapsulation Layers. These protocols are essentially responsible, together with the Physical Layer, for the actual networking of the system components. Notice, though, that most of the protocols presented in this section rely on an Internet and on a Transport Layer protocol for the end-to-end communication (see Table 3 for more details), which are typically IP and UPD/TCP. A complete analysis of available solutions for the Physical Layer is on the other hand outside the scope of this document.

The Datalink Layer provides Medium Access Control (MAC) and node-to-node data transfer, i.e. it creates a link between two nodes that are directly connected through a physical medium. The main responsibility of this layer is to guarantee access to the lower physical layer, as well as detect and correct, when possible, errors that may occur at lower level. Functions for flow control are typically implemented here. Some of the most common protocols which have gained quite success in the industrial fields are summarized in the following.

Additionally, some standard and non-standard protocols for routing and encapsulation for IoT-applications are presented. Routing protocols provide mechanisms to handle the packet transfer from source to destination. On the other hand, encapsulation protocols form the actual packets that are sent through the network. One of the problems in IoT-applications is that all components of the IoT-system need to be addressed. Typically, this is done by using standard IPv6-addresses, which are too long to fit in most of the IoT datalink frames that require much smaller packet overhead. In the following, some of the available protocols for encapsulation are presented.

Table 3: Relation between the IoT-protocol-stack (orange) and commonly available protocols w.r.t the ISO-OSI-model.

Application Layer		
App. Services Layer		OPC-UA, oneM2M, DDS, …
App. Protocols Layer		MQTT, SMQTT, DDS, CoAP, XMPP, …
Network	Transport	TCP,UDP, DCCP, …
	Internet	IP, ICMP, ECN, …
	Encapsulation	6LowPAN, 6TiSCH, 6Lo, …
	Routing	RPL, CORPL, CARP, …
	Datalink and Physical	WiFi, BLE, LTE-A, …
Device Layer		

3.3.1 IEEE 802.15.4 and IEEE 802.15.4e

IEEE 802.15.4 is a low data rate wireless connectivity solution with focus on very low complexity and extended battery lifespan that is in the range of multiple months to multiple years. Potential applications of the solution include sensors, interactive toys, smart badges, remote controls, and home automation. IEEE 802.15.4 is a simple packet-based radio protocol aimed at very low-cost, battery-operated devices that can intercommunicate and send low-bandwidth data to a centralized device. The protocol supports data rates of 1 Mbps, 850, 250, 100, 40, and 20 kbps. The data rate is dependent on the operating frequency as well as on the coding and modulation scheme. The transmission range varies from tens of meters up to 1 km, the latter introduced with IEEE 802.15.4g. The protocol is fully acknowledged for transfer reliability. The basic frame size is limited to 127 bytes in the original specification, and the philosophy behind that is two-fold: to minimize power consumption and to reduce the probability of frame errors. However, with IEEE 802.15.4g, the maximum frame size is increased to 2047 bytes.

The IEEE 802.15.4 standard only defines the functions of the physical and MAC layers. It serves as the foundation for several protocol stacks, some of which are non-IP, including Zigbee, Zigbee RF4CE, Zigbee Pro, WirelessHART, ISA100.11a, and RPL.

IEEE 802.15.4e is the next-generation 802.15.4 wireless mesh standard. It aims to improve on its predecessor in two focus areas: lower energy consumption and increased reliability. The standard introduces a new MAC layer to 802.15.4 while maintaining the same physical (PHY) layer. Hence, it can be supported on existing 802.15.4 hardware. Two key capabilities are added: time synchronization and channel hopping, hence the acronym TSCH. Time synchronization addresses the requirement for better energy utilization whereas channel hopping aims at increasing the reliability of communication

3.3.2 IEEE 802.11ah (Low Energy WiFi)

IEEE 802.11ah is a light (low-energy) version of the original IEEE 802.11 wireless medium access standard that defines a WLAN-system operating at sub-1 GHz license-exempt band. Due to the favorable propagation characteristics of the low frequency spectrum, this protocol is capable of providing improved transmission range compared to conventional WiFi. Additionally, in order to meet IoT-requirements, it was designed with a reduced overhead in comparison to the original WiFi. It also benefits from lower energy consumption, thus, allowing the realization of large networks of devices, e.g. sensors that cooperate to share signals. At the current state of the art, this protocol is the main Bluetooth competitor, since it can provide at the same time high data rates and wide coverage range.

3.3.3 WirelessHART

WirelessHART is a datalink protocol that operates on the top of IEEE 802.15.4. It provides a secure and reliable MAC protocol that uses encryption to cypher exchanged messages and checks their integrity in order to guarantee a high reliability level. Similar to other protocols, it used Time Division Multiple Access (TDMA) to provide fair share of the underlying communication medium.

In 2010, WirelessHART was approved by the International Electrotechnical Commission (IEC) unanimously, making it the first wireless international standard as IEC 62591.

3.3.4 Bluetooth Low Energy (BLE)

Bluetooth Low Energy (BLE) – also called Bluetooth Smart – is a short-range wireless communication protocol widely used for WPANs. Its main goal is to provide considerably reduced power consumption and costs, while maintaining communication ranges similar to Bluetooth. In comparison to it, the power consumption can be up to ten time less; its latency though can increase up to 15 times compared to classic Bluetooth. Notice that BLE is not compatible with classic Bluetooth. However, since both standards use the same radio frequencies, they can share the same physical antenna, which allows for the development of dual-mode devices that can operate with both BLE and classic Bluetooth.

The protocol follows a client/server architecture, where a client is a device that requests pieces of information, e.g. a smart-phone or a computer, and a server is a device that provides information, e.g. sensors. Thanks to an advertising mechanism, data are transferred only when necessary and/or available, such that considerably less energy for transmission is consumed. Additionally, nodes are usually awake only when they are communicating and they go to sleep otherwise to save their power

3.3.5 Zigbee

ZigBee smart energy is an IEEE 802.15.4-based specification for a suite of high-level communication protocols used to create WPANs with small, low-power digital radios, e.g. for home automation, medical device data connection, and low data rate, as well as close proximity. It supports a wide range of network topologies including star, peer-to-peer, or cluster-tree. A coordinator controls the network and is the central node in a star topology, the root in a tree or cluster topology and may be located anywhere in peer-to-peer.

ZigBee standard defines two stack profiles: ZigBee and ZigBee Pro. These stack profiles support full mesh networking and work with different applications allowing implementations with low memory and processing power. ZigBee Pro offers more features including security using symmetric-key exchange, scalability using stochastic address assignment, and better performance using efficient routing mechanisms.

Its low power consumption limits the transmission distance to approximately 10-100 m line-of-sight, i.e. the only way to pass data over longer distances is to use intermediate devices to reach the more distant ones.

3.3.6 LTE-A

Long-Term Evolution Advanced (LTE-A) is a set of standards designed to fit M2M communication and IoT applications in cellular networks. LTE-A is a scalable, lower- cost protocol compared to other cellular protocols. It uses OFDMA (Orthogonal Frequency Division Multiple Access) as a MAC layer access technology, which divides the frequency into multiple bands, which can be used independently. Notice that LTE-A is not the same as LTE. In fact, LTE is an actively developing technology, whose evolution was planned over a long time interval. LTE-A brings step by step toward the realization of 5G, which at the current state of the art is still on the specification phase. How long this migration process to a fully available 5G network will take is still an open question, which is difficult to answer at this point. It is though hard to imagine that this technology will be commercially available before the next 5 to 10 years.

3.3.7 RPL

Routing Protocol for Low-Power and Lossy Networks (RPL) is a distance-vector protocol that can support a variety of datalink protocols, including the ones discussed in the previous section. It builds a Destination Oriented Directed Acyclic Graph (DODAG) that has only one route from each leaf node to the root in which all the traffic from the node will be routed to. At first, each node sends a DODAG Information Object (DIO) advertising itself as the root. This message is propagated in the network and the whole DODAG is gradually built. When communicating, the node sends a Destination Advertisement Object (DAO) to its parents, the DAO is propagated to the root and the root decides where to send it depending on the

destination. When a new node wants to join the network, it sends a DODAG Information Solicitation (DIS) request to join the network and the root will reply back with a DAO Acknowledgment (DAO-ACK) confirming the join.
RPL is probably the most commonly used standard protocol for IoT-applications.

3.3.8 CORPL

CORPL – cognitive RPL – is a non-standard extension of RPL which is designed for cognitive networks and uses DODAG topology generation but with two new modifications to RPL. CORPL utilizes opportunistic forwarding to forward the packet by choosing multiple forwarders (forwarder set) and coordinates between the nodes to choose the best next hop to forward the packet to. DODAG is built in the same way as RPL. Each node maintains a forwarding set instead of its parent only and updates its neighbor with its changes using DIO messages. Based on the updated information, each node dynamically updates its neighbor priorities in order to construct the forwarder set, [35].

3.3.9 CARP

Channel-Aware Routing Protocol (CARP) is a distributed routing protocol designed for underwater communication, which can be used for IoT-application thanks to its lightweight packets. It considers explicitly link quality, which is computed based on historical successful data transmission gathered from neighboring sensors, in order to select the forwarding nodes. There are two scenarios: network initialization and data forwarding. In network initialization, a HELLO packet is broadcasted from the sink to all other nodes in the networks. In data forwarding, the packet is routed from sensor to sink in a hop- by-hop fashion. Each next hop is determined independently.
The main problem with CARP is that it does not support reusability of previously collected data. In other words, if sensor data are required only when they change significantly, then the CARP protocol is not particularly beneficial to this specific application due to the implemented data forwarding mechanism. An enhancement of CARP was done in E-CARP by allowing the sink node to save previously received sensory data, thus, reducing the communication overhead considerably, [36].
Although explicitly designed for IoT-applications, CARP has been not standardized yet and it was proposed just in literature.

3.3.10 6LoWPAN

6LoWPAN stands for IPv6 over Low power WPAN. It is probably the most commonly used standard for encapsulation, since it is capable of efficiently enclose IPv6 long headers in IEEE802.15.4 small packets, which cannot exceed 128 bytes. The specification supports different length addresses, low bandwidth, different network topologies, power consumption, low cost, scalable networks, mobility, unreliability and long sleep time.
Additionally, this standard provides header compression to reduce transmission overhead, fragmentation to meet the 128-byte maximum frame length in IEEE802.15.4, and support for multi-hop delivery. Frames in 6LoWPAN use four types of headers: No 6loWPAN header (00), Dispatch header (01), Mesh header (10) and Fragmentation header (11). In No 6loWPAN header case, any frame that does not follow 6loWPAN specifications is discarded. Dispatch header is used for multicasting and IPv6 header compressions. Mesh headers are used for broadcasting; while Fragmentation headers are used to break long IPv6 header to fit into fragments of maximum 128-byte length.

3.3.11 6TiSCH

The 6TiSCH group of the Internet Engineering Task Force (IETF) is working on developing standards to allow IPv6 to pass through Time-Slotted Channel Hopping (TSCH) mode of IEEE 802.15.4e datalinks.
The basic idea behind this protocol is to define a channel distribution usage matrix consisting of available frequencies in columns and available time-slots for network scheduling operations in rows. The resulting matrix is distributed to all the network nodes and, therefore, it is globally known to all, such that nodes within the same interference domain can negotiate their scheduling.
In this way, scheduling becomes an optimization problem where time slots are assigned to a group of neighboring nodes sharing the same application. This protocol is relatively new and still not completely standardized yet. A second official draft was proposed at the beginning of 2018.
In its current version, the protocol does not specify how the scheduling shall be done. This decision is left to the developer, in order to choose the proper solution for the specific realized application and provide the necessary flexibility for IoT-systems. Therefore, the scheduling can be either centralized or distributed, [37].

3.3.12 6Lo

IPv6 over Networks of Resource-constrained Nodes (6Lo) working group in IETF is developing a set of standards on transmission of IPv6 frames on various datalinks. Although, 6LowPAN and 6TiSCH, which cover IEEE 802.15.4 and IEEE 802.15.4e, were developed by different working groups, it became clear that there are many more datalinks to be covered and so 6Lo working group was formed. At the time of this writing most of the 6Lo specifications have not been finalized and are in various stages of drafts. For example, IPV6 over Bluetooth Low Energy Mesh Networks, IPv6 over IEEE 485 Master-Slave/Token Passing (MS/TP) networks, IPV6 over DECT/ULE, IPV6 over NFC, IPv6 over IEEE 802.11ah, and IPv6 over Wireless Networks for Industrial Automation Process Automation (WIA-PA) drafts are being developed to specify how to transmit IPv6 datagrams over their respective datalinks, [38]. Two of these 6Lo specifications “IPv6 over G.9959” and “IPv6 over Bluetooth Low Energy” have been approved as RFC.

3.3.13 IPv6 over BLE

[39] provides the specifications for IPv6 over BLE. It reuses most of the 6LowPAN compression techniques. However, since the Logical Link Control (LLC) and Adaptation Protocol (L2CAP) sublayer in Bluetooth already provides segmentation and reassembly of larger payloads into 27 byte L2CAP packets, fragmentation features from 6LowPAN standards are not used in this protocol. Another significant difference is that BLE does not currently support formation of multi-hop networks at the link layer. Instead, a central node acts as a router for the lower-powered peripheral nodes.

As it might be noticed from the amount of available standard and non-standard protocols, IoT-solutions are quite fragmented. It is difficult to find an overall solution that is capable of covering the requirements of heterogeneous IoT-systems. In particular, most of the available solutions are designed to be used with low-power networks, e.g. distributed sensors, with strong energy and bandwidth limitation, which might operate over a large range, but are not explicitly thought for industrial applications (IIoT), where sensors, actuators and distributed control strategies are used.

3.4 Device Layer [EZERIS]

In this section, the components that make up IoT devices - also referred to as “smart things” - will be explained. along with some of the most common protocols within these systems. The device layer of an IoT architecture is the foundation from which all other layers are built upon. Generally speaking, an IoT system is deployed usually when a network of sensors, actuators, or interconnected and independent subsystems are necessary to reach a goal. In this regard, the IoT devices that make up this system compose the Device Layer of the infrastructure, and is the first point-of-contact between the environment to monitor and the digital system.

3.4.1 IoT Device

IoT devices are generally composed of smaller systems which by themselves serve specific purposes, such as sensors, actuators, and microcontrollers. However, utilizing these systems in tandem make IoT devices very flexible, allowing them to be used in a multitude of domains such as home automation, healthcare, automotive, agriculture, smart cities, and many others, including the industrial domain.

A fully-fledged IoT-enabled device is composed of a stack of interconnected hardware and software components with the capacity of connecting to the internet and whose main roles are to capture and communicate data as well as to physically act upon the environment. In order to capture data from the environment, hardware sensors are employed. To perform physical actions that modify the environment, actuators are put in place, such as an actuator that controls a water valve. In order to connect the hardware components with the software components, devices such as microcontrollers (MCU) and system-on-a-chip (SoC) are used. These components allow for interfacing with the hardware components via onboard firmware and software stored on random-access and read-only memory areas which allows for a fine-grained control over the hardware components. Generally speaking, the role of the MCUs and SoCs are of utmost importance to IoT devices, given that they can also be used to connect to other systems by making use of network component stacks to provide access to the internet, which is a fundamental requirement of an IoT device.

Technical Committee ISO/IEC JTC 1/SC 41: Internet of Things and related technologies has recently published the ISO/IEC 30141 standard in August of 2018 with the goal of creating a reference architecture for IoT systems, which provides, among other things, security methodologies, a standardized vocabulary, IoT characteristics and conceptual models to be used during the design and implementation stages of said systems. The following conceptual model greatly summarizes the high-level components which interact with IoT devices:

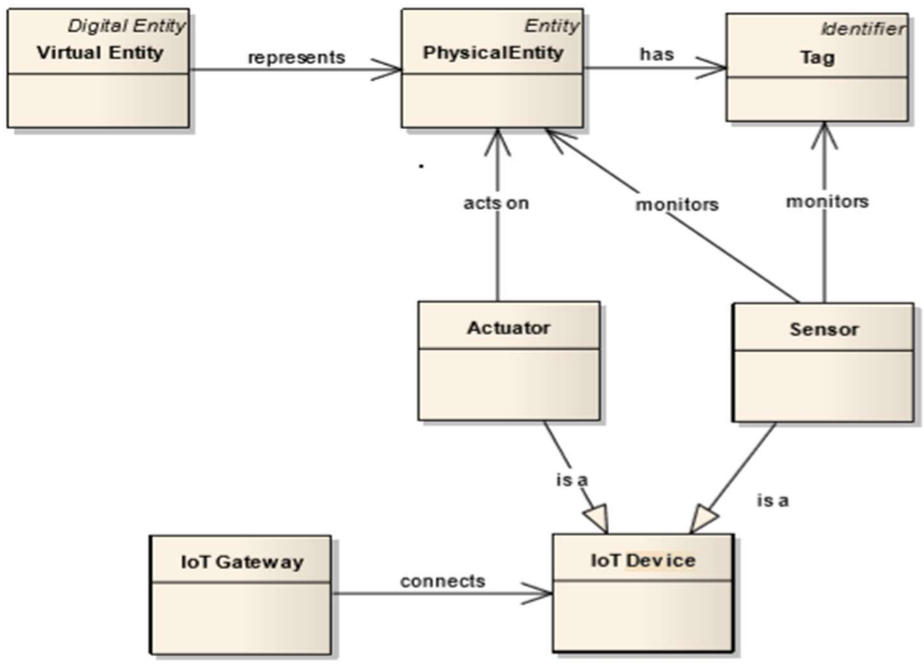


Figure 33: IoT conceptual model

Following a top down approach, it can be said that a virtual entity represents a physical entity. Physical entities are real world “things” that can be measured or acted upon by other “things” or systems. The virtual entity, on the other hand, is the virtual or digitized representation of said physical entity within a digital system. Typically, physical entities that participate in an IoT system contain identifying tags such as RFID to facilitate the integration of the physical entity into the system. These physical entities are monitored by sensors, but depending on the sensor, it may also be able to monitor the tag instead and fulfill the same purpose. Additionally, actuators perform actions upon the physical entity or its immediate environment to affect it in some way. In this conceptual diagram, actuators and sensors are both classified as specialized types of IoT devices, however it is important to note that not all sensors and actuators fall into this description. For instance, there exist “dumb” sensors which have no way of producing information on their own, and do not connect to a network or the internet by themselves either, which disqualify them from being considered IoT devices. Finally, the IoT gateway is used to connect multiple IoT devices to a Wide Area Network.

