Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis No. 3744

Architectural Design of an Abstraction Layer for the Integration of Heterogeneous Cyber-Physical Systems

Jasmin Alexandra Guth

Course of Study:	Wirtschaftsinformatik
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisors:	Dipl.-Inf. Lukas Reinfurt M.Sc. Michael Falkenthal
Commenced:	October 1, 2015
Completed:	March 23, 2016
CR-Classification:	C.3, D.2.11, D.4.7, I.2.11, H.3.4

Abstract

Recently Cyber-Physical Systems (CPS) gained increasing attention and popularity in the field of information technology (IT). The definite aim of CPS is the integration of physical processes with computation. CPS depend on multiple disciplines and arise of a complex interaction of embedded systems, application systems, and infrastructures, with the interaction of humans and technology, where this complex overall interaction is based on their interconnection and integration. Embedded computers and networks monitor and control the physical processes, usually using feedback-loops, where physical processes affect computations and vice versa. The abstraction layer consists of a reference architecture, and an abstract class model. The aim of the abstraction layer is to provide an architectural basis for the design and the integration of heterogeneous CPS. To derive a reference architecture diverse open-source, as well as proprietary CPS platform architectures are analyzed and compared. Subsequently the reference architecture is validated, by mapping it onto the architectures of the considered CPS platforms. The aim of the reference architecture is to provide an universal basis for the architectural design of CPS. Following this, the features of the considered CPS platforms are analyzed and compared, and subsequently an abstract class model is derived. The abstract class model is also validated, by verifying if the operations of the abstract class model are existent within the considered CPS platforms. The aim of the abstract class model is to provide a basis for the essential functionality, and interconnection of the components of a CPS platform.

Contents

List of Figures

List of Tables

1 Introduction

The Internet of Things (IoT) paradigm gained increasing attention and popularity in the field of information technology (IT) in the last quarter-century [Cha13]. Recently the term Cyber-Physical Systems (CPS) is mentioned coincidentally. IoT describes the vision of a completely interconnected world, where the physical world is connected to the internet with the aid of computing devices [Dom+09] [Kha+12]. CPS elaborate this with the purpose to enable monitoring, interpretation and a possible reaction to physical activities [Sal+15]. To achieve that, the physical world has to be perceived by physical devices, which are usually connected to the internet in an autonomous and secure way [Che+12]. Those devices can send the perceived data through the internet to their defined target, and can possibly be controlled remotely [Zhe+11] [Fan+10] [Che+12] [Kha+12].

Nowadays you can find different software approaches of connected ecosystems for various target audiences [Pos16]. There are CPS available for companies, which enable interconnected production processes, just as the approach of Smart Cities or CPS for consumers. Smart Home and Connected Cars are some collective terms for different solutions provided for consumers. Since the idea and the core logic of CPS are the same, independent of the target audiences, and the scope of the CPS, they can be compared and either expanded or curtailed.

1.1 Problem Domain and Motivation

At the current state of research there is a wide range of open-source and proprietary software solutions available to establish and operate CPS. Within the Postscapes IoT Awards 2014/2015 around 160 approaches participated in 13 categories [Pos16]. The consequence of a missing standard for the design of CPS is, that the software solutions may differ strongly, and hence cannot be combined or integrated. The intention of integrating heterogeneous CPS is, to enable the communication, and especially the data exchange between different CPS platforms. This enables the provision of consistent data [Cha+94]. This Thesis' aim is to develop the basis, i.e., an universal abstraction layer, for the design, and the integration of heterogeneous CPS. The abstraction layer comprises a reference architecture, which represents the basis for the architectural

design of CPS, as well as an abstract class model, providing a reference for the essential functionality, and interconnection of the components of a CPS platform.

1.2 Research Issues and Contributions

To accomplish the abstraction layer, different open-source and proprietary software solutions are considered and precisely analyzed. By selecting different providers, an overall impression is ensured. Subsequently the components, the architecture, and the various features are focused. Conspicuous during the initial research is, that the naming of the components, and crucial fundamentals differ strongly. For instance one platform uses the device in terms of a hardware entity, where sensors and actuators can be connected, and which can send and receive messages via a gateway to/of the platform. Another platform uses the device in terms of a smart device, which has integrated sensors and actuators, which already pre-processes the data gathered, and can directly communicate with the platform. Accordingly this circumstance is considered during the research and elaboration.

1.3 Research Method

Within this Thesis seven different CPS solutions are analyzed and compared. They are determined accordingly the following aspects: Considered within the selection of the CPS solutions are software products listed in [Kop15], which represents similar approaches to an IoT solution introduced in [Gil16]. Since there are not enough detailed information about this solution available, it is not further considered. Additionally considered within the selection of the CPS platforms are the nominated CPS solutions of the Postscape's IoT Awards [Pos16]. Initially the determining factor is the public availability of detailed information about the architecture, and the provided features.

The research of the seven CPS solutions is composed as follows: The architecture of the CPS solutions are described and compared. The conclusion of the comparison is consolidated within a correlation matrix, where the correlation of the diverse concepts of the components, and the abstraction levels are consolidated [Zim+13]. Subsequently the reference architecture is derived. To verify the reference architecture it is mapped to the analyzed CPS solutions. Within the next step, the features of the different CPS solutions are compared with a correlation matrix, as well. Thereby the correlation of the provided operations are consolidated, so the intersection of the functions become apparent [Zim+13]. Based on those results, the abstract class model is designed. To validate the abstract class model, the operations of the classes are mapped to the

considered CPS solutions within a correlation matrix. Therein the applicability of the abstract class model is derivable.

1.4 Structure of the Document

This Thesis is segmented into seven chapters. A short introduction to CPS and the motivation, the research issues, the research method and the structure of the document are outlined in Chapter 1. The fundamentals of CPS and the analyzed CPS solutions are characterized within Chapter 2. The precise analysis of the considered CPS solutions and the derived reference architecture is the content of Chapter 3. The validation of the reference architecture of Chapter 3 with the considered CPS solutions is described in Chapter 4. The features of the considered CPS solutions are analyzed more precisely to derive the abstract class model within Chapter 5. The abstract class model is validated with the analyzed CPS solutions in Chapter 6. Ultimately a summary of the outcome of this Thesis, a critical discussion of the results, and a perspective on further research are given within Chapter 7.

2 Fundamentals and Related Work

The following Chapter defines the scope of CPS, and introduces the seven considered CPS platforms. The first section deals with the definition of CPS, and then describes the state-of-the-art research, related protocols, and standards in this area. In the second section each CPS solution is introduced in detail, which creates the basis for the analysis, and the derivation of the abstraction layer.

2.1 Cyber-Physical Systems

The definite aim of CPS is the integration of physical processes with computation [Lee08]. CPS depend on multiple disciplines, and arise of a complex interaction of embedded systems, application systems, and infrastructures with the interaction of humans and technology, where this complex overall interaction is based on their interconnection and integration [Tal08]. Physical Processes are monitored and controlled by embedded computers and networks, where the physical processes can affect computations, and vice versa, using feedback-loops [Lee08] [Rag15].

2.1.1 State-of-the-Art Research

As CPS are gaining increasing attention on the market, as well as in research, many different angles have been evaluated. Beside the research about the economical, process-related and social effects, you can separate the technical researches into the following categories: architecture, security, protocol, and standards issues. Security issues are excluded from this Thesis, as they would exceed the boundary of this work.

2.1.2 Protocols

To interconnect the components of CPS through the internet, different protocols are used. The following list highlights the protocols, which are supported by the considered

CPS platforms, and therefor are described to provide the basis for the analysis of the CPS platforms. Tough the list has no claim of completeness.

The *Hypertext Transfer Protocol version 1.1 (HTTP/1.1)* [Fie+99] defines a standard for a stateless request/response data transfer protocol [Woj16]. As it is based on TCP, *HTTP/1.1* is reliable and connection oriented [Gou+02]. *HTTP/2* [Bel+15b] is a standard extending *HTTP/1.1*, which enables an improved performance. *HTTP/2* allows concurrent exchanges on the same connection, and it can establish unsolicited push of representation from servers to clients [Bel+15a]. *HTTPS* [Res+00] is a secure version of *HTTP*. Before the HTTP messages get sent to TCP, they are first sent to a security layer, where the message gets encrypted. As a result *HTTPS* is secure, coincidentally flexible, and easy to administer. [Gou+02]

The *Constrained Application Protocol (CoAP)* [She+14] is a document transfer protocol, especially drafted for resource-limited devices and networks [Tob14]. It implements the Representational State Transfer (REST) [Fil00] architectural style and enables a transparent mapping onto HTTP, but extends it with native push notifications, and many-to-many communications [Dio14]. Clients communicate with servers through connectionless datagrams, following a client/server model [Tob14]. Most commonly UDP is used as the underlying protocol, but CoAP can also be used on top of SMS or other packet-based communication protocols [Fri13] [Dio14]. Furthermore there are approaches, which enable the usage of TCP and TLS as the transport protocol for CoAP [Lem+14] [ARM11].

Message Queuing Telemetry Transport (MQTT) [OAS14] is an open standard of an publish-subscribe-messaging protocol for machine-to-machine (M2M) communication. Specifically designed for resource-constrained devices and low bandwidth, *MQTT* can be used efficiently in embedded systems [And13] [Gaz+15]. The sender and receiver communicate via an *MQTT*-based broker over TCP. Using the publish-subscribe architecture pattern, it enables to connect hundreds of thousands clients with a single broker [Luz+15] [Tob14]. Following this, *MQTT* enables a one-to-many communication [Dio14]. *MQTT*-based brokers support three Quality of Service (QoS) levels for message delivery: QoS 0 reliable (fire-and-forget), QoS 1 at-least-once, and QoS 2 exactly once [And13] [Ban+13].

WebSocket [Fet+11] is a network protocol to establish a full-duplex communication channel over a single TCP connection. Following this, *WebSocket* enables a low latency delivery in both directions, where the messages consist of text and binary application data. [Gri13]

Diameter [Faj+12] is a connection oriented and extendable protocol for authentication, authorization and accounting of communication partners in a network [Gos12].

The *Advanced Message Queuing Protocol (AMQP)* [OAS12] is an open standard application layer protocol for message-oriented middleware [Gaz+15]. It comprises both, a network protocol, and a protocol model. The network protocol specifies the entities, i.e., the producer, consumer, and the broker, which interoperate with each other. The protocol model defines the representation of messages, and the commands to interoperate among the entities [Luz+15]. It provides reliable, guaranteed, in-order, point-to-point, and store-and-forward message delivery [Sub+08] [Luz+15]. Furthermore the data content of an *AMQP* message is opaque and immutable, the messages are self-contained, and additionally the size of a message is not limited [Luz+15].

The *Simple or Streaming Text Oriented Messaging Protocol (STOMP)* [Sto16] is an interoperable protocol designed for asynchronous message exchange between clients via mediating servers. *STOMP* messages are defined as text based wire-format [Sto16].

2.1.3 Standards

To support the communication of the CPS components, standards are established. The following list describes those supported by the analyzed CPS platforms, and like the protocol list it has no claim of completeness.

The *European Telecommunications Standards Institute's Machine-to-Machine (ETSI M2M)* [ETS13] standard provides specifications for M2M services and applications, where it focuses on aspects of the IoT. *ETSI M2M* defines a functional architecture and the interfaces required to support end-to-end services. The Service Capability Layer (SCL) is formed by a set of functionalities within the M2M core, and can be utilized by M2M applications through a REST API. The SCL sits on top of the connectivity layers, and the API for applications is based on REST principles allowing scalability, unreliable connections, and binding to, e.g., HTTP or CoAP. However *ETSI M2M* does not specify how to support transport protocols [Che+14]. Within the functional architecture each physical equipment is represented as an instance of an SCL. Hence each device, gateway, and the network will find a corresponding entity within the SCL, so the abstract model will be a collection of Device SCL, Gateway SCL, and Network SCL. Furthermore each SCL instance is responsible for a subset of resources modeled and named according a recursive hierarchical tree [Gri+14].

The *European Telecommunications Standards Institute's oneM2M (oneM2M)* [ETS15] standard is a technical specification of the requirements of a generic distributed M2M software service layer. It provides standardized interfaces, so they are applicable to the entire ecosystem. *oneM2M* provides common service functions, which are exposed to applications via RESTful APIs. Even *oneM2M* is designed IP-based, it interworks with specific IP and non-IP technologies in the M2M area networks [One15]. Figure 2.1

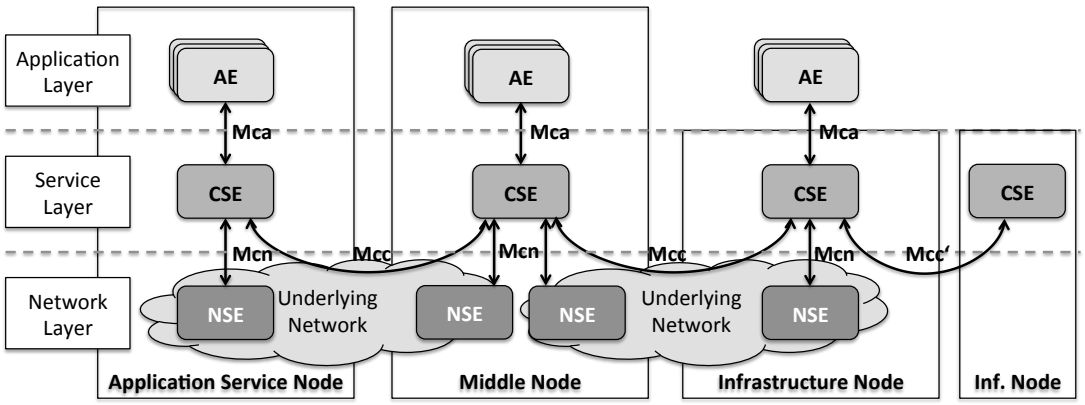


Figure 2.1: oneM2M Architecture based on [Ell14]

shows the architecture, which contains different nodes, i.e., an Application Service Node, a Middle Node, and an Infrastructure Node, where a node is the logical equivalent of a physical, or possibly virtualized device. Each node has one or more Network Services Entity (NSE) on the Network Layer, which are connected with each other through the Underlying Network. Besides the pure data transport, they provide services to the Common Services Entities (CSEs). A CSE provides the set of service functions that are common to the M2M environments, and each node has a CSE on the Service Layer. Additionally each node has multiple Application Entities (AEs) on the Application Layer, which provide application logic for the end-to-end M2M solutions. A Reference Point describes one or more interfaces between two service providers. There are four interfaces described[Ell14]:

1. The Mca which is the reference point between an AE and a CSE of the same node.
2. The Mcn describes the interface between a CSE and a NSE of the same node.
3. The Mcc is a reference point between two CSEs of a different node.
4. The Mcc' which describes the interface of two CSEs of the same node.

Open Mobile Alliance's Lightweight M2M (OMA LWM2M) [Ope15] is an industry standard for the device management of M2M or IoT devices, which was approved and published in December 2013. *OMA LWM2M* provides an efficient device-server interface, based on open IETF standards. Based on CoAP and Datagram Transport Layer Security (DTLS) with bindings to UDP and SMS, it is optimized for the communication over sensors- and cellular networks. *OMA LWM2M* provides an extensible object and resource model for application semantics, which allows to enable application data exchanges, in addition to the core device management features, such as firmware upgrade or connecitivity monitoring [She14] [Der+15].

The *Open Mobile Alliance's Next Generation Service Interfaces (OMA NGSI)* [Ope12b] are context management function specifications of the *NGSI* Enabler, which provides access to information about Context Entities through interfaces. *NGSI* defines two abstract interfaces with the following operations: *NGSI-9*, the Context Entity Discovery Interface, and *NGSI-10*, the Context Information Interface. *NGSI-9* comprises the Register Context Entity Operation, which enables the context management component to allow registering and updating context entities, their attributes, and availability. Furthermore *NGSI-9* embraces the Discover Context Entity Operation, enabling an actor to discover available context entities, and their attributes. *NGSI-10* comprises the Update Context Operation, which enables an application acting as a context producer to provide or update context information to the context management component, and the Query Operation, enabling applications to act as context consumers to query for context information. Additionally both *NGSI-9* and *NGSI-10* comprise the Subscribe and Notify based Context Entity Discovery Operation, which enables an application to issue a subscription to the context management component on behalf of another application, such that the second application receives the respective notification upon the availability of new context entities, or changes to available context entities, and their attributes [Ope12b] [Ope12a].

2.2 State-of-the-Art Technologies

This Section describes the considered CPS platforms. The first four platforms are open-source solutions and the following three are proprietary solutions. As it prepares the basis for the following analysis, an overall insight is given, and the architectural components are described.

2.2.1 OpenMTC

The *Open Machine Type Communications (OpenMTC)*[1] platform implements an open, cloud-enabled CPS solution, provided by Fraunhofer FOKUS and Technische Unversität Berlin. *OpenMTC* is designed to act as a horizontal convergence layer in terms of an M2M middleware for machine type communication, supporting multiple vertical domains, i.e., market segments like automotive, and eHealth. They can be deployed independently, or as part of a common platform. As it is designed to act as an M2M middleware, the aim is to provide a standard-compliant platform for M2M services. The purpose of *OpenMTC* is to interconnect various sensors and actuators from different vertical domains with a

[1] http://www.open-mtc.org

cloud-enabled, open platform, which aggregates the collected data, forwards the data to the target applications, and mediates instructions to end devices for an event-based control.

Figure 2.2 shows the architecture of *OpenMTC*. It consists of two common M2M capability layers: the *OpenMTC* Front-End in the field domain and the *OpenMTC* Back-End in the infrastructure domain, i.e., a cloud-based platform. The *OpenMTC* Front-End enables the connection of sensors and actuators to the platform, and therefor it communicates with the Connectivity component of the *OpenMTC* Back-End. This communication is enabled either by a direct, managed or un-managed access and transport via HTTP, CoAP, WebSocket or MQTT, or by a managed connectivity via the OpenEPC components. OpenEPC provides a network layer mobility concept, which forwards the data traffic, and ensures the access control, and which provides the Policy and Charging Rules Function (PCRF), the Acess Network Discovery and Selection Function (ANDSF), as well as the Home Subscriber Server (HSS) [Cor16b]. The PCRF and the ANDSF are part of the OpenEPC Policy Engine and Control Entities, which make policy based decisions for the connectivity, the access control and the resources allocated for mobile devices [Cor16a]. The HSS is part of the OpenEPC Subscription Data Entities, which store update, and transmit notifications on the users' subscription profile towards the other EPC authorization entities, and which supply information and mechanisms for the authentication of mobile devices [Cor16a]. Furthermore the *OpenMTC* Front-End provides an access point for the connection of further applications. The *OpenMTC* Back-End provides the core funcitonality of the platform, and the connection of further applications and other M2M platforms. It supports different capabilities, which are defined by the ETSI M2M and oneM2M specification, namely the Device SCL, Gateway SCL, and Network SCL. Additionally OpenMTC supports the OMA NGSI 9 and 10 interfaces for context management on the gateway and the back-end server.

By supporting the transport protocols HTTP, CoAP, Diameter, and WebSocket, *OpenMTC* allows to support different type of domain-specific applications with various interaction models, i.e., push/pull and subscribe/notify. Furthermore this enables applications and end-devices to interact in real-time over the web, or any other IP-based network, even via multicast [Fra15].

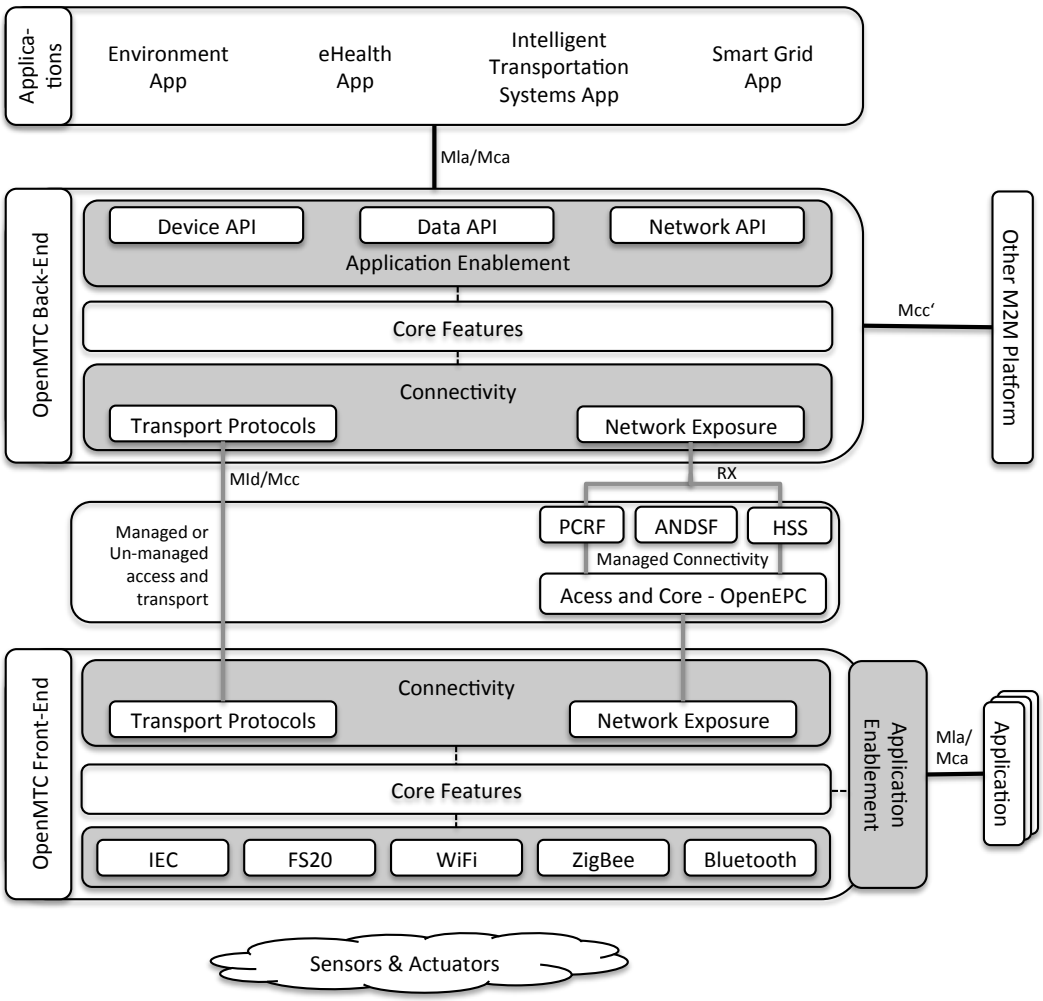


Figure 2.2: OpenMTC Architecture based on [Fra15]

2.2.2 FIWARE

FIWARE[2] is an open, cloud-based infrastructure for CPS, funded by the European Union and the European Commission. It is an enhanced OpenStack-based cloud, which hosts capabilities and the *FIWARE* Catalogue, containing a rich library of components, called Generic Enablers (GE). Figure 2.3 depicts the overall architecture, i.e., the GEs of FIWARE. The dottet boxes represent external entities. Beside the rich library of GEs, the Catalogue also contains tools and best practices. The GEs of the IoT component are spread over two different domains: the IoT Back-End and the IoT Edge, which

[2]https://www.fiware.org

are depicted within Figure 2.4. The IoT Back-End is hosted in a cloud datacenter, and comprises a set of functions, logical resources, and services, i.e., the IoT Device Management, the IoT Discovery, and the IoT Broker. It is connected to the Data Context Broker via an API, which ensures that the IoT resources are translated into NGSI Context Entities. The Data Context Broker enables to publish and subscribe to context information, i.e., it provides the required functionality for the communication between the devices, users, and applications. The IoT Device Management is the central enabler at the IoT Back-End for most common scenarios. It is responsible for connecting physical devices to a *FIWARE* platform, managing IoT-related NGSI Context Entities, and the IoT Edge Management.

The IoT Broker GE takes care of communicating with the different IoT Devices and Gateways to retrieve the needed information on behalf of the applications, using the NGSI protocol. It is an IoT Back-End enabler, interacting with the whole IoT deployment to satisfy the requests, and foreseen to run on a machine in a data center, where it serves as a middleware, which enables fast and easy access to IoT data. It is a stateless component, neither it stores context information, nor context availability information. The IoT Discovery is responsible for the context availability registrations from IoT Agents, i.e., making it the access point for information about entities and their attributes. The role of IoT Agents can be played by the Data Handling GE (IoT Edge), the Device Management, or other IoT Back-End Systems.