3.4.2 Sensors

Sensors are devices capable of detecting changes in the environment and measuring physical phenomenon, such as temperature or pressure, and are also capable of transmitting the acquired data as an electric signal. Efficient sensors should be very sensitive to the physical phenomenon they are measuring and should not modify it during the measurement, whilst ignoring other physical phenomena.

Examples of sensors are thermometers, pressure sensors, light sensors, accelerometers, gyroscopes, motion sensors, gas sensors and many more.

A sensor can be described from several properties:

- **Range:** The maximum and minimum values of the phenomenon that the sensor can measure.
- **Sensitivity:** The minimum change of the measured parameter that causes a detectable change in output signal.

- **Resolution:** The minimum change in the phenomenon that the sensor can detect.

Most modern sensors communicate their acquired data by sending electrical impulses which can vary in voltage and current. These impulses can then be sent to a microcontroller unit (MCU) or system-on-a-chip (SoC) and these systems can convert them into understandable values in terms of the magnitude that has been measured.

3.4.3 Actuators

Actuators operate in a manner opposite to the behavior of sensors. An actuator takes an electrical input and transforms it into a physical phenomenon. While in typical IoT solutions a sensor is monitoring data, technicians in a control center or even autonomous systems can take decisions using this data and perform actions through actuators.

Some types of actuators are:

- **Servo Motor:** It is a closed-loop servomechanism that uses position feedback to control its motion and final position.
- **DC Motor:** It converts electrical into mechanical energy. When current is applied, the wire loops generate a magnetic field, which reacts against the outside field of the static magnets.
- **Stepper Motor:** It is a DC motors that moves in discrete steps.
- **Linear actuator:** It creates motion in a straight line. They are used in machine tools and industrial machinery.
- **Relay:** It is an electrically operated switch. Their advantage is that they take a relatively small amount of power to operate.
- **Solenoid:** It is a specially designed electromagnet used for on-off applications such as latching, locking or triggering. They are common in home appliances, office equipment and factory automation.

3.4.4 SOC

A SoC (System on a Chip) is single physical device with all the necessary components for a complete system. SoC components may include analog-to-digital converters, logic control circuits, memory modules, and digital, analog or mixed-signal functions. Due to their size and low power consumption, SoCs have become very popular in embedded systems.

In the IoT ecosystem the main responsibility of a SoC is the communication between sensors and actuators. A SoC must have different interconnection options to provide connectivity with different kind of sensors and actuators. Moreover, SoCs need to understand different protocols to send and receive data within systems.

Generally, the requirements for building IoT devices vary among industries such as agriculture or urban planning. In some industrial environments, devices may work in extreme conditions. For example, sensors may need to collect data underwater or withstand high temperatures. In addition, some devices will require special hardware components. In isolated places, for example, devices could require powerful antennas to cover communicating over long distances. Consequently, the deployment requirements vary greatly. SoCs are usually manufactured on common process platforms in large manufacturing facilities capable of producing hundreds of millions of chips per month, which means lower prices.

The challenge in the design of SoC is that its components are locked into a single manufacturing process which adds difficulties to upgrade or switch out components individually.

3.4.5 Smart Things

A smart thing is a real-world object (products, assets, devices, machinery, etc.) with enhanced interaction with its environment. Smart things have enhanced connectivity capacities, which allow them to exchange data with other smart things and external systems. Moreover, they allow objects to exist outside of its physical device and boundaries, in terms of virtual entities represented in digital systems. The data collected from these products can be then analyzed to aid in decision-making, enable operational efficiencies and continuously improve the performance of the product or other processes.

Some examples of Smart things are smart cars, smart meters, connected home appliances, interconnected pumps and valves, street lighting, and wearables.

Some characteristics of smart things are:

- **Awareness:** Smart things should be aware of their environment and be capable of sensing, reacting and interpreting events which occur in the real world.
- **Connectivity:** Smart things need to provide connectivity to its environment, other smart things and possibly external systems. This connectivity can be wired, wireless or both.
- **Autonomy:** Smart things should operate autonomously, to some extent.
- **Reconfiguration:** Smart things should be configurable in order to adapt to requirement changes.
- **Auto-management:** Smart things should have, to some extent, autonomy to manage local resources such as energy or data storage.

3.4.6 Communication interfaces for sensors and actuators

In this subsection, some of the available low-end communication protocols, such as I^2C, are discussed.

In order to interlink IIoT devices to allow for the exchange of information and commands, they must share a common communication protocol. The low-end communication protocols allow the communication between different physical devices through microcontrollers. A microcontroller is a self-contained system, usually composed of different integrated circuits (IC) which make up several attached components such as a Processing Unit and different types of memory (RAM and ROM), with the capacity of interfacing with devices and other hardware components such as LCD displays, relays, switches and sensors via the use of GPIO (general purpose input/output) pins, other specialized connection pins used for digital or analog communication, and different data interfaces.

Some microcontroller systems are more sophisticated, while others have minimal requirements. Due to the heterogeneous nature of the devices and their different use cases, there is a large amount of different types of microcontroller interfaces available. Figure 34 shows the taxonomy of a subset of different microcontroller interfaces.

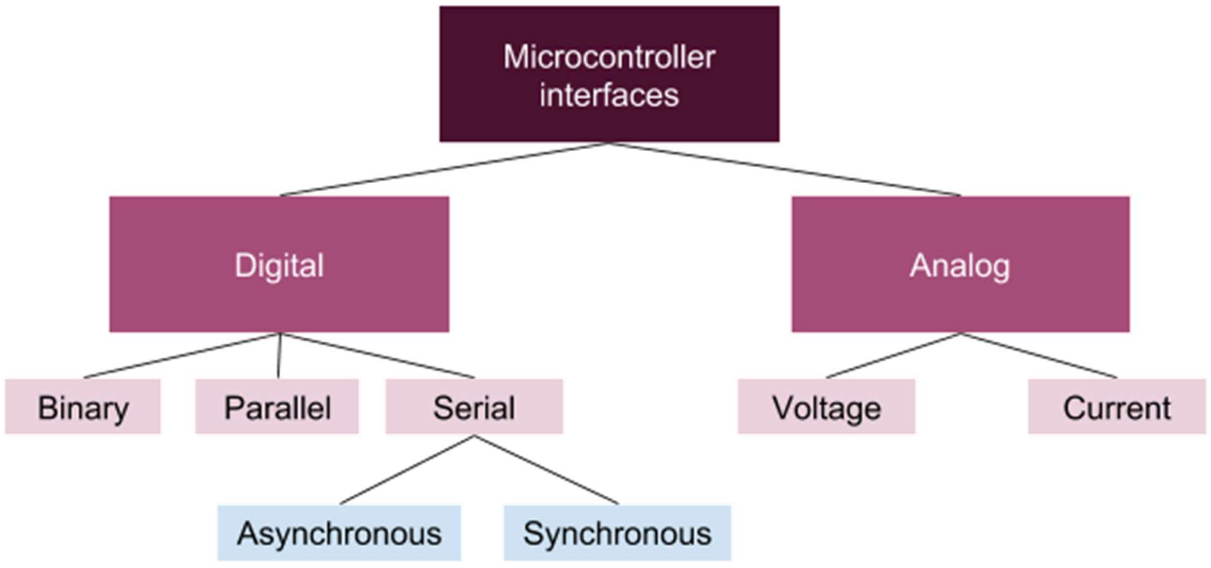


Figure 34: Taxonomy of microcontroller interfaces

These protocols can be classified into two main categories: (i) digital, and (ii) analog. On the one hand, digital microcontroller interfaces handle digital signals which are typically discrete in nature. One the other hand, analog interfaces handle analog signals, or continuous signals that are directly interpreted from voltage or current measurements.

Information between devices is transmitted through electric signals. Analog and digital signals are used to transmit information. In analog signals, data is translated into electric pulses of varying amplitude. Although an analog signal can only represent data within a range, there are an infinite number of possible values within. In digital signals, on the other hand, data is translated is into a discrete binary format (zero or one) where each bit is representative of two distinct amplitudes. This indicates that digital signals only have a finite set of possible values.

If we represent both digital and analog signals in a graph where time is plotted on the horizontal axis, an analog signal will take the form of smooth and continuous waves, whereas a digital signal would take the form of square waves.

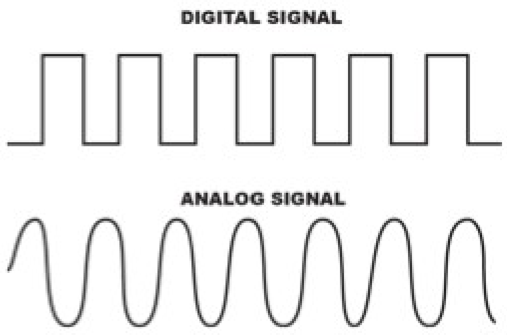


Figure 35: Digital and analog signal waves

Both types of signals have advantages and disadvantages. Digital signals, for example, can be immune to noise and are very well suited to translating digital information; nevertheless, they are limited to a finite range of discrete values they can represent, which can result in a loss of quality with respect to the signal source. On the other hand, analog signals are low cost, easily portable and are not limited to discrete values; however, they require more power and they are prone to noise, which is an important factor in industrial environments.

IIoT solutions need to deal with both analog and digital signals, inputs and outputs. Most microcontrollers can deal both with digital and analog signals with the use of analog-to-digital converters (ADC), which are used to transform an analog electric signal into a digital one.

3.4.7 Digital Interfaces

The digital interfaces can be separated into two main categories: parallel and serial. Whereas parallel interfaces transfer multiple streams of data at the same time across multiple wires, serial interfaces stream data, one single bit at a time in one wire. Parallel communication is fast, straightforward, and easy to implement but it requires many more input/output (I/O) lines resulting in additional cost. Serial communication sacrifices potential speed for a lower amount of utilized interfaces and resources on the microcontroller board. Additionally, there is also a specialized digital interface which is used to transmit single binary values, referred to as an On/Off interface.

3.4.7.1 Binary (On/Off)

The binary on/off interface is a simple digital interface which can control devices such as buttons or switches that may have two excluding states.

These kind of interfaces are simple, low-cost, fast and produce low programming overhead, which makes them a suitable decision for some scenarios. However, they lack functionality in situations when only two states are insufficient. In addition, this interface is only capable of managing one device and usually at a short distance.

3.4.7.2 Parallel

Parallel interfaces can send data between several electrical connections simultaneously.

Parallel communication is usually established through the use of multiple digital input/output pins on a microcontroller, which can be quite costly on available resources and may impose a serious problem depending on the specific case. The advantage of parallel communication channels is that they are able to transmit several bits in one clock transition of the microcontroller’s CPU, effectively increasing the data throughput. Some peripherals require data transmission to be done in parallel, such as certain LCD displays.

3.4.7.3 Serial

There are several serial protocols nowadays. Each of them can be mainly classified into (i) synchronous or (ii) asynchronous. On the one hand, synchronous serial interfaces include a clock signal in the transmission, so all plugged devices share a common clock. This makes a robust serial transfer but it also requires an extra wire. SPI and I^2C are examples of serial synchronous protocols. On the other hand, asynchronous serial protocols transfer data without an external clock signal minimizing the required wires and I/O pins. However, it requires extra effort to enable reliable transmission of data.

Serial protocols are easiest to implement because they require low pin counts in contrast with parallel buses which may require, for instance, eight or more. Another benefit is that serial protocols allow implementing large tasks with several inexpensive smaller processors. Serial interfaces allow processors to communicate without the need of shared memory and semaphores, and the problems they can create.

Serial interfaces are used to provide standardized logic levels between transmitters and receivers, they define the transmission medium and connectors, and specify timing and data rates. In addition, serial interfaces can perform serial-to-parallel and parallel-to-serial conversion.

3.4.7.3.1 Synchronous

Synchronous protocols require that the participants in the exchange of data use a clock signal to transmit data at the same rate. Synchronous protocols support high data transfer rate but they need a special line for the clock signal. These types of protocols usually require a master/slave configuration, where it is the responsibility of the master device to provide the clock signal to all the receivers or slave devices.

3.4.7.3.1.1 I^{2}C

The Inter Integrated Circuit (I^{2}C) is a de-facto world standard protocol used for communication between different components. The implementation of this protocol does not require any licensing fees since 2006.

The I^{2}C bus is developed using a simple bidirectional 2-wire bus. All the devices compatible with this protocol incorporate an on-chip interface which allows them to communicate directly with each other via the I^{2}C bus. This fact solves issues between the interfaces when designing digital control circuits.

Each device can operate as a transmitter (such as LCD driver), receiver or, in some cases, both (such as a memory).

Term	Description
Transmitter	Device which *sends* data
Receiver	Device which *receives* data
Master	Device which initiates a transfer, generates clock signals and terminates a transfer
Slave	Device addressed by a master
Multi-master	There are cases where more than one master can attempt to control the bus at the same time. An arbitration mechanism ensures that, if more than one master simultaneously tries to control the bus, only one is allowed to do so and the winning message is not corrupted
Synchronization	Procedure to synchronize the clock signals of two or more devices

Figure 36: I2C communication concepts

The I^{2}C protocol allows the master devices to communicate using a 7-bit or 10-bit address. Each device has a global and specific address that is assigned to the manufacturer of the device.

3.4.7.3.1.2 SPI

Serial Peripheral Interface (SPI) is a de-facto standard for synchronous communications used to send data between microcontrollers. It uses separate clock and data lines.

SPI is "full duplex" which means it has separate send and receive lines, so, it is possible to transmit and receive data at the same time.

The clock line is responsible in keeping communication synchronized. Given this, the frequency of the clock line also sets the maximum speed at which interconnected devices may send data.

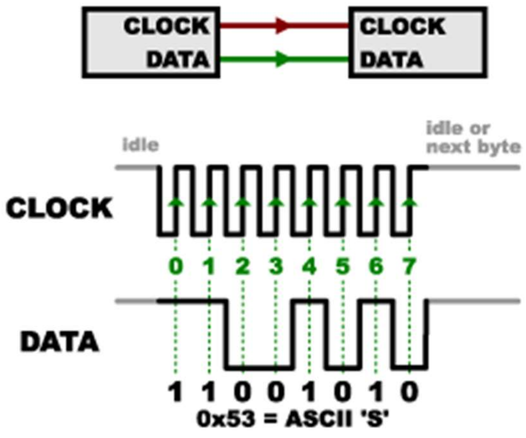


Figure 37: Diagram of data communication synchronized by a clock line

3.4.7.3.2 Asynchronous

Asynchronous serial protocols require mechanisms to ensure robust and error-free data transfers. These mechanisms include the use of synchronization bits, a baud rate or parity bits. These protocols are highly configurable but requires that that devices on a serial bus are configured to use the exact same configuration.

3.4.7.3.2.1 One-Wire

One-Wire is a device communications bus system designed that provides low-speed data and power over a single conductor. Its common use-case is for establishing communications across small and inexpensive sensors.

3.4.8 Analog Interfaces

Microcontrollers also typically have analog interfaces which can be used to establish communication with sensors or other devices that generate analog signals, which, as stated earlier, are continuous signals. This is a very common way of establishing communications, since a lot of the more used sensor types are designed in such a way that they communicate their measured value in terms of analog voltage or current. However, this analog signal won't do much by itself, and first needs to be filtered through an ADC, or analog-to-digital converter. These components convert the analog signal into a digital value and depending on the amount of bits that the ADC can handle, it will have either a higher or lower precision.

Data from sensors can also be sent using voltage or current as drivers, and this will very much depend on the application. If sending data over large distances via a wire, it is most common to use a current-emitting sensor. This is because the resistance of the long wire will not impact the amount of current that is received from the other end of the wire. A voltage, on the other hand, will drop over large distances and information may be lost through the transportation medium.

4 Data Management in IoT-systems [EZERIS]

Data management in IoT system is a challenging task due to (i) high velocity sampling rates required for some monitoring actions, (ii) data diversity, meaning data may vary from one domain to another, and (iii) massive volume could require certain activities such as monitoring in real time or maintaining a historical data archive. These characteristics results in the non-trivial task of deciding which storage systems are better suited to manage IoT data with minor caveats.

In section 4.1 different natures of data generated from sensors are presented. In section 4.2, an analysis is made comparing relational and non-relational database storage systems aimed at IoT systems, and in section 4.3, different strategies and mechanisms to support efficient data access are examined according to the different types of data that have been identified.

4.1 Sensor data

Sensors and actuators can communicate among themselves and interact with the environment. While physical entities sense, act and react to the real-world, they create data. It is possible to discern between two types of data, according to their nature. On the one hand, there exist sensors that monitor real-world events by performing periodic observations. This continuous monitoring usually requires large amounts of data storage, especially if the sampling rates are small. On the other hand, some sensors can react to external stimuli, and thus can be programmed to sample data when a specific event occurs. These kind of situations usually requires fast responses instead of large storage capabilities.

As the need for more scalable systems arises, storage solutions have seen a shift from relational storage systems to non-relational storage systems. Relational storage systems are based on a schema and relies on the structured query language (SQL) for defining and manipulating data. Relational storage systems are powerful because they are versatile and reliable; however, they can be restrictive because they rely on predefined schemas to determine the structure of the data. On the other hand, non-relational databases have reached great popularity because they can be better adapted in situations where data schema varies from one domain to another or even when the schema is not well-known. This is usually the case in IoT systems that deploy several different types of sensors or even sensor modules that can measure different phenomena at once.

4.2 Non-relational data storage

Non-relational data storage systems provide dynamic schemas for unstructured data. As data is stored in many ways, it is easier to store data without structure (each document could have its own structure) and generally, data is more scalable. As IIoT devices may generate large and heterogeneous amount of data, scalable schemas are more suitable for industrial and sensor environments.

In general, non-relational data storage presents the following benefits for IIoT:

- **Flexible Data Model:** Flexible data model, such as documents, graphs, key-value pairs make it easier to combine data of any structure because they allow dynamic modification of the schema without an impact on performance.
- **Scalability and Performance:** NoSQL can easily partition its datasets, allowing to scale out on commodity hardware deployed on premise or in the cloud.

- **Always-On Global Deployments:** NoSQL systems are designed to run across many nodes, including replication to automatically synchronize data across servers, racks, and data centers.

4.3 Data access

The following sections describe different storage-solutions that can be implemented in non-relational databases. Different storage solutions are better fitted to different tasks, and a brief analysis is presented after these descriptions to showcase some specific scenarios where some storage solutions are better fits for different-natured sensor data.

4.3.1 Document data store

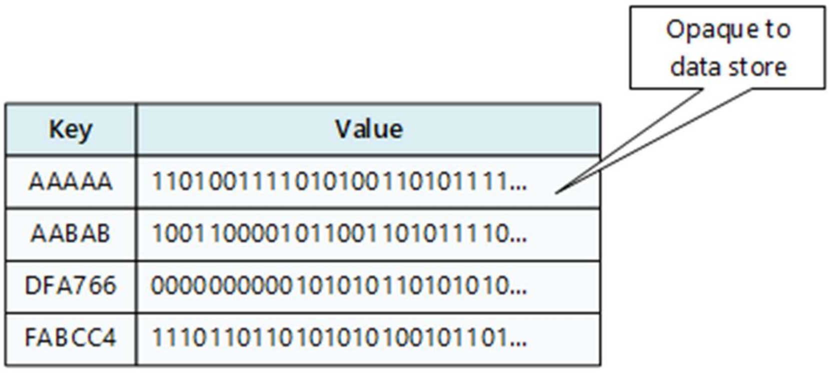
A document data store manages a set of named string fields and object data values in an entity referred to as a document. These data stores typically store data in the form of JSON (JavaScript Object Notation) documents. Each field value could be a scalar item, such as a number, or a compound element, such as a list or a parent-child collection.

4.3.2 Column-family

A column-family data store organizes data as a table, using columns and rows. This storage alternative is conceptually similar to relational databases. However, in this structure columns are divided into groups known as column families. Each column family holds a set of columns that are logically related that are typically retrieved or manipulated as a unit.

4.3.3 Key/value

The key is used to index the data by using a hashing function, responsible to provide an even distribution of hashed keys across the data storage. This organization method supports queries for selecting, inserting, and delete items. To update a element, the system may overwrite the existing data. Other relevant fact is that reading or writing are atomic operations, which can indicate that, if the value is large, this operation may take some time.


Opaque to data store

Key	Value
AAAAA	11010011110101001101011111...
AABAB	10011000010110011010111110...
DFA766	00000000001010101101010 10...
FABCC4	11101101101010101001011101...

Figure 38: Key/value store

This schema is not suitable for scenarios where it is needed to query by non-key values. Although it is typical in relational databases for filtering using a WHERE clause, in non-relational databases searching an element by a non-key could require a slow scan of all values.

4.3.4 Graph data

In a graph data structure there are two types of information: nodes and edges. While nodes represent entities, edges represent the relationship between them. Both nodes and edges can have properties that provide extra information. Edges can also have a direction indicating the nature of the relationship. Graphical structures allow to perform queries that traverse the structure and analyze the relationships between entities.

An example of a query in graph database could be "find all sensors which report directly or indirectly to the central server". Many graph databases provide a query language that can be used to traverse a network of relationships more efficiently.

4.3.5 Time-series

Time-series data storage allows to efficiently store data by time intervals. This organization method supports a very high number of writes, as they typically collect large amounts of data in real time from a large number of sources and they are very suitable in IoT scenarios where updates and deletes are rare.

Due to the nature of IoT platforms and the high sampling rate of sensors, a time-series database can grow rapidly. In addition, time-series systems can handle data that arrive out of order and data which arrive late comparative to their origin timestamp, as well as automatic indexing of data points and optimizations for queries described in terms of windows of time. This last feature enables queries to run across millions of data points and multiple data streams quickly, in order to support time series visualizations, which is a common way how time series data are consumed.

4.3.6 Object data

Object data stores are suited for the management of binary objects or blobs containing media-data such as images, text files, streaming content. Object are hashed by a unique ID. In addition, they can include metadata. This type of storage system is designed for supporting large files, therefore it provides large amounts of total storage to manage all files.

4.3.7 External index

External index data allows to combine different data types and it is suitable for indexing massive volumes of data. An example use case consists in the combination of file finding and searching within its data. Finding a specific file could be fast, but searching based on the contents of the file would require a scan of all of the files, which is slow. An external index allows for the creation of secondary search indices and then quickly find the path to the files that match some specified criteria. Another example application of an external index is with key/value stores that only index by the key. A secondary index can be constructed based on the values in the data, and quickly look up the key that uniquely identifies each matched item.

4.4 Application in IoT systems

Taking into account the previously described data storage solutions and access methods, specific IoT system scenarios can be directly mapped to one of these storage solutions to optimize the overall efficiency of the system. For instance, in a situation where multiple sensors are being sampled repeatedly throughout an indefinite amount of time, for example in a situation where the parameters of an object, location, process, system, or phenomenon are being monitored, and the data is kept as a historical dataset and there is no intention of updating or deleting any records, a time-series storage is an optimal solution. This decision will also support the visualization of said data further down the data value chain. In the case of having a diverse variety of sensor types, or modules composed of different sensors that sample their environment in response to a specific event or trigger, a document store best suits the task. This is because it is generally difficult to structure a general schema to which all sensor types and devices must adapt. Rather, an unstructured object may be stored in JSON format which allows for a much more flexible way of storing the data obtained from the sensor ecosystem.

5 Security Aspects of (I)IoT-platforms [NXP]

5.1 Introduction

Sensors, actuators, processors, networks and radios are key elements in building smart connected devices. These devices in turn enable real time sensing and contextual understanding, active modification and manipulation of the environment, and communicate with one another and humans.

In diverse fields from industry, to medicine, to transport, to smart cities and home automation, user interactions will change drastically and smart devices and services will become integral parts of daily lives. This new world of connectivity however brings with it openness to potentially millions of attackers. As society becomes more and more dependent on these systems, it becomes more vulnerable to malfunctions.

Smart factories will face similar challenges. Production sites, workshop floors, business procedures, designs, construction processes, and many of the products themselves will be "smart and connected". Sensors and extensive networks of computers, mostly as small computational components in industrial networks, will become mainstream for every step of the industrial production process. This in turn will allow connecting production to logistics, customers to products on the shop floor, and robots and workers to each other.

This chapter gives s generic overview on background, security concepts and principles for IIoT Solutions that will be considered when defining a security concept for the OPTIMUM platform to be created in this project. The OTIMUM specific security solution will be described in detail in deliverable D4.2 in scope of the work package WP4 / Task T4.2.

5.2 Use Cases, Resulting Security Requirements & IIoT Characteristics

5.2.1 What are typical IoT use cases?

Three are many areas for the IoT application solutions. Examples are:

- Smart Home,
- Car2X for autonomous driving
- Banking
- Healthcare
- … and many more

Nowadays using the IoT platform in Industrial area becomes more and more important. The corresponding domain is called ***Industrial Internet of Things (IIoT).***

A typical example of it is the so-called **Industry 4.0**:

It is applied to run and maintain flexible, efficient state-of-the-art production facilities, up-to-date and complete data about the status and properties of individual machines and components is essential. Data is not only required on the shop-floor but must also be provided, on a need-to- know basis, to other relevant parties. This will enable suppliers to adapt dynamically to changes in demand and production as well as enabling maintenance service providers for predictive maintenance activities. All this will happen autonomously with no, or minimal, human interaction. In the predictive maintenance use case, data from sensors may be collected from all of a company's machines via the cloud for analysis to determine exactly the right time for exchanging parts or performing maintenance, to prevent costly, and unnecessary downtime.

Further, it will optimize the service intervals and related costs.

<u>Access to the Operational Technology (OT) for (external) service providers is mandating the requirement to secure production against manipulation, assembly line stops and IP theft.</u>

5.2.2 What security related requirements result from these use cases?

The new and also the already existing solutions need to be adapted based on the answers to a set of specific questions:

- What attacks are relevant?
- What are the protection requirements?
- What is the impact of a successful attack against one node?

…and so on. For example, remote logical attacks to company routers are relevant and likely, while a local attack targeting the secret key of a single router within the company is less likely.

There will also be differences in the strength of the required security measures – the manipulation of a complex machine has a more severe impact on related costs and human safety, compared to a home HVAC device, unless it is used to attack the power grid.

In the end, there is no one solution that fits all purposes, rather a set of options is presented to select from for protecting various assets. Too many security measures will lead to an uncompetitive solution, while insufficient measures may mean significant risk to the business, safety or reliability of a process.

A system view is important when considering assets. The IIoT node itself might not be of high value to an attacker, but its computing & networking capability is of interest for integrating it into a botnet. The networking capability of IIoT nodes may enable scalable remote attacks, which can lead to a huge impact even if the impact on the single device is minimal.

5.2.3 What are the general characteristics of IIoT systems?

Many IIoT systems are highly autonomous. They act on their own and generate data during their operation, possibly without their owner even being aware. This autonomous operation opens IIoT systems to additional attack vectors. IIoT systems are also quite heterogeneous in terms of their intended use, their technical architecture, and their connectivity.

Further, the number of devices deployed and the amount of data being generated are expanding exponentially. Furthermore, the ability to remotely monitor, maintain and upgrade the millions of deployed IIoT devices in the long-term is a challenge. The IIoT nodes might be in difficult to access locations or even have no continuous connection to their host devices. IIoT devices need fast, secure over-the-air firmware and software updates, and measures to enable decommissioning at the end of their lifetimes. Vulnerabilities must be patchable within an appropriate amount of time.

Another degree of complexity arises from the required interoperability of devices from different vendors, without compromising security.

Finally, some IIoT devices will have a lifetime of 20 years or more. It is not likely that all IIoT devices will be supported, updated or patched for the complete lifetime in a cost-effective manner. These long lifetimes will present new security challenges that were not known at the time of production. Here, precautions should be considered and defined in the design stage.

5.3 Background, Security Concepts and Principles for IIoT Solutions

5.3.1 Why are IIoT systems attacked?

IIoT system are attacked for one of the following purposes:

- **IP theft from (part of) a system**
  For example, companies taking software from a device and using it to create counterfeit devices.
- **Intentional damage**
  Examples include causing cars to crash, opening lock gates on waterways to cause deliberate flooding or sending too much traffic to a system to overwhelm it (DDoS attack).
- **Reputation and notoriety**
  For example, security researchers trying to demonstrate their expertise by exposing vulnerabilities in a system. A more institutionalized purpose is academic honors: security researchers in universities need to demonstrate successful attacks to advance their careers. Similar motivations hold for security research companies.

5.3.2 When Do Attacks Happen?

A product can be attacked at any point in its lifecycle:

1. Production of devices and the components used in them
2. In the logistics chain
3. During installation
4. During use
5. After end of life (information or key extraction from a discarded device)

5.3.3 Types of Attacks

Attacks can be categorized as either ***remote*** or ***local***.

- ***Remote attacks*** can be carried out by sending commands over a network connection. The attacker does not need to be physically near the device under attack. Due to their scalability, these attacks are the most dangerous: with one PC, an attacker has the potential to attack millions of devices. Although the development of an attack may require significant expertise, carrying it out can be automated and executed by unsophisticated adversaries on a huge scale. In fact, there are black markets where such attacks are sold and some even include help desks in case the attacker who purchased the script has a problem deploying it!
  These are the attacks that are most often reported in the press and that most people have in mind when talking about attacks.
- ***Local attacks*** require physical access to the device meaning they are much less scalable and the attacker normally needs to be more skilled. However, it is possible that through a local attack the entire code of a product can be obtained. Analyzing that code may lead to the discovery of vulnerabilities which may be applied to similar devices through remote attacks.

Another way of categorizing attacks is to group them into ***physical*** or ***logical***.

- ***Physical attacks*** exploit vulnerabilities in a device either through direct manipulation or by observing its operation.

- ***Logical attacks*** only rely on messages sent to the device to cause damage.

The above categorizations have overlaps.

Analogously, while most local attacks are physical, some local attacks are logical. An example of a local logical attack is sending messages over an interface that is not available remotely, such as a debug interface.

5.3.4 Levels of Protection

The set of attacks to protect against shall be a balance between complexity and impact. Compare it to securing your house. It does not make sense to have a very strong lock on the front door while leaving the backdoor open. Also, if you have many valuables in your home that are attractive for thieves, you will want stronger safety measures than if you only have shabby second-hand furniture.

5.3.5 Confidentiality, Integrity, Authenticity and Availability

When considering security, it is important to understand the difference between ***confidentiality*** and ***integrity***.

The objective of confidentiality is to ensure that only authorized entities can read information while people who are not authorized cannot. For example, an encrypted email may only be read by the targeted recipients who have the correct keys.

The objective of integrity is to protect information against unauthorized modification. A related concept is ***authenticity*** where it is not only clear that data has not been modified, it is also clear where the data originates from. For example, sending an email that is cryptographically signed will allow the recipient to determine whether the message was altered between sender and recipient and confirm the identity of the sender.

While similar cryptographic means are used, it is important to recognize that confidentiality, integrity and authenticity are different. Encrypting a message will prevent unauthorized people from seeing the contents, but an adversary can change a part of the encrypted message and it will still decrypt. In general, the result will be nonsense, but it may also yield a legitimate looking message with different contents to the original. The error may not be detected, especially if it is used in machine to machine communication where only a few bytes are used to make a decision.

A common way to guarantee the ***integrity*** of data is to calculate a ***cryptographic hash***, normally just referred to as hash, over the data for which the integrity needs to be protected. A hash is a function to condense the data to be protected into a fixed, small number of bytes (the hash) in a way that it is not possible to construct another set of data that delivers the same hash without trying all possible datasets. This is much more complex than other types of checksums, such as those used to protect data from unintended corruption, for example, during transmission over an unreliable channel.

Another form of integrity is control flow integrity. The integrity of all data and code may be perfectly intact, but certain attacks may cause the flow of the software execution to be changed in a way that compromises the integrity of the device. The term run time protection refers to maintaining control flow integrity.

In some cases, confidentiality is essential to maintain integrity. For example, if the confidentiality of passwords is not maintained it may lead to adversaries gaining access to a system, enabling them to compromise its integrity.

Confidentiality and integrity are often mentioned together along with ***availability*** creating the abbreviation **CIA**. Availability is highly important especially for IIoT. It definitely is an issue to consider at the end to end system level, for example to ensure that an adversary is not deliberately overloading a system so that real messages cannot be processed or to jam a communication channel such that a device cannot send an important message.