The IoT Edge comprises all elements of the physical IoT infrastructure, i.e., IoT end-nodes (Devices), IoT gateways, and IoT networks. It is made of all on-field IoT infrastructure elements needed to connect physical devices to *FIWARE* Apps. The IoT Edge and its related APIs will facilitate the integration of new types of gateways and devices. An IoT end-node or Device is a hardware entity, component or system, which either measures or influences the property of a thing, or a group of things, or performs both activities. Sensors and actuators are devices, while complex physical devices with several sensors and actuators are named IoT end-nodes. Devices might use standard or proprietary protocols, which can be translated into any other protocol at the IoT Gateways. Part of the IoT Gateway is the GW Logic, which is responsible for the communication with the Back-End, and IoT and non-IoT devices. It includes functional components to handle registration or connection phases towards the Back-End or Platform, to translate incoming data and messages in an internal format, and to send the outgoing data or messages in ETSI M2M format. Furthermore it manages the communication with IoT resources. The Protocol Adapter GE, which is part of the IoT NGSI Gateway, deals with the incoming and outgoing traffic and messages between the IoT Gateway and registered devices, to be served by either the Gateway Device Management GE or the Data Handling GE. The Data Handing GE, also part of the IoT NGSI Gateway, addresses the need of filtering, aggregating, and merging real-time data form different sources. It can be operated stand-alone, but typically the Data Handling GE receives events from the

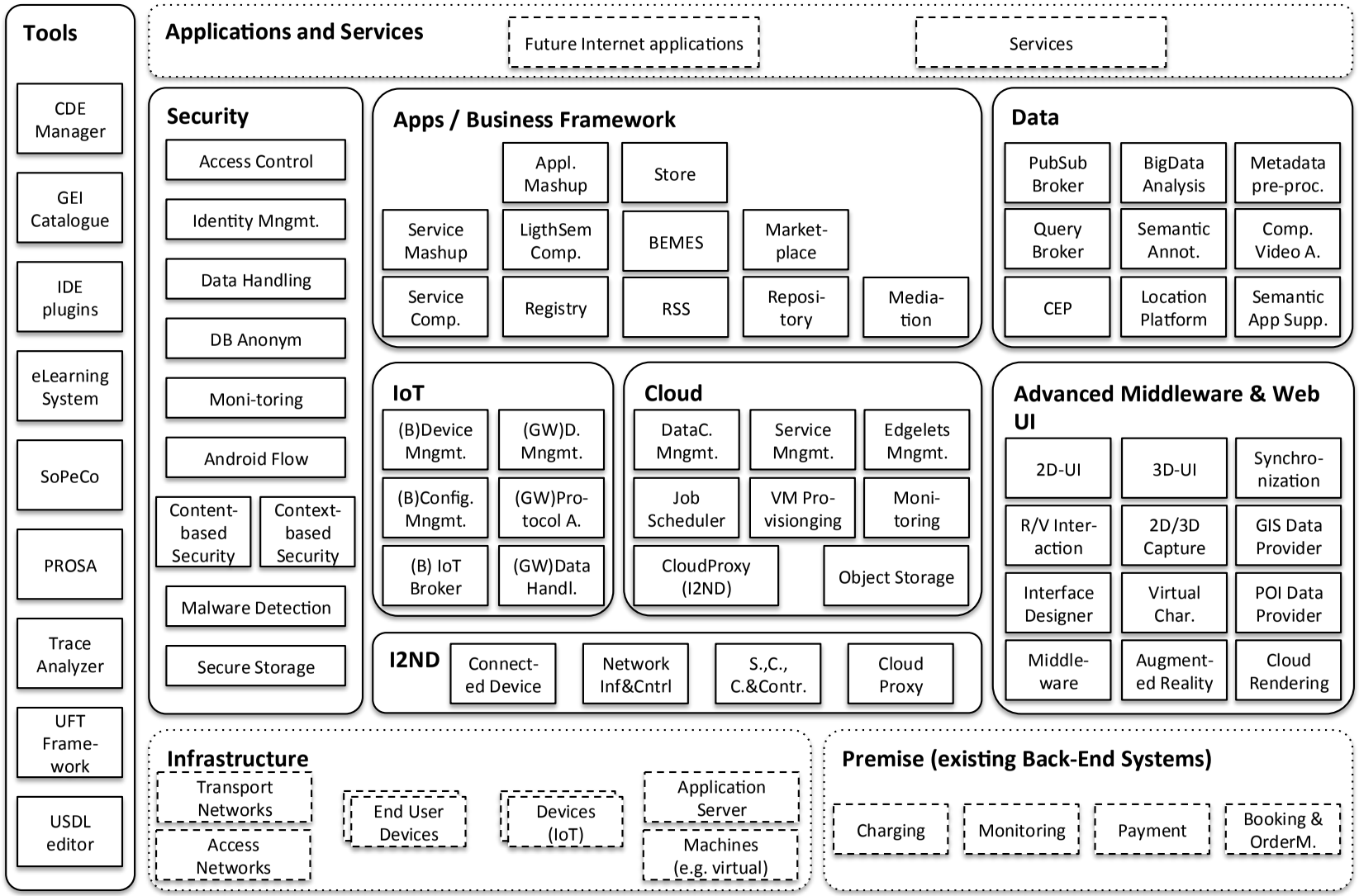


Figure 2.3: FIWARE Overall Architecture based on [FIW15a]

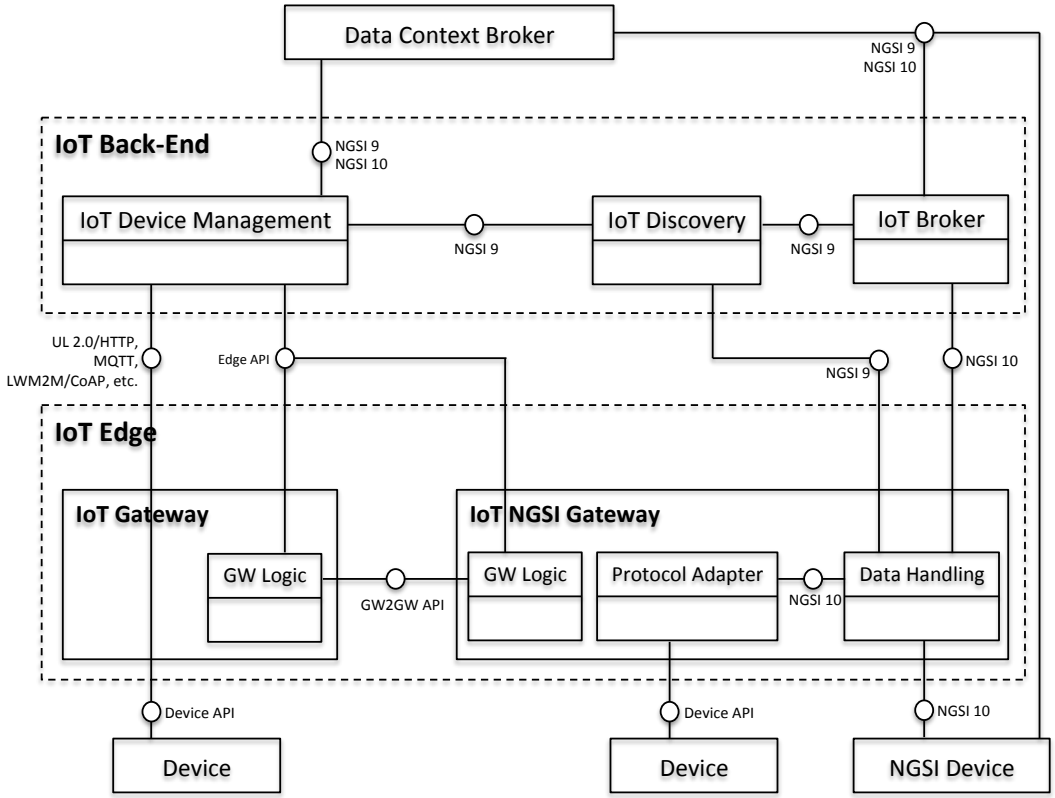


Figure 2.4: FIWARE Architecture based on [FIW15b]

Gateway protocol Adapter GE, and propagates the processed data towards the IoT Broker GE. Additionally it handles data streams from IoT devices that cannot continuously be online. *FIWARE* is based on OMA NGSI 9 and 10, and OMA LWM2M. It supports the transport protocols HTTP, MQTT, and COAP [FIW16a] [FIW16c] [FIW16b].

2.2.3 SiteWhere

SiteWhere[3] is an open CPS platform developed by SiteWhere LLC. It supplies a server, based on proven technologies, which acts as a controller for the processing of device data. The server can be installed on a local machine or it can run in the cloud. It is designed to scale up to billions of device events per day. *SiteWhere* uses a pluggable framework approach, which allows third parties to extend and customize the system.

[3]http://www.sitewhere.org

By providing REST services, *SiteWhere* allows external applications to interact with all facets of the system. Additionally it provides out of the box integration support for many popular frameworks and services.

As Figure 2.5 shows, *SiteWhere* is designed as a multitenant system. Multiple IoT applications (tenants) can be served from a single *SiteWhere* instance. To assure that, no data is intermingled between tenants, each system tenant has a separate data store. Additionally each tenant has a separate processing pipeline, which can be customized without affecting the processing of other tenants. Each *SiteWhere* Tenant Engine comprises a Device Management and a Communication Engine. The Communication Engine handles all functionality related to interacting with devices over "almost any transport protocol", explicitly named are MQTT, AMQP, Stomp, and WebSocket. This functionality includes the registration of new or existing devices, the receiving of events from connected devices, and the delivery of commands to connected devices.

SiteWhere never deals directly with a database. The system defines SPIs for the data operations, and expects datastore implementations to comply with the required interfaces. There are two core interfaces the datastore needs to implement: the IDeviceManagement and the IUserManagement. The IDeviceManagement contains all core device management calls including CRUD methods for sites, specifications, devices, events, etc. The IUserManagement contains all core user management calls including CRUD methods for users, authorities, etc. *SiteWhere* offers two highly tuned and very scalable persistence implementations: MongoDB [Mon16], where the implementation is optimized by the MongoDB team, to push in excess of 10,000 events per second on medium power cloud instance, and Apache HBase [The16], where the implementation is based on a customized schema, designed to scale linearly across a cluster that can be expanded if more capacity is needed. Both implementations conform to *SiteWhere* device management APIs, which provide a consistent view of the data independently of where it is stored.

SiteWhere defines SPIs for general asset types, and allows asset modules to be plugged in to provide asset definitions. *SiteWhere* assets represent objects in the physical world, i.e., people, places, and things, where Device specification assets are used to describe the hardware information or configuration for a type of device, and the device assignment assets are used to describe an entity associated with a device. It uses asset modules in a read-only manner, and only ever references entities based on a unique ID understood by the underlying asset module. *SiteWhere* enables integration by offering asset information via a pluggable asset management framework, which allows external systems to drive information. Tracking is enabled by providing an assignment history by capturing device assignments over time [Sit16a] [Sit16b].

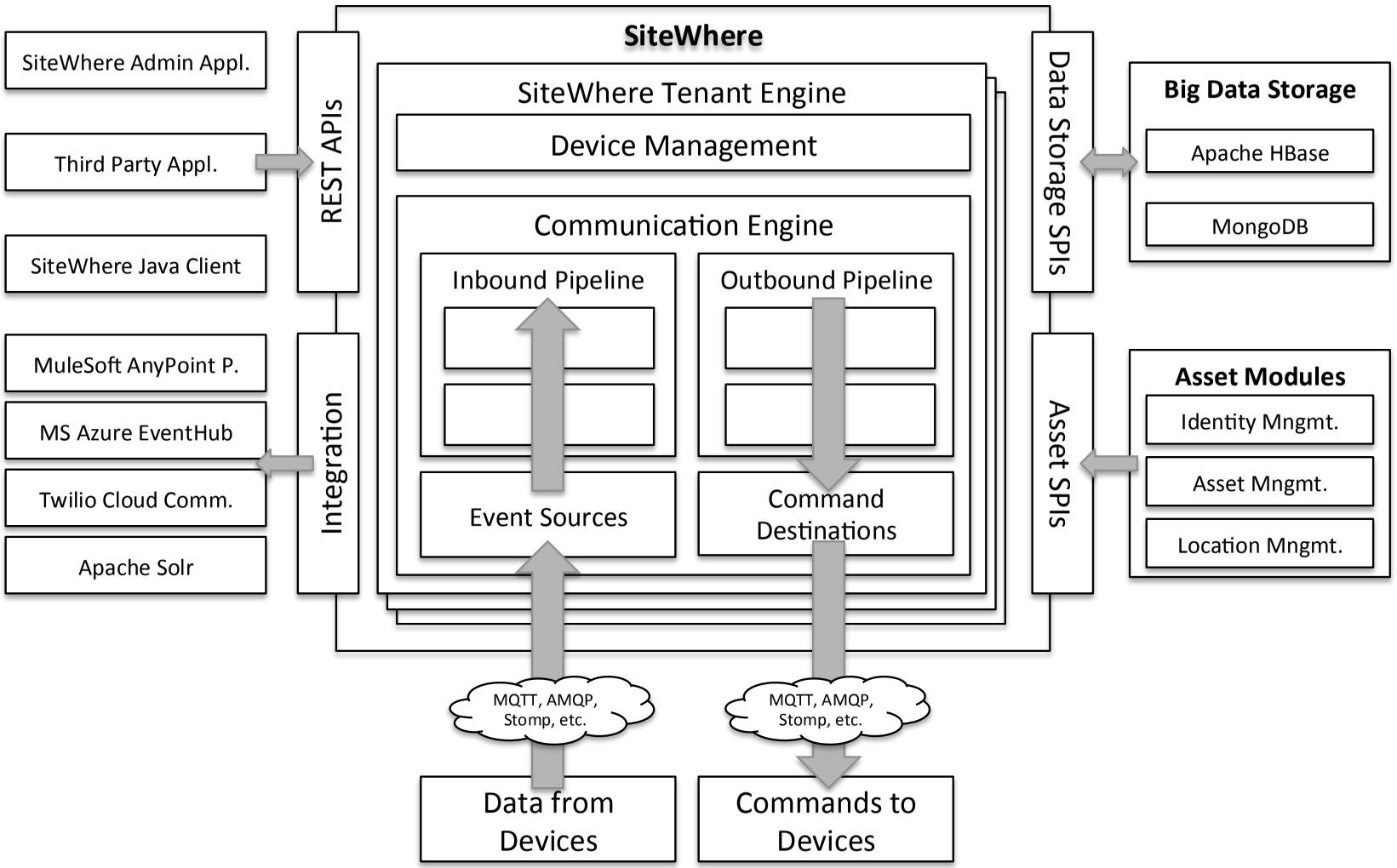


Figure 2.5: SiteWhere Architecture based on [Sit16d]

2.2.4 SmartThings

SmartThings[4] is an open smart home platform developed by SmartThings Inc. As it is designed for smart homes, it depicts only a subarea of CPS. The approach of *SmartThings* is to separate the intelligence from devices, where devices are limited to their primitive capabilities [Sma15b].

Figure 2.6 shows the architecture of *SmartThings*. Each device has capabilities, which define and standardize available attributes and commands for the device. The *SmartThings* Hub Connectivity connects the devices to the platform, and provides communication between all connected devices, the *SmartThings* cloud, and mobile application. The Client Connectivity enables Clients(-Devices) to connect to the platform. Using the Device Type Handler methods, the event-messages sent by the devices are translated into a normalized *SmartThings* event. The Subscription Processing component provides devices and users to subscribe to events, i.e., it enables the event processing. Within

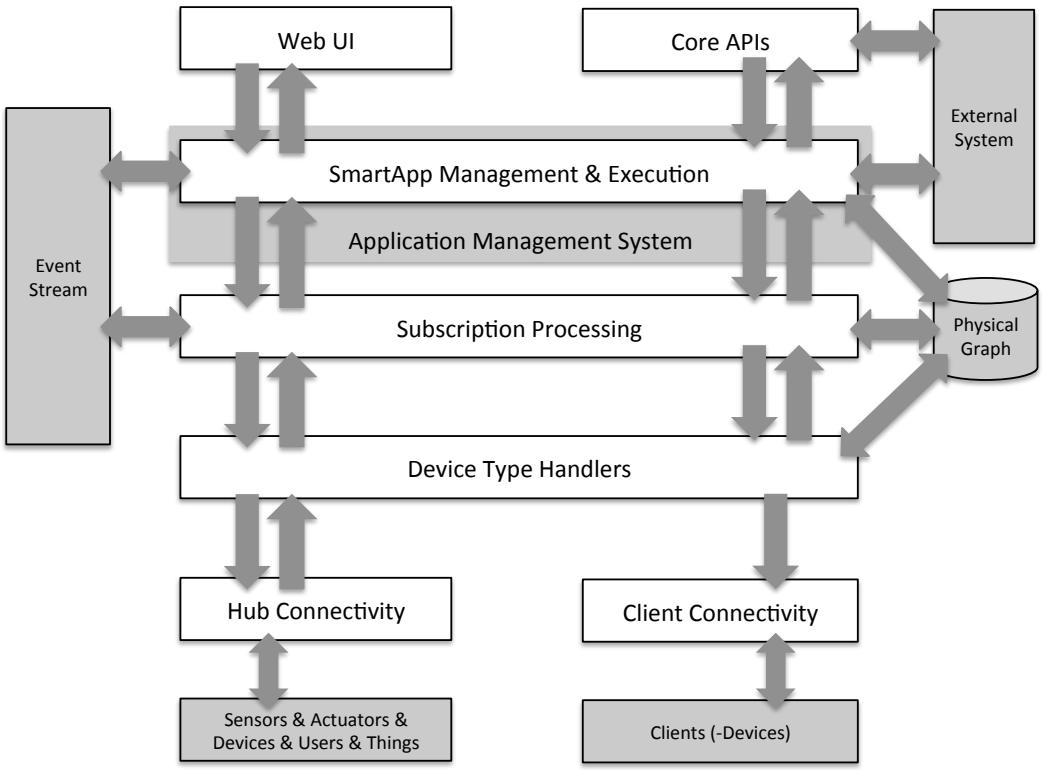


Figure 2.6: SmartThings Architecture based on [Sma15b]

[4]https://www.smartthings.com

the Application Management System, the SmartApp Management & Execution handles external calls to SmartApp endpoints, the execution of SmartApps, based on triggering subscriptions, and scheduled methods. Furthermore the Web UI and Core APIs provide features for monitoring the devices, hubs, locations and other aspects of the system. The Event Stream, External System, and Physical Graph components are external applications, which can be connected to the platform over the pictured access points of the components within the system [Sma16].

2.2.5 AWS IoT

Amazon Web Services IoT (AWS IoT)[5] is a managed cloud platform for CPS. It provides secure, bi-directional communication between internet-connected things, i.e., sensors, actuators, embedded devices, smart appliances, and the AWS cloud. Features of *AWS IoT* are collecting, storing and analyzing data, as well as the creation of applications to control devices.

Figure 2.7 shows the architecture of *AWS IoT*. It consists of the following six components: the Message Broker, the Rules Engine, the Thing Registry, the Thing Shadows Service, the Thing Shadow, and the Security & Identity Service. Within the figure of the architecture, the Thing Registry, the Thing Shadows Service, and the Thing Shadow are combined to the Thing Shadows component. Within the documentation of AWS IoT, the Device Gateway is mentioned as another component. Since it is used in the same way like the Message Broker, it is assumed, that the Device Gateway and the Message Broker are used as synonyms, which provide the same functionality [Ama16c] [Ama16e]. The Message Broker enables devices to securely and efficiently communicate with *AWS IoT*. It supports the publish-subscribe messaging pattern [Mil+10], which enables scalable, low-latency, and low-overhead communication [Ama16e]. By using the publish-subscribe model for message exchange, it enables one-to-one and one-to-many communication. MQTT and WebSockets are supported for publish and subscribe, and HTTPS for publish. Additionally it is possible to implement the support for proprietary or legacy protocols. The Message Broker provides a secure mechanism for things and IoT applications to publish and receive messages from each other. The Rules Engine provides message processing, and integration with other AWS services. By defining one or more actions to perform, based on the data in a message, a rule can be created. When a rule matches a message, the Rules Engine invokes the action using the selected properties. To republish messages to other subscribers the Message Broker can be used. The Thing Registry organizes the resources associated with each Thing. The Thing Shadow Service provides persistent representations of the Things in the AWS cloud. A Thing Shadow is a JSON

[5] https://aws.amazon.com/iot/

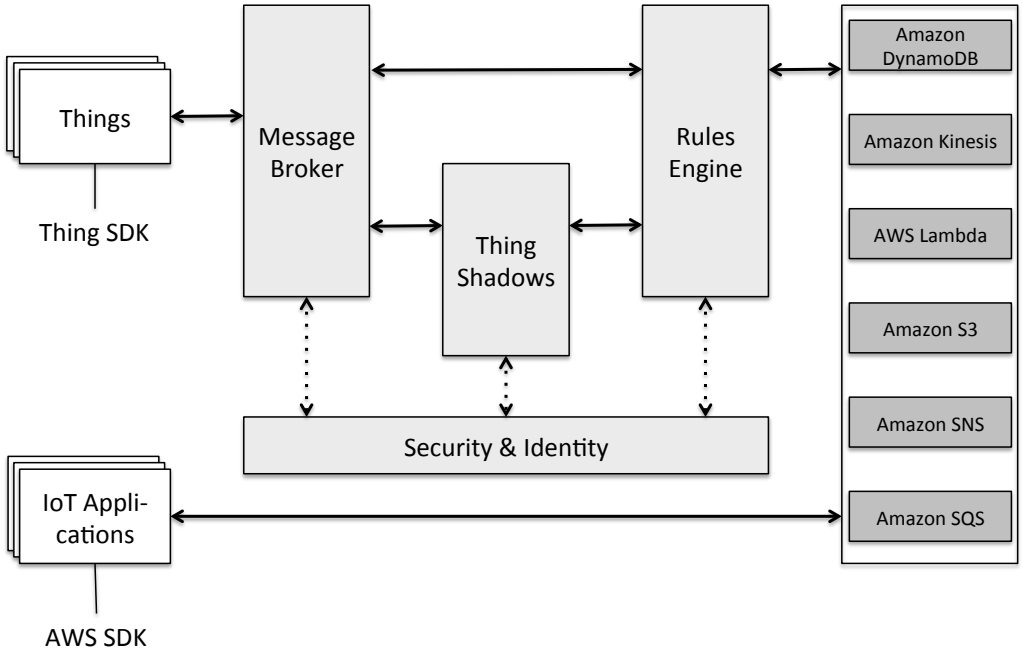


Figure 2.7: AWS IoT Architecture based on [Ama16c]

document, used to store and retrieve current state information for a Thing. By using the Thing Shadow, to get and set the state of a Thing, the Thing itself can be offline, and will synchronize as soon as it reconnects. The Security & Identity Service provides shared responsibility for security in the AWS cloud. The Message Broker and Rules Engine use AWS security features to send data securely to devices over the Thing Shadow, or other AWS services. Additionally the Security & Identity Service grants permissions, which AWS resources may be used, to perform the corresponding action of a Rule. *AWS IoT* provides interfaces to connect further applications to the platform [Ama16a] [Ama16d].

2.2.6 Microsoft Azure IoT Hub

Azure IoT Hub[6] is a managed, cloud-based service, which enables reliable and secure bi-directional communication between millions of IoT devices and a solution back-end, provided by Microsoft. Likewise Microsoft offers Azure IoT Suite, which is a collection of preconfigured solutions [Mic16c].

[6]https://azure.microsoft.com/en-us/services/iot-hub/

Figure 2.8 shows the architecture of a solution including the *Azure IoT Hub*. In a typical CPS solution the *Azure IoT Hub* is a managed service, and responsible for establishing bi-directional communication between devices and a cloud solution back-end, by implementing the service-assisted communication pattern [Vas14]. The aim of service-assisted communication is to establish trustworthy, bi-directional communication paths between a control system, i.e., the IoT Hub, and devices, which are deployed in untrusted physical space. Even intermittently connected devices can be used, as the messages are sent in a durable way . The *Azure IoT Hub* supports the transport protocols HTTP/1.1, AMQP, and MQTT, with which a device can communicate directly with a cloud gateway endpoint within the IoT Hub. If a device cannot use any of those protocols, it can connect through the Cloud Protocol Gateway, which performs protocol translation. Another optionally intermediary between the devices and the IoT Hub is a Field Gateway, which is deployed locally with the device, and provides local management services for the device by performing an active role in managing access and information flow.

The IoT Hub is connected to the Event Processing and Insight component, and to the Device Business Logic, Connectivity Monitoring component, which provides the core functionality of the platform. Furthermore the IoT Hub is connected to the Application Device Provisioning and Management component, which represents the access point for further possibly connected applications.

The REST APIs for the *Azure IoT Hub* offer acces to the device, messaging services, and the resource provider, i.e., the IoT Hub. Messaging services can be accessed from within an IoT service running in Azure, or directly over the internet from any application that can send HTTP/HTTPS requests, and receive HTTP/HTTPS responses [Mic16a] [Mic16c] [Mic16b].

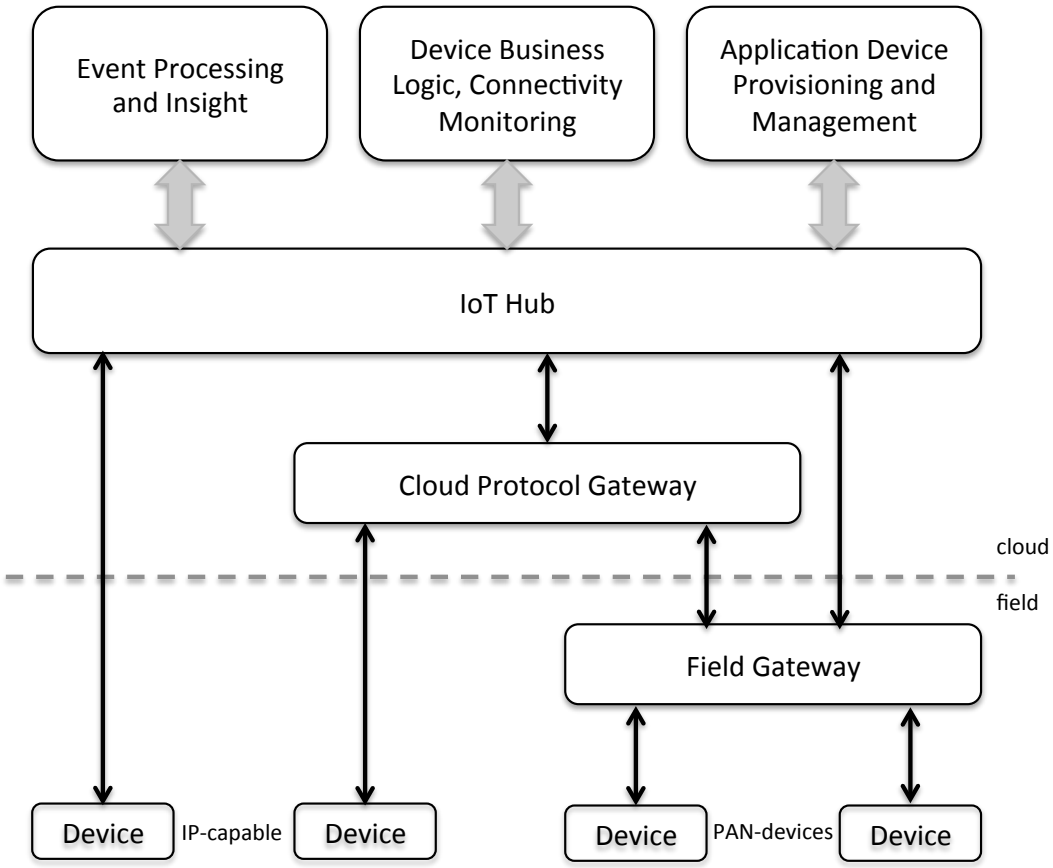


Figure 2.8: Azure IoT Hub Architecture based on [Bet16]

2.2.7 IBM Watson IoT Platform

IBM Watson IoT Platform[7] is a cloud-based platform solution for CPS, provided by IBM. It can be integrated into the IBM Bluemix cloud, and hence connect to their services. Since IBM recently released the IBM Watson IoT Platform, the available information, and documentation is used [Kuf16]. If the information is not updated, the documentation of the IBM IoT Foundation is considered, since it is the basis of the IBM Watson IoT Platform.

As the architecture of the *Watson IoT Platform* within Figure 2.9 depicts, it comprises four main building blocks: Connect, Information Management, Analytics and Risk Management. The *Watson IoT Platform* Connect component easily connects devices

[7] http://www.ibm.com/internet-of-things/iot-platform.html

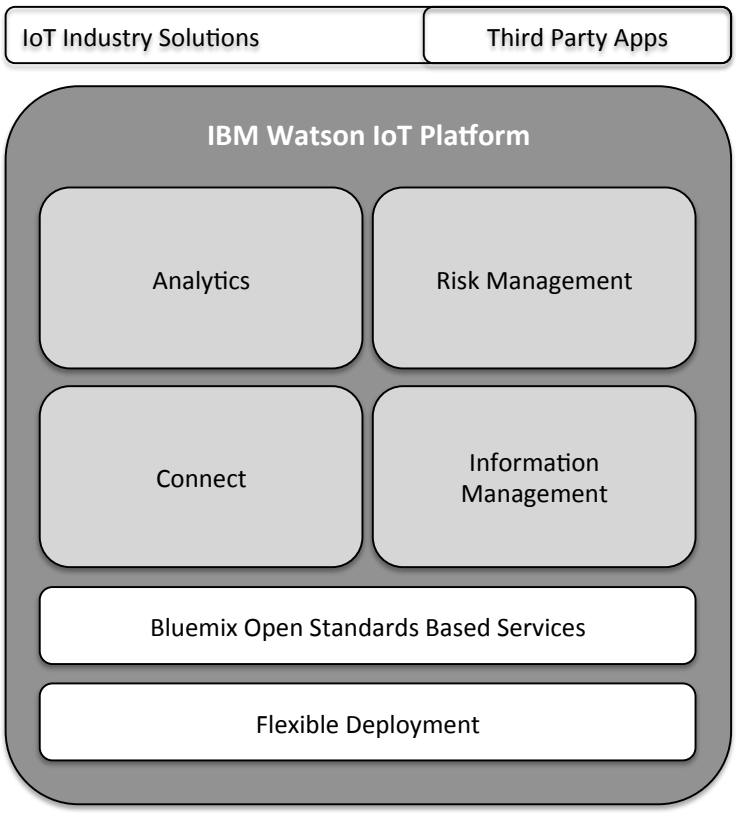


Figure 2.9: IBM Watson IoT Platform Architecture based on [IBM16a]

from chips to intelligent appliances to the platform. It performs device management functions, and scales through cloud-based services. The Information Management transforms and stores the data, collected from diverse data sources and platforms. The *Watson IoT Platform* Analytics enables to gain insight from huge volumes of IoT data, to make better decisions and optimize operations. Real-time analytics can be applied to monitor current conditions and respond accordingly. The Risk Management uses extensive dashboards and alerts, to manage risk and gather insights across the entire IoT landscape. It enables to act on notifications, and isolate incidents generated within the environment. Furthermore the *Watson IoT Platform* enables the connection of further third party applications, which are not part of the services provided by Bluemix. The *Watson IoT Platform* supports the transport protocols MQTT and HTTP [IBM16c].