5.3.6 Symmetric and Asymmetric Cryptography

Cryptography is used to protect the confidentiality and integrity of data.

There are two main classes of cryptography:

- **Symmetric cryptography**

  The same (secret) key is used for encryption and decryption. For integrity protection, the key is used to calculate a Message Authentication Code, or MAC, over the data, for example a message, which is then sent along with the data. When checking that the integrity of the data has not been compromised, the same key is used to recalculate the MAC, which is compared to the MAC sent with the message.

- **Asymmetric cryptography**

  Two keys are used: a private key and a public key. When something is encrypted using the public key, the private key is needed for decryption. For integrity protection, data can be signed with the private key and then verified by using the public key.

The advantage of symmetric cryptography is that it can be done very quickly on large amounts of data. Asymmetric cryptography is much slower and cannot be applied efficiently to large amounts of data but it has the advantage that key management is easier.

As the name suggests, the confidentiality of a public key is not important since it will only be used to encrypt or to verify. However, integrity of the public key is quite important. If an adversary can change a public key that someone is using to send messages, the adversary will be able to decrypt them, since he has obviously the private key that belongs to the public key he inserted.

5.3.7 Signing

In asymmetric cryptography, if a user is given a unique private key which they can use to sign data, other people can check the authenticity of the signature by using that user's public key. Although integrity protection is also possible with symmetric crypto by using a MAC as explained above, it is not possible to use it for signing, because both the creator of the MAC and the verifier need to have the key, so it is not possible to relate a MAC to one individual. It is impossible to tell from the MAC whether it was created by the sender or the receiver since they both have the key.

5.3.8 Certificates

A certificate consists of a public key and other data, such as the name of the person and the certificate expiration date, and is signed using the private key of the person or body that issued the certificate.

The user of the certificate will in some way have obtained the public key of the issuer. The main methods are by configuring it during production, or by obtaining it from another certificate. Of course, the validity of that certificate must be checked before using it. This way, a complete certificate tree can be built, but at the root of the tree, there is always a certificate that must be trusted on its face value.

5.3.9 Protecting keys versus protecting usage of keys

The confidentiality and integrity of cryptographic keys are at the heart of the security of any IIoT system. If keys are disclosed, the confidentiality and integrity of the system and devices are at stake. However, if the usage of these keys is not well protected, the security of the system is still at risk.

Consider a case where a key is well protected in a Secure Element (SE) that is in turn connected to a Micro Processing Unit (MPU). If malware can be loaded into the MPU that allows it to make use of the key in the SE, the system is still compromised since the malware can encrypt, decrypt and sign using the keys. The security of the specific device is compromised to the same extent as if the key itself had been exposed, although the key cannot be cloned into other devices. If this is an attack that can be done remotely, then the attack is scalable and even more powerful.

As with blacklists and whitelists, an authority governing and monitoring the overall system could take the necessary action. If no central authority exists, then stronger local protection may be necessary.

5.3.10 Protection against counterfeiting, including cloning

Copycat behavior is not new or unique to IIoT. Manufacturing companies can attempt to reverse engineer PCBs and produce copies, adding software they have extracted from an original device before selling their own counterfeit version.

We make a distinction between counterfeiting and a specific form of it, namely cloning. With cloning, a complete copy of the original device is created including all software and key material. In the more general concept of counterfeiting, a device with similar functionality is created, potentially starting from the same hardware design and the same software, but not copying the keys and possibly other data.

Counterfeiting can happen at the level of the complete IIoT device, or at the level of components.

Not only is the brand owner of the original device missing out on revenue, but they may also have to provide the back-end services for those counterfeit devices. In addition, there may be issues with the quality of the counterfeit, but the customer is not able to identify it as a counterfeit. There are incidents where companies have had to replace defective counterfeit devices by original devices under warranty.

5.3.11 Protection against IP theft

The software on IIoT devices is often highly valuable and will have taken considerable resources to develop. Copycat manufacturers may not just steal the entire software for use on cloned devices, they may steal parts of it which they modify or reuse in their own products.

The proposed NXP IIoT Hub has mechanisms to protect against IP theft as well. See Section 4.9 for further information.

5.3.12 End of life and decommissioning

In the IIoT market, the lifetime of a device may be significantly longer than assumed at its introduction to the market. That's especially true for IIoT devices 20y+ systems and devices. There is generally no managed end of life process for IIoT devices, unlike passports or banking cards that are actively terminated after a certain time.

What should be done with an expensive tractor that is mechanically fine, but for which there is no software update available anymore to defeat a new attack? This scenario may happen for a variety of reasons such as a vendor having gone out of business or the device running out of memory.

A more familiar example is WiFi keys stored in routers. After a router has broken or been discarded it may still be feasible to extract the key. If the key is not sufficiently protected after end of life, the network would be vulnerable to unauthorized use.

A general recommendation could be that users need to decommission their devices before taking them out of service. However, it is highly unlikely that everyone will do so and it may not even be possible if the device is broken.

Currently there is not much interest in this type of problem, but it will undoubtedly arise in future. There are already indications of increased awareness with a proposed bill in the United States which stipulate that a vendor selling to the US government must specify for how long a device will be supported, among other requirements.

5.4 Principles for IIoT Solutions

5.4.1 What attacks should be protected against?

It will not be possible to protect against every attack. Apart from the fact that many potential attacks will be unknown when a device is launched into the market, protection against certain attacks is too costly to implement without there being a realistic threat.

The picture below summarizes which attacks to protect against and the relative priorities. However, a trade-off should always be made between the risks and cost of protection.

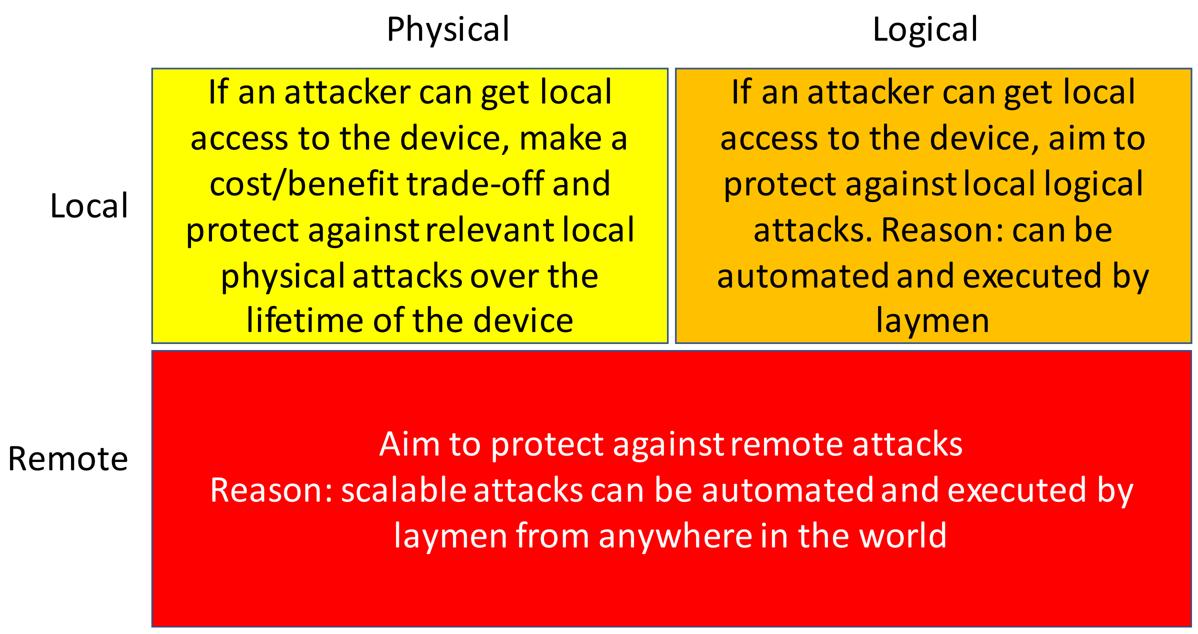

Physical
Logical
Local
If an attacker can get local access to the device, make a cost/benefit trade-off and protect against relevant local physical attacks over the lifetime of the device
If an attacker can get local access to the device, aim to protect against local logical attacks. Reason: can be automated and executed by laymen
Remote
Aim to protect against remote attacks
Reason: scalable attacks can be automated and executed by laymen from anywhere in the world

Figure 39: Protection to be provided for different classes of attack

IIoT devices will normally be connected to the internet, but this is not guaranteed and does not mean that they will always have an open and direct connection to the cloud as they could connect through edge nodes or have only an intermittent connection. However, in many cases an attacker can reach a device when he can access the right part of the network.

Since even very complicated attacks can be automated it is necessary to provide defense against remote attacks in most cases.

Resistance to local attacks is a different matter though:

1. If a local attack only causes loss or damage to the device-owner-turned-adversary and there is no damage to the rest of the system, there may be no need for protection.

   Consider a WiFi router in a private residence. It is likely that with a local attack the WiFi key can be extracted. However, there is no benefit to the owner as he chooses the key in the first place. If the router is stolen and keys are obtained from it by another adversary, its obvious nature will prompt the owner to change the WiFi keys in all devices previously connected to the router, creating a simple countermeasure.
2. If a local attack on an IIoT device can have a wider impact than described above, defense against local attacks needs to be considered. Conceptually, one only needs to defend against those attacks where the cost of the attack is lower than the anticipated gain. Applying this principle in practice is not as simple as it may seem:
   - The developer of IIoT device will generally not know where the IC will be used and the potential benefit of an attack
   - Even if the developer knows where the IIoT device is to be used, the most appropriate security will also depend heavily on the system level architecture
   - Some attacks will get easier and less costly over time
   - It is not always easy to quantify the gain for an attacker. For example, influencing an election or someone with a terrorist motive

Note that local attacks can also take place without the knowledge of the person who enables the actual attack. For example, when inadvertently connecting malware infected smartphones or USB sticks.

5.4.2 General Assets to Protect

These are some general assets to consider protecting:

- **Data Integrity and authenticity**
  Although there may be ways to tolerate corrupted information in a system, in general, protection of data integrity and authenticity is needed.
- **Control flow integrity and run time protection**
  If control flow integrity is compromised, the processing cycles of an IIoT device can be used for other purposes, such as becoming part of a botnet.
- **Confidentiality when needed (often the case)**
  For example, user data is normally to be treated confidentially, however the confidentiality of sensor readings in a factory may be of no particular concern.
- **Availability when needed**

For example, in a production line the unavailability of a device can have a big impact, whereas a non-functioning internet connected baby doll for a day or so is not catastrophic.

- **Control over the device**
  While defense against certain attacks may not be applicable, always consider protecting against local physical attacks that may give control over the device.
- **Privacy when applicable**
  For example, data that can be used to get personal information about people needs to be protected against misuse, whereas privacy is not generally an issue for sensors in a factory.

5.4.3 Recommendations for creating secure IIoT systems

NXP formulated a set of recommendations to be considered for IIoT security:

- Strive to use isolation whenever possible
  Isolate keys, crypto and essential functions by putting them on different cores and different memory
- Keep implementations as simple as possible
- Make sure the device can be updated remotely
  - Plan for capabilities (e.g. memory, crypto, processing) that will be sufficient during the foreseen lifetime of the device
  - Enforce rollback protection and do not allow factory resets, which may cause security patches to be bypassed
- Apply key diversification
  - Do not use the same private or symmetric key in more than one device or use the same key for more than a single purpose
- Devices that have limited capabilities should protect themselves by refusing any incoming connections. They should connect only to authenticated, and thus trusted, hosts or gateways that are capable of properly protecting themselves
  - Some devices may have such a limited set of capabilities that not all desired countermeasures can be implemented. In that case they should not accept any incoming network connection, but rather poll the server or gateway on their own initiative. This will reduce the attack surface for remote attacks. Note: This does of course assume that such devices will have enough capabilities to set up a secure connection to the server or gateway.
- Do not store any passwords or keys in the software
  - Software can be extracted, if not by a remote attack then by local attacks. Passwords in software will generally not be diversified, meaning that attacking one device can allow an attacker to compromise devices of the same type, often via a scalable remote attack
- Disable debug mode for production devices. This should be done before they leave the factory at the very latest.
- Take care of privacy, when applicable. When there is a privacy aspect, make sure it is taken care of.

5.4.4 Recommendation when deploying IIoT devices or IIoT systems

For those who deploy IIoT devices or IIoT systems, consider the following recommendations:

- <u>Know what to deal with</u>

Because it is difficult even for experts to know what security level a device offers, demand that product has been certified.

- Demand a security software update service
  This is essential for new versions of software to be downloaded. Vendors must be explicit about how long their customers can count on getting such software updates. Demand that this is stated in the contract.
- Manage end of life
  Decide what to do at the end of the security update service period, if a vendor goes out of business or if updates go beyond the hardware capabilities of the device, for example if there is insufficient memory available. Either stop using the device or take extra protection steps such as placing the device behind extra firewalls to prevent remote attacks and limiting physical access to the device to protect against local attacks or retrieve information periodically and delete some memory.
- Remove secrets at end of life
  When taking the device out of operation, make sure that any secret on the device is securely erased before releasing control of it. In some cases, nothing else is needed other than simply decommissioning the device in the server. In other cases, secrets may have to be actively removed or, if a device is broken so that nothing can be erased, it may need to be physically destroyed.

5.5 Common Architectural Elements in IIoT Solutions

5.5.1 Security by Design

Security by design is the key to enabling smooth integration of security features into IIoT devices. Integrating security into an existing product very often leads to unwanted overheads and security issues which cannot be avoided at later stages of the implementation. Security by design leads to a sound security concept, including a balanced split of security between hardware and software while still considering costs and performance. Doing so is not pure technical ability or a process, it is more a mindset to consider security during all design decisions in the development and lifecycle of a product. In businesses like banking or ID this mindset is already well established. In new business areas like IIoT there is a lack of the required security mindset, because there is a low demand from customers and regulators, and knowledge on the impact of security flaws is missing. IIoT device vulnerabilities are shown, almost on a daily basis, by attacks being published on devices ranging from Automotive, to pacemakers and light bulbs.

Each development project with a security scope must have dedicated people assigned to determining, specifying and reviewing the security architecture. They must act as the security advocates within the engineering organizations and promote the central security competence. Reviews uncovering security issues late in the development cycle or after release, therefore saving money and protecting the reputation of the developing company. As security by design is the responsibility of everyone in the development process, enabling it requires a lot of discipline and the necessary measures to establish it.

NXP recommend following a staged approach, tailored to the different levels of security maturity existing inside of NXP.

5.5.2 Security architectures and their security properties

For the IIoT use cases described previously, four basic requirements must be met: integrity and authenticity must always be taken care of, plus confidentiality and availability when they are required. These data requirements can be mapped into the three principles for IIoT solutions.

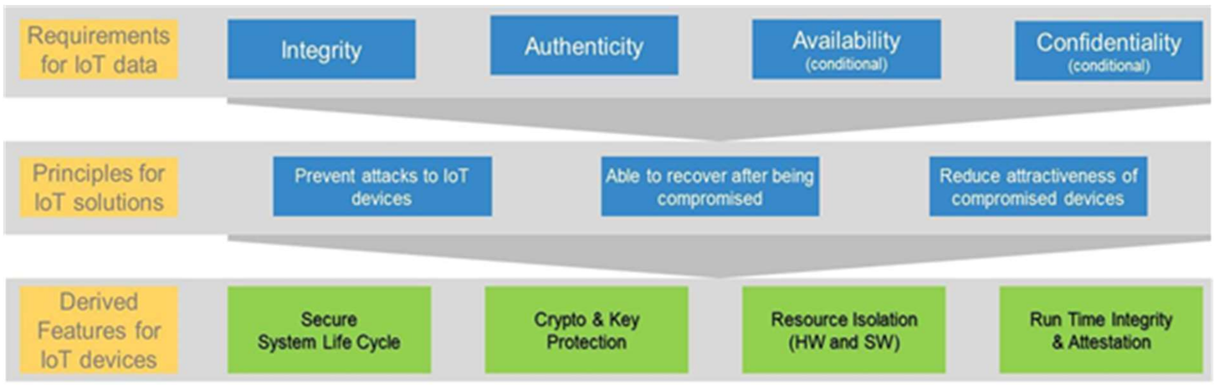

Requirements for IoT data
Integrity
Authenticity
Availability (conditional)
Confidentiality (conditional)
Principles for IoT solutions
Prevent attacks to IoT devices
Able to recover after being compromised
Reduce attractiveness of compromised devices
Derived Features for IoT devices
Secure System Life Cycle
Crypto & Key Protection
Resource Isolation (HW and SW)
Run Time Integrity & Attestation

Figure 40: IIoT device features derived from for IIoT data requirements

For example, to maximize data availability the devices must have a minimum attack surface, and if successfully attacked the device should attempt to recover. Recovery of the IIoT device could take the form of reduced functionality, bandwidth or other attributes to thwart or minimize future attacks.