3 Design of the Reference Architecture

The following Chapter analyses and compares the considered CPS solutions, described in Chapter 2.2. The architecture components, and the definitions of those components are explored. On basis of this analysis, the requirements of the reference architecture are derived, and the reference architecture is designed. Within the reference architecture every component is defined.

3.1 Analysis of the State-of-the-Art Technologies

Starting point of the analysis of the state-of-the-art technologies is the examination of the architecture of the considered CPS solutions.

Every platform has an external component “device”, and another internal component for managing the connection of the devices to the platform. The device component differes by the degree of abstraction. The device component of the platform FIWARE is used in terms of a smart device, i.e., it has integrated or connected sensors and/or actuators, and a low scale of intelligence to perform, e.g., initial filtering of the data, and it can directly communicate with the core logic of the platform. The platforms FIWARE, SiteWhere, AWS IoT, Microsoft Azure, and IBM Watson IoT Platform use the device component in terms of a device with integrated or connected sensors and/or actuators which gathers and routes the data, but without any intelligence. This means that they can only send data and receive commands. Contrary to that, the platforms OpenMTC, and Microsoft Azure use a device component, as well as a sensor and/or actuator component.

Every platform has a component for managing the connection of the devices to the platform. The level of detail of the representation is dependent of the degree of abstraction of the device component. Commonly, the connection component manages the devices and the connection of the devices, translates the messages sent from and to the devices into the required format, and routes the messages to the defined receiver.

Within every considered platform, the core functionality is found either on top of the connection component, or the core functionality comprises the connection component.

The core functionality of the platforms is composed of different smaller components, differing in the level of detail within the representation.

Every platform enables the connection of further applications, and data sources. Noticeable thereby is, that the platforms OpenMTC, SiteWhere, AWS IoT, SmartThings, as well as partly Microsoft Azure, and IBM Watson IoT Platform enable the whole representation, analysis, storage, etc. of the data over connected applications. Within FIWARE, Microsoft Azure, and the IBM Watson IoT Platform it is possible to connect further applications, and data sources, but the representation, analysis, and storage of the collected data is part of the platform. Regarding the last two mentioned platforms, the connected applications depict an extension of the existing functionality.

Based on the above described analysis and comparison of the seven considered CPS platforms, the following universal architecture components are identified: Device, connection, core functionality, and further applications and/or further data sources components. Following this, a precise mapping of those components to the components of the considered CPS platforms is possible. Correlation Matrix 3.1 depicts the summarized comparison of the architecture components. Subject to the universal description of the above exposed components, the correlation matrix identifies the name of the corresponding components of every platform. Since the analysis of the device component shows, that three diverse concepts of devices are used, namely sensors/actuators, devices, and smart devices, each of those concepts is considered. They comprise the first three rows of the correlation matrix. The last three rows comprise the remaining exposed components, namely the connection, the core functionality, and a further applications and/or further data sources components.

	OpenMTC	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT Hub	IBM Watson IoT Platform
Sensors/ Actuators	Sensors & Actuatros	-	-	Sensors & Actuators & Devices & Users & Things	Things	-	Connect
Devices	Front-End Core Features + Connectivity	Device / NGSI Device	Data from Devices / Commands to Devices	Sensors & Actuators & Devices & Users & Things	Things	Device	Connect
Smart Devices	Front-End Core Features + Connectivity + Back-End Connectivity	Device / NGSI Device	-	-	Things	Device	Connect
Connection Component	Front- + Back-End Connectivity	IoT Edge + bordering APIs	Communication Engine	Hub and Client Connectivity + Device Type Handlers	Message Broker (+ Thing Shadow + Security & Identity)	Cloud Protocol Gateway + Field Gateway	Connect
Core Functionality	Back-End Core Features + Application Enablement	IoT Backend + Data Context Broker	SiteWhere Tenant Engine	Application Management System + Subscription Processing	Rules Engine + Thing Shadow (+ Security & Identity)	IoT Hub + Event Processing and Insight + Device Business Logic, Connectivity Monitoring	Analytics + Risk Management + Information Management
Further Applications / Further Data Sources	Applications / Other M2M Platform	✔ (not represented in architecture)	Integration / REST APIs / Data Storage SPIs / Asset SPIs	Event Stream / External System / Physical Graph	Amazon Services / IoT Applications	Application Device Provisioning and Management	IoT Industry Solutions / Third Party Apps / Bluemix Open Standards Based Services / Flexible Deployment

Table 3.1: Correlation Matrix of the Considered Technology Architectures

3.2 Requirements of the Reference Architecture

Based on the analysis of the considered platforms, and the comparison of the architecture components in Table 3.1, the requirement of the components of the reference architecture can be derived. Initially it is important to distinguish and define the device component, as it is used in different ways. As the major intersection is the utilization as a device with integrated or connected sensors and/or actuators, which gathers and routes the data, but without any intelligence, this constitutes the basis for the further research.

The connection component should be able to manage the connection between the devices and the platform. Additionally it should be able to route and translate the messages. Therefor, the messages sent by the devices to the platform have to be translated into the required format of messages inside of the platform, and vice versa. The connection component should be extendable, so that even devices with a transport protocol or standard, different from the ones supported by the platform, can be used.

The core functionality component should comprise features to manage the devices, users, and possibly grouped entities, and to aggregate and utilize the messages received and sent. Since this component represents the logic of a platform, the features are precisely analyzed, and an abstract class model is designed and discussed within Chapter 5 and 6.

Furthermore the application component should enable the connection and usage of further applications, and that messages can be exchanged in the required form. Hence the application component should support the access through various messaging formats. Likewise the further data source component should enable to connect external data sources, and to retrieve and use their data.

3.3 Reference Architecture

Following the requirements of the reference architecture, Figure 3.1 shows the derived reference architecture of a CPS. The components are defined as follows:

Sensor & Actuator

A *sensor* is a hardware component gathering the physical space around it, e.g., by measuring the temperature, and sending the gathered data to the connected device, without any further logic. They are used to translate changes within the physical environment into electrical signals [Anj+02]. An *actuator* is a hardware component, which can perform a command, e.g., by turning up the heater. *Actuators* only receive

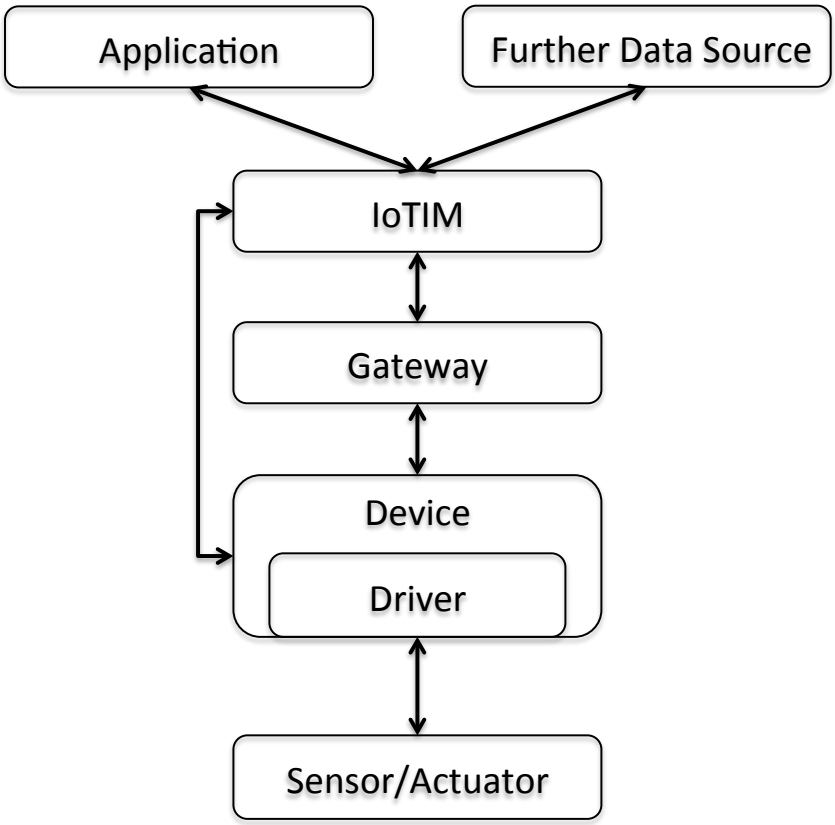


Figure 3.1: Reference Architecture

commands from their connected device, without any additional logic, they possibly send an acknowledgement, that they have received or even performed the command. They are used to act on the physical environment by translating electrical signals into some kind of physical action [Anj+02]. A *Sensor* or an *actuator* is always physically connected to a device, which then communicates with the platform, they never communicate directly with the platform.

Device

A *device* is likewise a hardware component, which has sensors/actuators physically connected or even integrated, and which can communicate with the platform. *Devices* forward the data received from the sensors to the platform, just as they forward the commands from the platform to the actuators. A *device* can communicate directly with the platform, if it supports the corresponding transport protocol, like HTTP, MQTT, or CoAP, and a compatible payload format, like JSON, or XML. Mostly they are connected to a gateway, which can translate the messages accordingly. The *device* comprises the *device driver*, which enables the communication of the *device* with the connected sensors and actuators.

Gateway

The *gateway* ensures the connection, and enables the communication between the devices and the platform. It manages the device connections, by supporting the required communication technologies, and transport protocols. Additionally the *gateway* translates the received messages into the payload format required by the platform, and likewise it translates the commands into a payload format supported by the devices. The required payload format can be based on a standard, like oneM2M, or any other self-defined payload format.

IoTIM

The core functionality of the platform is represented as the *Internet of Things Integration Middleware (IoTIM)* component. It comprises all functionality like, e.g., managing the devices, users, etc., or aggregating and utilizing the data. The features of the *IoTIM* are discussed in Chapter 5. The *IoTIM* receives the data from the devices mostly via the gateway, a direct communication with the devices is also possible. Likewise it can send the commands via the gateway, or directly to the devices. Furthermore the *IoTIM* is access point for APIs.

Application

The *Application* component represents further applications connected to the IoTIM through a common interface. Further applications can either be internal in terms of applications provided by the same provider as the IoT platform, where the integration is already designated, or they can be external in terms of third party applications.

Further Data Source

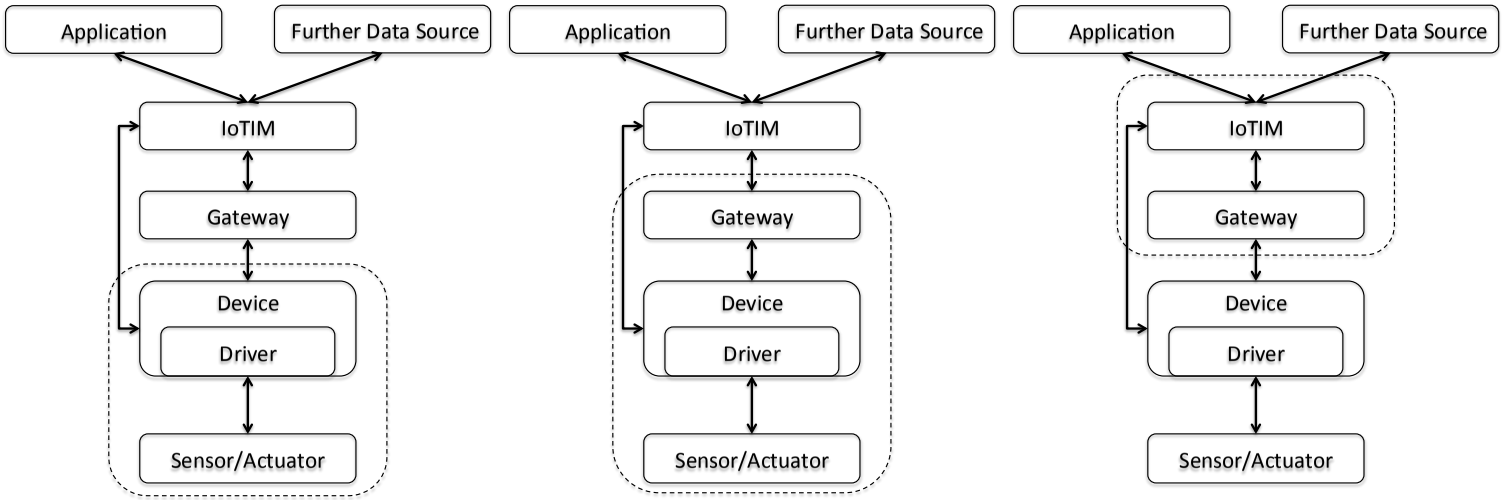
The *Further Data Source* component represents possibly connected external data sources, which can be connected to the IoTIM. Likewise within the applications, the further data sources can be internal in terms of data sources provided by the same provider as the IoT platform, or they can be external in terms of third party data sources.

The communication between the components is asynchronous, which enables non-blocking communication, as different parties do not have to wait for a response, and can process other tasks in the meantime. Within the reference architecture one component can play multiple roles, e.g., a device can be a gateway, sensor and actuator simultaneously. Furthermore Figure 3.2 illustrates three cases of multiple components comprised within one component, which are described in the following.

As described within the definition above, a device can have integrated sensors and actuators, depicted in Figure 3.2(a). A Smartwatch is a device with integrated light and movement sensors, hence it is a device with integrated sensors. But it needs, e.g., a Smartphone to act as a gateway, so that the Smartwatch can communicate with the

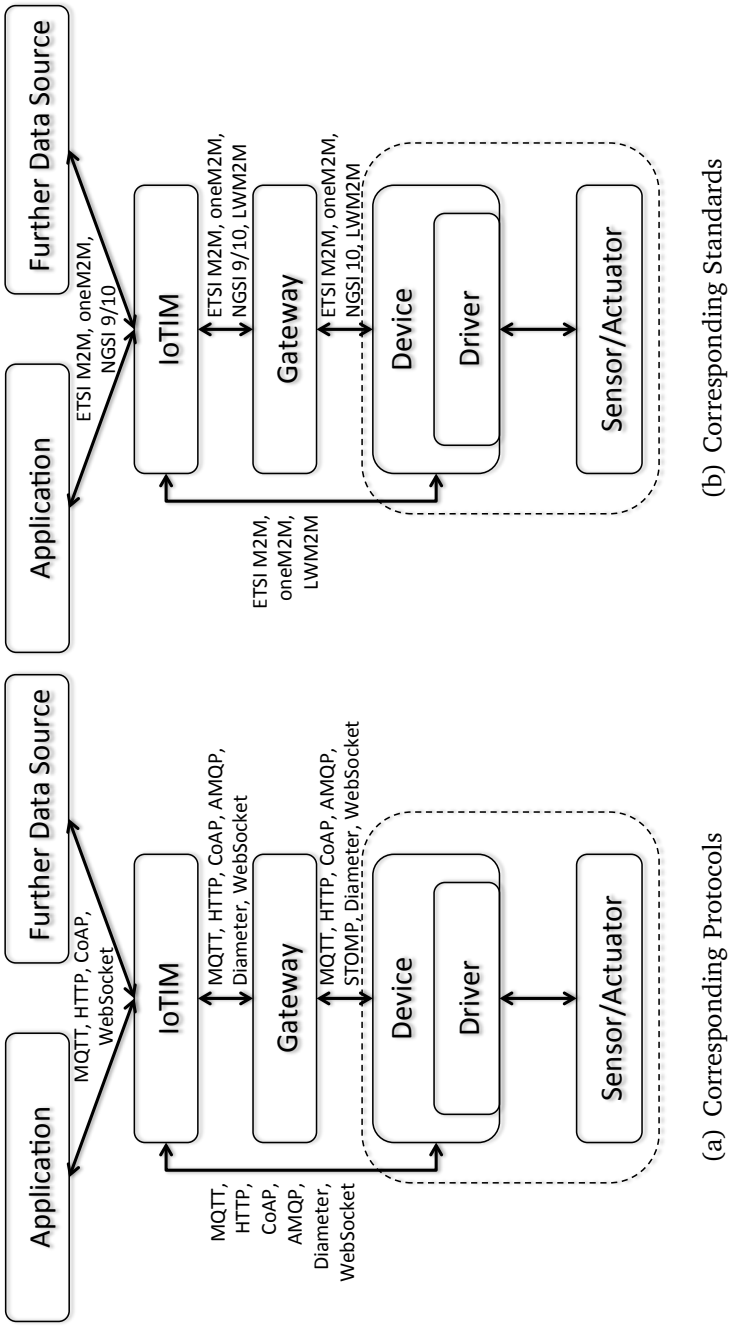
IoTIM. Likewise it is possible, that a device comprises a gateway, sensors and actuators, illustrated in Figure 3.2(b). A Smartphone can constitute a device comprising sensors, as well as comprising a gateway, as it can communicate directly with the IoTIM. If the Smartphone supports the required transport protocol and payload format, no translation of the gateway is needed. Additionally it can perform further functionality of a gateway, like managing the connection of devices, e.g., of a Smartwatch. Furthermore it is possible, that the IoTIM comprises the gateway, like, e.g., within the platform Microsoft Azure, depicted in the right Figure 3.2(c). This means that the platform integrates the functionality of a gateway, hence it cannot be separated.

Figure 3.3 shows the reference architecture with the corresponding transport protocols (a), which are defined in Section 2.1.2, and the reference architecture with the corresponding standards (b), which are defined in Section 2.1.3. Represented are only the transport protocols and standards used by the considered platforms. It illustrates which transport protocols and standards are used between the individual components. The transport protocols, MQTT, HTTP, CoAP, and WebSocket are used through every component of the reference architecture. AMQP and Diameter are used between the IoTIM, Gateway, and Device component, while STOMP is only used between the Device and the Gateway component. The connection of the IoTIM to the Application and Further Data Source component describes, that the platforms offer APIs to connect further applications and data sources, supporting the transport protocols MQTT, HTTP, CoAP, and WebSocket. The standards ETSI M2M, oneM2M, and NGSI 9/10 are used between the APIs, the IoTIM, and the Gateway component. Additionally between the IoTIM, and the Gateway component the LWM2M is used. Between the Gateway and the Device component the standards ETSI M2M, oneM2M, NGSI 10, and LWM2M are used. Between the IoTIM, and the Device component ETSI M2M, oneM2M, and LWM2M can be used to accomplish a direct communication. Likewise within the transport protocols, the connection of the IoTIM to the Application and Further Data Source components means, that the platforms provide APIs supporting the ETSI M2M, oneM2M, NGSI 9, and NGSI 10 standards, to connect further applications and data sources.



(a) Smartwatch (b) Smartphone (c) Microsoft Azure

Figure 3.2: Reference Architecture with Multiple Components Comprised within One Component



(a) Corresponding Protocols (b) Corresponding Standards

Figure 3.3: Reference Architecture with Corresponding Protocols and Standards

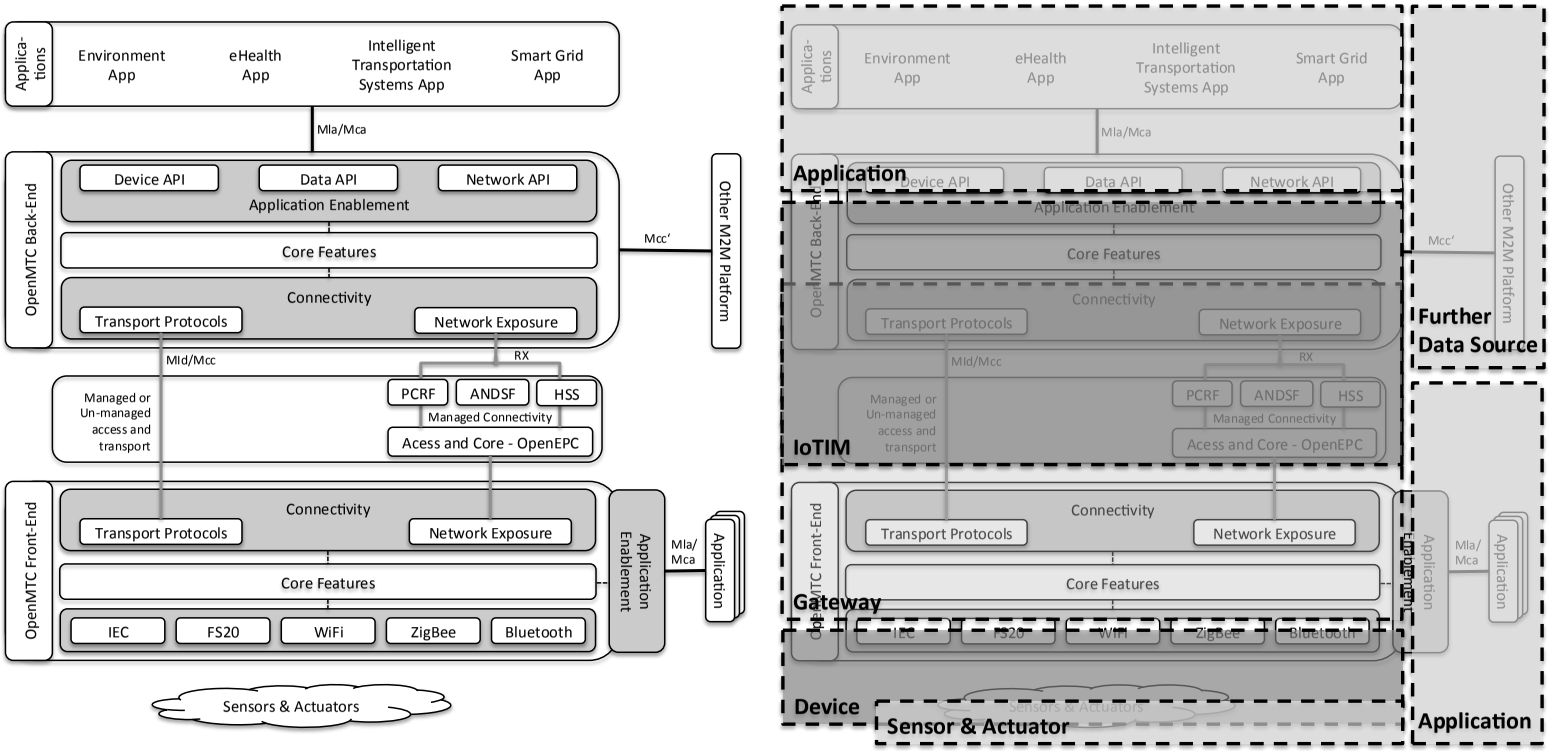
4 Validation of the Reference Architecture

This Chapter validates the developed reference architecture by mapping it to the architecture of every considered CPS solution. Differences are depicted, and a conclusion recapitulates the reference architecture and the applicability.

4.1 OpenMTC

Figure 4.1 shows the architecture of OpenMTC on the left side (a), and the mapping of the reference architecture with the underlying architecture of OpenMTC on the right side (b). OpenMTC enables the connection of sensors and actuators via communication technologies like WiFi, and Bluetooth. Since this correlates with the *Device* component, but not with the *Sensor & Actuator* component, they represent the *Device* component. Accordingly the *Sensor & Actuator* component is southbound partly overlapping it. Likewise partly comprised by the *Device* component is the lowest layer of the Front-End, which represent the communication technology of the devices. The *Gateway* component encompasses the Core Features of the Front-End, the Connectivity component inside the Front-End, and the Connectivity component of the Back-End, including the OpenEPC component in between. The OpenEPC component, the Core Features, and partly the Application Enablement of the Back-End are part of the *IoTIM*, since they provide the core logic of the platform. The *Application* component comprises the Application Enablement, and the Applications component of the Back-End, and the one of the Front-End, providing the connection, and the usage of further applications. Additionally the *Further Data Source* component consolidates the M2M Platforms connected to the Back-End.

Following this the reference architecture is applicable to the OpenMTC architecture.



(a) OpenMTC Architecture

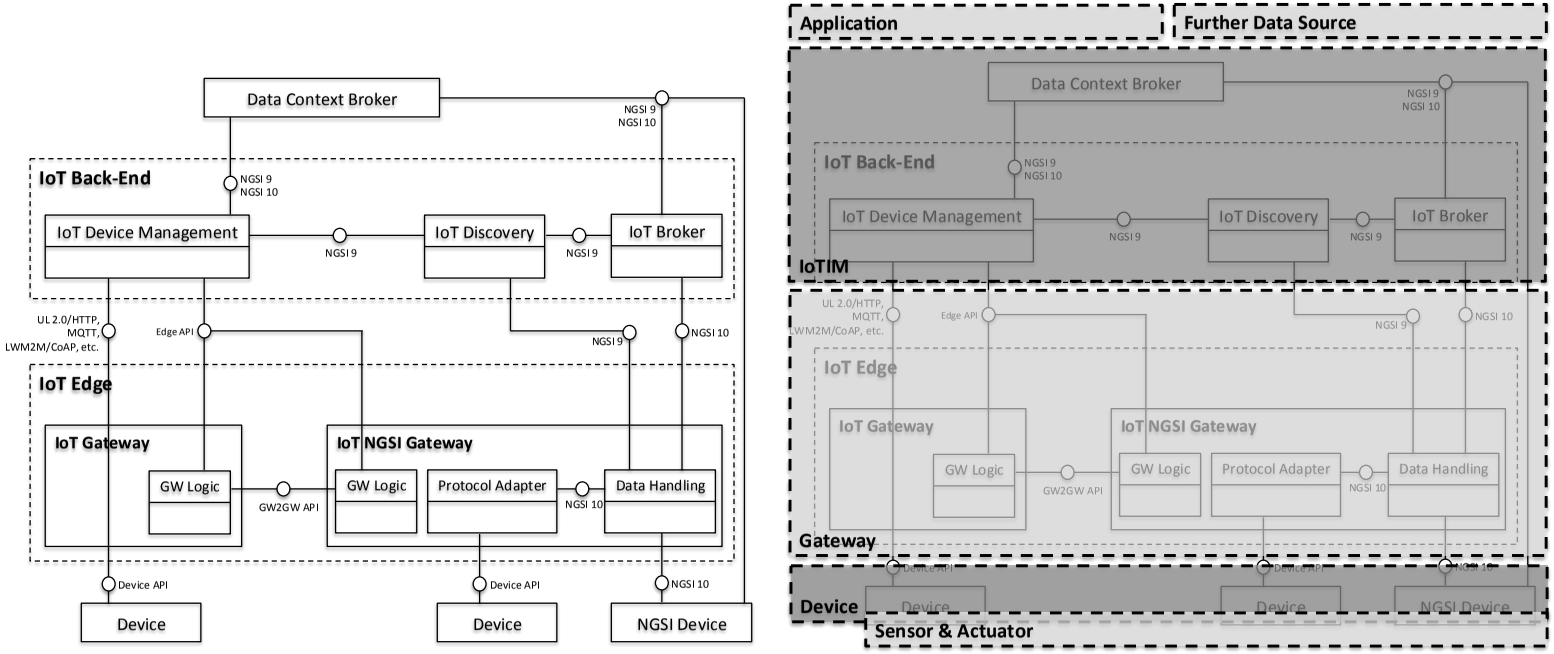
(b) OpenMTC Validation

Figure 4.1: OpenMTC Validation of the Reference Architecture based on [Fra15]

4.2 FIWARE

The architecture of FIWARE (a), and the mapping of the reference architecture (b) are shown in Figure 4.2. The FIWARE architecture does not represent sensors and/or actuators, as they follow the approach of devices with integrated and/or connected sensors and/or actuators. Hence the component *Sensor & Actuator* is partly overlapping the *Device* component. The *Device* component encapsulates all devices represented within the architecture. The *Gateway* comprises the IoT Edge, and the south- and northbound APIs, as they enable the translation of messages into the required format. The *IoTIM* encompasses the IoT Backend and the Data Context Broker, as they represent jointly the core logic of FIWARE's platform. Not depicted within the architecture are the possibly to the Data Context Broker connected applications and further data sources. The overall architecture depicting the GEs of FIWARE described within Section 2.2.2 shows, that further applications and data sources are possibly connected to the platform. Hence the *Application*, and the *Further Data Source* components are positioned northbound to the Data Context Broker.