In order to accomplish some form of recovery and also for its functionality in standard operation mode, an IIoT device needs to be able to establish a secure end-to-end communication with its backend server. This comprises of:

- mutual authentication of the end points, so that both end-points are certain that they are communicating with a trustworthy partner
- data integrity of the communication must be ensured which enables the opposite end-point to check if data has been changed, inserted or deleted in transit
- data confidentiality for all or some data, depending on the use case

The secure end-to-end communication is shown in the figure below, where SSS stands for Secure Sub System.

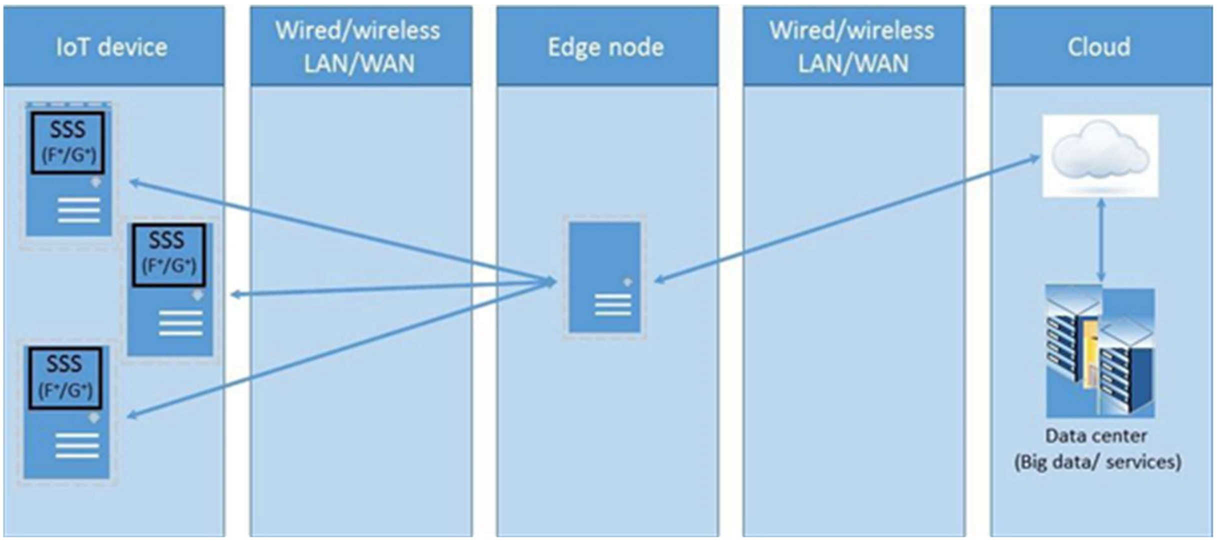


Figure 41: End-to-End communication for IIoT.

From these principles for IIoT solutions, the following requirements for IIoT devices are derived:

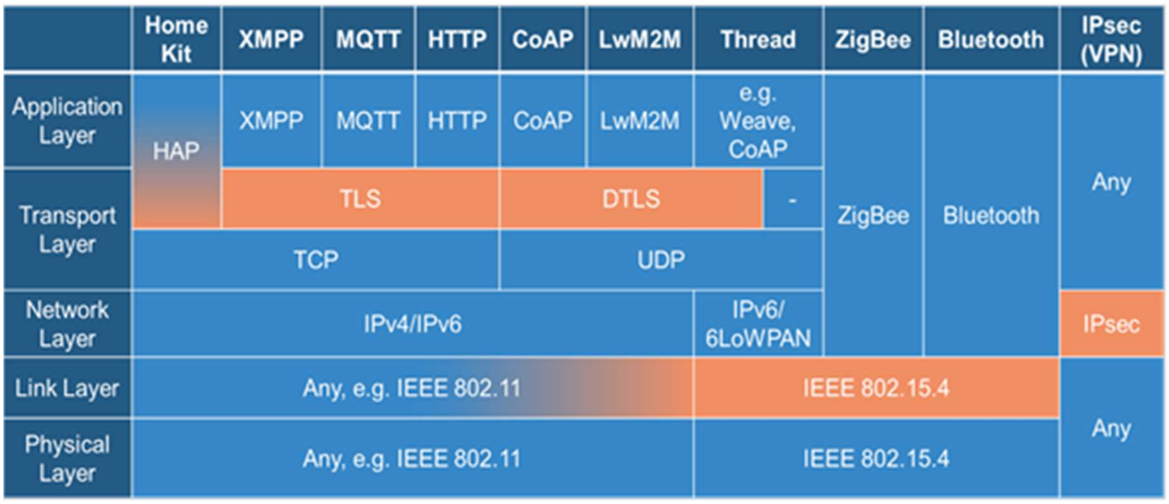
- **Secure System Lifecycle**
  Realized by secure boot, including the secure power-up, as a pre-condition to lifecycle management and configuration protection, with an irreversible path from virgin device, via secure debug, to a fully functional device in the field. It also includes the option to securely perform a remote update of FW/SW in the field with secure firmware over the air updates, secure logging and secure handling of field returns.
- **Crypto & Key Protection**
  Realized by state-of-the-art secure key storage and key usage by a rich/secure execution environment. Used in an IIoT device for encrypting, decrypting and run authentication procedures, creating and executing integrity checks, and returning a device to a safe state if it has been compromised. A device must also provide means for protected secure root key provisioning and storage of key and certificate credentials, and handle the security of derived keys.
- **Resource Isolation**
  To minimize the attack surface, the resources in HW, such as memory, and SW used for platform security features must provide isolation from other parts of the IIoT system.
- **Run Time Integrity & Attestation**
  Software must be protected against modification by remote attestation and modification of the program flow prevented by solid Anti-rollback, counterfeiting is a further security feature for this class

5.5.3 Network protocols

5.5.3.1 Overview of Existing Protocols

The landscape of IIoT network protocols and connectivity frameworks is very fragmented, countless group initiatives and standardization entities have created their own standards and frameworks geared towards the IIoT market. In addition to standards and open frameworks, individual companies are also defining and promoting completely closed eco-systems like Apple’s HomeKit. These diverse standards and frameworks differ in scope and the layers in the protocol stack which they address. Many protocol stacks reuse parts of other protocols and standards that can be combined with each other or instead provide multiple options to choose from. This further increases the complexity of the overall picture.

The figure below provides an overview of the most important protocol stacks and connectivity standards typically used by IIoT and their security protocol layers (shown in orange).



	Home Kit	XMPP	MQTT	HTTP	CoAP	LwM2M	Thread	ZigBee	Bluetooth	IPsec (VPN)
Application Layer	HAP	XMPP	MQTT	HTTP	CoAP	LwM2M	e.g. Weave, CoAP	ZigBee	Bluetooth	Any
Transport Layer		TLS			DTLS		-			
	TCP				UDP					
Network Layer	IPv4/IPv6						IPv6/ 6LoWPAN			IPsec
Link Layer	Any, e.g. IEEE 802.11						IEEE 802.15.4			Any
Physical Layer	Any, e.g. IEEE 802.11						IEEE 802.15.4			

Figure 42: Protocols and connectivity standards for IIoT with security protocol layers in orange

Generally, simpler IIoT end nodes like smart light bulbs typically use a wireless connectivity stack like ZigBee, Bluetooth or Thread to define all layers from physical to transport or even the application layer while more capable end nodes and gateways typically use more application layer oriented standards for direct connections to the cloud like CoAP, MQTT or XMPP on top of an IP-based protocol stack. Some higher layer oriented frameworks such as Weave, IoTivity, Fairhair and others build on one or more of these protocol stacks and are therefore omitted in the overview.

Network security protocols used for IIoT can be categorized based on their layer in the communication stack i.e. link, network or transport:

- **Link layer security** is typically used to ensure only authorized devices can enter a network. Communication is protected from eavesdroppers but it provides no end-to-end security like most networks where any device that is part of the network can decrypt any communication,

  Communication to peers that are not part of the same network have to be relayed (i.e. decrypted and forwarded after re-encryption) by an access point or gateway. Link layer security provided by the wireless connectivity standard (for example WiFi, Thread, ZigBee) is in most cases the only network security measure used for protecting the communication of simpler IIoT devices like smart light bulbs with a gateway.

- **Network layer security** (for example IPsec as used for VPN) provides protection of confidentiality and integrity for a specific communication link for example a connection between an IIoT device and a gateway. In contrast to link layer security the connection can cross different networks. There are approaches optimizing IPsec for use in IIoT, but deployment in typical network setups is complex and error prone and there is no widespread adoption of IPsec for IIoT deployments so far.

- **Transport layer security** is used to create a secure channel between two applications on two endpoints and provides full end-to-end security. The most widespread security protocols on the transport layer are Transport Layer Security (TLS) and Datagram

Transport Layer Security (DTLS) used for securing communications of IIoT devices but also for many other applications like web browsing. These protocols are commonly used for securing direct connections to cloud services. All major cloud providers that offer IIoT cloud services support TLS for accessing their services.
The Thread network uses DTLS for commissioning new devices into the network. The security layer of the proprietary HomeKit Accessory Protocol (HAP) can also be categorized as a transport layer security protocol.

6 Requirements from IIoT perspective [ifak (Table from ETRI)]

As a general definition of IoT, considering the IIoT platform as a comprehensive definition including device, network, and application service layers, it is necessary to build an IIoT platform considering the following conditions.

Objectification Scale: In the case of mechanical devices in a factory, basically, a unit of PLC equipment or a unit of controller constituting DCS can be defined as an object. If the control equipment is configured per a process unit, the objects are located in the control layer on behalf of the individual process. If the control equipment is configured per a specific facility unit, multiple pieces of control equipment are distributed in one process to perform control functions. In addition, sensor information management equipment that monitors the factory environment, and transportation equipment can be defined as objects.

Level of Intelligent Things: Since control equipment such as PLC or DCS operates according to the driving conditions and procedures programmed by the operator in advance, if the production process is changed or an exception occurs, it will respond according to preset control conditions or lead to operational failure. Assuming a typical scenario, when control equipment detects an exceptional situation, the operator intervenes to analyze the problem and take action accordingly. Therefore, the intelligence of objects needs to be distributed as a unit of things where human intervention occurs, and at a level where the situation analysis based on human decision making is carried out autonomously.

Real time Analysis: The process control of the production process focuses on input data monitoring and real-time control of events. In contrast, the IIoT platform is based on the building of the big data needed for data analysis, so the data source is transported to the top level. However, for real-time data analysis, it must be possible to analyze in the machine layer that directly controls the device, and the corresponding data processing and storage function should be placed in the machine layer. Assuming such an environment configuration, it may be effective to adopt a continuous query-based data analysis method that performs a query directly on the input stream rather than a snapshot query-based data analysis that targets data stored in the database.

Collaboration between Machines: Collaboration between machines is a process of requesting and receiving data between entities, regarding a machine as a single independent intelligent entity. In here, the data may be process-sharing information between facilities or flow control information between various processes in one production process. To collaborate between machines, individual machines must be aware of things related to themselves and should be capable of one-to-one or one-to-many communication and interaction so that they can distribute their status information to other machines or request if necessary.

Distributed Computing: In the IIoT environment, the distributed computing can basically be configured through the extension of the edge computing architecture. Edge computing is conceptually structured in a different way from DCS, which had previously been applied to factory automation. DCS has a structure in which individual control devices are distributed to perform independent process control and a central main server collects and analyzes events generated from the control devices in a batch manner. In contrast, the edge computing method has a structure in which distributed edge clouds have the role and authority as much as a central cloud in its area. Usually, edge computing architecture can be suitable for small-scale scenario, where meaningful analysis is possible for small amounts of data.

The requirements for developing the target technology can be derived by reference to the general considerations described above and use cases limited to the scope of development. Requirements drawn up to date are defined in WP 1.2, and the related requirements are summarized as follows.

Table 4. General Requirements

ID	Requirement Statement	WPs addressed
R-GEN-001	The system MUST be able to support wired and wireless communication between the control components.	WP2,WP3
R-GEN-002	Wireless communication MUST be highly available	WP2
R-GEN-003	Communication MUST be suitable for Safety-Related Functions (at least PL d according EN ISO 13849)	WP2, WP3, WP4, WP5
R-GEN-004	Communication data rate MUST be suitable for the defined Use Cases	WP2,WP3
R-GEN-006	The overall operational system SHOULD be dynamically scalable.	WP2,WP3
R-GEN-007	System MUST provide cooperation between operators and machines	ALL
R-GEN-009	The overall system MUST be modular and scalable	ALL
R-GEN-010	System MUST comply with existing safety and security regulations	ALL
R-GEN-014	Machines MUST be able to assist semi-autonomously (assisted movements) in factory environment	ALL
R-GEN-015	System MUST provide embedded control panels to support the new functions of the machines	ALL
R-GEN-019	The data communication MUST be secured from unauthorized access	WP2, WP4
R-GEN-020	Machines MUST be self-described	WP2
R-GEN-021	Integrity of localization data MUST be protected	WP3, WP4
R-GEN-022	Collision avoidance between transported loads and human operators MUST be implemented	WP2, WP3, WP4
R-GEN-023	Collision avoidance between transported loads and other machines or obstacles MUST be implemented	WP2, WP3, WP4
R-GEN-028	System SHOULD support external sensor module data storage and management	WP2
R-GEN-029	System SHOULD allow access to the stored sensor module data to external systems	WP2

Table 5. Detailed Requirements

ID	Requirement Statement	Nature	WPs addressed
R-DET-001	Control components MUST be cost effective	non-functional	ALL
R-DET-002	Control system MUST have short reaction times (real-time capabilities)	non-functional	WP2/WP3
R-DET-004	The control components MUST support short set up time	non-functional	WP2/WP3/WP4
R-DET-005	Wireless communication MUST be highly available	non-functional	WP2/WP3
R-DET-006	Wireless communication MUST be suitable for high transmitter density	non-functional	WP2
R-DET-007	M2M communication data rate w/o video-data SHOULD be in a specific range	non-functional	WP2
R-DET-008	The communication system SHOULD support fast communication set up time	non-functional	WP2
R-DET-009	The control components SHOULD be characterized with a Cat.3 Architecture	non-functional	WP2, WP3, WP4

R-DET-010	Control system's HW components SHOULD have compact dimensions	non-functional	ALL
R-DET-011	Control system MUST be modular and scalable	non-functional	WP2, WP3, WP4
R-DET-013	The control System SHOULD allow deterministic guarantees	non-functional	ALL
R-DET-016	System SHOULD provide machine cooperation functions	functional	ALL
R-DET-017	Manual system operability MUST be guaranteed in case of communication or system break down	functional	ALL
R-DET-019	Interoperability with existing machines and machines from other manufactures SHOULD be guaranteed	non-functional	WP2, WP3, WP4
R-DET-020	Wired control-panels CAN be available on the machines	non-functional	ALL
R-DET-022	Wireless HMI devices SHOULD be made available to the operator for the remote control of the machine	non-functional	ALL
R-DET-023	GUI for wireless HMI devices SHOULD be provided	non-functional	WP2, WP3, WP4
R-DET-024	HMI SHOULD provide manual selection of machines for the execution of specific tasks	functional	WP2, WP3, WP5
R-DET-025	HMI CAN allow specification of tasks and their corresponding properties, e.g. size and weight of load be transported	functional	WP2, WP3, WP6
R-DET-030	System SHOULD collect machine usage information	functional	WP2, WP3, WP4
R-DET-031	System CAN store and manage data for data-analysis and reporting	functional	WP2, WP3, WP4, WP5
R-DET-033	System SHOULD allow remote firmware updates	functional	WP2, WP3, WP4
R-DET-035	Wireless communication between machines SHOULD be provided	functional	WP2
R-DET-037	Secure and safety compliant communication proto-cols MUST be used to connect wireless control- and HMI-devices	non-functional	WP2
R-DET-040	Connection-oriented solutions for low-level communication SHOULD be implemented	non-functional	WP2
R-DET-043	A standard M2M protocol SHOULD be used for the inter-machine communication	non-functional	WP2
R-DET-044	Standard M2M communication protocols SHOULD be used for communication at high-level (enterprise level)	non-functional	WP2
R-DET-051	Disconnections and/or interruptions of communication between operator and machines MUST preserve human safety	non-functional	WP2
R-DET-067	Machines MUST be able to avoid collisions with other stationary machines, operators and obstacles	functional	WP2, WP3, WP4
R-DET-068	Machines MUST be able to avoid collisions with other moving machines and moving operators	functional	WP2, WP3, WP4
R-DET-072	System SHOULD allow the definition of virtual exclusion zones	functional	WP2, WP4, WP5
R-DET-082	Control system MUST be able to collect incoming data stream in real time (e.g. sampling rate given any sensor).	functional	WP2, WP3
R-DET-084	The machines MUST be able to monitor their circumstance and detect specific events in real time.	functional	WP2, WP3
R-DET-086	The machine SHOULD be able to filter and manage unstable data (data loss, collection error, collection delay, etc.) that occur during the data collection process.	functional	WP2
R-DET-118	System SHOULD support machine data collection, data management, and data communication to/from machines	Functional	WP2

7 Specification of the IoT-platform Architecture for OPTIMUM [ERSTE, Thorsis, ifak]

7.1 Provided Functionalities

The analysis of the requirements specified in deliverable D1.2 and summarized in Section 6 allows to define the following functionalities for the IIoT-platform:

- Connectivity and information exchange (high priority)
- Reliability of the communication (high priority)
- Wireless communication (high priority)
- Providing standard-interfaces (high priority)
- Support remote HMI (high priority)
- Security (high priority)
  - Machine authentication
  - User authentication
  - HMI authentication
  - Authentication of localization devices
- Interoperability with existing machines (medium priority)
- Compatibility with existing standards (medium priority)
- Data storage, data management (medium priority)
  - Position information of connected devices
- Dynamic resource discovery (low priority)
  - List of machines connected to the system
  - Descriptions of machine connected to the system
- Remote firmware update (low priority)

The functionalities are ordered by priority, from "high" till "low". These values are derived directly from D1.2 depending on the priority assigned to the specifications contained in the document.

From the list above, it is clear that the main functionality provided by the IIoT-platform is the secure and reliable interconnection of different industrial devices, machines and HMIs, possibly via wireless communication. In the following a general description of the IIoT-platform architecture is given. More details about the available components are presented in Section 7.3.

7.2 General Description

In Figure 43 the overall structure of the OPTIMUM-architecture is presented. As illustrated, the system is built hierarchically: at the low-level the communication between the industrial devices, e.g. cranes and fork lifters, but also sensors and actuators, takes place; at high-level, enterprise solutions, such as data storage systems and cloud-systems, are present. The user/operator is practically located at the center of the system and s/he can control different devices and trigger specific tasks by using a remote or a local HMI.

The specifications analysis shows that at low-level delays and information losses in the communication might strongly compromise the system performance, such for example in the use-case "Follow Machine", where two cranes must cooperate to realize a tandem function and carefully carry together a heavy load. For this reason, solutions and protocols that can guarantee connection-oriented real-time communication are preferred in the OPTIMUM-project.

For this purpose, in accordance with the specified requirements, the lightweight MQTT-protocol is selected as good candidate for the communication between the components of the industrial device, such as DCP and IIoT-platform (see Figure 44). Although MQTT is not proven to be real-time, it provides small packet overhead and, therefore, it can guarantee an efficient use of the communication medium. Performance analysis to investigate the real-time capabilities of MQTT are planned during the execution of WP6.

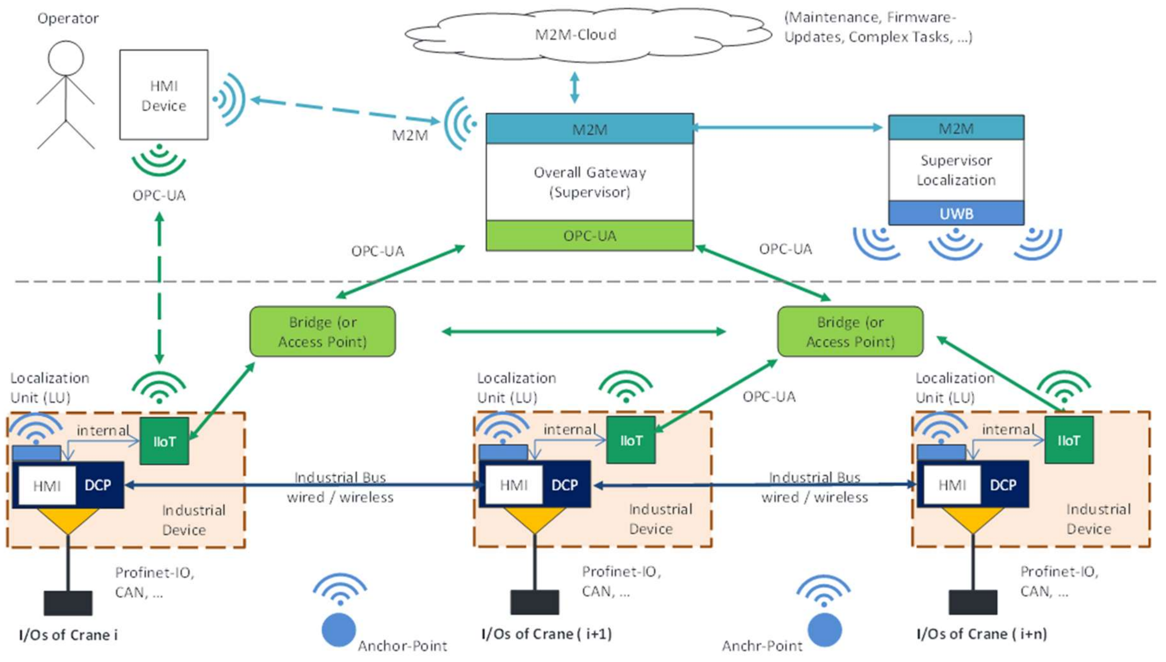


Figure 43: Interaction between industrial devices, HMI and cloud data system.

At high level (or enterprise level), the communication is not time-critical. Therefore, OCP-UA was selected as candidate protocol for the M2M communication. As illustrated in Figure 44, the IIoT-platform is responsible for translating the MQTT messages into the OPC-UA protocol and vice versa, such that interoperability between different industrial devices can be guaranteed. The binding between external systems such as the 3D-Simulation and the Data Management System is also realized via OPC-UA.

The localization sensors are directly connected to the embedded hardware via CAN-bus. They communicate with the DCP and provide directly information to the control units, without using the IIoT-platform. Safety requirements related to the position between industrial devices, such as collision avoidance, are solved directly at this level by the localization system – which directly provides proximity zones – together with the DCP, without requiring the intervention of the IIoT-platform. In other words, the position information of other industrial devices is exchanged via the IIoT-platform only for the execution of specific complex tasks that are not time-critical. Further information on how collision avoidance is realized w.r.t. the implemented localization system is provided in the documentation of the demonstrators (WP6).

It shall also be noticed that, from the Physical Layer point of view, the choice of what technology shall be used for the communication between the industrial devices is left open. Different systems can, therefore, use different technologies, such as WiFi, LTE 5G, or Ethernet, as long as the available infrastructure provides TCP/IP functionalities, i.e. the IIoT-platform architecture of OPTIMUM is independent from the actual communication medium which can be both wired and wireless but requires its resources to be addressable with proper IPs.

Further information about the components of the IIoT-platform are specified in the following section.

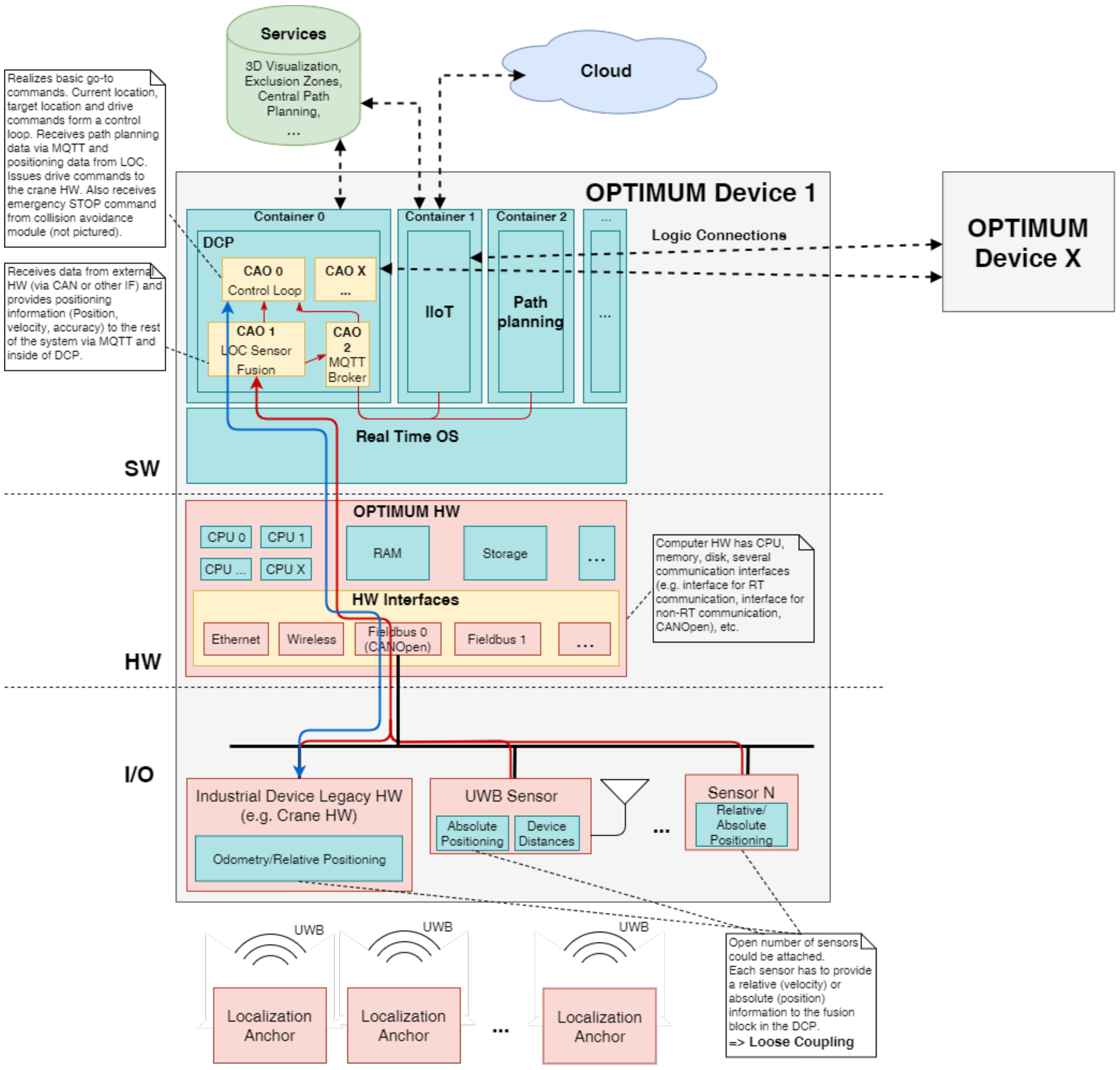

Services
3D Visualization, Exclusion Zones, Central Path Planning, ...
Cloud
Realizes basic go-to commands. Current location, target location and drive commands form a control loop. Receives path planning data via MQTT and positioning data from LOC. Issues drive commands to the crane HW. Also receives emergency STOP command from collision avoidance module (not pictured).
Receives data from external HW (via CAN or other IF) and provides positioning information (Position, velocity, accuracy) to the rest of the system via MQTT and inside of DCP.
OPTIMUM Device 1
Container 0
Container 1
Container 2
...
DCP
CAO 0
Control Loop
CAO X
...
CAO 1
LOC Sensor Fusion
CAO 2
MQTT Broker
IIoT
Path planning
Logic Connections
OPTIMUM Device X
Real Time OS
SW
OPTIMUM HW
CPU 0
CPU 1
CPU ...
CPU X
RAM
Storage
...
HW Interfaces
Ethernet
Wireless
Fieldbus 0 (CANOpen)
Fieldbus 1
...
Computer HW has CPU, memory, disk, several communication interfaces (e.g. interface for RT communication, interface for non-RT communication, CANOpen), etc.
HW
I/O
Industrial Device Legacy HW (e.g. Crane HW)
Odometry/Relative Positioning
UWB Sensor
Absolute Positioning
Device Distances
Sensor N
Relative/ Absolute Positioning
UWB
Localization Anchor
Open number of sensors could be attached. Each sensor has to provide a relative (velocity) or absolute (position) information to the fusion block in the DCP.
=> Loose Coupling

Figure 44: Overall hardware-software representation of the OPTIMUM industrial devices and its binding with the existing embedded hardware as well as other OPTIMUM subsystems.

7.3 Architecture Overview

The IIoT-platform is composed by different blocks which represents logical components implementing one or more sets of functions. An overview of the IIoT-platform architecture is presented in Figure 45.

It is assumed that the industrial device provides a real time operating system (OS), where the IIoT-platform is implemented as a stand-alone container (see Figure 44) with interfaces to the other external – w.r.t. the IIoT-platform – components. Functions like process- and thread-execution, memory management, network management at Physical, Datalink, Transport and Network Layer – w.r.t. to the ISO-OSI-Model – are provided explicitly by the OS.

In the remaining of this chapter, the single components of the IIoT-platform are described in more detail. Notice that these are the minimal components to fulfil the requirements of D1.2 and realize the functionalities specified in Section 7.1, but they might not necessarily be the only components of the IIoT-platform. In fact, depending on the specific implementation, additional functionalities might be required and further components at Application Level can be added. Nevertheless, the presented IIoT-architecture provides sufficient degrees of freedom to further extend the results of the OPTIMUM-project also after its conclusion.

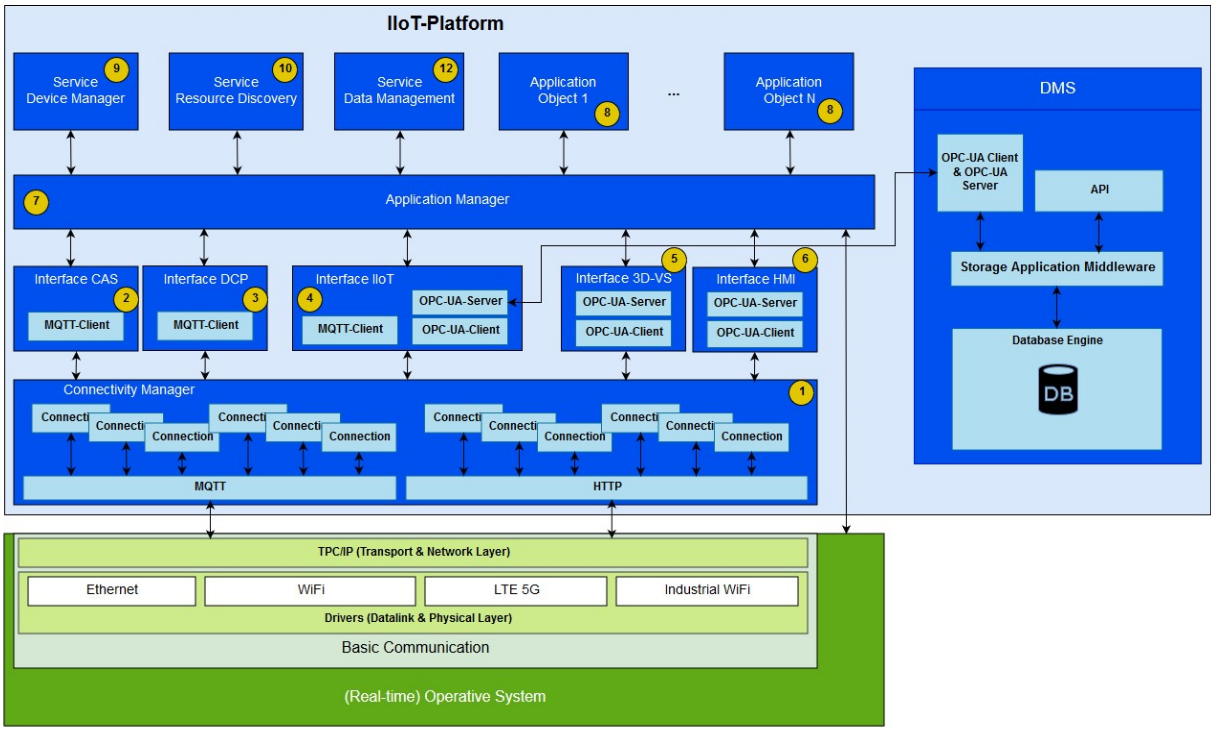

IIoT-Platform
Service Device Manager 9
Service Resource Discovery 10
Service Data Management 12
Application Object 1 8
...
Application Object N 8
DMS
OPC-UA Client & OPC-UA Server
API
Storage Application Middleware
Database Engine
DB
7 Application Manager
Interface CAS 2 MQTT-Client
Interface DCP 3 MQTT-Client
Interface IIoT 4 OPC-UA-Server MQTT-Client OPC-UA-Client
Interface 3D-VS 5 OPC-UA-Server OPC-UA-Client
Interface HMI 6 OPC-UA-Server OPC-UA-Client
Connectivity Manager 1
Connection
MQTT
HTTP
TPC/IP (Transport & Network Layer)
Ethernet
WiFi
LTE 5G
Industrial WiFi
Drivers (Datalink & Physical Layer)
Basic Communication
(Real-time) Operative System

Figure 45: Architecture of the IIoT-platform.

7.3.1 Connectivity Manager (CM)

It is assumed that the OS and the communication infrastructure provide TCP/IP functionalities, i.e. basic communication for the Physical, Datalink, Transport and Network Layer are provided by the OS. The Connectivity Manager (CM) is responsible to control and manage all possible internal and external connections w.r.t. the industrial device. In other words, the CM ensure that a connection, for instance between the IIoT-platforms of two different industrial devices, is reliable and still alive. If the connection is performed over WiFi, Ethernet, or another communication technology is irrelevant from the CM point of view.