Consequently the reference architecture maps to FIWARE's architecture, and is applicable to it.



(a) FIWARE Architecture

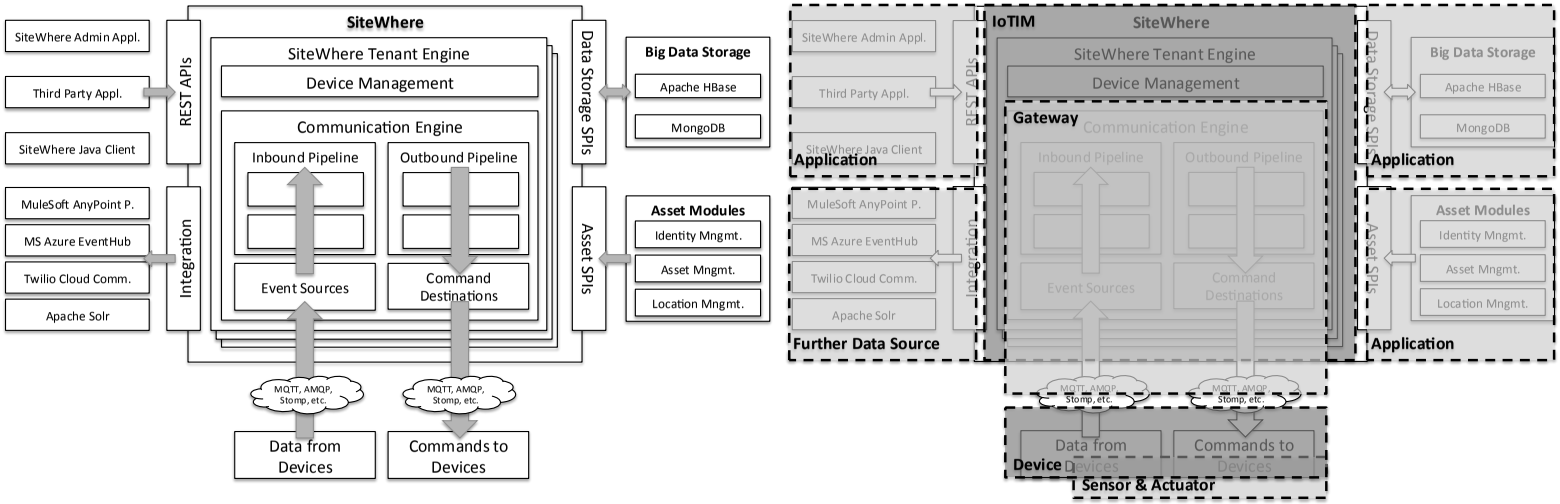
(b) FIWARE Validation

Figure 4.2: FIWARE Validation of the Reference Architecture based on [FIW15b]

4.3 SiteWhere

Figure 4.3 depicts the architecture of SiteWhere on the left (a), and the mapping with the reference architecture on the right (b). Since the SiteWhere architecture follows the approach of devices with integrated and/or connected sensors and/or actuators, they are not represented within the architecture. Hence the component *Sensor & Actuator* overlaps part of the *Device* component. The data from devices, and commands to devices components of the architecture represent the *Device* component of the reference architecture. The *Gateway* encapsulates the Communication Engine, which ensures the communication between the devices and the platform. The SiteWhere Tenant Engine is comprised by the *IoTIM*, as it represents the core functionality of the platform. On the right side of the platform the *Application* components are positioned, as well as another *Application* component on the left side, as all of those blocks provide the connection of further applications. Additionally the *Further Data Source* component comprises the Integration block on the left site, since it provides the connection and usage of further data sources.

Since the core functionality of SiteWhere integrates the connection component, the components *Gateway* and *IoTIM* are not precisely separable. Nevertheless the reference architecture is applicable to the SiteWhere architecture.



(a) SiteWhere Architecture

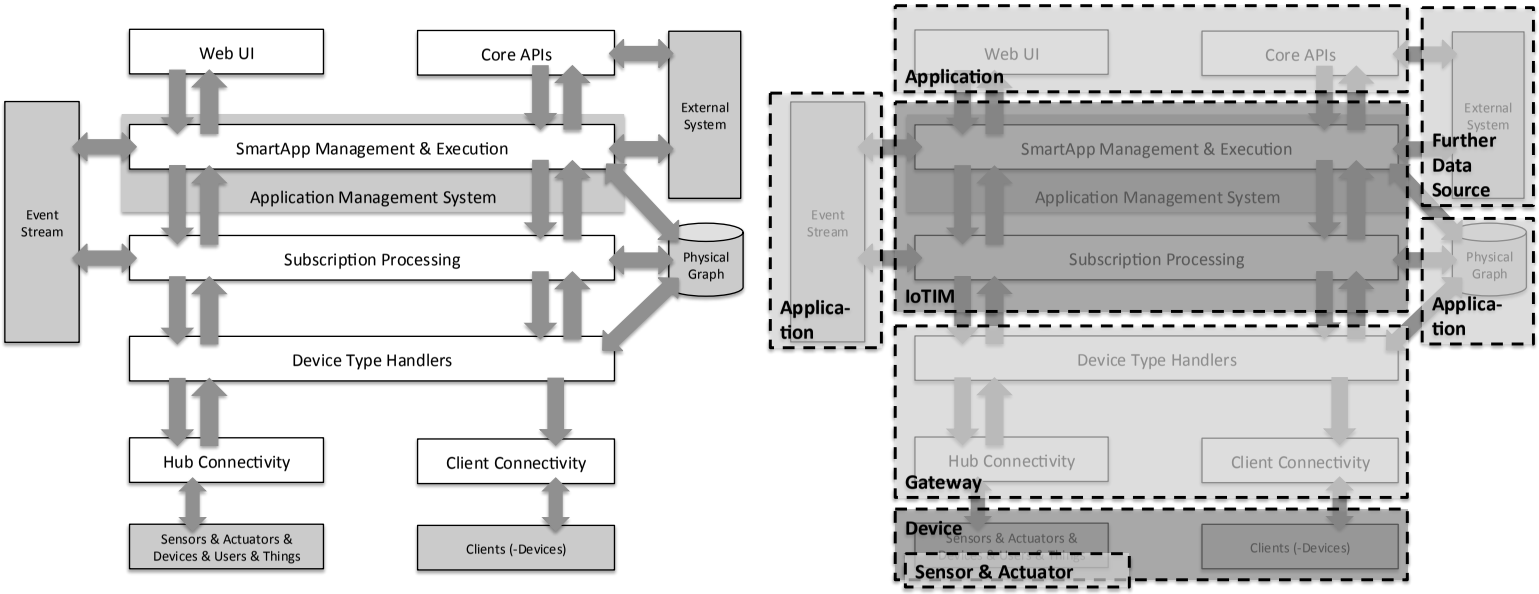
(b) SiteWhere Validation

Figure 4.3: SiteWhere Validation of the Reference Architecture based on [Sit16d]

4.4 SmartThings

On the left side (a) of Figure 4.4 the SmartThings architecture is imaged, and the mapping with the reference architecture is shown on the right side (b). Within the SmartThings architecture, sensors and actuators are represented within one component, together with devices, users and things. Because of that, the *Sensor & Actuator* component is only a part of this component. The *Device* component of the reference architecture comprises this joint component, as well as the Clients (-Devices) component. The Clients (-Devices) component does not correlate to the definition of the *Device* component, since they represent devices using the SmartThings App to control the platform, like, e.g., a Smartphone. But it is considerable, that they also send the data of their sensors to the IoTIM, and the user's control actions performed on the device can also be considered as an event. The *Gateway* encapsulates the Hub- and Client Connectivity, and the Device Type Handlers, which manage the connection of the devices, and the devices themselves. Encapsulated by the *IoTIM* are the Application Management System and the Subscription Processing, where the core functionality of the platform is placed. Within the SmartThings platform different applications can be connected, they are embraced by the *Application* component of the reference architecture. Thereby the Core APIs enable the connection, and the Event Stream, the Web UI, and the Physical Graph are the possibly connected applications. The External System component represents the *Further Data Source* component of the reference architecture.

Following the reference architecture can likewise be mapped to the SmartThings architecture, and hence is applicable to it.



(a) SmartThings Architecture

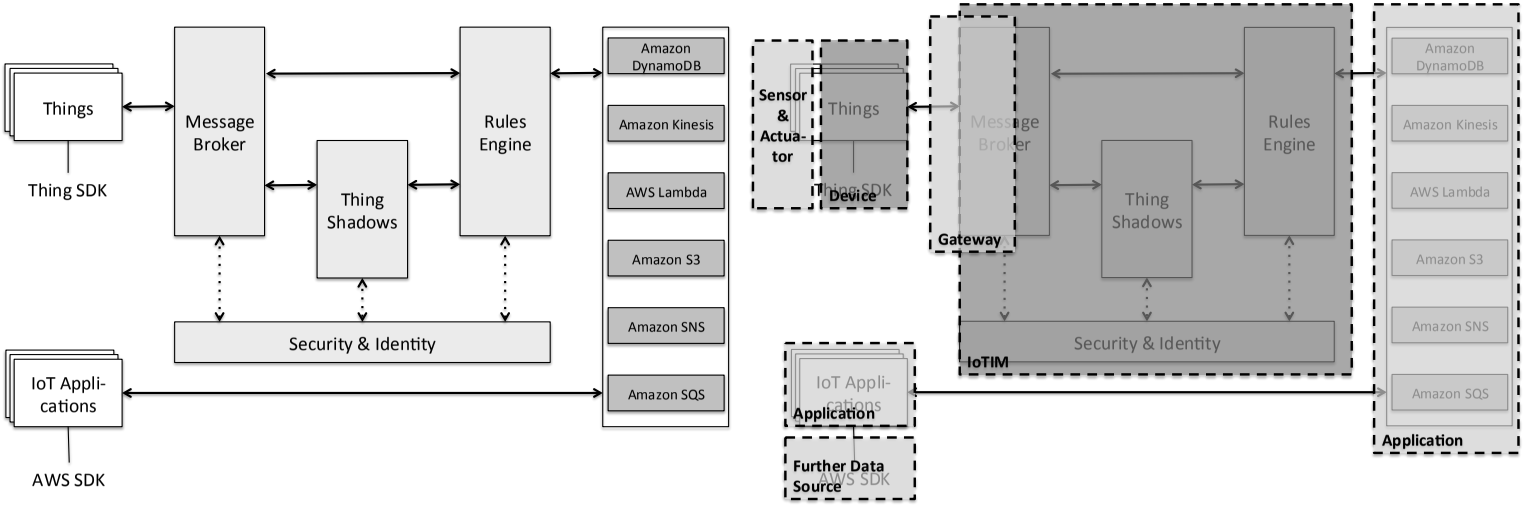
(b) SmartThings Validation

Figure 4.4: SmartThings Validation of the Reference Architecture based on [Sma15b]

4.5 AWS IoT

Figure 4.5 shows the architecture of AWS IoT (a), and the mapping of the reference architecture (b). The Things component of the AWS IoT architecture represents the *Sensor & Actuator*, as well as the *Device* component, since AWS IoT does not differ those components more precisely within their architecture. The *Gateway* sits between the Things and the Message Broker components, and comprises a part of the Message Broker component, as it manages the devices, and their connection to the platform. The Message Broker, the Thing Shadow, the Rules Engine, and the Security & Identitiy component are encapsulated by the *IoTIM*, since they provide the core functionality. Within the AWS IoT architecture possibly connected applications are represented in the IoT Applications component, and inside the collection of Amazon services on the right side. They are embraced by the *Application* component of the reference architecture. The *Further Data Source* component is represented by the AWS SDK.

Because of the approach of AWS IoT to integrate the connection component into the core functionality of the platform, the components *Gateway* and *IoTIM* are not precisely separable. Nevertheless the reference architecture is applicable to the AWS IoT architecture.



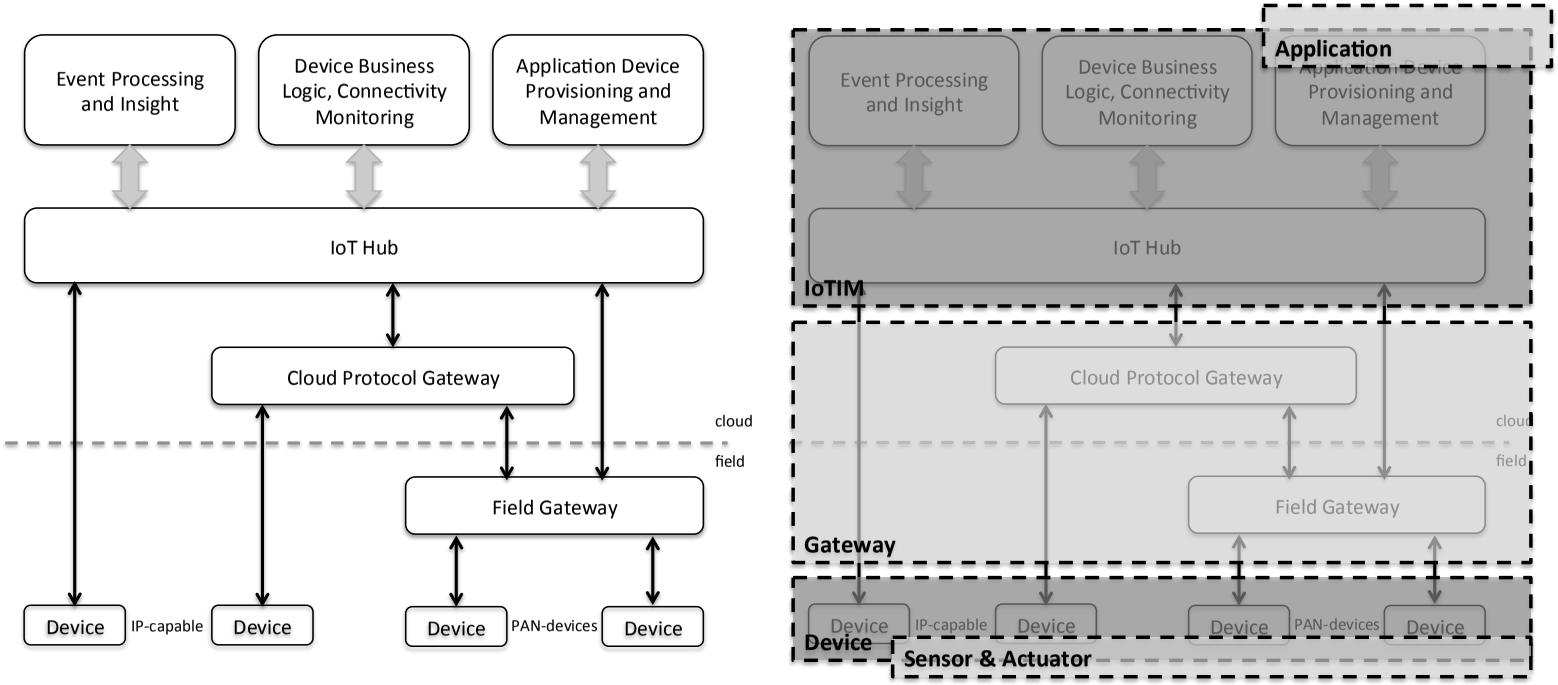
(a) AWS IoT Architecture

(b) AWS IoT Validation

Figure 4.5: AWS IoT Validation of the Reference Architecture based on [Ama16c]

4.6 Microsoft Azure IoT Hub

Within Figure 4.6 the Microsoft Azure's IoT Hub architecture is depicted on the left (a), and the mapping of the reference architecture on the right (b). As the architecture does not represent sensors and actuators, the *Sensor & Actuator* component of the reference architecture overlaps the *Device* component, which comprises all devices of the Microsoft Azure IoT Hub architecture. The *Gateway* encapsulates the Field Gateway and the Cloud Protocol Gateway, where the connection of the devices, and the message translation is handled. The core functionality, and hence the *IoTIM* component of the reference architecture is the IoT Hub, the Event Processing and Insight, the Device Business Logic, Connectivity Monitoring, and the Application Device Provisioning and Management. The last component of the underlying architecture is likewise partly the *Application* component of the reference architecture, as it enables the connection of further applications. The *Further Data Source* component is not represented within the Azure IoT Hub architecture, nevertheless the connection of further data sources is enabled. Even though the *IoTIM* and the *Application* components cannot be separated precisely, and the *Further Data Source* component is not represented within the architecture, the reference architecture is applicable to the architecture of the Microsoft Azure IoT Hub.



(a) Azure IoT Hub Architecture

(b) Azure IoT Hub Validation

Figure 4.6: Azure IoT Hub Validation of the Reference Architecture based on [Bet16]

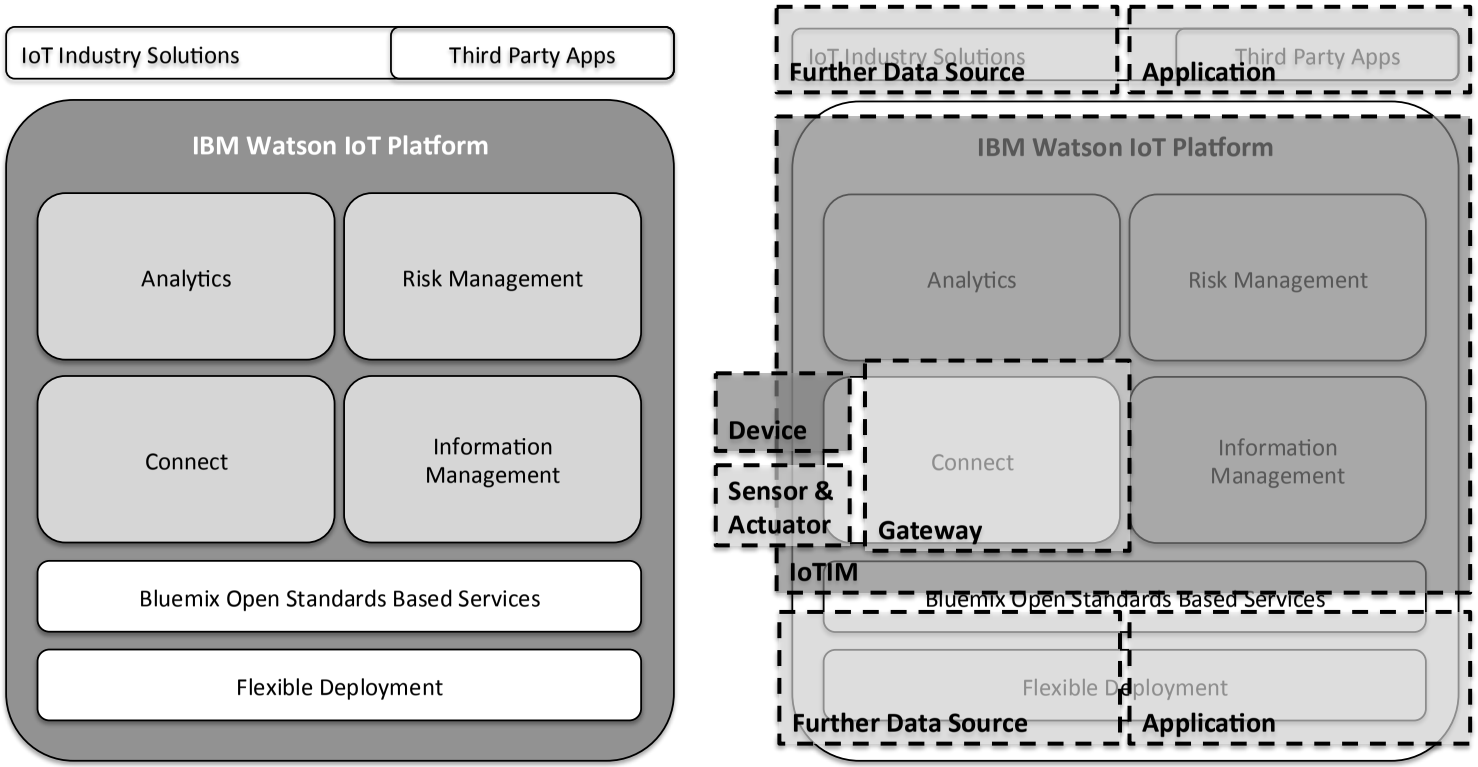
4.7 IBM Watson IoT Platform

Figure 4.7 shows the architecture of the IBM Watson IoT Platform (a), and the mapping of the reference architecture (b). Considering the Connection component of IBM's Watson IoT Platform it represents partly the *Sensor & Actuator* and the *Device* components of the reference architecture. Mainly it depicts the *Gateway*, since the connection of the devices, and the translation of messages are managed within the Connection component. The *IoTIM* comprises the Analytics, the Risk Management, the Information Management, and the Bluemix Open Standards Based Services, as they provide the core functionality of the platform. The IBM Watson IoT Platform enables the connection of further applications and data sources, within the compoenents IoT Industry Solutions, Third Party Apps, Flexible Deployment, as well as partly the Bluemix Open Standards Based Services. They are encapsulated by the *Application* and the *Further Data Source* components of the reference architecture.

Since the IBM Watson IoT Platform follows the approach of providing core functionality of the platform within connected IBM Bluemix Open Standards Based Services, a precise differentiation between the *IoTIM* and the *Application* component is not possible. Nevertheless the reference architecture is applicable to the IBM Watson IoT Platform.

4.8 Conclusion

The validation of the reference architecture with the considered CPS solutions has shown, that it is applicable to the architectures. In some cases a precise differentiation is not possible, caused by the integration of multiple components within one. Nevertheless the core components of the reference architecture are apparent. Hence the components of the reference architecture do not have to be strictly seperable. The derived reference architecture as imaged in Figure 3.1 can be used as a universal reference of the architecture of a CPS. Therefor it is important to respect the definition of the reference architecture components, described in Section 3.3.



(a) IBM Watson IoT Platform Architecture

(b) IBM Watson IoT Platform Validation

Figure 4.7: IBM Watson IoT Platform Validation of the Reference Architecture based on [IBM16a]

5 Design of the Abstract Class Model

The following Chapter describes how the abstract class model is designed, based on the considered CPS solutions. Initially the features of the CPS solutions are contemplated and compared. Adapted from that, the features and the correlations, discovered within every considered CPS platform, are specified, and the abstract class model is derived.

5.1 Analysis of the Features

The OpenMTC platform could not be considered for the analysis and comparison of the features, since the pages of the documentation have no content [Fra14]. The analysis of the features is based on the SiteWhere platform, as the classification of the features here is distinct, and well structured [Sit16c]. Within this Section the single categories of the SiteWhere features, called services, are described and compared to the FIWARE, SmartThings, AWS IoT, and the IBM Watson IoT Platform. The comparison is aided with the corresponding Section of the correlation matrix. The index declaration of the correlation matrix can be found within the appendix A.

The Asset Management service of SiteWhere manages objects in the physical world, i.e., people, places, and things, called Asset Categories. Those can be assigned to devices, to describe their association. The methods are depicted in the left column of Table 5.1 [Sit16c]. The Asset Management service provides methods to create, retrieve, delete, and update Asset Categories. FIWARE has the greatest intersection with those methods, as to every SiteWhere Asset Management method at least a comparable method of FIWARE can be mapped [FIW15c]. The second largest intersection is represented by the IBM Watson IoT Platform, whereat only comparable functionality can be found [IBM16b]. SmartThings only has one comparable method within the 20 methods of the SiteWhere Asset Management service [Sma15a]. No comparable methods has AWS IoT [Ama16b] [Ama16d] [Ama16c].

The SiteWhere Batch Operations service handles actions that operate on multiple devices, executing asynchronously, and providing a mechanism for monitoring progress over time, listed in the left column of Table 5.2 [Sit16c]. The SiteWhere Batch Operations service comprises methods to create, retrieve, and schedule a batch operation. Again,

FIWARE has at least a comparable functionality for each method of the SiteWhere Batch Operations service [FIW15c]. AWS IoT provides two comparable methods within the six SiteWhere Batch Operation methods [Ama16b] [Ama16d] [Ama16c]. Even though the IBM Watson IoT Platform has a functionality category named Bulk Operations, it has no comparable features to this SiteWhere service [IBM16b]. Likewise SmartThings has no comparable functionality [Sma15a].

The Device Assignment service of SiteWhere provides 25 methods to manage the association between a device and an optionally related asset, shown in Table 5.3 [Sit16c]. It comprises methods to create, retrieve, update, and delete a device assignment, as well as schedule command invocations, and create various events, based on the device assignment. As above, FIWARE provides at least a comparable functionality for each Device Assignment method [FIW15c]. The IBM Watson IoT Platform has comparable and a few same methods [IBM16b]. Likewise SmartThings and AWS IoT both provide some comparable features [Sma15a] [Ama16b] [Ama16d] [Ama16c].

The methods within SiteWhere's Device Command Invocation service comprise targeted functionality, such as listing invocation responses, and providing summary information about an invocation [Sit16c]. The methods are depicted in Table 5.4, whereat most related operations are covered by the Device Assignments service. Within this category only FIWARE and AWS IoT provide comparable functionality [FIW15c] [Ama16b] [Ama16d] [Ama16c].

The Device Commands service listed in Table 5.5, specifies methods SiteWhere can use to interact with a given hardware configuration, i.e., a device [Sit16c]. It comprises methods to delete, retrieve, and update a device command. FIWARE provides the same functionality within two different methods [FIW15c]. Comparable functionality is comprised by AWS IoT [Ama16b] [Ama16d] [Ama16c]. Both SmartThings and IBM Watson IoT Platform have no comparable functionality [Sma15a] [IBM16b].

SiteWhere Device Groups are used to create an association between related devices. The corresponding service comprises methods to create, retrieve, update and delete a device group, and methods to add and delete elements to/from a device group. The operations are depicted in Table 5.6 [Sit16c]. The same functionality, and additionally comparable functionality is provided by FIWARE [FIW15c]. The IBM Watson IoT Platform comprises comparable functionality for each method of the Device Groups service [IBM16b]. SmartThings has some comparable features, whereat AWS IoT comprises no comparable features [Sma15a] [Ama16b] [Ama16d] [Ama16c].