With respect to the available requirements, the CM provides specific support for different standards:

- MQTT, for the internal and external communication at industrial device level, e.g. the communication between DPC and IIoT-platform of the same industrial device, or of two different devices.
- HTTP and/or CoAP, both for the external communication at enterprise level, e.g. for the communication with the Data Management System or the CAS.

By providing support to different protocols, the CM allows the implementation of the standard OPC-UA, which in OPTIMUM is, in particular, used for the Data Management System and the 3D-virtualization and Simulation System as well as M2M-standards like oneM2M or etsiM2M. Support for the direct binding of OCP-UA with TCP/IP is also provided. Therefore, the CM implements the basis for both IoT-stacks for oneM2M and OPC-UA as well as all functionalities which are required for the instantiation of (logical) Connections among the components of the industrial devices.

Connection

A Connection is a logical remote or local link between components of the industrial devices. Whenever connections are remote, if not already provided by other components, a "keep alive" mechanism to guarantee that the link is never interrupted shall be implemented. Whenever no further response from the other industrial device is received, the connection is considered broken and the corresponding associated service is interrupted or preempted. Solutions to automatically awake a preempted service whenever the logical connection is restored shall be provided by the CM.

7.3.2 Interface Context-Awareness-System (CAS)

To interact with the Context-Awareness-System (CAS) the IIoT-platform includes a specific communication interface. All data which need to be exchanged to internal or external CAS shall be directed through this interface. The interface includes an MQTT-client for lightweight communication, such that the coupling between IIoT and CAS is loose.

7.3.3 Interface Distributed Control Platform (DCP)

The interactions between IIoT and DCP are realized via this interface. All data which are needed from or by the DCP are communicated via MQTT. Therefore, an MQTT-client for the loose coupling between DCP and IIoT is included in the Interface DCP. At the DCP-side an MQTT-broker shall be provided.

7.3.4 Interface IIoT

The IIoT of one industrial device can require communication with the IIoT of another one. For this reason, a specific interface for the communication between the IIoT-platforms of two different devices is provided. Since the communication can happen both at high and low level, the Interface IIoT component provided both services for MQTT, by implementing an MQTT-client, and for OPC-UA, by implementing both an OPC-UA-client and an OPC-UA-server. Both types of connections are loose and non-time-critical for the execution of specific tasks.

7.3.5 Interface 3D-Virtualization-System (3D-VS)

To bind the 3D-simulation software with the industrial device a specific interface is provided. In this case, since the communication is at high level, only OPC-UA functionalities are provided. An OPC-UA-client and an OPC-UA-server are implemented in the Interface 3D-VS. The communication is also in this case loose and non-time-critical.

Notice that a direct connection between 3D-VS and DCP in case of time-critical information exchange is also possible. This topic is although outside the scope of this document and related to the specification of the DCP. Therefore, it is not discussed in this deliverable.

7.3.6 Interface Human-Machine-Interaction (HMI)

The IIoT-platform shall provide also support for the connection of (remote) HMI devices. To this aim, the IIoT-platform integrates an Interface HMI, which implements the OPC-UA-protocol. Both an OPC-UA-client and OPC-UA-server are implemented.

Notice that the support and implementation of specific communication protocols for the remote HMI which rely on time-relevant data exchange for control purposes with the DCP, e.g. wirelessCAN, is outside the scope of this document.

7.3.7 Application Manager (AM)

Services or Application Objects as well as the other components implemented in the IIoT-platform may require instantiation, coordination, management, preemption, triggering, or access to low-level functions provided by the OS. To this intent, an Application Manager is implemented in the IIoT-Platform. This component is, in particular, responsible for managing the tasks between the Application Objects as well as the communication and data exchange with other internal and external components of the IIoT-platform, by commanding the Connection Manager the opening and closing of new connections. It grants the high level services access to the communication as well as to all functionalities provided by the OS and the other components of the industrial device, such as DCP and CAS.

7.3.8 Application Objects (AO)

An Application Object is a high level Service at Application Level. In practice, this realizes the high level or functionalities that the IIoT-platform shall provide.

According to the specified requirements, the IIoT-platform is responsible for providing three main services: Device Manager, Resource Discovery, Data Management.

These are only the basic services that were derived from the document D1.2. The IIoT-platform is although not limited to these and it can be extended with additional application objects or services, which are outside the scope of the OPTIMUM-project.

7.3.9 Service Device Manager

The Service Device Manager is responsible for the management of all information and functionalities which are strictly related to the industrial device, its function, its hardware and its software. In particular, these can be summarized as follows:

- Management, storage and reporting of the current device status
- Identification, isolation and reporting of errors, failures and malfunctions
- Management of software updates related to the IIoT-platform and its functionalities[2]
- Management and optimization of energy consumption
- Management of device and components authentication

7.3.10 Service Resource Discovery

The IIoT-platform shall also provide support for dynamic resource discovery. To this intent, the Service Resource Discovery is implemented as Application Object in the IIoT-platform. The functionalities of the Service Resource Discovery include:

- Store and report information about the attributes of the device (type, characteristic, available functions, etc.)
- Inform surrounding devices about the presence/activation of a new industrial device
- Manage the list of reachable available (active) devices (add, remove, store, etc.)

7.3.11 Service Data Management

The Service Data Management is an Application Object with the aim of providing following functionalities:

[2] Updates related to other components such as DCP, OS, e.g. firmware updates, are not in the scope of the Service Device Manager.

- Storage and management of local data essential for the industrial device
- Collect and temporary store data that must be transferred to the data system
- Communicate and exchange data with the data- and cloud-system
- Identify and sort out incorrect, invalid and/or inconsistent data before submitting them to the data system

8 Specification of the Data Management System for OPTIMUM

8.1 Provided Functionalities

The Data Management System will allow integration with External Sensing Devices

One of the main responsibilities of the Data Management System (DMS) is to allow integration and connectivity with External Sensing Devices (ESD), which in turn shall allow non-monitored machines and processes to access the OPTIMUM environment. In order to integrate ESDs into the OPTIMUM environment and kick-start the data storage and management processes handled by the DMS, the ESD needs only to integrate with the industrial device via the MQTT interface of its internal IIoT subcomponent. Figure 46 illustrates this integration process:

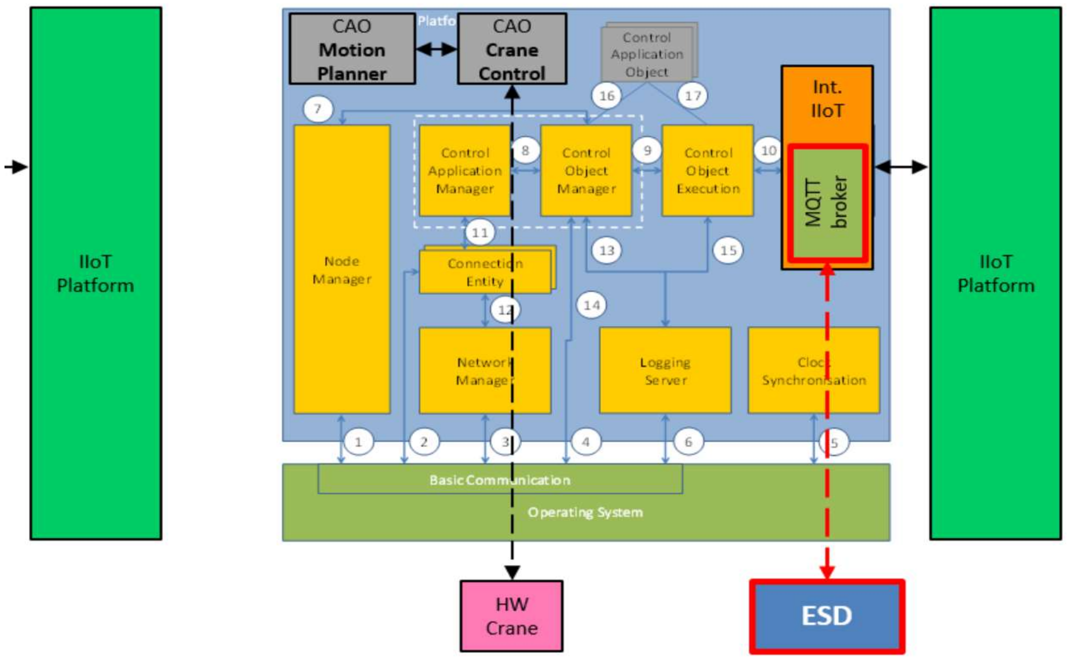


Figure 46: ESD integration with OPTIMUM DCP via MQTT broker

The Data Management System will provide multiple storage solutions, such as local and cloud approaches.

In order to appeal to different needs and situations, the Data Management System will provide functionality to store data associated to a machine or process locally. The following figure illustrates this process:

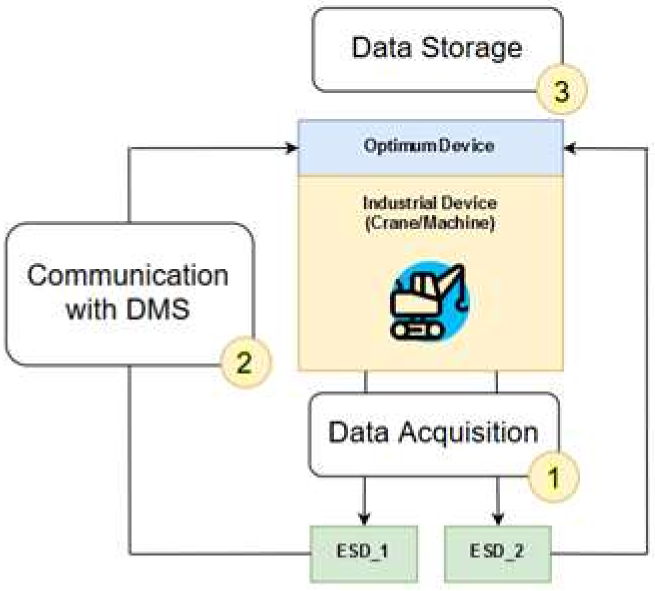


Figure 47: ESD data flow and local storage

The initial setup of the situation described in Figure 47 is that an industrial device is linked or associated to an existing industrial process or hardware, such as a crane, press, conveyor belt, or a whole process such as cutting metallic pieces. The workflow regarding data generation and storage relevant to the industrial hardware or process is as follows:

1. **Data Acquisition:** Once the initial scenario is established, the first step corresponds to data acquisition. The data acquisition step involves using the ESDs associated to the industrial hardware or process in question. There could be one or more ESDs associated to this data acquisition process. For instance, one ESD could be responsible for measuring the time taken to complete a process, while another ESD could be responsible for other metrics such as throughput and amount of produced waste. Additionally, these metrics could also be managed by a single ESD. This is where the flexibility of ESDs come into play and it is responsibility of the ESD user or manufacturer to produce a device that suits its needs.

2. **Communication with DMS:** Having collected the data, the next step would be for the involved ESDs to establish communication with the DMS component located in the IIoT platform of the industrial device. This will be done by having the ESD connect to the MQTT broker situated within the DCP component of the industrial device.

3. **Data Storage:** When the DMS receives the incoming data from the different ESDs, it is this component's responsibility to locally store the data within the IIoT-platform for later use.

This data acquisition and storage procedure can be repeated for all the machines and process that wish to be monitored or integrated into the OPTIMUM environment:

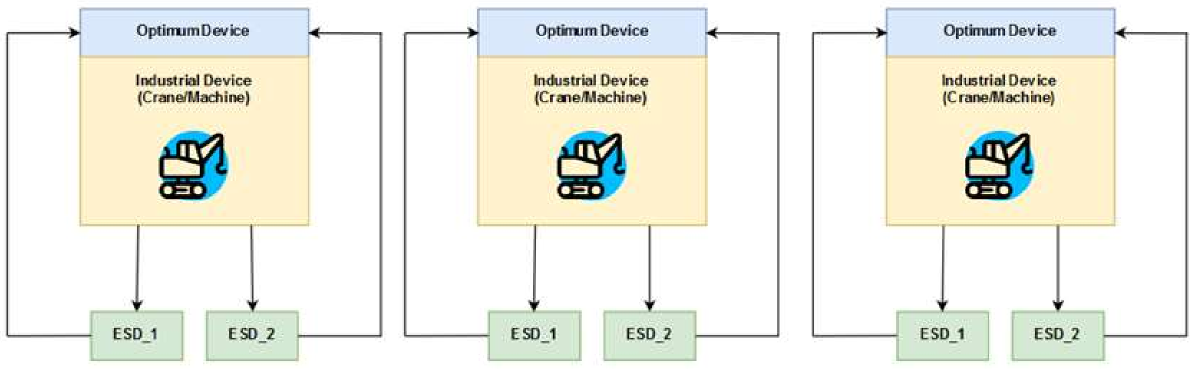

Optimum Device
Industrial Device (Crane/Machine)
ESD_1
ESD_2
Optimum Device
Industrial Device (Crane/Machine)
ESD_1
ESD_2
Optimum Device
Industrial Device (Crane/Machine)
ESD_1
ESD_2

Figure 48: Multiple industrial devices with associated ESDs

With respect to accessing the data by a cloud application or system, the process would simply add another step as shown in Figure 49:

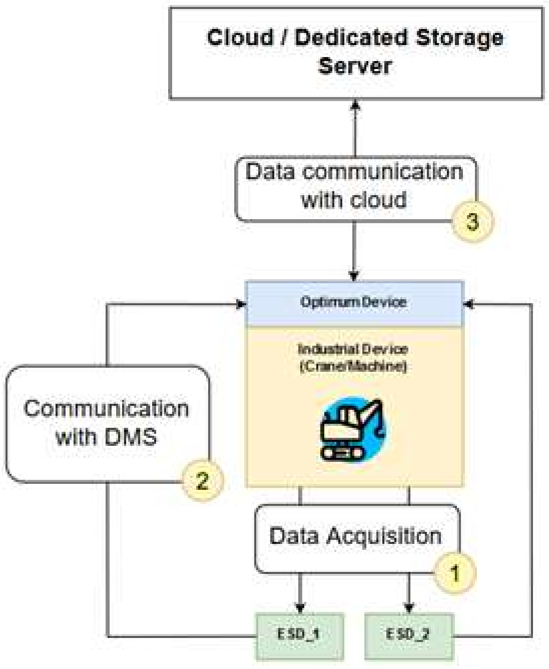

Cloud / Dedicated Storage Server
Data communication with cloud
3
Optimum Device
Industrial Device (Crane/Machine)
Communication with DMS
2
Data Acquisition
1
ESD_1
ESD_2

Figure 49: Cloud communication in industrial device monitoring

Regarding this functionality, all the steps are the same up until data are correctly stored in the DMS within the industrial device's IIoT platform. However, the DMS will offer an API that provides access to its stored data and thus a way to interface with cloud solutions.

8.2 General Description

This section provides a general description of the responsibilities of the proposed Data Management System for the OPTIMUM project, as well as further detailing the concept of External Sensor Module.

In general, the main responsibility of OPTIMUM's DMS is to store and manage data coming from External Sensing Devices or any general IoT sensor with compatible interfacing capabilities. In this context, we will refer to an ESD as an IoT device created for the sole reason of integrating an existing machine or process that does not inherently have any sensing capabilities, with the industrial device, and ultimately, into the OPTIMUM environment.

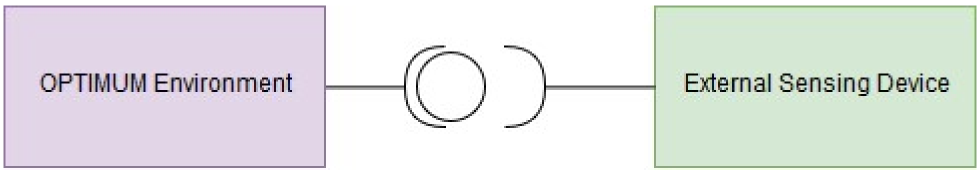


Figure 50: Integration between OPTIMUM environment and ESD

In terms of architecture, an ESD is made up of one or more sensors and possesses the ability to communicate its acquired sensor data with the OPTIMUM environment.

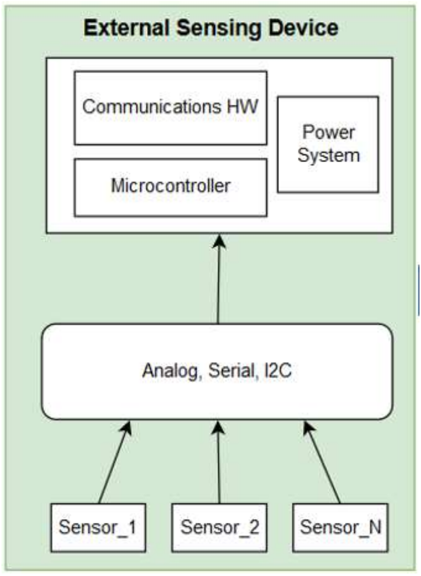


Figure 51: High level ESD component diagram

These IoT devices are external to the OPTIMUM system, and they may be prototyped or manufactured by third-parties. However, the idea is to allow the industrial device to interface with these ESDs and store their data within the DMS. This will broaden the scope of OPTIMUM's usability by allowing many types of machines and equipment, without having to worry about their inherent interfacing capabilities, to be integrated into the OPTIMUM environment via the use of ESD's acting as a middleware or intermediate step.