SiteWhere Device Specifications are applied, to capture characteristics of a given hardware configuration, including a list of commands that may be invoked on the device. The corresponding methods of this service are create, retrieve, update, and delete a device specification, and retrieve the attached commands, listed in Table 5.7 [Sit16c]. In

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Asset Management								
Create a New Asset Category	OS Clustering (2)	Semantic Support (3)						DT: Create device type
Create a New Hardware Asset in Category		NGSI (1)						DT: Create device type
Create a New Location Asset in Category		Pub/Sub Semantic Extension (4); Location Server (8);						D: Get device location information
Create a New Person Asset in Category		NGSI (1)						DT: Create device type
Delete an Existing Asset Category	OS Clustering (2)	Semantic Support (3)						DT: Delete device type
Delete an Existing Category Asset		NGSI (1)						DT: Delete device type
Get a Category Asset by Unique Id		NGSI (1)						DT: Get device type
Get an Asset by Unique Id		NGSI (1)		Attribute: value				DT: Get device type
Get an Asset Category by Unique Id	OS Clustering (2)	Semantic Support (3)						DT: Get device type
Get an Asset Module by Unique Id	OS Clustering (2)							DT: Get device type
List Asset Categories that Match Criteria	OS Clustering (2)							DT: List device types
List Asset Modules that Match Criteria	OS Clustering (2)							BO: List devices
List Assignments Associated with an Asset		NGSI (1)						BO: List devices
List Category Assets that Match Criteria	OS Clustering (2)							DT: List device types
Refresh the List of Asset Modules	OS Clustering (2)							
Search for Assets in an Asset Module	OS Clustering (2)							
Update an Existing Asset Category	OS Clustering (2)	Semantic Support (3)						DT: Update device type
Update an Existing Hardware Asset in Category		NGSI (1)						DT: Update device type
Update an Existing Location Asset in Category		Device Sensors (10)						DT: Update device type
Update an Existing Person Asset in Category		NGSI (1)						DT: Update device type

Table 5.1: Correlation Matrix to Compare the Features within SiteWhere Category Asset Management

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**
Batch Operations								
Create Batch Command Operation Based on Criteria		Service Composition (6)						
Create New Batch Command Invocation	Service Composition (6)	Complex Event Processing (5)				CreateTopicRule		
Get Batch Operation by Unique Token		Service Composition (6)						
List Batch Operation Elements		Service Composition (6)						
List Batch Operations	Service Composition (6)					ListTopicRules		
Schedule Batch Command Operation Based on Criteria	Service Composition (6)	Complex Event Processing (5)						

Table 5.2: Correlation Matrix to Compare the Features within SiteWhere Category Batch Operations

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Device Assignments								
Add Data to Device Assignment Data Stream		Semantic Annotation (7)						
Create a New Device Assignment	Semantic Annotation (7)			Attribute: name		CreateThing		
Create Alert Event for Device Assignment	OS Clustering (2)					CreateTopicRule		
Create Command Invocation Event for Assignment	Service Composition (6)					CreateTopicRule		
Create Command Response Event for Assignment		Service Composition (6)				CreateTopicRule		
Create Data Stream for Device Assignment		Semantic Annotation (7)						DD: Add device diagnostic log
Create Location Event for Device Assignment		Location Server (8)						
Create Measurements Event for Device Assignment		Service Composition (6)						
Delete an Existing Device Assignment	Semantic Annotation (7)							
Get All Data from Device Assignment Data Stream		Semantic Annotation (7)					DD: Get all device diagnostic logs	
Get Data from Device Assignment Data Stream		Semantic Annotation (7)					DD: Get device diagnostic log	
Get Device Assignment By Token	Semantic Annotation (7)			Attribute: value		DescribeThing		BO: List devices
Get Device Assignment Data Stream by Id		Semantic Annotation (7)						
List Alert Events for Device Assignment	Service Composition (6)					List / GetTopicRule		
List Assignment Measurements as Chart Series		Service Composition (6)						
List Command Invocation Events for Assignment	Service Composition (6)					GetTopicRule		
List Command Response Events for Assignment		Service Composition (6)				GetTopicRule		
List Data Streams for Device Assignment		Semantic Annotation (7)						DD: Get all device diagnostic logs
List Events for Device Assignment		Service Composition (6)		Device: events			HER: View events across all devices of	
List Location Events for Device Assignment		Service Composition (6)		Event: location; locationId				HER: View events across all devices of a specific type
List Measurement Events for Device Assignment		Service Composition (6)						HER: View events across all devices of a specific type
Mark Device Assignment as Missing		Semantic Annotation (7)						
Release an Active Device Assignment		Semantic Annotation (7)						
Schedule Command Invocation	Service Composition (6)							
Update Device Assignment Metadata	Semantic Annotation (7)					UpdateThing		

Table 5.3: Correlation Matrix to Compare the Features within SiteWhere Category Device Assignments

this case FIWARE does not cover all methods with a comparable functionality. Probably they would be comprised by the Administration REST API, where the page of the documentation has no content [FIW15c]. The IBM Watson IoT Platform provides comparable features for most of the methods [IBM16b]. SmartThings and AWS IoT both only provide two comparable methods [Sma15a] [Ama16b] [Ama16d] [Ama16c].

Within the SiteWhere platform a device is a representation of a connected physical hardware entity, that conforms to a known device specification, the corresponding functionality is depicted in Table 5.8 [Sit16c]. The Devices service comprises methods to create, retrieve, update and delete a device, and a device element mapping. Noticeable within this category is, that nearly every method is supported by every platform. FIWARE and the IBM Watson IoT Platform provide mostly even the same functionality, whereat SmartThings and AWS IoT provide comparable features [FIW15c] [IBM16b] [Sma15a] [Ama16b] [Ama16d] [Ama16c]. Additionally the Devices service provides a method to add multiple events for a device, which is not covered by any of the compared platforms.

The SiteWhere Events service is used to directly access event information by an unique event id. All other event-related methods are available in the Device Assignments service, since SiteWhere events are always posted in the context of an assignment [Sit16c]. Table 5.9 shows that except of AWS IoT, every platform comprises a comparable functionality [Ama16b] [Ama16d] [Ama16c] [FIW15c] [IBM16b] [Sma15a].

SiteWhere enables the usage of external search engines to operate on it's data, which allows SiteWhere to enrich the result data if necessary, and presents a single view of the data, whether stored in SiteWhere, or indexed in an engine optimized for adhoc queries [Sit16c]. The methods corresponding to External Search Providers are listed in Table 5.10. FIWARE and the IBM Watson IoT Platform both provide comparable functionality [FIW15c] [IBM16b]. SmartThings and AWS IoT comprise no methods for the usage of external search providers [Sma15a] [Ama16b] [Ama16d] [Ama16c].

SiteWhere Grated Authorities are permissions assigned to users, to allow access to various pieces of the system functionality. They are used to restrict the access to the REST services [Sit16c]. The methods of the Granted Authorities service, comprising the creation and retrieval of authorities, are listed in Table 5.11. FIWARE and AWS IoT both comprise comparable functionality to the Granted Authorities services [FIW15c] [Ama16b] [Ama16d] [Ama16c]. SmartThings and the IBM Watson IoT Platform do not provide comparable methods [Sma15a] [IBM16b].

SiteWhere defines Scheduled Jobs as system actions, that are performed on a specified schedule. The Scheduled Jobs service comprises methods to create, retrieve, update, and delete a scheduled job, depicted in Table 5.12 [Sit16c]. FIWARE provides comparable functionality for all methods of the Scheduled Jobs ser-

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Device Command Invocations								
Get Command Invocation by Unique Id	Service Composition (6)					GetTopicRule		
Get Command Invocation Summary		Service Composition (6)				GetTopicRule		
List Responses for Command Invocation		Service Composition (6)				ListTopicRules		

Table 5.4: Correlation Matrix to Compare the Features within SiteWhere Category Device Command Invocations

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Device Commands								
Delete Device Command by Unique Token	Complex Event Processing (5); Service Composition (6)					DeleteTopicRule		
Get Device Command by Unique Token	Complex Event Processing (5); Service Composition (6)					GetTopicRule		
Update an Existing Device Command	Complex Event Processing (5); Service Composition (6)					ReplaceTopicRule		

Table 5.5: Correlation Matrix to Compare the Features within SiteWhere Category Device Commands

vice [FIW15c]. SmartThings and AWS IoT comprise some comparable functionality [Sma15a] [Ama16b] [Ama16d] [Ama16c]. The IBM Watson IoT Platform does not provide any comparable methods [IBM16b].

The SiteWhere Schedules provide the ability to run jobs at another point in time, rather than immediately firing an action [Sit16c]. The service comprises methods to create, retrieve, update, and delete a schedule, they are listed in Table 5.13. FIWARE and AWS IoT both provide comparable functionality to the Schedules service [FIW15c] [Ama16b] [Ama16d] [Ama16c]. SmartThings provides some comparable methods, and the IBM Watson IoT Platform does not comprise any comparable methods at all [Sma15a] [IBM16b].

The Sites service methods of SiteWhere are used to organize devices that are related, so that their events can be looked at from a grouped perspective. The methods are depicted in Table 5.14 [Sit16c]. The service comprises methods to create, retrieve, update, and delete a site, as well as retrieving alerts, command invocations, and responses for a site. Furthermore it provides an operation to create a zone within a site, which is related to the Zones service. Again FIWARE provides comparable functionality for every method of the Sites service [FIW15c]. SmartThings, AWS IoT, and the IBM Watson IoT Platform comprise a few comparable methods [Sma15a] [Ama16b] [Ama16d] [Ama16c] [IBM16b].

SiteWhere's System Information service provides two methods to retrieve information about the running SiteWhere instance, i.e., the server runtime state, and the version information, listed in Table 5.15 [Sit16c]. FIWARE provides the same functionality, the IBM Watson IoT Platform comprises comparable methods [FIW15c] [IBM16b]. Both SmartThings and AWS IoT do not comprise methods comparable to the System Information service of SiteWhere [Sma15a] [Ama16b] [Ama16d] [Ama16c].

SiteWhere provides an architecture, where multiple IoT applications can run concurrently in separate containers, called Tenants, containing its own data and processing pipeline [Sit16c]. The corresponding methods of the Tenant service are depicted in Table 5.16. The service comprises methods to create, retrieve, update, and delete a tenant. Additionally it provides a method to send a command to a Tenant Engine. FIWARE provides even multiple comparable methods of the Tenant service of SiteWhere [FIW15c]. The IBM Watson IoT Platform comprises some comparable functionality. SmartThings and AWS IoT do not comprise any comparable methods [IBM16b] [Sma15a] [Ama16b] [Ama16d] [Ama16c].

SiteWhere users represent entities, which are authorized to use the system. The corresponding User service comprises methods to create, retrieve, update, and delte an user, and to retrieve the authorities for an user. The operations are listed in Table 5.17 [Sit16c]. Again FIWARE comprises the same functionality as the SiteWhere Users service [FIW15c]. AWS IoT and the IBM Watson IoT Platform both provide some

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**
Device Groups								
Add Elements to Device Group	OS Clustering (2)	Semantic Support (3)		Device Handler: definition; addChildDevice				BO: Register multiple new devices
Create New Device Group	OS Clustering (2)	Semantic Support (3)						DT: Create device type
Delete Device Group by Unique Token	OS Clustering (2)	Semantic Support (3)						DT: Delete device type
Delete Elements from Device Group	OS Clustering (2)	Semantic Support (3)		SmartApp: deleteChildDevice				BO: Delete multiple devices
Get a Device Group by Unique Token	OS Clustering (2)	Semantic Support (3)						DT: Get device type
List Device Groups that Match Criteria	OS Clustering (2)	Semantic Support (3)						DT: List device types
List Elements in a Device Group	OS Clustering (2)	Semantic Support (3)		SmartApp: getChildDevices				BO: List devices
Update an Existing Device Group	OS Clustering (2)	Semantic Support (3)						DT: Update device type

Table 5.6: Correlation Matrix to Compare the Features within SiteWhere Category Device Groups

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Device Specifications								
Create Device Command for Specification	OS Clustering (2)			Device Handler: <command name>				
Create New Device Specification		NGSI (1)		Device Handler: definition				DT: Create device type
Delete Existing Device Specification		NGSI (1)						DT: Delete device type
Get Specification GPB by Unique Token								
Get Specification GPB File by Unique Token								
Get Specification by Unique Token		NGSI (1)						DT: Get device type
List Device Commands by Namespace						List / Get TopicRules		
List Device Commands for Specification	OS Clustering (2)					List / Get TopicRules		
List Specifications that Match Criteria	OS Clustering (2)							DT: List device types
Update Existing Device Specification		NGSI (1)						DT: Update device type

Table 5.7: Correlation Matrix to Compare the Features within SiteWhere Category Device Specifications

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**
Devices								
Add Multiple Events for Device								
Create New Device	ETSI M2M mId (9)			Device Handler: definition; Device: name		CreateThing	D: Add device	
Create New Device Element Mapping		NGSI (1); OS Clustering (2)		Attribute: name		CreateThing		
Delete Device Based on Unique Hardware Id	ETSI M2M mId (9)			SmartApp: deleteChildDevice		DeleteThing	D: Remove device	
Delete Existing Device Element Mapping		NGSI (1); OS Clustering (2)		SmartApp: deleteChildDevice				
Get Current Assignment for Device						DescribeThing		
Get Device by Unique Hardware Id	ETSI M2M mId (9)		Device: id	Device: name; SmartApp: getChildDevice		DescribeThing	D: List devices	
List Assignment History for Device								
List Devices in Device Group	OS Clustering (2)			SmartApp: getChildDevices			BO: List devices	
List Devices in Device Groups with Role		OS Clustering (2)						BO: List devices
List Devices that Match Criteria	ETSI M2M mId (9)					ListThings	D: List devices	
List Devices Using a Given Specification	OS Clustering (2)					ListThings	D: List devices	
Update an Existing Device	ETSI M2M mId (9)					UpdateThing	D: Update device	

Table 5.8: Correlation Matrix to Compare the Features within SiteWhere Category Devices

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Events								
Get Event by Unique Id		Query Broker (11)		Event: id; description; descriptionText				HER: View events for a specific device; EC: Get list of events for a specific device; EC: Get last event in a specific event for a specific device

Table 5.9: Correlation Matrix to Compare the Features within SiteWhere Category Events

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**
External Search Providers								
List Available Search Providers		Apps Marketplace Search (12)						ES: Invoke an operation on an external service that has been integrated with the IOTF Platform
Search for Events in Provider		Apps Marketplace Search (12)						

Table 5.10: Correlation Matrix to Compare the Features within SiteWhere Category External Search Providers

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Granted Authorities								
Create a New Authority		Security Chapter AccessControl GE Authorization (13)				CreatePolicy		
Get Authority by Id		Security Chapter AccessControl GE Authorization (13)				GetPolicy (Describe Certificate)		
List Authorities that Match Criteria		Security Chapter AccessControl GE Authorization (13)				ListPrincipalThings; ListThingPrincipals; ListPolicies		

Table 5.11: Correlation Matrix to Compare the Features within SiteWhere Category Granted Authorities

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**	**Same Funcitonality**	**Comparable Functionality**
Scheduled Jobs								
Create New Scheduled Job		Complex Event Processing (5); Mediator GE (14)		SmartApp: schedule		CreateTopicRule		
Delete Scheduled Job		Complex Event Processing (5); Mediator GE (14)				DeleteTopicRule		
Get Scheduled Job by Token		Complex Event Processing (5); Mediator GE (14)				GetTopicRule		
List Scheduled Jobs Matching Criteria		Complex Event Processing (5)						
Update Existing Scheduled Job		Complex Event Processing (5); Mediator GE (14)						

Table 5.12: Correlation Matrix to Compare the Features within SiteWhere Category Scheduled Jobs

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Schedules								
Create a New Schedule		Query Broker (11); Mediator GE (14)				CreateTopicRule		
Delete a Schedule		Query Broker (11); Mediator GE (14)				DeleteTopicRule		
Get Schedule by Token		Query Broker (11); Mediator GE (14)				GetTopicRule		
List Schedules that Match Criteria		Query Broker (11); Mediator GE (14)				ListTopicRule		
Update an Existing Schedule		Query Broker (11); Mediator GE (14)				ReplaceTopicRule		

Table 5.13: Correlation Matrix to Compare the Features within SiteWhere Category Schedules

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Sites								
Create New Site		OS Clustering (2); Pub/Sub Semantic Extension (4); Location Server (8); Device Sensors (10)		Location: id; latitude; longitude; etc.				
Create New Zone for Site		OS Clustering (2); Pub/Sub Semantic Extension (4); Location Server (8); Device Sensors (10)						
Delete Site by Token		OS Clustering (2); Pub/Sub Semantic Extension (4); Location Server (8); Device Sensors (10)						
Get Site by Unique Token		OS Clustering (2); Pub/Sub Semantic Extension (4); Location Server (8); Device Sensors (10)		Location: name				
List Alerts for Site		OS Clustering (2); Location Server (8)				List / Get / Create TopicRule		
List Command Invocations for Site		OS Clustering (2); Location Server (8)				List / Get / Create TopicRule		
List Command Responses for Site		OS Clustering (2); Location Server (8)				List / Get / Create TopicRule		
List Device Assignments for Site		OS Clustering (2)						D: Get device location
List Locations for Site		OS Clustering (2); Pub/Sub Semantic Extension (4); Location Server (8); Device Sensors (10)						D: Get device location information
List Measurements for Site		OS Clustering (2); Location Server (8)						
List Sites Matching Criteria		OS Clustering (2); Location Server (8)						
List Zones for Site		OS Clustering (2); Pub/Sub Semantic Extension (4); Location Server (8); Device Sensors (10)						
Update Existing Site		OS Clustering (2); Pub/Sub Semantic Extension (4); Location Server (8); Device Sensors (10)						

Table 5.14: Correlation Matrix to Compare the Features within SiteWhere Category Sites

	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
SiteWhere	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
System Information								
Get Server Runtime State	OS Compute (15)							
Get Version Information	OS Compute (15)							OC: Get oranization details; D: Get device management information

Table 5.15: Correlation Matrix to Compare the Features within SiteWhere Category System Information

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Tenants								
Create New Tenant		Semantic Support (3); OS Networking (16); OS Shared File Systems (17)						
Delete Existing Tenant		Semantic Support (3); OS Networking (16); OS Shared File Systems (17)						
Get Tenant by Authentication Token		OS Networking (16); OS Shared File Systems (17)						OC: Get organization details
Get Tenant by Unique Id		Semantic Support (3); OS Networking (16); OS Shared File Systems (17)						OC: Get organization details
Get Tenant Engine Configuration		OS Networking (16); OS Shared File Systems (17)						
List Tenants that Match Criteria		Semantic Support (3); OS Networking (16); OS Shared File Systems (17)						
Send Command to Tenant Engine		OS Networking (16); OS Shared File Systems (17)						
Update an Existing Tenant		Semantic Support (3); OS Networking (16); OS Shared File Systems (17)						

Table 5.16: Correlation Matrix to Compare the Features within SiteWhere Category Tenants

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Users								
Create New User	OS Identity (18)							
Delete User by Username	OS Identity (18)							
Get Authorities for User	OS Identity (18)					GetPolicy; (Describe Certificate)		OC: Get organization details
Get User by Username	OS Identity (18)							OC: Get organization details
List Authorized Tenants for User	OS Identity (18)					ListPolicies		OC: Get organization details
List Users Matching Criteria	OS Identity (18)					ListThings		
Update Existing User	OS Identity (18)							

Table 5.17: Correlation Matrix to Compare the Features within SiteWhere Category Users

SiteWhere	FIWARE		SmartThings		AWS IoT		IBM Watson IoT Platform	
	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality	Same Funcitonality	Comparable Functionality
Zones								
Delete Zone by Unique Token		Location Server (8); Device Sensors (10); OS Image Service (19)						
Get Zone by Token		Location Server (8); Device Sensors (10); OS Image Service (19)						D: Get device location information
Update an Existing Zone		Location Server (8); Device Sensors (10); OS Image Service (19)						D: Update device location information

Table 5.18: Correlation Matrix to Compare the Features within SiteWhere Category Zones

comparable methods [Ama16b] [Ama16d] [Ama16c] [IBM16b]. SmartThings does not provide any comparable functionality [Sma15a].

Within the SiteWhere platform a zone is a user-defined geographical area, that carries a special meaning within the IoT application, like, e.g., a secure area in an airport, where only certain personnel should be allowed. The Zones service provides functionality to monitor such zones, and to react on inadvertent events, e.g., to fire an alert if an unauthorized person enters the secure zone of an airport [Sit16c]. The corresponding methods of the Zones service are depicted within Table 5.18. Noticeable, the service only comprises methods to retrieve, update, and delete a zone. The operation to create a zone is covered by the Sites service, since zones are attached to sites. FIWARE and the IBM Watson IoT Platform both comprise comparable functionality [FIW15c] [IBM16b]. SmartThings provides a comparable method to retrieve a zone, and AWS IoT does not provide any comparable functionality to the SiteWhere Zones service at all [Sma15a] [Ama16b] [Ama16d] [Ama16c].

FIWARE, SmartThings, AWS IoT, and the IBM Watson IoT Platform all provide further features. As the comparison has shown, SiteWhere also provides functionality, which are not covered by the remaining ones. Considering those further features, there is no functionality, which is covered by most of the remaining platforms. Hence it is assumed, that they represent additional functionality, which does not influence the core functionality, and which is not considered within the following research.

5.2 Requirements of the Features and the Abstract Class Model

As the analysis and comparison of the features has shown, some categories are covered by all platforms, and hence are essential for a CPS platform. Within this Section those categories are reviewed, and the requirements of the abstract class model are derived.

Obviously a service to manage the devices of a CPS is one of the most important functions, as every platform provides methods comparable to the Devices service of SiteWhere (5.8). Especially methods to create, retrieve, update, and delete devices are essential to a CPS platform. Additionally the comparison within the Device Assignments service, and the Device Specification service of SiteWhere has indicated, that the platform should be able to classify, and to provide further information about the devices with the aid of a device specification (5.7), description, and assignments (5.3). Another important feature is the ability to aggregate devices and/or users into logical or geographical groups, as shown within the comparison of the Device Groups (5.6), the Sites (5.14), and the Zones (5.18) services of SiteWhere. Furthermore this should enable not only to look at the data

from a grouped perspective, but also to enable the platform to send commands to every participant of a group, site, or zone. Noticeable is, that the SiteWhere platform uses sites and zones in an overlapping term, i.e., a site is used for location-aware devices and their events, and a zone is a user-defined geographical area. The compared platforms do not differ within sites and zones in the exact same manner.

Another important, already mentioned feature are commands (5.5), invocations (5.4), batch operations (5.2), and schedules (5.13, 5.12). They prepare the core functionality of the platform in terms of the communication between the devices and the platform, and the control of the devices. Obviously commands enable the control of devices, and invocations enable an automated, predefined control of devices. The comparison has shown, that batch operations are not provided by every platform. Nevertheless they are crucial for an eased usage of a CPS platform, as they enable sending one command to multiple devices. Likewise schedules are instrumental in enabling an eased usage, as they assure to establish a time-table for commands.

Events are like devices essential to a CPS platform. They are the key enabler of monitoring and controlling the physical world, which defines the aim of CPS. Furthermore they prepare the basis for the communication of devices with the platform. The comparison within the Events (5.9), and Asset Management (5.1)services of SiteWhere indicated that, as well.

The analysis of the Users service (5.17) of SiteWhere has shown, that beside SmartThings every platform provides functionality to manage the users of the platform. Since the user interacts with, and configures the platform, a component which manages the users should be provided. It is assumed that the SmartThings platform likewise provides some kind of a user management, which is not represented within the features.

The Tenants (5.16), System Information (5.15), and External Search Providers (5.10) services of SiteWhere are not covered by every considered platform. Nevertheless they provide beneficial functionality. The deployment of tenants in terms of a separate container, so that multiple IoT applications can run concurrently are presumably also used by the remaining platforms. For instance AWS IoT and the IBM Watson IoT Platform provide accounts for their clients, where they can access only their own application. It can be of interest for the user to retrieve information system, when, e.g., the application instance has connection problems, or bad response times. Respective the External Search Providers service, it is likewise presumable, that comparable functionality is provided by the remaining platforms, since all of them provide an access point for further applications within their architectures (3.1).

5.3 Abstract Class Model

Following the analysis and the requirements of the features, the abstract class model is derived. Figure 5.1 shows the whole abstract class model, below the individual components, and the correlations are described in detail. A simple connection between two classes means that they have an interconnection, and can access the other class, e.g., to use the attributes or operations in a desired way.

Figure 5.2 shows the excerpt of the platform class, and the connected classes of the upper part of the abstract class model. Initially the platform has the attributes platformID, which is the unique identifier, and the attribute description, which allows to characterize the platform textual. The platform class represents the underlying basis, where all other classes are connected to, and at least partly dependent.

Northbound with a composition connected to the Platform is the Platform Information class. This class cannot exist without the Platform class, and every instance of the Platform class has one instance of the Platform Information class assigned. The Platform Information class provides operations to retrieve information about the Platform itself, the server, and the version of the instance running on the platform. As mentioned within Section 5.2, this class provides beneficial information to the user, if necessary.

Likewise northbound via an aggregation is the AuthorityManager class, which can be connected with zero to multiple platform instances, as the managed authorities can possibly be applied to more than one platform instance. Subsequently one platform instance can be connected to zero to multiple AuthorityManagers. The AuthorityManager contains a list of all authorities, and comprises operations to create, retrieve, update, and delete an instance of the Authority class. The Authority class is connected to the AuthorityManager class via a composition, i.e., the Authority class cannot exist without the AuthorityManager class, and one instance of the AuthorityManager can have zero to multiple instances of the Authority class associated. The Authority class contains the attributes authority ID, which is the unique identifier, a name, and a description of the authority. Those two classes build the basis for managing the access control and the security.

Furthermore northbound connected to the Platform is the Pluggable Service class, which contains an operation to retrieve available pluggable services. They can be internal, i.e., services of the same provider as the platform, or external in terms of third party services. One Platform can have zero to multiple instances of the Pluggable Service class associated, and the Pluggable Service can be connected to zero to multiple Platform instances. The Pluggable Service class represents the access point of further possibly connected applications, like observed within every analyzed platform.