As a practical example, it is helpful to consider a factory that is not yet equipped with modern tools and machinery and is subject to using antiquated equipment. Whereas modern machinery will possibly be equipped with interfacing capabilities with other systems, for instance, to monitor the tools usage statistics, system metrics, or overall health, outdated equipment will have no possibility of doing so. Given that overhauling the factory's whole set of tools and machines to newer models in order to propel that initial undertaking into an Industry 4.0 workspace may be expensive and poses its own limiting barriers, a more adequate approach would be to introduce ESD's into the factory floor which will be capable of externally monitoring a machine or process' metrics, which can thusly be integrated into the OPTIMUM system.

Additionally, OPTIMUM's DMS will allow communicating with cloud applications and systems in order to centralize the data that it has obtained.

In conclusion, the Data Management System of OPTIMUM will allow the storage and management of data obtained from External Sensing Devices and other compatible IoT devices and provide functionalities to centralize the data in a cloud environment.

8.3 Architecture Overview

This section aims to describe the overall architecture of the Data Management System by taking a look at the intended integration with the other OPTIMUM components, including the External Sensing Devices as well as the DCP within the industrial device.

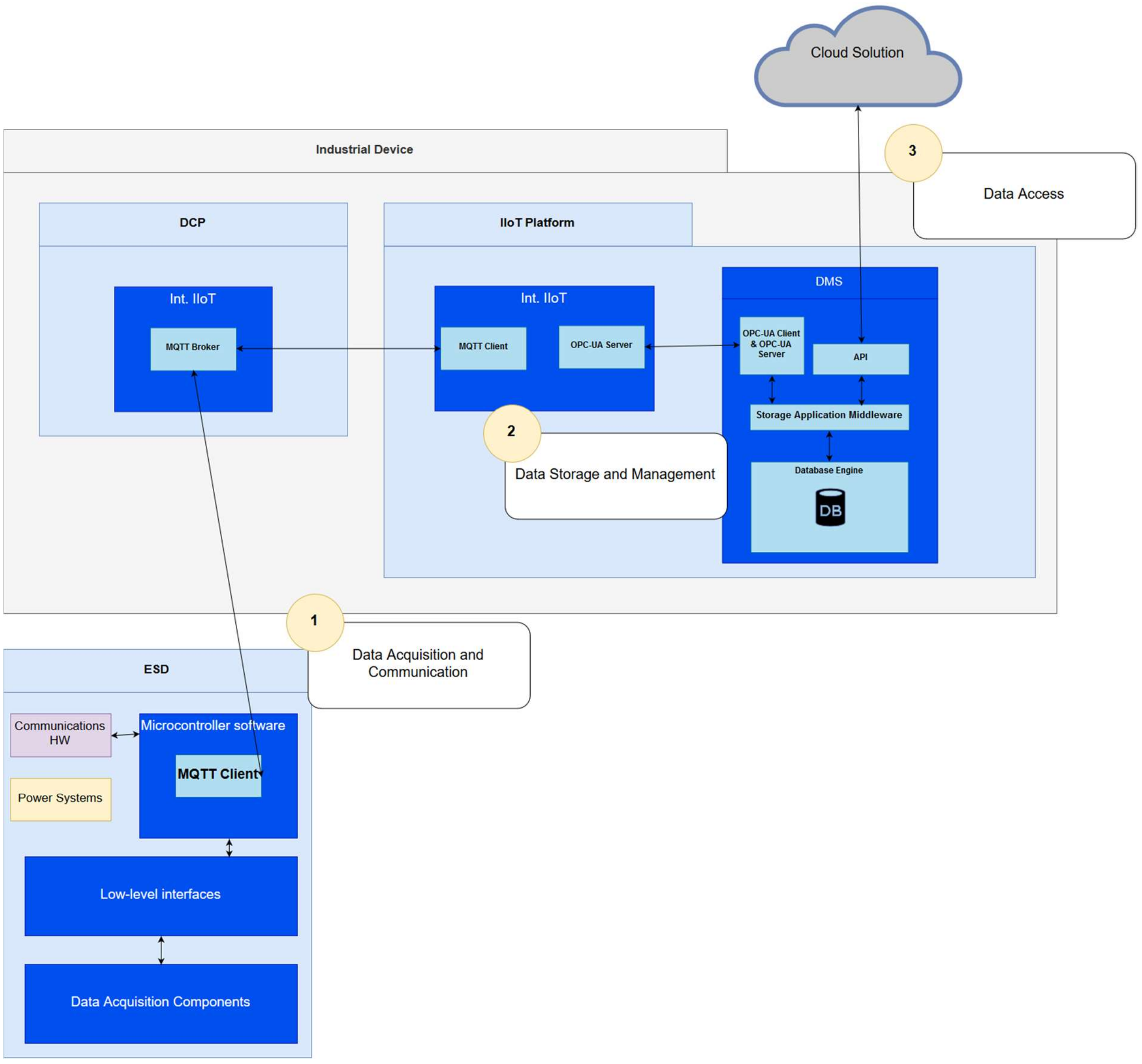


Figure 52: Architectural overview of ESD, DCP and DMS

As we can see from the diagram, the first integration occurs between the ESD and the DCP of the industrial device. This is where data acquisition occurs, and the integration is done via the DCP's MQTT broker. Afterwards, the integration between the DMS and the DCP takes place, also using the MQTT broker provided by the DCP component. This integration allows for data storage and management by including the DMS into the overall view. Finally, the DMS will contain an API that allows external systems such as cloud solutions to access the data stores.

To further detail the DMS components, the following diagram can be observed:

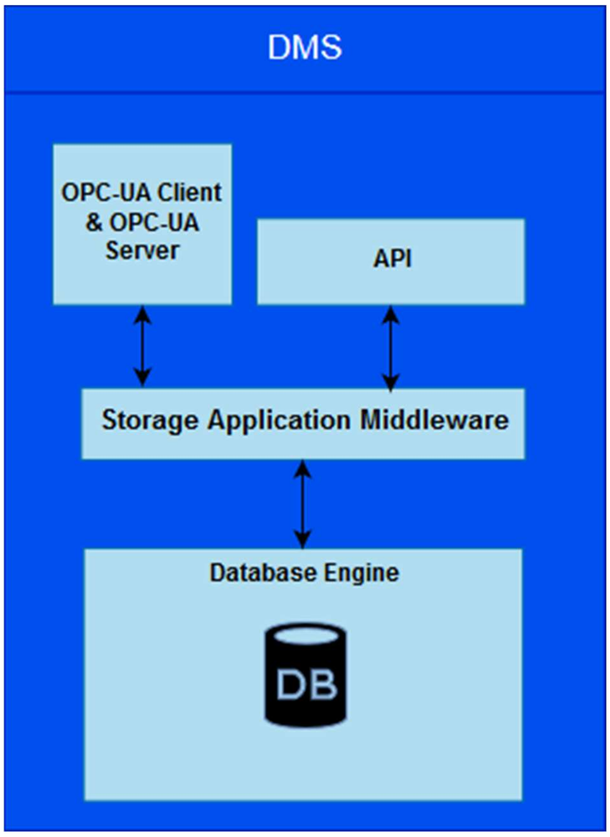


Figure 53: DMS architecture

- **OPC-UA Client & Server:** the DMS comes equipped with OPC-UA client and server instances that will allow it to obtain and share data from the IIoT Platform's OPC-UA server located within the Interface IIoT component.
- **Storage Application Middleware:** this application middleware is the logical connection between the data received from the MQTT client and the underlying database engine. This is the software that manages the logic between the received data and the storage media.

- **Database Engine:** the database engine is responsible for managing the data and storing it in the physical media of the device. This component handles the basis of all storage and management functions.

- **API:** the API provides a set of endpoints to the application middleware, which in turn decides which functionality can be ported from the database engine. With this API, external systems such as solutions stored in the cloud can access the data stored within the DMS via the permitted functionality established by the application middleware.

9 Abbreviations

3D-VS	3D-visualization software OR 3D-virtualization software
AND	Application dedicated node
AE	Application entity
AM	Application manager
AMQP	Advanced message queuing protocol
AO	Application objects
API	Application programming interface
ASN	Application service node
B2B	Business to business
B2C	Business to customer
BLE	Bluetooth low energy
CARP	Channel-aware routing protocol
CAS	Context awareness system
CIA	Confidentiality, integrity, availability
CM	Connectivity manager
CoAP	Constrained application protocol
CORPL	Cognitive RPL
CSE	Common service entity
CSF	Common service function
DAO	Destination advertisement object
DAO_ACK	DAO acknowledgment
DCP	Decentralized control platform
DCS	Distributed Control System
DDS	Data distribution service
DIO	DODAG information object
DIS	DODAG information solicitation
DODAG	Destination oriented directed acyclic graph
DTLS	Datagram transport layer security
EPC	Electronic product code
ERP	Enterprise resource planning
ICT	Information and communication technology
IEC	International Electrotechnical commission
IETF	Internet engineering task force
IIoT	Industrial Internet of Things

IN	Infrastructure node
IoT	Internet of things
IP	Internet protocol
GE	Generic enabler
HAP	HomeKit accessory protocol
HMI	Human-machine-interaction (device or interface)
HVAC	Heating, ventilation and air conditioning device
JSON	JavaScript object notation
LLC	Logical link control
LTE	Long term evolution
M2M	Machine to machine
MAC	Medium access control OR Message authentication code
MD	Middle node
MPU	Micro Processing Unit
MQTT	Message queuing telemetry transport
MS/TP	Master-slave/token-passing
NGN	Next generation networks
NSE	Network service entity
OASIS	Organization for the advancement of structured information standards
OPC-UA	Open platform communications unified architecture
OS	Operative system
OT	Operational technology
PHY	Physical layer
PLC	Programmable logic controller
QoS	Quality of service
REST	Representational state transfer
RFID	Radio frequency identifier
RPL	Routing protocol for low power and lossy networks
SDL	Service description language
SE	Secure element
SoA	Service oriented application
SoC	Service oriented computing
SoM	Service oriented middleware
SSL	Secure socket layer
TCP	Transmission control protocol
TDMA	Time division multiple access
TLS	Transport layer security
TSCH	Time synchronization channel
UDP	User datagram protocol
URI	Uniform (or unique) resource identifier
URL	Uniform resource locator
VDMA	Verband Deutscher Maschinen- und Anlagenbau
VM	Virtual machine
VPN	Virtual private network
WIA-PA	Wireless network for industrial automation process automation

WLAN	Wireless local area network
XML	Extensible markup language
XMPP	Extensible messaging and presence protocol
WPAN	Wireless personal area network
WPx	Work package x, where x corresponds the work package number
w.r.t.	With respect to
WSN	Wide sensor network

10 References

[1] Chebudie, Abiy Biru, Minerva, Roberto, Rotondi, Domenico. (2014). Towards a definition of the Internet of Things (IoT). IEEE Internet Initiative, May 2015.

[2] Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. Future generation computer systems, 29(7), 1645-1660.

[3] Atzori, L., Iera, A., & Morabito, G. (2010). The internet of things: A survey. Computer networks, 54(15), 2787-2805.

[4] Singh, D., Tripathi, G., & Jara, A. J. (2014, March). A survey of Internet-of-Things: Future vision, architecture, challenges and services. In Internet of things (WF-IoT), 2014 IEEE world forum on (pp. 287-292). IEEE.

[5] Derhamy, H., Eliasson, J., Delsing, J., & Priller, P. (2015). A survey of commercial frameworks for the internet of things. In IEEE International Conference on Emerging Technologies and Factory Automation: 08/09/2015-11/09/2015. IEEE Communications Society.

[6] Lucero, S. (2016). IoT platforms: enabling the Internet of Things. White paper.

[7] Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Clarke, S. (2016). Middleware for Internet of Things: A survey. IEEE Internet of Things Journal, 3(1), 70-95.

[8] Zdravković, M., Trajanović, M., Sarraipa, J., Jardim-Gonçalves, R., Lezoche, M., Aubry, A., & Panetto, H. (2016, February). Survey of Internet-of-Things platforms. In 6th International Conference on Information Society and Techology, ICIST 2016 (Vol. 1, pp. 216-220).

[9] Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Clarke, S. (2016). Middleware for Internet of Things: A survey. IEEE Internet of Things Journal, 3(1), 70-95.

[10] Ray, P. P. (2016). A survey on Internet of Things architectures. Journal of King Saud University-Computer and Information Sciences.

[11] Da Xu, L., He, W., & Li, S. (2014). Internet of things in industries: A survey. IEEE Transactions on industrial informatics, 10(4), 2233-2243.

[12] Guth, J., Breitenbücher, U., Falkenthal, M., Leymann, F., & Reinfurt, L. (2016, November). Comparison of IoT platform architectures: A field study based on a reference architecture. In Cloudification of the Internet of Things (CIoT) (pp. 1-6). IEEE.

[13] Sivaraman, V., Gharakheili, H. H., Vishwanath, A., Boreli, R., & Mehani, O. (2015, October). Network-level security and privacy control for smart-home IoT devices. In Wireless and Mobile Computing, Networking and Communications (WiMob), 2015 IEEE 11th International Conference on (pp. 163-167). IEEE.

[14] Linden Tibbets and Jesse Tane. (2012). IFTTT. [Online].

[15] Ngu, A. H., Gutierrez, M., Metsis, V., Nepal, S., & Sheng, Q. Z. (2017). IoT middleware: A survey on issues and enabling technologies. IEEE Internet of Things Journal, 4(1), 1-20.

[16] Analytics, I. (2015). IoT Platforms The central backbone for the Internet of Things. White paper.

[17] D. Niewolny (2013), How the internet of things is revolutionizing healthcare, White paper.

[18] Jayaraman, P. P., Palmer, D., Zaslavsky, A., & Georgakopoulos, D. (2015, April). Do-it-Yourself Digital Agriculture applications with semantically enhanced IoT platform. In Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2015 IEEE Tenth International Conference on (pp. 1-6). IEEE.

[19] Sicari, S., Rizzardi, A., Grieco, L. A., & Coen-Porisini, A. (2015). Security, privacy and trust in Internet of Things: The road ahead. Computer networks, 76, 146-164.
[20] Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., ... & Scholz, U. (2009). Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In Software engineering for self-adaptive systems (pp. 164-182). Springer, Berlin, Heidelberg.
[21] OPC Foundation, IEC 62541 - OPC UA Architecture Specification - Part 1: Overview and Concepts, V1.04, OPC Foundation, 2017.
[22] Cisco, Time Sensitive Networking: A Technical Introduction, Cisco Public, White Paper, 2017
[23] J. Blackman, What is OPC UA TSN, and how does it work? (The long read, with CISCO and SAP), Enterprise IoT Insights, June 2018
[24] R. Jamal, OPC-UA over Time Sensitive Network (TSN) – Standard für Industrial IoT, Proc. of VIP – Virtuelle Instrumente in der Praxis – 2017, pp. 288-292, 2017
[25] P. Lutz and S. Gerstl, OPC-UA over TSN – gemeinsame Sprache für die Zukunft, Elektronik Praxis, November 2017
[26] http://portals.omg.org
[27] Publication IIC:PUB:G5:V1.01:PB:20180228 of Industrial Internet Consortium
[28] http://mqtt.org/
[29] https://www.hivemq.com
[30] https://tools.ietf.org/html/rfc7252
[31] https://tools.ietf.org/html/rfc6120
[32] ISO/IEC 19464:2014 - AMQP standard (https://www.iso.org/standard/64955.html)
[33] https://www.amqp.org/
[34] https://www.rabbitmq.com/
[35] A. Aijaz and A. Aghvami, "Cognitive machine-to-machine communications for internet-of-things: A protocol stack perspective", IEEE Internet of Things Journal, vol. 2, no. 2, pp. 103-112, April 2015
[36] Z. Zhou, B. Yao, R. Xing, L. Shu, and S. Bu, "E-CARP: An energy efficient routing protocol for UWSNs in the internet of underwater things", IEEE Sensors Journal, vol. PP, no. 99, 2015
[37] D. Dujovne, T. Watteyne, X. Vilajosana, and P. Thubert, "6TiSCH: Deterministic IP-enabled industrial internet (of things)", IEEE Communications Magazine, vol. 52, no. 12, pp. 36-41, December 2014
[38] IETF, "IPv6 over Networks of Resource-constrained Nodes (6lo)"
[39] J. Nieminen, et al, "IPv6 over Bluetooth Low Energy", IETF RFC 7668, October 2015
[40] T. Salman and R. Jain, "Networking protocols and standards for internet of things", Internet of Things and Data Analytics Handbook, pp. 215-238, 2015