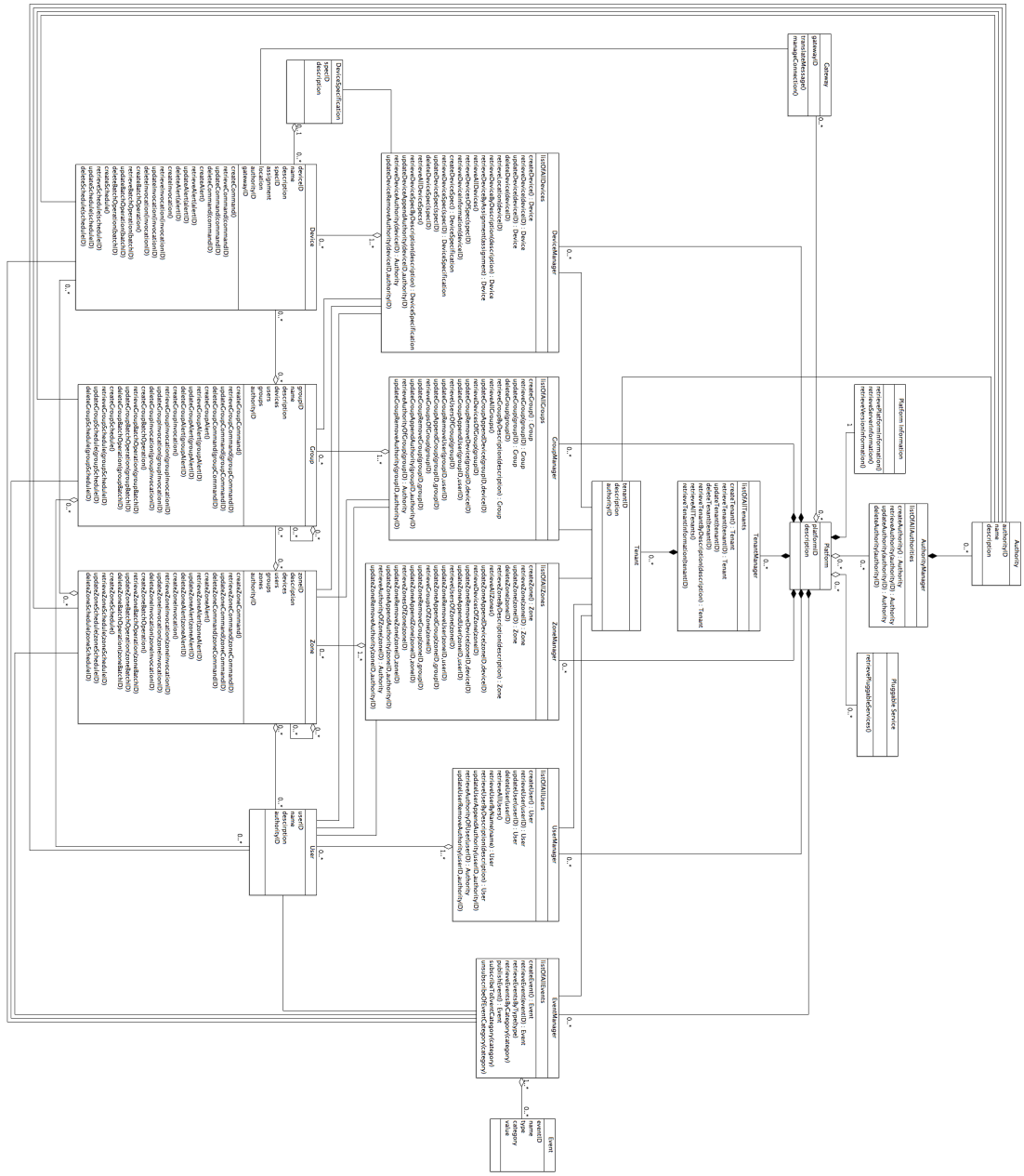


Figure 5.1: Abstract Class Model

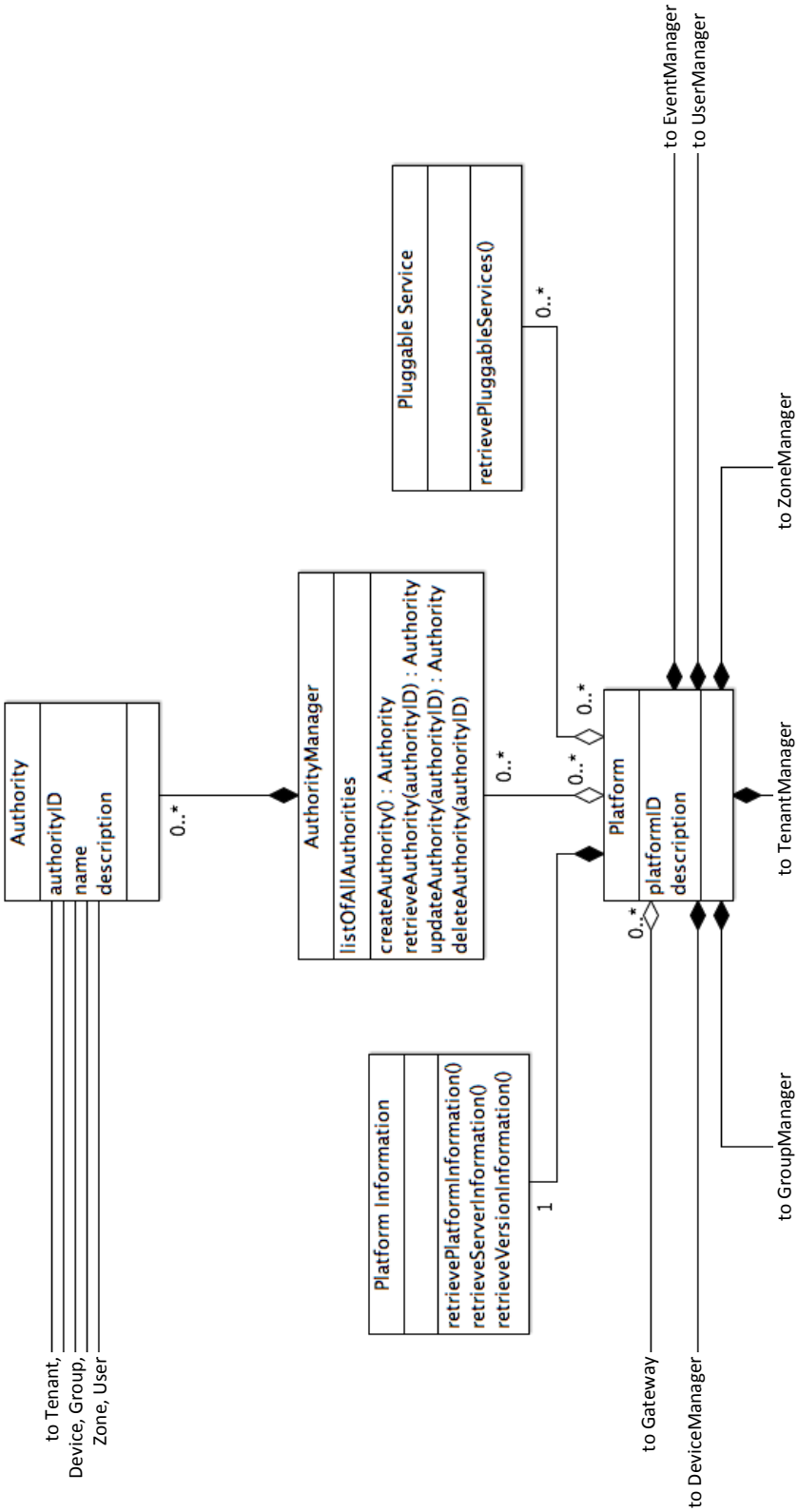


Figure 5.2: Abstract Class Model Excerpt Platform

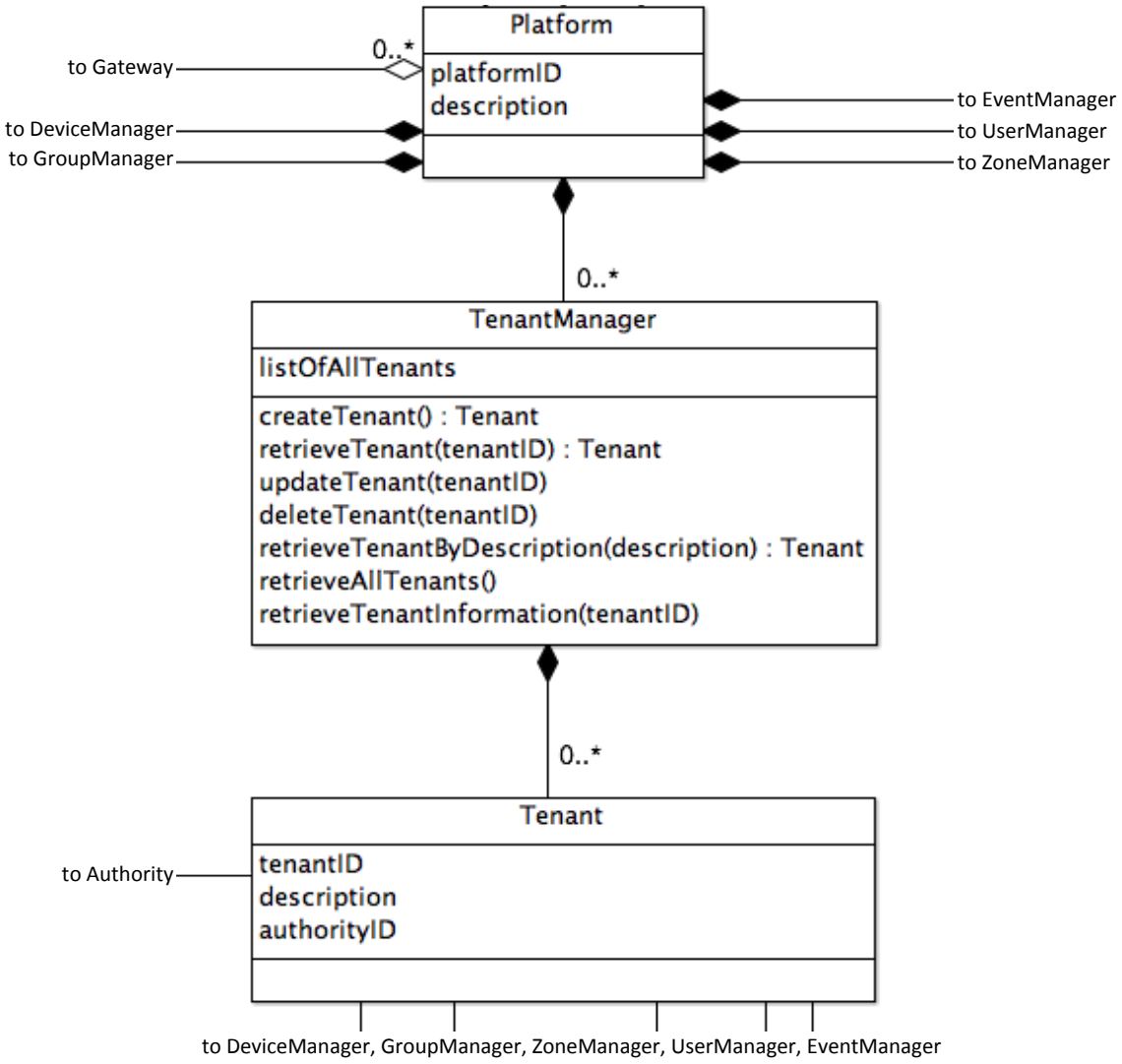


Figure 5.3: Abstract Class Model Excerpt Tenant

Figure 5.3 shows, that the Platform is southbound via an composition connected to the TenantManager class, which is further connected to the Tenant class. A Tenant describes an enclosed container within a platform, to enable that multiple instances can run concurrently, but absolutely isolated. Since the possibility to provide separate containers within one platform seemed to be common and reasonable, this approach is adopted.

The TenantManager provides operations to manage all tenants of a platform. It cannot exist without the Platform, and one Platform instance can have zero to multiple instances of the TenantManager assigned. The TenantManager contains a list of all Tenants, and comprises operations to create, retrieve, update, and delete a single Tenant instance, as

well as an operation to retrieve all Tenants, and to retrieve information about a Tenant. The Tenant class is connected to the TenantManager with a composition, so the Tenant cannot exist without the TenantManager, and one instance of the TenantManager can be connected to zero to multiple instances of a Tenant. It contains the attributes tenantID, which is the unique identifier, a description of the Tenant and an authorityID, which points to the Authority class, defining which Authority is assigned to the Tenant.

Figure 5.4 shows the excerpt of the DeviceManager, the Device, and the DeviceSpecification class. The Device class represents physical devices, connected to the platform. The DeviceManager provides operations to manage those devices, and the DeviceSpecification class enables a classification of the devices. The DeviceManager class is northbound connected with a composition to the Platform, i.e., the DeviceManager cannot exist without the Platform, and a Platform instance can be associated to zero to multiple instances of the DeviceManager. Additionally the DeviceManager is connected to the Tenant, which means that the DeviceManager, it's attributes, and operations can be accessed via the Tenant. The DeviceManager contains a list of all Devices, and provides operations to create, retrieve, update, and delete a Device instance, and to retrieve information about a device instance. Furthermore it comprises operations to create, retrieve, update, and delete an instance of the DeviceSpecification class, and to append, retrieve appendend, and remove an Authority to/of an instance of a Device. Following this, the DeviceManager is connected to the DeviceSpecification class, which contains the attributes specID, the unique identifier of the specification, and a description. Obviously the DeviceManager is connected to the Device class through an aggregation, i.e an instance of the Device class belongs to one to many instances of the DeviceManager class, and a DeviceManager can have zero to multiple Device instances associated. Furthermore the DeviceManager is connected to the Group, Zone, and User class.

The Device class is connected to the DeviceSpecification class, so that one instance of a Device can belong to zero or one instances of the DeviceSpecification, and an instance of the DeviceSpecification can be associated to zero to multiple instances of a Device. The Device class contains the attributes deviceID, which is the unique identifier of an instance of a device, and the attributes name, description, and specID, which points to the associated DeviceSpecification. Furhtermore a Device has an assignment, a location, an authorityID, which points to the associated Authority, and an optional gatewayID, which points to the Gateway the device uses to communicate with the platform. If the device communicates directly with the platform, the gatewayID is empty. Following this, the Device is connected to the Authority, and the Gateway class. The Device class contains operations to create, retrieve, update, and delete Commands, Alerts, Invocations, Batch Operations, and Schedules. Accordingly the Device class is connected to the EventManager class, since events can trigger those operations. Additionally the Device class is connected through aggregations to the Group, and Zone classes, to enable that a Device can be contained in zero or multiple Groups and/or Zones.

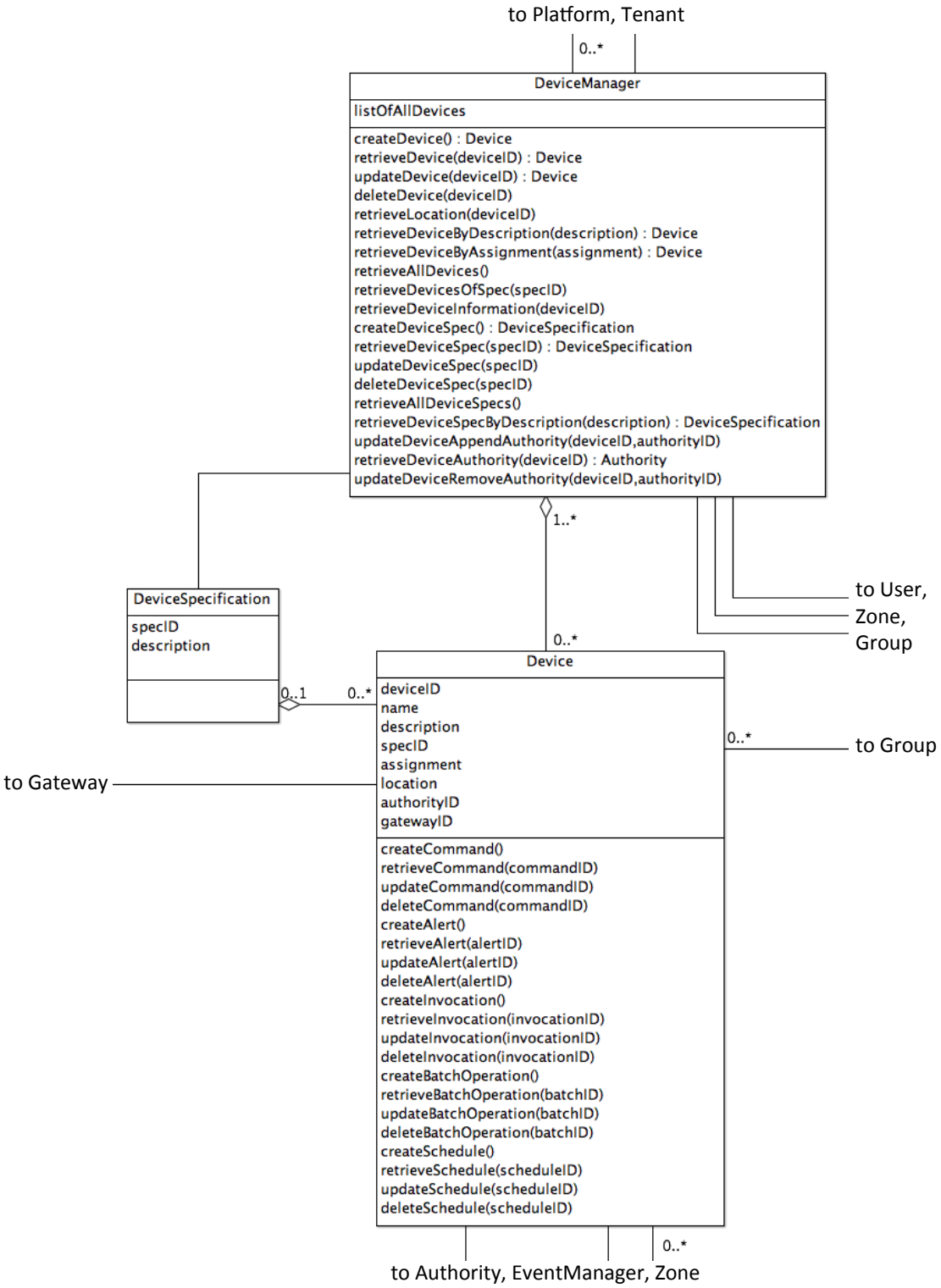


Figure 5.4: Abstract Class Model Excerpt Device

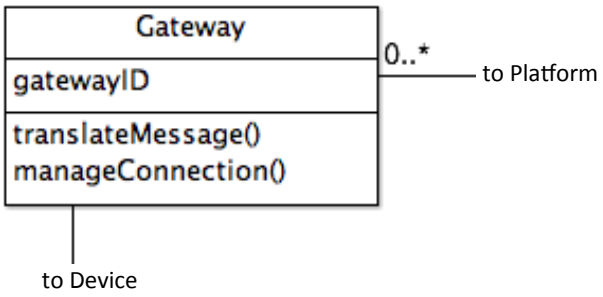


Figure 5.5: Abstract Class Model Excerpt Gateway

As already mentioned, the Device class is connected to the Gateway class, if the Device requires the Gateway to enable the communication of devices with the platform. Accordingly the Gateway is connected to the Platform with an aggregation, i.e., an instance of the Gateway class belongs to zero to multiple Platform instances, and a Platform instance can be associated with zero to multiple instances of a Gateway. The Gateway class is shown within Figure 5.5. It contains the gatewayID, which is the unique identifier of the instance of the Gateway class. The Gateway comprises operations to translate a message, and to manage the connection of the devices.

Figure 5.6 depicts the GroupManager, and Group class, which enable the aggregation of devices into logical groups, where a group can likewise contain groups. For instance, the heaters within a room can be contained within a group. The GroupManager is northbound connected with a composition to the Platform, so a Platform instance can be associated to zero to multiple GroupManager instances, and the GroupManager cannot exist without the Platform. The GroupManager is also connected to the Tenant, to enable that the operations of the GroupManager can be accessed through the Tenant. The GroupManager contains a list of all Groups, and comprises operations to create, retrieve, update, and delete an instance of a Group, as well as operations to append, retrieve appendend, or remove a Device, User, Group, or Authority of a Group. Accordingly it is connected to the Group class, where one GroupManager can be associated to zero to multiple Group instances, and one Group instance belongs to one to multiple GroupManager instances.

Furthermore the Group class is northbound connected to the DeviceManager, to enable that the Group class can access the operations of the DeviceManager. The Group class contains the attributes groupID, which is the unique identifier of the instance of a Group, name, and description of the instance, as well as a list of each, Devices, Users, and Groups the instance of the Group contains. Additionally it contians an authorityID, which points to the Authority class, hence the Group class is connected to the Authority class. A Group can be aggregated into zero to multiple Groups, following it is connected to itself

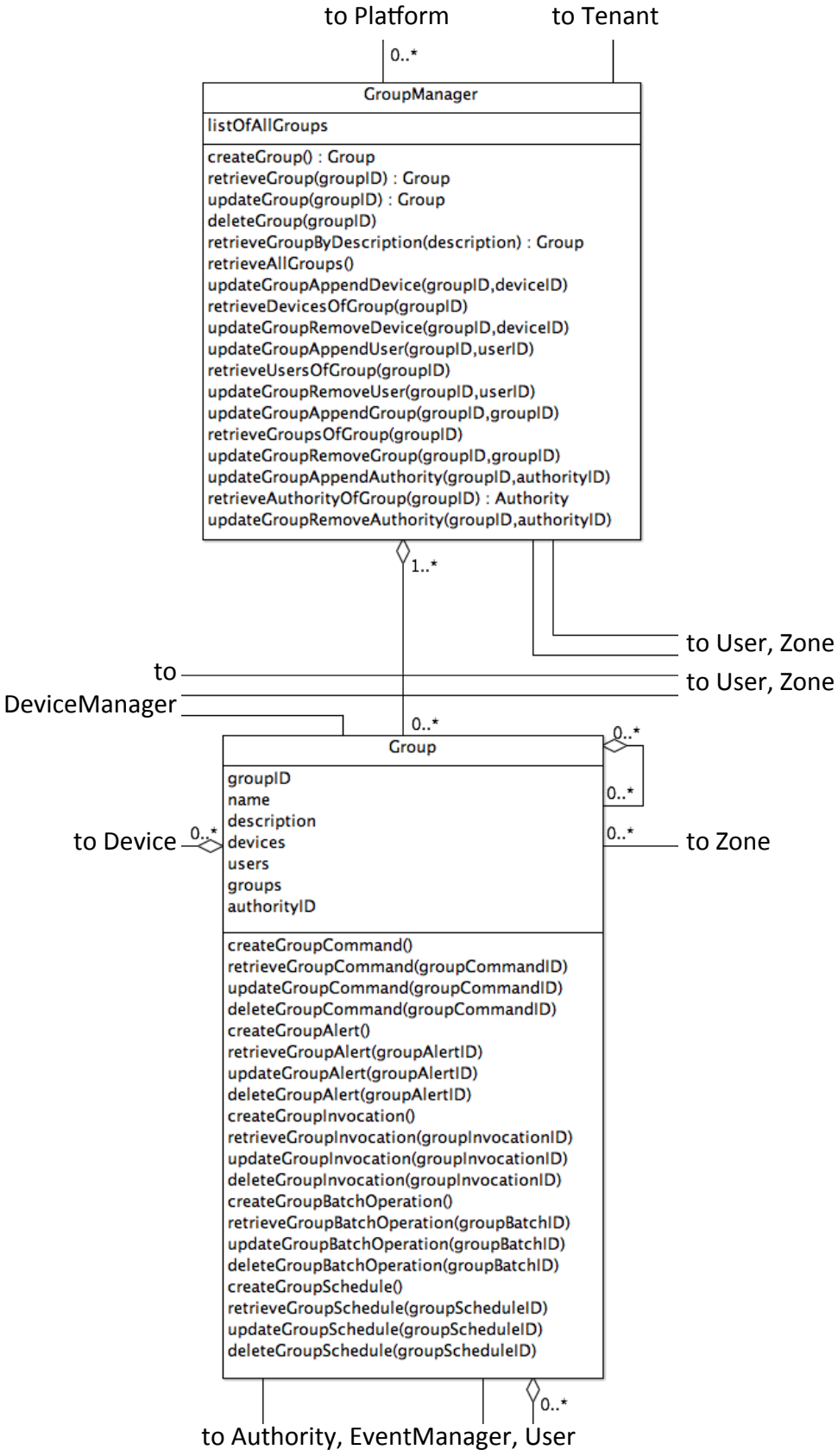


Figure 5.6: Abstract Class Model Excerpt Group

through an aggregation. The class provides similar to the Device class operations to create, retrieve, update, and delete Group Commands, Group Alerts, Group Invocations, Group Batch Operations, and Group Schedules. Therefor it is likewise connected to the EventManager class. Beside the already described aggregation to the Device class, the Group class is furthermore connected through an aggregation to the Zone and User class, i.e., one instance of a Group can belong to zero to multiple Zone instances, and it can comprise zero to multiple User instances.

Figure 5.7 depicts the excerpt of the ZoneManager, and the Zone class, which enable to divide Devices, Users, and Groups into geographical or logical Zones. Likewise within the Groups, a Zone can contain Zones. A Zone can, e.g., be a room within a building, or the ground floor of a building, where the room is part of the ground floor. The ZoneManager class is northbound connected to the Platform, where the connection is a composition, i.e., the ZoneManager cannot exist without the Platform, and an instance of the Platform can be associated to zero to multiple instances of the ZoneManager. Likewise the DeviceManager and the GroupManager, the ZoneManager is connected to the Tenant, which means that the operations of the ZoneManager can be accessed via the Tenant. Containing a list of all Zones, the ZoneManager provides operations to create, retrieve, update, and delete a Zone, as well as operations to append, retrieve appendend, or remove Devices, Users, Groups, Zones, and an Authority to/of a Zone. Southbound the ZoneManager is connected to the Zone class via an aggregation, which means that one instance of the ZoneManager can be associated to zero to multiple instances of a Zone, and one Zone belongs to one to multiple ZoneManagers.

Beside the connection to the ZoneManager, the Zone class is northbound connected to the GroupManager, and the DeviceManager, to achieve, that the Zone can access their operations. The Zone class contains the attributes zoneID, which is the unique identifier of the instance of the Zone, a description, and a list of each, Devices, Users, Groups, and Zones the instance of a Zone contains, as well as an authorityID, which points to the Authority class. Following the Zone class is connected to the Authority class. Like the Device and Group class the Zone class provides operations to create, retrieve, update, and delete Zone Commands, Zone Alerts, Zone Invocations, Zone Batch Operations, and Zone Schedules. Hence the Zone class is connected to the EventManager, since the events are used to trigger those operations. The Zone class is connected through an aggregation to the Device class, where one instance of a Zone contains zero to multiple instances of a Device, and one instance of a Device is associated to zero to multiple Zone instances. Furthermore the Zone class is connected to the User class via an aggregation, which means that one instance of a Zone can be associated with zero to multiple instances of a User, and one instance of a User belongs to zero to multiple Zones. Additionally the Zone class can aggregate Groups, where one Zone instance contains zero to multiple Group instances, and one Group instance is associated to zero to multiple Zone instances. Again the Zone class is connected to itself, so an instance of

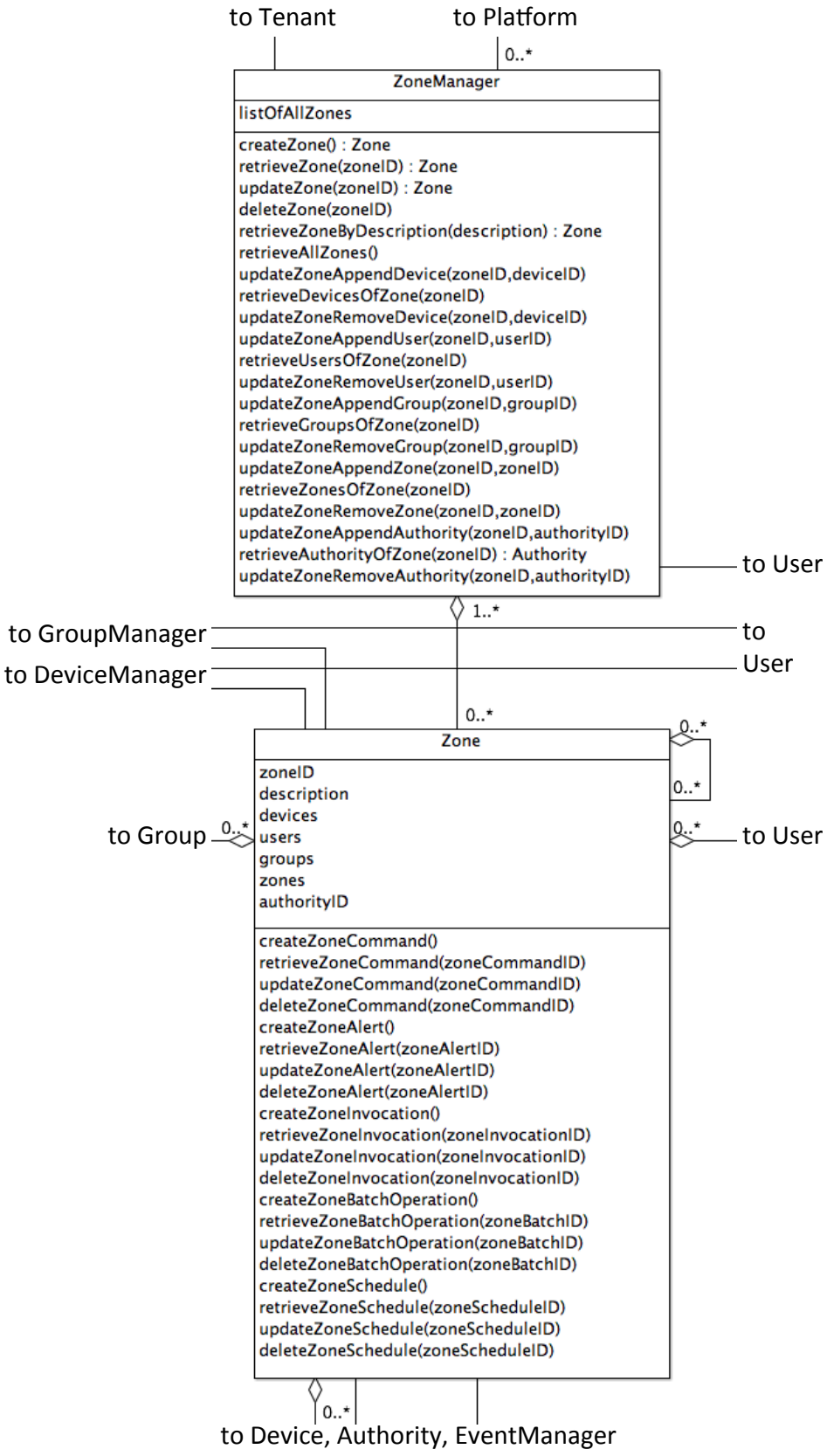


Figure 5.7: Abstract Class Model Excerpt Zone

a Zone can belong to zero to multiple Zone instances, i.e., Zones can be aggregated into Zones.

Figure 5.8 shows the excerpt of the UserManager, and the User classes. They provide the functionality to involve various users within the platform. The UserManager class is northbound connected to the Platform via a composition, so that a UserManager cannot exist without the Platform, and a Platform can be connected to zero to multiple instances of a UserManager. Likewise the other Managers, it is connected to the Tenant, so the operations of the UserManager can be accessed through the Tenant. The UserManager contains a list of all Users, and provides operations to create, retrieve, update, and delete instances of a User, and to append, retrieve appended, and remove an Authority to/of a User. Accordingly the UserManager is connected to the User class through an aggregation. One instance of the UserManager can be associated with zero to multiple instances of a User, and one instance of a User belongs to one to multiple instances of a UserManager.

The User class contains the attributes userID, which is the unique identifier of an instance of the User, name, description, and authorityID, which points to the Authority class. Hence the User class is connected to the Authority class. Furthermore the User class is connected to the EventManager, the ZoneManager, the GroupManager, and the DeviceManager, so the operations of the Manager classes can be accessed through the User class.

Figure 5.9 depicts the excerpt of the EventManager, and the Event classes. They provide all necessary operations regarding the handling of occurring events, and hence provide the basis for the functionality of a CPS. The EventManager class is northbound connected to the Platform and the Tenant. The connection to the Platform is a composition, which means that the EventManager cannot exist without the Platform, and one instance of the Platform can be associated to zero to multiple instances of the EventManager. The connection to the Tenant means, that the operations of the EventManager can be accessed through the Tenant. Southbound the EventManager is connected to the User, the Zone, the Group, and the Device class, to enable that those classes can access the operations of the EventManager class. It contains a list of all Events, and comprises operations to create, retrieve, and publish an instance of an Event, as well as an operation to subscribe, and one to unsubscribe to/of an Event Category. The EventManager is connected to the Event class through an aggregation, so that one instance of an EventManager can be associated to zero to multiple instances of an Event, and one instance of an Event belongs to one to multiple instances of an EventManager.

The Event class contains the attributes eventID, which is the unique identifier of an instance of the Event class, the name, the type, the category and the value of the instance. The type and the category attributes enable to allocate an event to a context within it

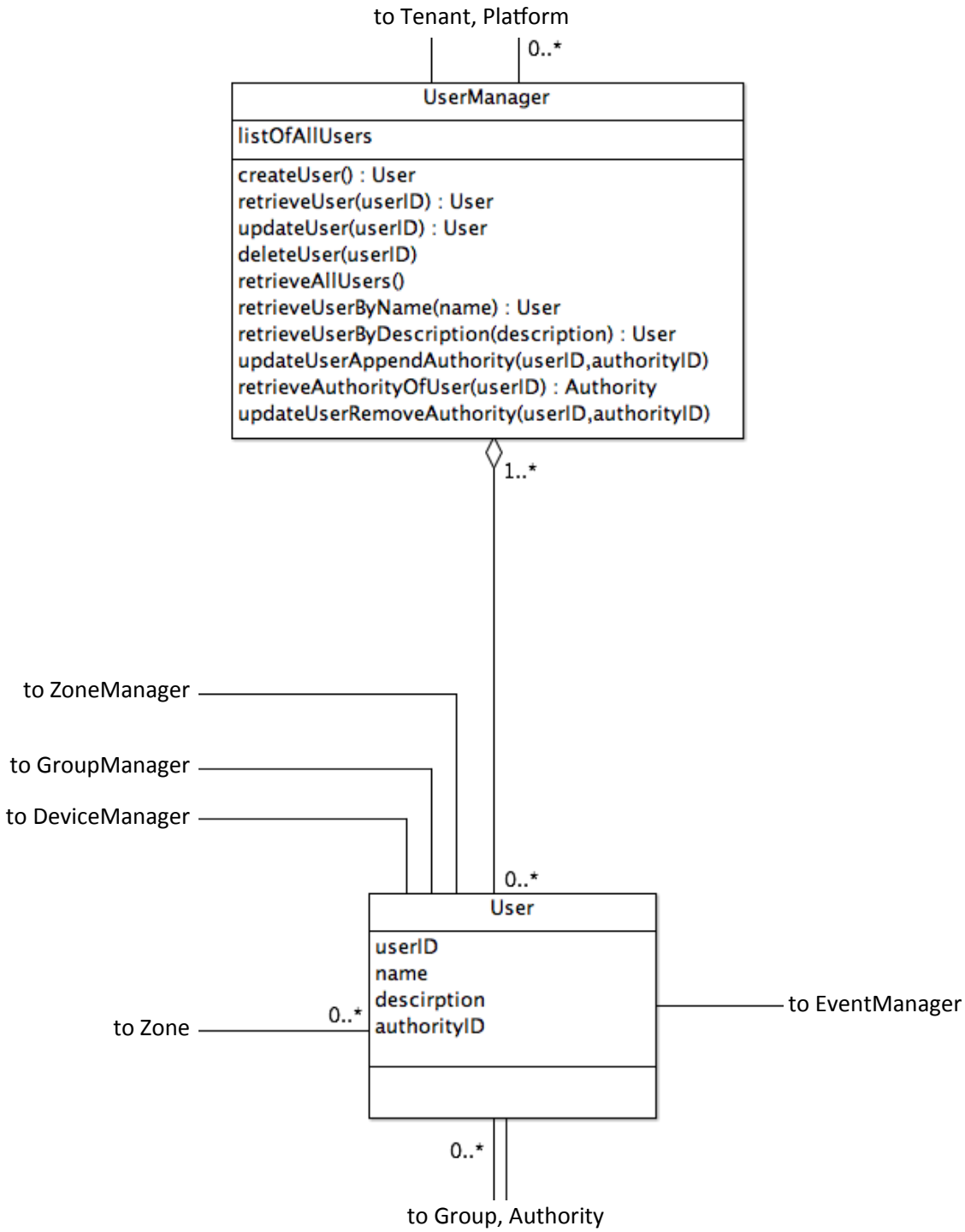


Figure 5.8: Abstract Class Model Excerpt User

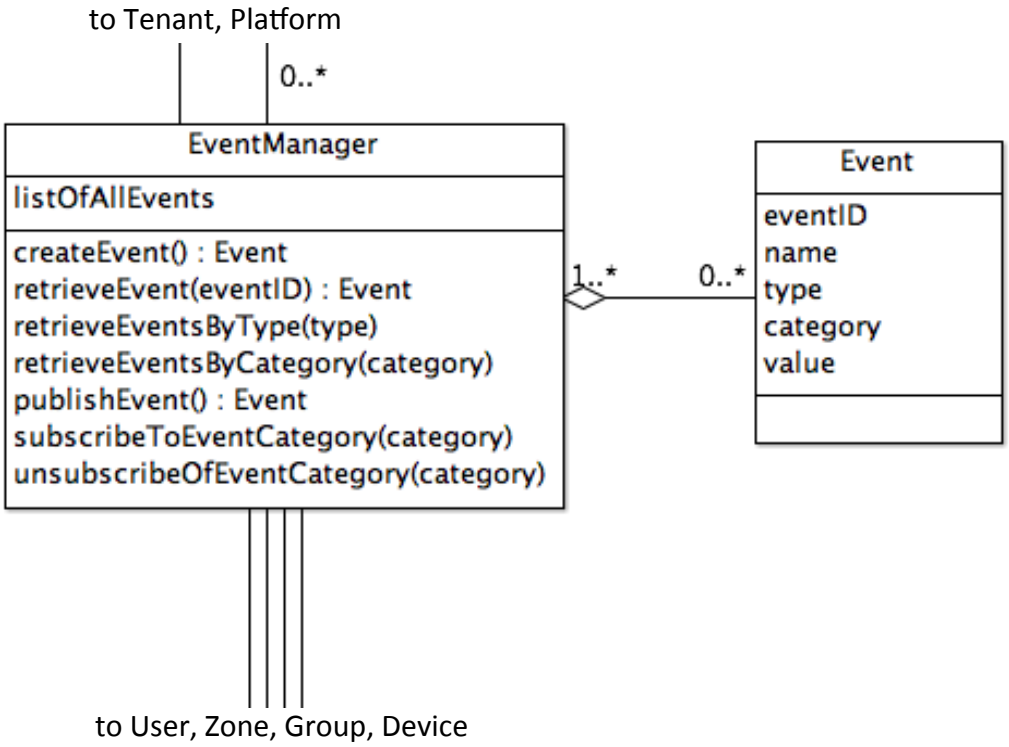


Figure 5.9: Abstract Class Model Excerpt Event

is occurred, i.e., to append context information to an event. Both attributes are used within the EventManager to retrieve Events.

6 Validation of the Abstract Class Model

This Chapter deals with the validation of the abstract class model, designed in Chapter 5.3. As already mentioned within the Chapter before, the documentation of the REST API's of OpenMTC is not available, hence it cannot be considered within the validation.

6.1 Validation of the Operations of the Classes

Within the following correlation matrices a check mark means, that the same functionality is existent within the platform, a bracketed check mark labels comparable existent functionality. A check mark star indicates, that presumably the same or at least a comparable functionality is existent, which is not represented within the documentation of the platform. A dash means, that the platform does not comprise any comparable functionality.

Correlation Matrix 6.1 shows the validation of the Device class. Both FIWARE and AWS IoT comprise the same functionality, the Azure IoT Hub and SmartThings provide at least comparable functionality. SiteWhere only encompasses some of the same functionality, therefor it is presumable, that it also provides the remaining methods, which are not documented. Likewise the Watson IoT Platform presumably comprises the functionality of the Device class, since they are essential for the communication of the platform with a device.

Within Table 6.2 the correlation matrix of the DeviceManager class is represented. FIWARE and the IBM Watson IoT Platform both provide almost the same functionality for all methods, the remaining are comparable. SmartThings likewise encompasses at least comparable functionality, the three methods regarding the authority of a device are presumably existent, but not mentioned within the documentation. AWS IoT comprises for most of the methods at least a comparable functionality, except for the location and device information retrieval, which is not covered. SiteWhere also provides the same or comparable functionality, beside the methods for the device information retrieval, and the ones regarding the authority, which are not existent. Microsoft's Azure IoT Hub encompasses the same and comparable functionality for half of the methods of the

Abstraction Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
Device						
createCommand()	✔	✔*	✔	✔	✔	✔*
retrieveCommand(commandID)	✔	✔	✔	✔	(✔)	✔*
updateCommand(commandID)	✔	✔	(✔)	✔	(✔)	✔*
deleteCommand(commandID)	✔	✔	(✔)	✔	✔	✔*
createAlert()	✔	✔	(✔)	✔	(✔)	✔*
retrieveAlert(alertID)	✔	✔*	(✔)	✔	(✔)	✔*
updateAlert(alertID)	✔	✔*	(✔)	✔	(✔)	✔*
deleteAlert(alertID)	✔	✔*	(✔)	✔	(✔)	✔*
createInvocation()	✔	✔*	(✔)	✔	✔	✔*
retrieveInvocation(invocationID)	✔	✔	(✔)	✔	✔	✔*
updateInvocation(invocationID)	✔	✔*	(✔)	✔	✔	✔*
deleteInvocation(invocationID)	✔	✔*	(✔)	✔	✔	✔*
createBatchOperation()	✔	✔	(✔)	✔	✔	✔*
retrieveBatchOperation(batchID)	✔	✔	(✔)	✔	✔	✔*
updateBatchOperation(batchID)	✔	✔*	(✔)	✔	✔	✔*
deleteBatchOperation(batchID)	✔	✔*	(✔)	✔	✔	✔*
createSchedule()	✔	✔	✔	✔	✔	✔*
retrieveSchedule(scheduleID)	✔	✔	(✔)	✔	✔	✔*
updateSchedule(scheduleID)	✔	✔	(✔)	✔	✔	✔*
deleteSchedule(scheduleID)	✔	✔	✔	✔	✔	✔*

Table 6.1: Correlation Matrix to Validate the Abstract Class Model Device

DeviceManager. Not covered are the methods regarding the Device Specification, since the Azure IoT Hub does not use anything comparable.

Within the validation of the Group class, shown in Table 6.3, AWS IoT comprises the same functionality for every method. FIWARE, SmartThings, and the Azure IoT Hub provide comparable functionality for all of the class methods. Likewise SiteWhere encompasses comparable functionality for most of the methods, presumably it also covers the remaining ones, since they are in conjunction with the covered methods. The IBM Watson IoT Platform does not document any comparable functionality, like within the Device class. Since it supports the grouping of devices, and users, the methods are presumably covered.

Correlation Matrix 6.4 shows the validation of the GroupManager class. Besides AWS IoT, which provides comparable functionality for all of the methods, FIWARE, the Azure IoT Hub, and the Watson IoT Platform all comprise at least comparable functionality. SmartThings provides at least comparable functionality, except for the methods regarding the authority, which are presumably covered, and the ones regarding the users, which are not encompassed, since SiteWhere does not support to have multiple users involved within the platform. Likewise SiteWhere only encompasses at least comparable functionality, beside the ones dealing with the authority. They are not covered by SiteWhere, since the platform does only support to append authorities to users.

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
DeviceManager						
createDevice(): Device	✔	✔	✔	✔	✔	✔
retrieveDevice(deviceID): Device	✔	✔	✔	✔	✔	✔
updateDevice(deviceID): Device	✔	✔	(✔)	✔	✔	✔
deleteDevice(deviceID)	✔	✔	✔	✔	✔	✔
retrieveLocation(deviceID)	(✔)	(✔)	✔	–	–	✔
retrieveDeviceByDescription (description): Device	✔	✔	(✔)	✔	✔	✔
retrieveDeviceByAssignment (assignment): Device	✔	✔	(✔)	✔	✔	✔
retrieveAllDevices()	✔	✔	(✔)	✔	✔	✔
retrieveDevicesOfSpec(specID)	✔	✔	(✔)	✔	(✔)	✔
retrieveDeviceInformation (deviceID)	✔	–	(✔)	–	✔	✔
createDeviceSpec(): DeviceSpecification	✔	✔	✔	(✔)	–	✔
retrieveDeviceSpec(specID): DeviceSpecification	✔	✔	(✔)	(✔)	–	✔
updateDeviceSpec(specID)	✔	✔	(✔)	(✔)	–	✔
deleteDeviceSpec(specID)	✔	✔	(✔)	(✔)	–	✔
retrieveAllDeviceSpecs()	✔	(✔)	(✔)	(✔)	–	(✔)
retrieveDeviceSpecByDescription (description): DeviceSpecification	✔	✔	(✔)	(✔)	–	✔
updateDeviceAppendAuthority (deviceID, authorityID)	✔	–	✔*	(✔)	(✔)	(✔)
retrieveDeviceAuthority (deviceID): Authority	✔	–	✔*	(✔)	(✔)	(✔)
updateDeviceRemoveAuthority (deviceID, authorityID)	✔	–	✔*	(✔)	(✔)	(✔)

Table 6.2: Correlation Matrix to Validate the Abstract Class Model DeviceManager

Like within the Group class, Table 6.5 shows, that AWS IoT provides the same functionality like the Zone class of the abstract class model. FIWARE, SmartThings, and SiteWhere comprise at least comparable functionality. Regarding the SiteWhere platform, some of the methods are not documented, but presumably existent. The Azure IoT Hub does not support combining devices into zones, hence it does not provide any comparable functionality. Since the IBM Watson IoT Platform does support the approach of zones, presumably the methods are covered.

The validation of the ZoneManager is shown in Correlation Matrix 6.6. FIWARE, AWS IoT, and the Watson IoT Platform comprise comparable functionality for all of the methods. Again SmartThings provides comparable functionality, besides the methods regarding the authority and the user methods. The authority methods are presumably covered, since the authority has to be managed by a platform. The user methods are not covered at all, as SmartThings does not support to involve multiple users within the platform. Likewise within the validation of the GroupManager, SiteWhere encompasses

Abstraction Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
Group						
createGroupCommand()	(✔)	✔*	(✔)	✔	(✔)	✔*
retrieveGroupCommand (groupCommandID)	(✔)	(✔)	(✔)	✔	(✔)	✔*
updateGroupCommand (groupCommandID)	(✔)	(✔)	(✔)	✔	(✔)	✔*
deleteGroupCommand (groupCommandID)	(✔)	(✔)	(✔)	✔	(✔)	✔*
createGroupAlert()	(✔)	(✔)	(✔)	✔	(✔)	✔*
retrieveGroupAlert(groupAlertID)	(✔)	✔*	(✔)	✔	(✔)	✔*
updateGroupAlert(groupAlertID)	(✔)	✔*	(✔)	✔	(✔)	✔*
deleteGroupAlert(groupAlertID)	(✔)	✔*	(✔)	✔	(✔)	✔*
createGroupInvocation()	(✔)	✔*	(✔)	✔	(✔)	✔*
retrieveGroupInvocation (groupInvocationID)	(✔)	(✔)	(✔)	✔	(✔)	✔*
updateGroupInvocation (groupInvocationID)	(✔)	✔*	(✔)	✔	(✔)	✔*
deleteGroupInvocation (groupInvocationID)	(✔)	✔*	(✔)	✔	(✔)	✔*
createGroupBatchOperation()	(✔)	(✔)	(✔)	✔	(✔)	✔*
retrieveGroupBatchOperation (groupBatchID)	(✔)	(✔)	(✔)	✔	(✔)	✔*
updateGroupBatchOperation (groupBatchID)	(✔)	✔*	(✔)	✔	(✔)	✔*
deleteGroupBatchOperation (groupBatchID)	(✔)	✔*	(✔)	✔	(✔)	✔*
createGroupSchedule()	(✔)	(✔)	(✔)	✔	(✔)	✔*
retrieveGroupSchedule (groupScheduleID)	(✔)	(✔)	(✔)	✔	(✔)	✔*
updateGroupSchedule (groupScheduleID)	(✔)	(✔)	(✔)	✔	(✔)	✔*
deleteGroupSchedule (groupScheduleID)	(✔)	(✔)	(✔)	✔	(✔)	✔*

Table 6.3: Correlation Matrix to Validate the Abstract Class Model Group

at least comparable functionality of the ZoneManager methods, except for the ones regarding the authority, which are not covered.

Correlation Matrix 6.7 shows the validation of the UserManager class of the abstract class model. SmartThings does not comprise any comparable functionality at all, since it does not support multiple users involved within the system. FIWARE, SiteWhere, AWS IoT and the Azure IoT Hub all provide at least comparable functionality. The IBM Watson IoT Platform supports only one user per instance of the platform, i.e., an admin to setup and manage the system. Hence it is possible to retrieve the instance of the user, and the authority assigned, but all other methods are not covered.

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
GroupManager						
createGroup(): Group	✔	✔	✔	(✔)	✔	✔
retrieveGroup(groupID): Group	✔	✔	✔	(✔)	✔	✔
updateGroup(groupID): Group	✔	✔	(✔)	(✔)	✔	✔
deleteGroup(groupID)	✔	✔	(✔)	(✔)	✔	✔
retrieveGroupByDescription (description): Group	✔	✔	(✔)	(✔)	✔	✔
retrieveAllGroups()	✔	✔	(✔)	(✔)	✔	✔
updateGroupAppendDevice (groupID, deviceID)	✔	✔	✔	(✔)	✔	✔
retrieveDevicesOfGroup(groupID)	✔	✔	✔	(✔)	✔	✔
udateGroupRemoveDevice (groupID, deviceID)	✔	✔	✔	(✔)	✔	✔
updateGroupAppendUser (groupID, userID)	✔	✔	–	(✔)	✔	✔
retrieveUsersOfGroup(groupID)	✔	✔	–	(✔)	✔	✔
updateGroupRemoveUser (groupID, userID)	✔	✔	–	(✔)	✔	✔
updateGroupAppendGroup (groupID, groupID)	(✔)	(✔)	(✔)	(✔)	(✔)	(✔)
retrieveGroupsOfGroup(groupID)	(✔)	(✔)	(✔)	(✔)	(✔)	(✔)
updateGroupRemoveGroup (groupID, groupID)	(✔)	(✔)	(✔)	(✔)	(✔)	(✔)
updateGroupAppendAuthority (groupID, authorityID)	(✔)	–	✔*	(✔)	(✔)	(✔)
retrieveAuthorityOfGroup (groupID): Authority	(✔)	–	✔*	(✔)	(✔)	(✔)
updateGroupRemoveAuthority (groupID, authorityID)	(✔)	–	✔*	(✔)	(✔)	(✔)

Table 6.4: Correlation Matrix to Validate the Abstract Class Model GroupManager

The validation of the EventManager is shown within Table 6.8. FIWARE, SmartThings, and the Azure IoT Hub provide at least comparable functionality for every operation of the EventManager. The IBM Watson IoT Platform comprises comparable functionality, except for the subscribe to and unsubscribe of an event operation. Since the platform supports to publish events, it is presumable that it also covers the subscribe and unsubscribe operation. The documentation of AWS IoT gives no information about any operations regarding events. As they are essential for the communication, and an accurate usage of a CPS platform, they are presumably covered. SiteWhere provides comparable functionality, except for the subscribe to, and unsubscribe of an event operation, which are not covered.

Table 6.9 shows the correlation matrix of the TenantManager class of the abstract class model. FIWARE, SiteWhere, and the Azure IoT Hub all provide the same functionality, except for the operation to retrieve all tenants, where those platforms provide comparable functionality. SmartThings comprises comparable functionality for all of the

Abstraction Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
Zone						
createZoneCommand()	(✔)	✔*	(✔)	✔	–	✔*
retrieveZoneCommand (zoneCommandID)	(✔)	(✔)	(✔)	✔	–	✔*
updateZoneCommand (zoneCommandID)	(✔)	(✔)	(✔)	✔	–	✔*
deleteZoneCommand (zoneCommandID)	(✔)	(✔)	(✔)	✔	–	✔*
createZoneAlert()	✔	(✔)	(✔)	✔	–	✔*
retrieveZoneAlert(zoneAlertID)	✔	✔*	(✔)	✔	–	✔*
updateZoneAlert(zoneAlertID)	✔	✔*	(✔)	✔	–	✔*
deleteZoneAlert(zoneAlertID)	✔	✔*	(✔)	✔	–	✔*
createZoneInvocation()	✔	✔*	(✔)	✔	–	✔*
retrieveZoneInvocation (zoneInvocationID)	✔	(✔)	(✔)	✔	–	✔*
updateZoneInvocation (zoneInvocationID)	✔	✔*	(✔)	✔	–	✔*
deleteZoneInvocation (zoneInvocationID)	✔	✔*	(✔)	✔	–	✔*
createZoneBatchOperation()	(✔)	(✔)	(✔)	✔	–	✔*
retrieveZoneBatchOperation (zoneBatchID)	(✔)	(✔)	(✔)	✔	–	✔*
updateZoneBatchOperation (zoneBatchID)	(✔)	✔*	(✔)	✔	–	✔*
deleteZoneBatchOperation (zoneBatchID)	(✔)	✔*	(✔)	✔	–	✔*
createZoneSchedule()	(✔)	(✔)	(✔)	✔	–	✔*
retrieveZoneSchedule (zoneScheduleID)	(✔)	(✔)	(✔)	✔	–	✔*
updateZoneSchedule (zoneScheduleID)	(✔)	(✔)	(✔)	✔	–	✔*
deleteZoneSchedule (zoneScheduleID)	(✔)	(✔)	(✔)	✔	–	✔*

Table 6.5: Correlation Matrix to Validate the Abstract Class Model Zone

operations of the TenantManager. Within AWS IoT the approach of tenants is existent, but not visible for the clients, since each tenant is explicitly assigned to one client. Hence AWS IoT does not cover any of the operations of the TenantManager class. Likewise the IBM Watson IoT Platform uses the approach of tenants, whereby the clients can still retrieve the tenant itself, and information about it, but the remaining operations are not covered.

Within Correlation Matrix 6.10 the validation of the AuthorityManager is represented. FIWARE and the Azure IoT Hub both provide the same functionality for all operations of this class. AWS IoT provides at least comparable functionality. The IBM Watson IoT Platform encompasses comparable functionality, except for the deletion of authorities, which is not supported, since the authority is assigned at the point of the creation of a

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
ZoneManager						
createZone(): Zone	✔	✔	✔	(✔)	–	✔
retrieveZone(zoneID): Zone	✔	✔	✔	(✔)	–	✔
upateZone(zoneID): Zone	✔	✔	(✔)	(✔)	–	✔
deleteZone(zoneID)	✔	✔	(✔)	(✔)	–	(✔)
retrieveZoneByDescription (description): Zone	✔	✔	(✔)	(✔)	–	(✔)
retrieveAllZones()	(✔)	(✔)	(✔)	(✔)	–	(✔)
updateZoneAppendDevice (zoneID, deviceID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
retrieveDevicesOfZone(zoneID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
udateZoneRemoveDevice (zoneID, deviceID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
updateZoneAppendUser (zoneID, userID)	(✔)	(✔)	–	(✔)	–	(✔)
retrieveUsersOfZone(zoneID)	(✔)	(✔)	–	(✔)	–	(✔)
updateZoneRemoveUser (zoneID, userID)	(✔)	(✔)	–	(✔)	–	(✔)
updateZoneAppendGroup (zoneID, groupID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
retrieveGroupsOfZone(zoneID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
updateZoneRemoveGroup (zoneID, groupID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
updateZoneAppendZone (zoneID, zoneID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
retrieveZonesOfZone(zoneID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
updateZoneRemoveZone (zoneID, zoneID)	(✔)	(✔)	(✔)	(✔)	–	(✔)
updateZoneAppendAuthority (zoneID, authorityID)	(✔)	–	✔*	(✔)	–	(✔)
retrieveAuthorityOfZone (zoneID): Authority	(✔)	–	✔*	(✔)	–	(✔)
updateZoneRemoveAuthority (zoneID, authorityID)	(✔)	–	✔*	(✔)	–	(✔)

Table 6.6: Correlation Matrix to Validate the Abstract Class Model ZoneManager

device. SiteWhere comprises the creation, and retrieval of authorities, while it does not cover comparable operations to update or delete an authority. The documentation of SmartThings gives no information about the authority handling, therefor the operations are presumed to be existent.

Table 6.11 shows the correlation matrix of the classes Pluggable Service, Platform Information, and Gateway. FIWARE, SiteWhere, the Azure IoT Hub, and the Watson IoT Platform all comprise a functionality to retrieve pluggable services. Since SmartThings, and AWS IoT provide the connection of further applications, the operation to retrieve pluggable services is presumably covered. All platforms, except AWS IoT, provide at least comparable functionality to retrieve information about the platform, server, and

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
UserManager						
createUser(): User	✔	✔	–	(✔)	✔	–
retrieveUser(userID): User	(✔)	(✔)	–	(✔)	✔	(✔)
updateUser(userID): User	✔	✔	–	(✔)	✔	–
deleteUser(userID)	✔	✔	–	(✔)	✔	–
retrieveAllUsers()	(✔)	(✔)	–	(✔)	✔	–
retrieveUserByName(name): User	✔	✔	–	(✔)	✔	✔
retrieveUserByDescription (description): User	✔	✔	–	(✔)	✔	–
updateUserAppendAuthority (userID, authorityID)	✔	(✔)	–	(✔)	(✔)	–
retrieveAuthorityOfUser (userID): Authority	✔	✔	–	(✔)	(✔)	✔
updateUserRemoveAuthority (userID, authorityID)	✔	(✔)	–	(✔)	(✔)	–

Table 6.7: Correlation Matrix to Validate the Abstract Class Model UserManager

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
EventManager						
createEvent(): Event	(✔)	(✔)	✔	✔*	(✔)	(✔)
retrieveEvent(eventID): Event	✔	✔	✔	✔*	(✔)	✔
retrieveEventsByType(type)	✔	(✔)	(✔)	✔*	(✔)	(✔)
retrieveEventsByCategory (category)	✔	(✔)	(✔)	✔*	(✔)	✔
publishEvent(): Event	✔	✔	✔	✔*	✔	✔
subscribeToEventCategory (category)	✔	–	✔	✔*	✔	✔*
unsubscribeOfEventCategory (category)	✔	–	✔	✔*	✔	✔*

Table 6.8: Correlation Matrix to Validate the Abstract Class Model EventManager

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
TenantManager						
createTenant(): Tenant	✔	✔	(✔)	–	✔	–
retrieveTenant(tenantID): Tenant	✔	✔	(✔)	–	✔	✔
updateTenant(tenantID): Tenant	✔	✔	(✔)	–	✔	–
deleteTenant(tenantID)	✔	✔	(✔)	–	✔	–
retrieveTenantByDescription (description): Tenant	✔	✔	(✔)	–	✔	–
retrieveAllTenants()	(✔)	(✔)	(✔)	–	(✔)	–
retrieveTenantInformation (tenantID)	✔	✔	(✔)	–	✔	✔

Table 6.9: Correlation Matrix to Validate the Abstract Class Model TenantManager

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
AuthorityManager						
createAuthority(): Authority	✔	✔	✔*	✔	✔	(✔)
retrieveAuthority (authorityID): Authority	✔	✔	✔*	✔	✔	(✔)
updateAuthority (authorityID): Authority	✔	–	✔*	(✔)	✔	(✔)
deleteAuthority(authorityID)	✔	–	✔*	(✔)	✔	–

Table 6.10: Correlation Matrix to Validate the Abstract Class Model AuthorityManager

Abstract Class Model	FIWARE	SiteWhere	SmartThings	AWS IoT	Microsoft Azure IoT	IBM Watson IoT Platform
Pluggable Service						
retrievePluggableServices()	✔	✔	✔*	✔*	✔	✔
Platform Information						
retrievePlatformInformation()	(✔)	(✔)	(✔)	–	✔	✔
retrieveServerInformation()	✔	✔	(✔)	–	(✔)	✔
retrieveVersionInformation()	✔	✔	(✔)	–	✔	✔
Gateway						
translateMessage()	✔*	✔*	✔	✔*	✔*	✔*
manageConnection()	✔*	✔*	✔	✔*	✔*	✔*

Table 6.11: Correlation Matrix to Validate the Abstract Class Model Pluggable Service, Platform Information, and Gateway

the version. Within AWS IoT no comparable functionality is existent. The Gateway class operations are only mentioned explicitly within the documentation of SmartThings, and therefor the same functionality exists. All remaining platforms do not have any comparable functionality documented. Since a connection component was part of every architecture, they presumably have comparable functionality. Furthermore it provides the underlying basis for the communication of the devices with the platform, and hence is essential for a CPS.

6.2 Conclusion

Following the validation of the abstract class model, most of the operations of the classes can be found within the considered platforms. The differences within the Zone, and ZoneManager result of the various approaches to group devices, and users. Likewise the distinction of the concept of a user is indicated within the validation. The approach of tenants is existent within every platform, where the impacts to the user of the platforms differ. Since there is no standardized approach to manage the authorities, every platform uses a different solution. This becomes apparent within the validation

of the AuthorityManager. Nevertheless the validation of the abstract class model has shown, that it is applicable to the considered platforms. Hence it can be applied as an reference for the design of a CPS.

7 Discussion and Further Research

The following Chapter summarizes the previous research. Furthermore it discusses the conclusions, and subsequently possible further research is outlined.

7.1 Résumé

Within this Master's Thesis initially CPS, and applied protocols and standards are introduced. The considered CPS solutions are described, and analyzed. Thereupon the reference architecture is derived, and validated. Subsequently the features of the considered CPS solutions are analyzed, and the abstract class model is derived and validated, as well.

The conclusion of the validation of the CPS reference architecture is, that the derived reference architecture is applicable to every considered CPS platform, and hence can be used as an universal reference architecture. The aim is to provide an architectural basis for the design of an CPS, and a reference for the content of the components. The reference architecture contains of six components: the Sensor/Actuator, the Device (including the Driver), the Gateway, the IoTIM, the Application, and the Further Data Source. An Sensor is a hardware entity, translating changes of the physical environment into electrical signals, and sending those to the device. An Actuator is likewise a hardware entity, receiving commands send by the device, and acting on the physical environment by translating electrical signals into some kind of physical action. Sensors/Actuators are always physically connected to a Device, which is likewise a hardware entity. A Device communicates either directly, or via a Gateway with the IoTIM. A Gateway translates the messages of the device into the required format, and passes it on to the IoTIM. The IoTIM is the core logic component of a CPS. The features of the IoTIM are represented within the abstract class model. The Application, and the Further Data Source components enable the connection of further applications, and data sources.

Following the validation of the abstract class model, it is likewise applicable to the considered CPS platforms, and hence can be referenced as basis for the design of a CPS. The abstract class model consists of 19 classes, where seven are managing classes, i.e., they exist to instantiate, and manage their underlying classes. Namely the managing

classes with their underlying classes are: TenantManager and Tenant, AuthorityManager and Authority, DeviceManager and Device, GroupManager and Group, ZoneManager and Zone, UserManager and User, and EventManager and Event. The remaining classes are: DeviceSpecification, Gateway, Platform, Platform Information, and Pluggable Service. As the platform class represents the initial point, every class is dependent of the Platform. A Tenant enables to build autonomous instances of a CPS within one platform. Groups and Zones are used to aggregate Devices and Users into logical or geographical units. Furthermore the DeviceSpecification can be used to categorize the devices. The Gateway enables the translation of messages, and manages the connection of the devices. The EventManager provides operations to create, retrieve, publish, and subscribe to events, which is the basis of the communication between the Platform and the Devices. The AuthorityManager is responsible for the authorization of Devices, Users, Groups, Zones, and Tenants. The Platform Information provides operations to retrieve information about the platform, the server, or the version. Pluggable Services are possibly connected further applications, which can be internal, i.e., they are provided by the same provider as the platform, or they are external in terms of third party applications.

7.2 Discussion

To derive the CPS reference architecture and the abstract class model, seven CPS platforms are considered. Questionable is, if thereby a significant, universal conclusion can be made. Since the platforms are open-source, as well as proprietary, at least various segments are covered. The comparison of the platforms, and the validation has shown, that the core approach is alike. The deviations are the result of the diverse concepts pursued, like, e.g., the support of groups and zones, to aggregate devices into logical or geographical units. Likewise the definition of devices is diverse, as some platforms use devices in terms of a device, which has included sensors and actuators, and some platforms differ within devices, and sensors and actuators. Furthermore the gateway component is represented as an separate component, or integrated within the core logic of the platform.

Another critical aspect is, that some of the documentations of the platforms seem to be incomplete, and hence the validation therefor is assumed, based on the available information.

7.3 Further Research

Within this Master's Thesis it is noticeable, that the introduced standards are not applied by every platform, and only some transport protocols are supported. Since they build the basis for the communication of the devices with the platform, they could represent a not completely used potential, which should be researched in detail. Thereby it is interesting, to analyze if the standards and protocols are used, if they are used in an effective way, and if not, how this could be achieved.

Excluded within this Thesis is the security issue, which composes another further research sector. Since the considered platforms within this research followed diverse approaches to handle the authorization, an analysis of those approaches could be a starting point for further research.

Within the comparison of the features of the platforms, and the validation of the abstract class model it became apparent, that users in terms of an involved participant of the system are not universalized. Following this, an analysis of how users should be integrated within CPS could be another research issue.

Bibliography

[Ama16a] Amazon Web Services. *AWS IoT*. https://aws.amazon.com/de/iot/how-it-works/. 2016. (Visited on 02/29/2016).

[Ama16b] Amazon Web Services. *AWS IoT API Reference*. http://docs.aws.amazon.com/iot/latest/apireference/Welcome.html. Mar. 2016. (Visited on 09/03/2016).

[Ama16c] Amazon Web Services. *AWS IoT Developer Guide*. http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html. 2016. (Visited on 09/03/2016).

[Ama16d] Amazon Web Services. *AWS IoT Documentation*. https://aws.amazon.com/documentation/iot/. 2016. (Visited on 02/03/2016).

[Ama16e] Amazon Web Services. *AWS IoT FAQs - Amazon Web Services*. https://aws.amazon.com/iot/faqs/. 2016. (Visited on 03/17/2016).

[And13] Piper Andy. *Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP*. https://blogs.vmware.com/vfabric/2013/02/choosing-your-messaging-protocol-amqp-mqtt-or-stomp.html. Feb. 2013. (Visited on 02/29/2016).

[Anj+02] M. Anjanappa, K. Datta, and T. Song. "Introduction to Sensors and Actuators." In: *The Mechatronics Handbook*. 2nd ed. Austin, Texas: CRC Press, 2002, (16–1)–(16–14).

[ARM11] ARM Limited. *ARM Information Center - How to start Device Server to support CoAP over TCP*. https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwi8uJLonZ_LAhWJDZoKHZsmAXUQFggdMAA&url=https%3A%2F%2Frepositorium.sdum.uminho.pt%2Fbitstream%2F1822%2F34905%2F1%2FDisserta%25C3%25A7%25C3%25A3o_Pedro%2520de%2520Barbosa%2520Mendon%25C3%25A7a%2520Diogo_2014.pdf&usg=AFQjCNFoPBPnMFpxblqEAeU-bF0mqRF2NQ. 2011. (Visited on 03/20/2016).

[Ban+13] S. Bandyopadhyay and A. Bhattacharyya. “Lightweight Internet protocols for web enablement of sensors using constrained gateway devices.” In: *2013 International Conference on Computing, Networking and Communications (ICNC)*. Jan. 2013, pp. 334–340.

[Bel+15a] M. Belshe, R. Peon, and M. Thomson Ed. *Hypertext Transfer Protocol version 2*. https://tools.ietf.org/html/draft-ietf-httpbis-http2-17. Feb. 2015. (Visited on 02/29/2016).

[Bel+15b] M. Belshe, BitGo, R. Peon, Google, Inc., and M. Thomson, Ed. *RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)*. https://tools.ietf.org/html/rfc7540. May 2015. (Visited on 03/16/2016).

[Bet16] Dominic Betts. *Microsoft Azure - Übersicht über Azure IoT Hub*. https://azure.microsoft.com/de-de/documentation/articles/iot-hub-what-is-iot-hub/. Mar. 2016. (Visited on 02/03/2016).

[Cha13] Hakima Chaouchi. *The Internet of Things: Connecting Objects*. en. John Wiley & Sons, Feb. 2013.

[Cha+94] Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey Ullman, and Jennifer Widom. “The TSIMMIS Project: Integration of Heterogeneous Information Sources.” In: *Information Processing Society of Japan (IPSJ 1994)*. Tokyo, Japan, Oct. 1994.

[Che+12] Feng Chen, Changrui Ren, Qinhua Wang, and Bing Shao. “A process definition language for Internet of things.” In: *2012 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*. July 2012, pp. 107–110.

[Che+14] H. W. Chen and F. J. Lin. “Converging MQTT Resources in ETSI Standards Based M2M Platform.” In: *IEEE Internet of Things (iThings), 2014 IEEE International Conference on, and Green Computing and Communications (GreenCom), IEEE and Cyber, Physical and Social Computing(CPSCom)*. Sept. 2014, pp. 292–295.

[Cor16a] Core Network Dynamics GmbH. *OpenEPC - What is EPC?* http://www.openepc.com/home/what-is-epc/. Mar. 2016. (Visited on 03/17/2016).

[Cor16b] Core Network Dynamics GmbH. *OpenEPC – The OpenEPC Project*. http://www.openepc.com/. Mar. 2016. (Visited on 03/17/2016).

[Der+15] H. Derhamy, J. Eliasson, J. Delsing, and P. Priller. “A survey of commercial frameworks for the Internet of Things.” In: *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*. Sept. 2015, pp. 1–8.

[Dio14] Pedro de Barbosa Mendonca Diogo. *A Complete Internet of Things Solution for Real-Time Web Monitoring*. Oct. 2014.

[Dom+09] John Domingue, Dieter Fensel, Paolo Traverso, and FIS, eds. *Future Internet - FIS 2008: first Future Internet Symposium, FIS 2008 Vienna, Austria, September 29-30, 2008 ; revised selected papers*. eng. Lecture notes in computer Science 5468. Berlin: Springer, 2009.

[Ell14] Omar Elloumi. *OneM2M Service Layer Platform - Initial Release*. http://www.onem2m.org/onem2m-showcase/showcase-presentations. Dec. 2014. (Visited on 03/03/2016).

[ETS13] ETSI. *ETSI TS 102 690 V2.1.1*. http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/02.01.01_60/ts_102690v020101p.pdf. Oct. 2013. (Visited on 03/16/2016).

[ETS15] ETSI. *ETSI TS 118 101 V1.0.0*. http://www.etsi.org/deliver/etsi_ts/118100_118199/118101/01.00.00_60/ts_118101v010000p.pdf. Feb. 2015. (Visited on 03/16/2016).

[Faj+12] Fajardo, Arkko, Loughney, and Zorn. *RFC 6733 - Diameter Base Protocol*. https://tools.ietf.org/html/rfc6733. Oct. 2012. (Visited on 03/16/2016).

[Fan+10] Tongrang Fan and Yanzhao Chen. "A scheme of data management in the Internet of Things." In: *2010 2nd IEEE International Conference on Network Infrastructure and Digital Content*. Sept. 2010, pp. 110–114.

[Fet+11] I. Fette and A. Melnikov. *RFC 6455 - The WebSocket Protocol*. https://tools.ietf.org/html/rfc6455. Dec. 2011. (Visited on 03/16/2016).

[Fie+99] R. Fielding, UC Irvine, J. Gettys, and J. Mogul. *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1*. https://tools.ietf.org/html/rfc2616. June 1999. (Visited on 03/16/2016).

[Fil00] Roy Thomas Fileding. "Architectural Styles and the Design of Network-based Software Architectures." PhD thesis. Irvine: University of California, 2000.

[FIW15a] FIWARE. *FIWARE Architecture*. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_Architecture. Aug. 2015. (Visited on 03/17/2016).

[FIW15b] FIWARE. *Internet of Things (IoT) Services Enablement Architecture*. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Internet_of_Things_(IoT)_Services_Enablement_Architecture. July 2015. (Visited on 03/16/2016).

[FIW15c] FIWARE. *Summary of FIWARE API Open Specifications*. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Summary_of_FI-WARE_API_Open_Specifications. Dec. 2015. (Visited on 09/03/2016).

[FIW16a] FIWARE. *FIWARE*. https://www.fiware.org. Feb. 2016. (Visited on 02/29/2016).

[FIW16b] FIWARE. *FIWARE Wiki*. https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Main_Page. Mar. 2016. (Visited on 03/03/2016).

[FIW16c] FIWARE. *Quick FIWARE Tour Guide*. http://fiwaretourguide.readthedocs.org/en/latest/. Mar. 2016. (Visited on 02/03/2016).

[Fra14] FOKUS Fraunhofer. *OpenMTC R3 Documentation Device Management Overview*. http://www.open-mtc.org/Downloads/API-Documentation/dm/index.html. 2014. (Visited on 09/03/2016).

[Fra15] FOKUS Fraunhofer. *OpenMTC*. http://www.open-mtc.org. 2015. (Visited on 02/29/2016).

[Fri13] Peter Friess. *Internet of Things: Converging Technologies for Smart Environments and Integrated Ecosystems*. en. River Publishers, June 2013.

[Gaz+15] V. Gazis, M. Gortz, M. Huber, A. Leonardi, K. Mathioudakis, A. Wiesmaier, F. Zeiger, and E. Vasilomanolakis. “A survey of technologies for the internet of things.” In: *Wireless Communications and Mobile Computing Conference (IWCMC), 2015 International*. Aug. 2015, pp. 1090–1095.

[Gil16] Marc Gille. *GitHub - marcgille/thing-it-node: REST, WebSocket, Service Definition and Event Processing for Devices like Raspberry Pi, C.H.I.P, Intel Computestick etc.*. https://github.com/marcgille/thing-it-node. Mar. 2016. (Visited on 03/21/2016).

[Gos12] Subrata Goswami. *Internet Protocols: Advances, Technologies and Applications*. en. Springer Science & Business Media, Dec. 2012.

[Gou+02] David Gourley, Brian Totty, Marjorie Sayer, Anshu Aggarwal, and Sailu Reddy. *HTTP: The Definitive Guide: The Definitive Guide*. en. "O'Reilly Media, Inc.", Sept. 2002.

[Gri+14] L. A. Grieco, M. Ben Alaya, T. Monteil, and K. Drira. “Architecting information centric ETSI-M2M systems.” In: *2014 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. Mar. 2014, pp. 211–214.

[Gri13] Ilya Grigorik. *High Performance Browser Networking*. Sept. 2013.

[IBM16a] IBM. *IBM Internet of Things Architecture Overview*. https://www.iot-academy.info/mod/page/view.php?id=478. Feb. 2016. (Visited on 03/03/2016).

[IBM16b] IBM. *IBM IoT Foundation API*. https://docs.internetofthings.ibmcloud.com/swagger/v0002.html. Mar. 2016. (Visited on 09/03/2016).

[IBM16c] IBM. *IBM Watson Internet of Things Platform*. http://www.ibm.com/internet-of-things/iot-platform.html. Mar. 2016. (Visited on 03/03/2016).

[Kha+12] R. Khan, S.U. Khan, R. Zaheer, and S. Khan. "Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges." In: *2012 10th International Conference on Frontiers of Information Technology (FIT)*. Dec. 2012, pp. 257–260.

[Kop15] Oliver Kopp. *Similar Approaches - marcgille/thing-it-node*. https://github.com/marcgille/thing-it-node. Dec. 2015. (Visited on 03/20/2016).

[Kuf16] Bernard Kufluk. *The IBM Watson IoT Platform arrives*. https://developer.ibm.com/iotfoundation/blog/2016/02/12/the-ibm-watson-iot-platform-arrives/. Feb. 2016. (Visited on 08/03/2016).

[Lee08] E.A. Lee. "Cyber Physical Systems: Design Challenges." In: *2008 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC)*. May 2008, pp. 363–369.

[Lem+14] Simon Lemay, Hannes Tschofenig, Zebra Technologies, and Carsten Bormann. *A TCP and TLS Transport for the Constrained Application Protocol (CoAP)*. https://tools.ietf.org/html/draft-tschofenig-core-coap-tcp-tls-01. Sept. 2014. (Visited on 03/19/2016).

[Luz+15] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni. "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks." In: *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*. Jan. 2015, pp. 931–936.

[Mic16a] Microsoft. *Azure IoT Hub*. https://azure.microsoft.com/en-us/services/iot-hub/. Mar. 2016. (Visited on 03/03/2016).

[Mic16b] Microsoft. *Azure IoT Hub Documentation*. https://azure.microsoft.com/de-de/documentation/services/iot-hub/. Mar. 2016. (Visited on 03/03/2016).

[Mic16c] Microsoft. *Azure IoT Suite*. https://azure.microsoft.com/en-us/solutions/iot-suite/. Mar. 2016. (Visited on 01/03/2016).

[Mil+10] Peter Millard, Peter Saint-Andre, and Ralph Meijer. *Publish-Subscribe*. en. http://xmpp.org/extensions/xep-0060.html. XMPP Extension Protocol. July 2010. (Visited on 03/17/2016).

[Mon16] MongoDB, Inc. *MongoDB for GIANT Ideas*. https://www.mongodb.org. 2016. (Visited on 03/16/2016).

[OAS12] OASIS Standard. *OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0*. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf. Oct. 2012. (Visited on 03/16/2016).

[OAS14] OASIS Standard. *MQTT Version 3.1.1 - OASIS Standard*. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html. Oct. 2014. (Visited on 03/16/2016).

[One15] OneM2M Partners. *oneM2M White Paper - The Interoperability Enabler for the Entire M2M and IoT Ecosystem*. http://www.onem2m.org/images/files/oneM2M-whitepaper-January-2015.pdf. Jan. 2015. (Visited on 03/03/2016).

[Ope12a] Open Mobile Alliance. *Next Generation Service Interfaces Architecture - Approved Version 1.0*. http://technical.openmobilealliance.org/Technical/Release_Program/docs/NGSI/V1_0-20120529-A/OMA-AD-NGSI-V1_0-20120529-A.pdf. May 2012. (Visited on 03/16/2016).

[Ope12b] Open Mobile Alliance. *NGSI Context Management - Approved Version 1.0*. http://technical.openmobilealliance.org/Technical/release_program/docs/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf. May 2012. (Visited on 03/03/2016).

[Ope15] Open Mobile Alliance. *Ligthweight Machine to Machine Technical Specifciation - Candidate Version 1.0*. http://technical.openmobilealliance.org/Technical/Release_Program/docs/LightweightM2M/V1_0-20151214-C/OMA-TS-LightweightM2M-V1_0-20151214-C.pdf. Dec. 2015. (Visited on 03/16/2016).

[Pos16] Postscapes. *2014/15 Internet of Things Awards*. http://postscapes.com/internet-of-things-award/2014/winners. Mar. 2016. (Visited on 04/03/2016).

[Rag15] Francesco Rago. "A Smart Adaptable Architecture Based on Contexts for Cyber Physical Systems." In: *Procedia Computer Science*. Complex Adaptive Systems San Jose, CA November 2-4, 2015 61 (2015), pp. 301–306.

[Res+00] E. Rescorla and RTFM, Inc. *RFC 2818 - HTTP Over TLS*. https://tools.ietf.org/html/rfc2818. May 2000. (Visited on 03/16/2016).

[Sal+15] Flora Salim and Usman Haque. "Urban computing in the wild: A survey on large scale participation and citizen engagement with ubiquitous computing, cyber physical systems, and Internet of Things." In: *International Journal of Human-Computer Studies*. Transdisciplinary Approaches to Urban Computing 81 (Sept. 2015), pp. 31–48.

[She+14] Z. Shelby, ARM, K. Hartke, C. Bormann, and Universität Bremen TZI. *RFC 2752 - The Constrained Application Protocol (CoAP)*. http://tools.ietf.org/html/rfc7252. June 2014. (Visited on 03/16/2016).

[She14] Zach Shelby. *OMA Lightweight M2M Protocol (OMA LWM2M) Tutorial*. https://www.youtube.com/watch?v=g-41ZdcTnXc. May 2014. (Visited on 03/03/2016).

[Sit16a] SiteWhere, LLC. *SiteWhere*. http://www.sitewhere.org. 2016. (Visited on 01/03/2016).

[Sit16b] SiteWhere, LLC. *SiteWhere Documentation*. http://documentation.sitewhere.org/overview.html. 2016. (Visited on 02/03/2016).

[Sit16c] SiteWhere, LLC. *SiteWhere REST Services Documentation*. http://documentation.sitewhere.org/rest/single.html. Mar. 2016. (Visited on 09/03/2016).

[Sit16d] SiteWhere, LLC. *SiteWhere System Architecture*. http://www.sitewhere.org/documentation/system-architecture/. 2016. (Visited on 10/30/2015).

[Sma15a] SmartThings, Inc. *SmartThings API Documentation*. http://docs.smartthings.com/en/latest/ref-docs/reference.html. 2015. (Visited on 09/03/2016).

[Sma15b] SmartThings, Inc. *SmartThings Documentation*. http://docs.smartthings.com/en/latest/architecture/index.html. 2015. (Visited on 01/03/2016).

[Sma16] SmartThings, Inc. *SmartThings*. https://www.smartthings.com. 2016. (Visited on 01/03/2016).

[Sto16] Stomp. *STOMP Protocol Specification, Version 1.2*. http://stomp.github.io/stomp-specification-1.2.html. Feb. 2016. (Visited on 02/29/2016).

[Sub+08] H. Subramoni, G. Marsh, S. Narravula, Ping Lai, and D. K. Panda. “Design and evaluation of benchmarks for financial applications using Advanced Message Queuing Protocol (AMQP) over InfiniBand.” In: *Workshop on High Performance Computational Finance, 2008. WHPCF 2008*. Nov. 2008, pp. 1–8.

[Tal08] Carolyn Talcott. “Cyber-Physical Systems and Events.” en. In: *Software-Intensive Systems and New Computing Paradigms*. Ed. by Martin Wirsing, Jean-Pierre Banâtre, Matthias Hölzl, and Axel Rauschmayer. Lecture Notes in Computer Science 5380. Springer Berlin Heidelberg, 2008, pp. 101–115.

[The16] The Apache Software Foundation. *Apache HBase*. https://hbase.apache.org. Mar. 2016. (Visited on 03/18/2016).

[Tob14] Jaffey Toby. *MQTT and CoAP, IoT Protocols*. https://eclipse.org/community/eclipse_newsletter/2014/february/article2.php. Feb. 2014. (Visited on 02/29/2016).

[Vas14] Clemens Vasters. "*Service Assisted Communication*" *for Connected Devices*. Feb. 2014. (Visited on 03/16/2016).

[Woj16] Maik Wojcieszak. “Internet- und Anwendungsprotokolle als IoT-Grundlage.” deutsch. In: *iX Developer 1/2016 - Internet der Dinge* (Jan. 2016), pp. 68–73.

[Zhe+11] Jun Zheng, D. Simplot-Ryl, C. Bisdikian, and H.T. Mouftah. “The internet of things [Guest Editorial].” In: *IEEE Communications Magazine* 49.11 (Nov. 2011), pp. 30–31.

[Zim+13] Alfred Zimmermann, Kurt Sandkuhl, Michael Pretz, Michael Falkenthal, Dierk Jugel, and Matthias Wißotzki. “Towards an integrated serviceJoriented reference enterprise architecture.” In: ACM, 2013, pp. 26–30.

A Appendix

1)	NGSI Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FI-WARE_NGSI_Open_RESTful_API_Specification)
2)	OpenStack Clustering API v1 (http://developer.openstack.org/api-ref-clustering-v1.html) Senlin (https://wiki.openstack.org/wiki/Senlin)
3)	Semantic Support Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Semantic_Support_Open_RESTful_API_Specification)
4)	Publish/Subscribe Semantic Extension Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Publish/Subscribe_Semantic_Extension_Open_RESTful_API_Specification)
5)	Complex Event Processing Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Complex_Event_Processing_Open_RESTful_API_Specification)
6)	Service Composition Open API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Service_Composition_Open_API_Specification_(PRELIMINARY))
7)	Semantic Annotation Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Semantic_Annotation_Open_RESTful_API_Specification)
8)	Location Server Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Location_Server_Open_RESTful_API_Specification)
9)	ETSI M2M mld Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/ETSI_M2M_mId_RESTful_API_Specification_(PRELIMINARY))
10)	Device Sensors API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Device_Sensors_API_Specification_(PRELIMINARY))
11)	Query Broker Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Query_Broker_Open_RESTful_API_Specification)
12)	Apps Marketplace Search RESTful API (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Apps.MarketplaceSearchREST)
13)	Security Chapter AccessControl GE Authorization Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE.OpenSpecification.Security.AccessControlGE.Authorization.Open_RESTful_API_Specification)
14)	Mediator GE Open RESTful API Specification (https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Mediator_GE_Open_RESTful_API_Specification_(PRELIMINARY))
15)	OpenStack Compute API (http://developer.openstack.org/api-ref-compute-v2.1.html)
16)	OpenStack Networking API v2.0 (http://developer.openstack.org/api-ref-networking-v2.html)
17)	OpenStack Shared File Systems API (http://developer.openstack.org/api-ref-share-v2.html)
18)	OpenStack Identity API v3 (http://developer.openstack.org/api-ref-identity-v3.html)
19)	OpenStack Image Service API v2 (http://developer.openstack.org/api-ref-image-v2.html)

Table A.1: Index Declaration of the Comparison Correlation Matrix: FIWARE

OC	Organization Configuration
BO	Bulk Operations
DT	Device Types
D	Devices
DD	Device Diagnostics
PD	Problem Determination
HER	Historical Event Retrieval
DMR	Device Managment Requests
UM	Usage Management
SS	Service Status
EV	Event Cache
C	Connectivity
ES	Exsternal Services

Table A.2: Index Declaration of the Comparison Correlation Matrix: IBM Watson IoT Platform

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature