A GeoEvent-driven Architecture based on GeoMQTT for the Geospatial IoT

Stefan Herlé

Veröffentlichung des Geodätischen Instituts der
Rheinisch-Westfälischen Technischen Hochschule Aachen
Mies-van-der-Rohe-Straße 1, 52074 Aachen

NR. 71

2019
ISSN 0515-0574

A GeoEvent-driven Architecture based on GeoMQTT for the Geospatial IoT

Von der Fakultät für Bauingenieurwesen der Rheinisch-Westfälischen Technischen Hochschule Aachen zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von

Stefan Herlé

Berichter:
Universitätsprofessor Dr.-Ing. Jörg Blankenbach
Universitätsprofessor Dr.-Ing. Ralf Bill

Tag der mündlichen Prüfung: 24.10.2019

Veröffentlichung des Geodätischen Instituts der
Rheinisch-Westfälischen Technischen Hochschule Aachen
Mies-van-der-Rohe-Straße 1, 52074

Nr. 71

2019
ISSN 0515-0574

ACKNOWLEDGMENTS

I have written this thesis during my employment as a research associate at the Geodetic Institute and Chair for Computing in Civil Engineering & Geo Information Systems of RWTH Aachen University (gia). Over the years I have learned great deal, had a lot fun and have been able to work in a very pleasant atmosphere.

First, I would like to express my sincere gratitude to my supervisor Univ.-Prof. Dr.-Ing. Jörg Blankenbach for supporting me with patience, motivation and knowledge throughout my doctoral studies. His guidance and advice helped me to succeed in implementing the concept and writing this thesis. Furthermore, I thank him for suggesting the catchy protocol name "GeoMQTT". Likewise, I would also like to thank Univ.-Prof. Dr.-Ing. Ralf Bill for accepting the task of Korreferent for my doctoral thesis. His insightful comments and encouragement helped me to question my work from various perspectives and improve my thesis.

I would also like thank the many fellows who supported me during my dissertation project. My colleagues at the Geodetic Institute were not only great sources for ideas and inspiration, but also provided invaluable team spirit in our extracurricular activities such as climbing, go-karting or footgolf. Special thanks go to my officemates Robert Löhring and Marius Laska, who provided me with advice and assistance, and Vlad for creating the GeoMQTT logo. Further, I would like to thank Lydia Gecsey, Holger Rudat and Eberhard Vogel-Stirnberg for their precious IT and general support. Thanks also to colleagues from sister institutes for supporting me over the course of the dissertation, especially colleagues of the EarlyDike project, who inspired me greatly. Thanks to them for the lively discussions in our project meetings and their cooperation overall. Furthermore, I would also like to thank my "coffee fellows" for providing me with much-needed daily distractions.

Finally, I thank my family and friends for supporting me spiritually throughout writing this thesis. Thank you for always having an open ear for my concerns or ideas, even at the breakfast table.

Aachen, November 2019
Stefan Herlé

One thing left to say:

“There’s still much to do; still so much to learn. [...] Engage!”
— Capt. Jean-Luc Picard, *Star Trek: TNG*

SUMMARY

In the Internet-of-Things (IoT) vision things of the physical world communicate with humans and other objects by means of the Internet. The concept extends the traditional Internet of Computers (IoC) that uses computers such as desktop PC, notebooks or smartphones as access points to the Internet by a multitude of end points represented by physical objects. Since things, human beings but also events in the physical world possess various spatial properties, their actions, observations or happenings inhere geospatial information. Hence, the exchange of digital geospatial information in the IoT leads to the Geospatial IoT. The Geospatial IoT is fundamentally different from established geoinformation technologies. Things in the Geospatial IoT provide geodata with a much higher frequency and quantity. Its integration is foreseen as the most revolutionary change in the history of geoinformation technologies. The ubiquitous geodata collection and provisioning through the Geospatial IoT as an all-encompassing infrastructure holds huge potential for better spatially understanding, modeling and visualizing of our natural and artificial ecosystems. However, a lot of challenges occur in integrating spatial and spatiotemporal data from IoT devices into new or already established systems. Novel requirements concerning architectures, messaging mechanisms and technologies such as scalability or efficiency appear, which must be analyzed and considered when implementing an appropriate infrastructure.

This thesis addresses different research questions regarding an infrastructure for the Geospatial IoT and its integration into established geoinformation technologies. Approaching these issues, typical concepts, architectures and buildings blocks of the IoT are investigated first. This holds especially for recent developments in the field of IoT communications on different layers. Based on these typical structures, a concept for a Geospatial IoT architecture is designed. The conceptual architecture uses GeoEvents as a base data type for exchanging spatiotemporal messages between actors in the Geospatial IoT. The data type is derived from real-world geospatial events and states, which represent occurrences and states of continuants. Basically, a GeoEvent is a four tuple with a name, a spatial component, a temporal component and a message payload. With that event type defined, a GeoEvent-driven architecture for the Geospatial IoT can be specified. It follows an event-driven pattern, so that GeoEvents are send to interested consumers when they occur. Thereby, consumers may specify their interest in GeoEvents by GeoSubscriptions. In the architecture, GeoEvents are distributed by a GeoEvent processing engine, which evaluates the meta information of the GeoEvents against filters of the GeoSubscriptions.

The IoT communication protocol Message Queuing Telemetry Transport (MQTT) is chosen to implement the GeoEvent-driven architecture for the Geospatial IoT. It meets requirements for resource-constraint devices such as small message size and efficiency, but has also beneficial architectural properties such as the messaging paradigm or the scalability of the server. In the thesis, the topic-based publish/-subscribe protocol MQTT is extended by a GeoEvent and a GeoSubscription data type. In the developed extension called GeoMQTT, new messages are introduced to encode these data types. A GeoMQTT broker distributes the GeoEvents between the clients based on the GeoSubscriptions. To achieve this, filtering capabilities on temporal and spatial components are integrated, so that GeoMQTT can be used as a content-based publish/subscribe protocol. Several clients in different programming languages are implemented, as well as a GeoMQTT-SN extension for unreliable networks. The latter one focuses especially on small message sizes. GeoMQTT is evaluated with respect to multiple requirements for IoT environments. The expressiveness of the message types as well as the efficiency of the subscribe mechanisms meet the requirements for the Geospatial IoT. Further, scalability is ensured by the GeoMQTT broker and, thus, prepared for a dynamically increasing number of things in the Geospatial IoT. Finally, the spatiotemporal messaging protocol GeoMQTT is integrated in established geoinformation technologies. For instance, a GeoMQTT plug-in for QGIS is implemented to receive GeoEvents in a desktop Geographic Information System (GIS) in real-time. The Web Processing Service (WPS) for executing geoprocesses is extended by new input and output data types, namely GeoPipes, so that processes on geospatial data streams can be invoked. Additionally, bridges to Sensor Web services and to REST servers provide end users or software agents access to historical or new GeoEvents by means of the World Wide Web (WWW).

The thesis approaches and solves some of the challenges and tasks occurring in building an infrastructure for the Geospatial IoT. The prototypical implementation of the GeoEvent-based architecture and its related services show that things in the Geospatial IoT can be interconnected efficiently by a spatiotemporal messaging mechanism. Real-time access is realizable by services and established geoinformation technologies may also participate. Nevertheless, there are still many ongoing issues to solve, which are addressed in the end of the thesis.

KURZFASSUNG

Im Internet-of-Things (IoT) (Internet der Dinge, IdD) vernetzen sich physische Dingen mit anderen Objekt und Menschen mittels Informations- und Kommunikationstechnologien des Internets. Damit erweitert das IoT Konzept das traditionelle Internet der Computer um eine Vielzahl von Zugangspunkten. Alltägliche physische Objekte können benutzt werden um weltweit mit anderen Menschen oder Dingen zu kommunizieren. Da Objekte, Menschen und Ereignisse der realen Welt räumliche Eigenschaften besitzen, wohnen ihren Handlungen, Beobachtungen, Zuständen oder Geschehnissen ebenfalls Geoinformationen inne. Durch den Austausch digitaler Geoinformationen (Geodaten) im IoT kann auch von einem Geospatial IoT gesprochen werden. Dabei ist das Geospatial IoT fundamental verschieden von konventionellen Geoinformationstechnologien. Dinge im IoT liefern ihre Geodaten mit einer viel höheren Frequenz und Menge. Die Integration dieser Dinge wird als der größte revolutionäre Wandel in der Geschichte von Geoinformationstechnologien angesehen. Dabei hat die allgegenwärtige Sammlung und Bereitstellung von Geodaten durch ein Geospatial IoT großes Potential. Natürliche und von Menschen gemachte Ökosysteme können so besser räumlich erfasst, modelliert, visualisiert und verstanden werden. Es sind allerdings noch viele Herausforderungen zu lösen um räumliche und raumzeitliche Echtzeitdaten von IoT Geräten in neue oder etablierte Systeme zu integrieren. Anforderungen an Architektur und Nachrichtenaustauschmechanismus wie etwa Skalierbarkeit oder Effizienz müssen analysiert, evaluiert und umgesetzt werden, um eine geeignete Infrastruktur für ein Geospatial IoT aufzubauen.

Diese Doktorarbeit fokussiert sich auf verschiedenste Forschungsfragen im Zusammenhang mit einer Infrastruktur für ein Geospatial IoT und der Integration von IoT Dingen, Daten und Datenströmen in etablierte Geoinformationstechnologien. Dazu werden zunächst typische Konzepte, Architekturen und Bausteine des IoT untersucht. Insbesondere werden neuste Entwicklungen bei der IoT Kommunikation betrachtet. Basierend auf diesen Analysen wird ein Konzept für eine Geospatial IoT Architektur entwickelt. Die konzeptionelle Architektur benutzt als Basisdatentyp GeoEvents, um Akteure in einem Geospatial IoT über einen raumzeitlichen Nachrichtenaustauschmechanismus zu verknüpfen. Dieser Datentyp ist abgeleitet von raumzeitlichen Ereignissen aus der realen Welt. Ein GeoEvent ist ein 4-Tupel mit einem Namen, einer räumlichen und einer zeitlichen Komponente, sowie eines Nachrichtenrumpfes. Mittels GeoEvents kann eine GeoEvent-getriebene Architektur (GeoEvent-driven Architecture) für ein Geospatial IoT entworfen werden. Diese

folgt einem ereignisgesteuerte Muster, sodass GeoEvents direkt zum Konsumenten geschickt werden, sobald diese auftreten. Dabei können Konsumenten ihr Interesse in GeoEvents mittels einer GeoSubscription ausdrücken. Ein GeoEvent-Prozessor evaluiert die Metainformationen jedes eingehenden GeoEvents gegen jede GeoSubscription und verteilt diese an interessierte Clients.

Das IoT Protokoll Message Queuing Telemetry Transport (MQTT) wird in dieser Arbeit genutzt um diese GeoEvent-getriebene Architektur prototypisch zu implementieren. MQTT erfüllt die Anforderungen zur Unterstützung von Ressourcen-beschränkten Geräten wie eine kleine Nachrichtengröße oder gute Performanz, aber auch Architektureigenschaften wie der Nachrichtenaustauschmechanismus und die Skalierbarkeit der Server. Das themen-basierte Publish/Subscribe Protokoll MQTT wird durch die Datentypen GeoEvents und GeoSubscriptions erweitert. Neue Nachrichtentypen werden in Geospatial MQTT (GeoMQTT) eingeführt, um die angedachten Mechanismen umzusetzen. Ein GeoMQTT Broker verteilt die GeoEvents an Clients basierend auf den GeoSubscriptions. Dazu werden Filter-Möglichkeiten auf den Metainformationen von GeoEvents eingeführt, sodass GeoMQTT ein inhaltsbasiertes Publish/Subscribe umsetzt. Neben GeoMQTT Clients in verschiedenen Programmiersprachen, wird die Erweiterung GeoMQTT for Sensor Networks (GeoMQTT-SN) für unzuverlässige Netzwerke entwickelt, die insbesondere auf geringe Nachrichtengrößen achtet. Die GeoMQTT Erweiterung wird in Hinblick auf mehrere Anforderungen für IoT Systeme evaluiert. Sowohl die Ausdrucksstärke für die Modellierung von GeoEvents als auch die Nachrichtengröße und die Effizienz der Filtermechanismen erfüllen die Anforderungen. Auch die Skalierbarkeit ist durch den GeoMQTT Broker sichergestellt. Schließlich wird das GeoMQTT Protokoll in konventionelle Geoinformationstechnologien integriert. Beispielsweise wird ein GeoMQTT Plug-in für QGIS implementiert, um GeoEvents in Echtzeit in einem Desktop GIS zu empfangen. Des Weiteren wird der WPS Dienst, der zum Ausführen von Geoprozessen genutzt wird, um neue Ein- und Ausgabeformate für GeoEvents erweitert, sodass Prozesse auf raumzeitlichen Datenströmen ausgeführt werden können. Zusätzlich werden Brücken zu Sensor Web und Representational State Transfer (REST) Diensten vorgestellt, um archivierte oder neue GeoEvents durch Methoden des WWW abzurufen bzw. zu veröffentlichen.

Diese Arbeit behandelt und löst einige Herausforderungen und Anforderungen, die bei dem Aufbau einer Infrastruktur für das Geospatial IoT auftreten. Die prototypische Implementierung einer GeoEvent-getriebenen Architektur und zugehöriger Dienste zeigt die Möglichkeit Dinge und Systeme in einem Geospatial IoT über raumzeitliche Nachrichtenaustauschmuster zu verbinden. Zusätzlich können diese raumzeitlichen Nachrichten von verschiedenen Diensten und konventionellen Geoinformationstechnologien empfangen und angeboten werden. Dennoch sind noch weitere Herausforderungen für eine Implementierung des Geospatial IoT zu lösen, auf die am Ende der Arbeit eingegangen wird.

CONTENTS

CHAPTER 1

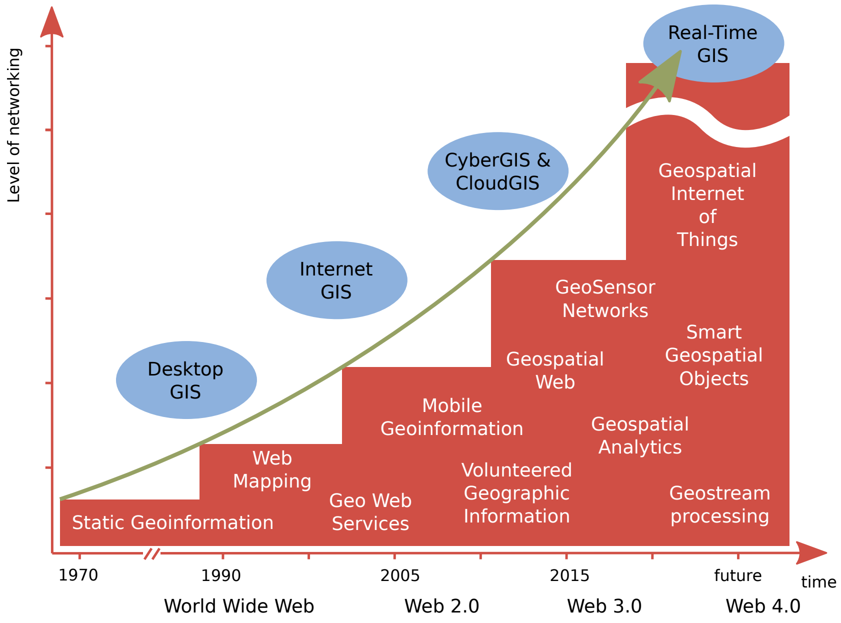
INTRODUCTION

1.1 MOTIVATION

The Internet-of-Things (IoT) is regarded as one of the most disruptive technologies of the century (Alkhatib et al., 2014). Society, industry as well as academia and the public have already shifted their attention towards the IoT evolution to enhance everyday activities. Aside from technology, new products, business models and services are going to emerge from the opportunities the IoT and its related concepts offer. The IoT evolution is driven by advances in sensors and actuators but also increased interconnectivity for IoT devices. In a few years, embedded technology that uses sensors and actuators will shape the way humans and machines interact and communicate with each other. Gartner (2017) forecasts the number of connected devices will rise to 20.4 billion in 2020, with 12.8 billion things in the consumer market and 7.4 billion things in the business sector. But the IoT is more than just sensors and actuators embedded into environmental, urban or personal applications. Through interaction with the physical world, these sensors record tremendous amount of data, but to understand and enhance improvements, computing capacities in back-end computers and data centers are required as well. IoT use cases such as smart city applications rely heavily on sensors and actuators. However, realizing their full potential requires comprehensive analysis of a multitude of tremendous, fast data streams in real-time. IoT platforms consists several technologies, but sensors and actuators form the basis of interactions with the physical world.

Taking a closer look at current, evolving or future IoT applications, most use cases involve the use of geospatial data. For instance, smart city or smart farming applications often involve activities such as managing spatial assets or infrastructures, interconnecting moving devices and vehicles, monitoring environmental parameters or living organisms, all while optimizing energy consumptions. In these everyday activities, spatiotemporal data forms an essential building block towards improving efficiency. Relevant data can be generated by in-situ sensors, devices such as Global Navigation Satellite System (GNSS) receivers, remote sensors such as satellite imagery, or surveillance cameras. Massive real-time geospatial data is generated from various sources that must be embedded into context, and analyzed in real-time according to the demands of each application. It becomes evident that these applications can be considered as part of the **Geospatial IoT**, in which sensor information, processes, analysis and consequent actions are driven by geospatial data and spatiotemporal data streams.

The Geospatial IoT is fundamentally different from traditional geoinformation technologies. Given the level of networking in the Geospatial IoT, further innovations in geoinformation and Geographic Information System (GIS) will develop at an increasingly accelerated rate (see Figure 1.1). With rising numbers of IoT devices, the quantity of producers and consumers of spatial and spatiotemporal data will increase exponentially.



Source: Author's illustration

Figure 1.1: *Evolution of Geoinformation-Technology*

While in the early days of digital geoinformation, the data were static and the first desktop GIS utilizes them to visualize and analyze several spatial occurrences. With the emergence of the WWW in the 1990s, concepts of Internet GIS evolved from static web mapping and more advanced web portals in WebGIS (Abel et al., 1998) to geo web services. These services, such as the widely-used Web Map Service (WMS), were developed to retrieve georeferenced map images using HTTP (Doyle, 2000). Geo web services form the basis for Spatial Data Infrastructures (SDIs), which enable the discovery and use of geospatial data by users. Following these developments, the INSPIRE initiative (Infrastructure for Spatial Information in Europe) became the leading effort in Europe to make spatial and geographical information accessible and interoperable. Here, standardized geo web services are the central building block of the initiative.

The movement of static WWW towards the Web 2.0, in which user-created and driven content combined with technological advances in mobile computing, led to further developments in data collection and consumption in other areas. Examples of such developments can be found in the field of Volunteered Geographic Information (VGI) (Goodchild, 2007), participatory sensing (Burke et al., 2006) or Mobile GIS as well as Location-Based Services (LBSs) (Küpper, 2005; Blankenbach, 2007). The introduction of cloud computing and its related "as a Service" (*aaS) models relocated IT infrastructure components from individual computers to computer networks. CloudGIS and CyberGIS are recent innovations, which have evolved from cloud computing technology (Wang, 2010; Naghavi, 2012; Egenhofer et al., 2016).

The next logical evolutionary step is the incorporation of real-time and big data in GIS. Positioning technology such as GNSSs have shown that the position of objects can be monitored in real-time, so that in the future it will be possible to know "where everything is, at all times" (Goodchild, 2010). Geo Sensor Networks (GSNs) and devices in the Geospatial IoT may publish real-time geospatial data and initiate geo data streams. According to ESRI president Jack Dangermond (2017), the integration of real-time data from the IoT directly into a GIS layer stack is the most revolutionary change in the history of GIS and brings great opportunities. Real-time GIS developments will affect every kind of geoinformation technology from the software level (desktop GIS or geo web service) and the geospatial data management or the spatiotemporal data modeling, over geospatial analytics and geocomputation frameworks to dynamic visualization and geographic knowledge discovery (Yue & Jiang, 2014).

While the potential for better spatially understanding, modeling and visualizing our natural and artificial ecosystems through using IoT as an all-encompassing infrastructure is enormous (Kamilaris & Ostermann, 2018), a lot of challenges remain in integrating spatial and spatiotemporal data from IoT devices. The Geospatial IoT needs a sophisticated and scalable infrastructure to cope with an onslaught of geo sensors publishing data in real-time. Such an infrastructure must be based on suitable concepts and communication mechanisms to provide a stable and interoperable architecture.

1.2 Related Research

In academic literature, the Geospatial IoT has become a popular research subject in the GIScience community. The research aspects under this subject are quite wide-ranging and new challenges and opportunities are a frequent occurrence (Rieke et al., 2018). Kim (2018) identifies five core research areas for Geospatial IoT platforms:

1. Geospatial IoT edge computing: sensor data generation and processing performed in IoT devices.
2. Geospatial IoT device integration: integration and acquisition of heterogeneous geospatial data in IoT platforms.
3. Geospatial IoT data acquisition & management: querying and processing geo data streams from IoT devices as well as spatiotemporal analysis and storage management for large amounts of spatiotemporal data.
4. Geospatial IoT service provider: standardized web service interfaces e.g. for accessing Geospatial IoT data.
5. Geospatial IoT applications: development of various kinds of Geospatial IoT applications (smart city, LBS etc.).

Research in these areas aims to address various challenges, that arise in the development of Geospatial IoT architectures. These challenges include data acquisition, storage, analysis, and disclosure, in addition to services related to spatial, locational and sensor data. Resolving these problems requires interdisciplinary cooperation, which will in turn improve the efficiency and development of Geospatial IoT applications. In the future, IoT applications which rely on geospatial data acquisition, distribution and analyses will shape our everyday processes and activities.

In research literature, one can find several approaches towards resolving challenges that occur in the development of Geospatial IoT applications. These approaches range from optimizing structures in GSN to providing appropriate visualization techniques for spatiotemporal data streams. In this thesis, we focus on the integration of Geospatial IoT devices in an appropriate architecture and simple processing of the emitted geo data. However, since a decoupled view on these fields is impossible, the other research areas are also tackled on the side.

Geospatial IoT device and data integration depend on communication between every actor in the Geospatial IoT - from small devices in GSN over a platform with multiple servers to the end user or device. Smart objects publish data or events to notify other actors to the latest sensor measurements or actuating tasks. To illustrate this integration process, Serbanati et al. (2011) introduced a conceptual model of smart objects in an IoT reference model. Researchers in this study view smart objects as a connecting concept between a physical object and a digital proxy, which can be accessed with appropriate technical interfaces. Zakaria et al. (2015) merge the concept of smart objects of the IoT with geographic objects in space to form smart geographic objects (short SGC). These objects, equipped with embedded technology, communicate with nearby devices or IoT platforms. However, none of these studies focus on the actual implementation of the required technical communication process and its exchange patterns.

Integration of devices and their data can be realized by various mechanisms and protocols such as CoAP or MQTT, which utilize different messaging mechanisms. Further exploration of these protocols, messaging systems and their associated advantages and disadvantages can be found further on in this thesis. The integration of sensors and IoT devices into an Internet platform can be found in numerous technical solutions. For instance, Bröring et al. (2010) proposed a concept called Sensor Bus to connect sensor data to Internet services on the Sensor Web by using protocols such as IRC or XMPP. Commercial solutions, such as Microsoft's Azure IoT Hub[1] or Amazon's AWS IoT[2] apply common IoT protocols to connect Internet-connected devices with cloud services. Unlike in the Geospatial IoT, however, these solutions offer plain messaging but do not consider the characteristics and advantages of geospatially enhanced data and data streams.

Focusing on geospatially enhanced data and streams, some proposed systems can be found in the literature advancing the dissemination of messages based on the spatiotemporal characteristics of the data. Chen et al. (2003), Grothe (2010) and Chen & Shang (2018) have developed such systems for use cases in different areas. These systems usually employ publish/subscribe message exchange patterns, where distribution is based on spatiotemporal filtering. Similar research on this topic has been conducted by Simonis (2006), Echterhoff & Everding (2008), Echterhoff (2010) or Huang (2014) in the Sensor Web domain. Together these studies resulted in the Open Geospatial Consortium (OGC) Publish/Subscribe Standard 1.0 interface specification, which supports the publish/subscribe message exchange pattern in OGC web services (Braeckel et al., 2016). Here, spatiotemporal criteria may be supplied to filter out interests in certain events. However, all these approaches are hardly suitable for resource-restricted devices deployed in the IoT. For the Geospatial IoT, several researchers proposed similar approaches: in Jin & Chen (2010) and Jin et al. (2013) spatiotemporal events act as a base data type for message dissemination. Publish/subscribe middleware is implemented to distribute messages based on spatiotemporal and logical filtering. Both systems are, however, only specified to handle a limited number of clients.

The integration of Geospatial IoT data in GIS technology is a topic with a wealth of research, ranging from simple spatial visualization to data geoprocessing. Shi et al. (2010) used Google Maps to implement a digital home control system for a smart home solution. Aceves & Larios (2012) realized a web service layer to provide georeferenced smart city data in a web browser application. Similarly, Ribeiro et al. (2015) implemented a GIS web-based platform for wireless in-situ geo sensor data visualization and distributed processing by accessing OGC interface standards such as the Sensor Observation Service (SOS) or the Web Processing Service (WPS). Processing of Geospatial IoT data with GIS technologies can be

[1] https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-protocols

[2] https://docs.aws.amazon.com/en_us/iot/latest/developerguide/protocols.html

found in Thakur et al. (2015) or Wiener et al. (2016). Both devised architectures to deal with spatiotemporal data streams. In Kmoch et al. (2016), processes acting on sensor data were integrated and published with the OGC WPS interface in a SDI.

However, a fully-integrated architecture for the Geospatial IoT - one, that is based on a common spatiotemporal data type and communication mechanism - is only in the beginning stages of development. Contemporary architectures lack in integrating IoT devices and existing GIS technology. This thesis aims to analyze and describe how a more sophisticated architecture would look like, specifically one that functions within the networking parameters of restricted environments.

1.3 OBJECTIVES AND QUESTIONS

The Geospatial IoT is a system that integrates disparate actors such as smart objects, embedded IoT devices, their sensors, their spatiotemporal data streams, cloud infrastructures and web services. The goal of this thesis is to develop an appropriate architecture for the Geospatial IoT based on spatiotemporal data types and communication mechanisms that can be utilized by every device in the system. Several substantial requirements must be defined to both, the conceptual model and communication mechanisms to allow the participation of every object and device. These communication mechanisms should be based on advanced protocols and standards.

Within the architecture, communication mechanisms should be developed to enhance and optimize the dissemination of spatiotemporal messages by exploiting their meta information. Actors and systems should be able to prioritize messages based on spatial regions and points in time. Since the number of devices and number of spatiotemporal data streams can be massive in Geospatial IoT applications, the networking mechanisms must be constructed to handle concurrent participating clients on a huge scale. A high level of networking and the dynamic expansion of the Geospatial IoT require flexible properties such as a facile scalability. Different kinds of devices should be able to participate, such as resource-restricted devices (e.g. single-board computers or sensor nodes), and ordinary computers. This calls for a specific level of interoperability. Aside from a concept for the architecture, requirements for an appropriate communication mechanism must be derived. These requirements should be based on the theoretical background and challenges of the IoT. Finally, a prototypical implementation should be created. The following research questions regarding the concept of the IoT architecture can be raised:

- What is the spatial nature of things in the IoT?
- What are the spatiotemporal data types in the Geospatial IoT?

- How can an architecture employ these data types, what are the related concepts?
- In what form may actors specify their interest in spatiotemporal messages?

A prototypical implementation may ask for the following questions:

- What are the requirements to implement a communication mechanism for the Geospatial IoT based on the conceptual model?
- Which IoT protocol meets the requirements for our concept?
- How to implement the desired concept and functionalities?
- In which way can we evaluate the prototypical implementation?

This implementation should be evaluated regarding the derived requirements. The evaluation should investigate the capabilities of the developed communication mechanisms according to the applicability for the desired Geospatial IoT architecture, especially whether the mechanisms have the capabilities to interconnect smart objects of all kinds. At this, the question raises whether introduced spatiotemporal filtering capabilities can be exploited efficiently to optimize message dissemination between system participants. We focus further on the expressiveness of the conceived spatiotemporal data types for the Geospatial IoT as well as connectability and scalability of the proposed system.

An additional objective of this thesis features the linkage of Geospatial IoT devices, their data and the proposed architecture as well as the developed messaging mechanisms with established GIS technologies. Like mentioned in the motivation for this thesis, GIS concepts and techniques have been evolved with the technological development phases of computers, the Internet and the Web: from desktop-based over Internet GIS to geo web services and GIS applications in the cloud. The next logical step in this evolution is the integration of spatiotemporal data in real-time, which is induced by devices in the IoT. Thus, existing GIS technologies must adapt and bind the techniques that drive the IoT revolution. This holds for desktop GIS, but also distributed GIS applications such as geo web services. Therefore, we aim at investigating the extendability of these established geospatial concepts and technologies. The research deals with the capabilities of connecting desktop GIS and geo web services such as the OGC WPS interface for processing geospatial data with the proposed Geospatial IoT architecture. This leads to various questions, for instance:

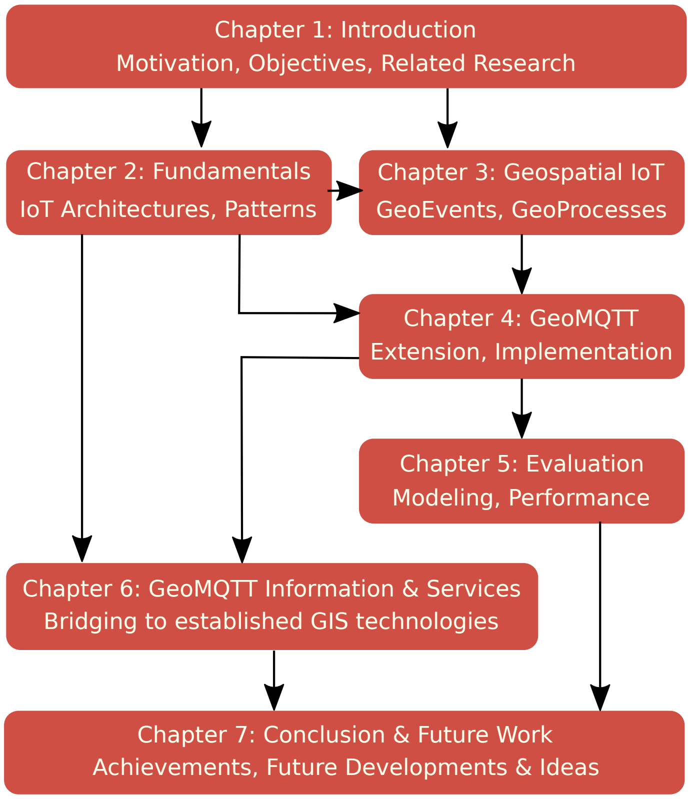
- Is a seamless integration of the Geospatial IoT in GIS technology desirable, practical and effective?
- How adaptable and flexible are existing GIS applications and standards to novel mechanisms of the Geospatial IoT?
- Can existing GIS technology cope with the amount and the velocity of spatiotemporal data and data streams of the Geospatial IoT?
- Are new innovative solutions required to manage the integration of GIS methods and the Geospatial IoT?

The thesis tries to approach these questions raised by the different research areas in Geospatial IoT platforms. The approach involves the implementation of a platform based on a conceptual architecture and conceived messaging models for the Geospatial IoT. Questions about the efficient integration of Geospatial IoT devices, geospatial computing capabilities and GIS technology are to be examined with the help of this platform.

1.4 THESIS STRUCTURE

Tackling the objectives and questions raised before, the outline of this thesis is inspired by the presented research agenda. Hence, it is structured into 7 chapters, which are conceptually arranged from theory chapters dealing with concepts of the IoT and the Geospatial IoT, implementation and evaluation parts of the proposed architecture and communication mechanisms, to an implementation chapter about integrating Geospatial IoT concepts in GIS technology. The interrelationships between the chapters are depictured by the diagram in Figure 1.2.

After the introduction (Chapter 1), which describes the motivation and objectives of this thesis, Chapter 2 focuses on the fundamentals of the IoT. First, the vision of the IoT is presented with its characteristics, drivers, challenges and disruptive opportunities. Since the fundamental building block of the IoT are Internet-connected things, we give also a profound view on the concept of smart things. This conceptual view is followed by specific research on IoT architectures and patterns. The four building blocks of IoT architectures are investigated. This includes (1) IoT devices, sensor and actuators, (2) M2M communication protocols in the IoT, (3) information and service models for the IoT and (4) visualization and analysis of IoT data. Due to the specific objectives of this thesis, the focus is on the communication patterns, mechanisms and protocols that are commonly used in IoT systems. It serves especially as a profound basis for the following two chapters. The third chapter (Chapter 3) develops the idea of the Geospatial IoT and transforms the fundamentals of the generalized IoT in


Chapter 1: Introduction
Motivation, Objectives, Related Research
Chapter 2: Fundamentals
IoT Architectures, Patterns
Chapter 3: Geospatial IoT
GeoEvents, GeoProcesses
Chapter 4: GeoMQTT
Extension, Implementation
Chapter 5: Evaluation
Modeling, Performance
Chapter 6: GeoMQTT Information & Services
Bridging to established GIS technologies
Chapter 7: Conclusion & Future Work
Achievements, Future Developments & Ideas

Source: Author's illustration

Figure 1.2: *Thesis outline and interrelationships between chapters*

a spatiotemporal context. The proposed architecture implements a feedback loop system from observation of real-world phenomenon, over analyzing situations and making decisions, to actuating based on these decisions. It is driven by events, which are first observed and subsequently distributed in the system. Thus, we explore the notion of spatiotemporal events and processes in the chapter. Especially, the spatial and the temporal components of events are discussed by investigating their nature and the relationships between events in both components. We define data types and concepts such as GeoEvent and GeoPipes for our proposed GeoEvent-driven architecture and conduct a requirement analysis for a suitable communication mechanism and protocol to implement a prototypical architecture for the Geospatial IoT. The implementation chapter (Chapter 4) first evaluates the different M2M communication protocols presented in Chapter 2 with respect to this requirement analysis. Subsequently, the best matching protocol (MQTT) and its core features are explained with more detail. To achieve the desired functionalities, the original protocol should be extended by specific functions, which is specified afterwards. We present the GeoMQTT extension by describing precisely the introduced new message types and mechanisms. By doing so, first the structures of the new data packets are depictured and, second, the introduced spatiotemporal filtering capabilities are described. Both rely on the conceptual architecture established in Chapter 3. Further, we give insights in the implemented GeoMQTT framework and corresponding software. Also, the extension GeoMQTT-SN for unreliable networks is described, which allows to connect sensor nodes in GSN to the GeoMQTT system. This covers especially the different introduced control packets, which aim for an as small as possible size. In the following evaluation chapter (Chapter 5) the proposed architecture in conjunction with the implemented protocol is assessed based on constructed scenarios in the Geospatial IoT. The evaluation serves to examine whether the requirements are met by the advanced protocol. We focus here on different aspects: first, the modeling capabilities of the introduced message types in GeoMQTT are investigate according to the nature of physical events and processes developed in Chapter 3. Second, the message types and their characteristics are compared to other common encoding and messaging standards with similar modeling capabilities, especially in terms of message size. Finally, the protocol is tested under load to investigate first the performance and efficiency of the proposed Geospatial IoT communication pattern and, secondly, the scalability of the whole system. In Chapter 6 the implemented Geospatial IoT information and service applications based on GeoMQTT are presented. This chapter focuses on the integration of Geospatial IoT data and devices with established GIS technologies. First, the integration of GeoMQTT in desktop GIS software (here QGIS) to enable the receipt of real-time events is implemented by a plug-in. Further, different bridges to existing architectures in the IoT are built: A RESTful access point to GeoMQTT provides the translation between an event-driven and a resource-oriented architecture, while an adapter for the Sensor Web Enablement (SWE) services offers the binding to a service-oriented architecture. The latter one can be used to request sensor measurements or sensor tasks by using HTTP-based services. Beside these bridges, an extension for the WPS, which can be used to offer geoprocesses as a service, is introduced. By integrating the GeoPipes concept in the service interface and its processes, the former static input and output data types are enhanced by data streams, so that they can process and issue continuous streams in real-time. In the conclusion chapter (Chapter 7), the achievements of this thesis are summarized as well as reviewed with respect to the postulated objectives and questions. Based on this premise, we develop and discuss further ideas to advance the proposed architecture for the Geospatial IoT.

CHAPTER 2

FUNDAMENTALS

Within the IoT vision, physical things are arranged in communication networks and connected to the Internet. These can be everyday objects or large machines. This chapter introduces the concept and history of the IoT briefly and points out its actors, challenges, opportunities and architectures. The latter one is discussed in detail to give insights in the four building blocks of IoT architectures. At this, the focus lies on the communication mechanisms, patterns and protocols in the IoT, since the practical part of the thesis relates on these concepts, techniques and technologies.

2.1 THE INTERNET-OF-THINGS (IOT)

In 1991, Mark Weiser published the article "The Computer for the 21st Century", in which he describes the seminal vision of future technological ubiquity (Weiser, 1991). It counts as the birth of the IoT vision. Weiser coined the vision of "ubiquitous computing", which describes a future, where computing devices will be replaced by intelligent objects in everyday life. In his article, he already envisioned the concept of smart homes and the technology of smart tablets, although still science-fiction during these times.

The term Internet-of-Things (IoT) was finally introduced in 1999 by Kevin Ashton, later co-founder of the MIT's Auto-ID Center. According to Ashton (2009), he used the term first as the title for a presentation at Procter and Gamble (P&G) when he linked the idea of Radio-Frequency Identification (RFID) to P&G's supply chain. His initial idea was to move the data gathering process from humans to computers, so that they can see, hear and smell the world for themselves (Ashton, 2009). He argued if computers knew everything, we could track and count everything, which would lead to a revolution in reducing waste, loss and costs.

Since then, the vision of the IoT began to emerge. Neil Gross predicted 1999 in an article published in the BusinessWeek, that the "earth will don an electronic skin", which "will use the Internet as scaffold to support and transmit its sensations". It consists of millions of embedded electronic measuring devices that will probe and monitor cities, species, atmosphere, highways, fleets of trucks or human bodies (Gross, 1999). During the next years, different developments supported this view, for instance, appliances such as refrigerators were connected to the Internet. In 2005 UN's International Telecommunications Union (ITU) published its first report on the topic shaping its vision (ITU, 2005):

> A new dimension has been added to the world of information and communication technologies (ICTs): from anytime, any place connectivity for anyone, we will now have connectivity for anything. Connections will multiply and create an entirely new dynamic network of networks – an Internet of Things

Nowadays, the IoT vision evolves steadily shaping whole industries (Industrial IoT (IIoT)) and everyday activities. Advanced concepts have been introduced by different domains. High-level concepts such as the Web of Things (WoT) or the Sensor Web are specializations or applications of the IoT. The WoT describes the idea of accessing surrounding devices through web applications (Duquennoy et al., 2009). This is similar to the vision of the Sensor Web, in which web services are offered to provide access to sensors, sensor networks and their data (Gibbons et al., 2003). Different services are implemented for discovery, access, alerting or planning. Thus, both WoT and Sensor Web rely on the IoT but also on web technologies especially the HTTP protocol. A closer look on these concepts is given in Section 2.5.

IoT as a Cyber-Physical System (CPS)

Other approaches introduce conceptual models for the IoT. For instance, the concept of Cyber-Physical System (CPS) is often used along with the IoT. CPSs describe the confluence of embedded systems, real-time systems, distributed sensor systems and controls. Computational capabilities are integrated in physical processes by embedded computers and networks monitoring and controlling physical processes. Feedback loops are usually utilized, in which physical processes affect computations and the other way around (Lee, 2008). However, the concept of CPS illustrates the theoretical foundation for complex and distributed system based on wired or wireless communication, but is not bound to a specific technology. Thus, some authors identify the IoT as a manifestation of a CPS (Ibarra-Esquer et al., 2017).

The concept of CPS follows a cyber-physical model of a control system, which can also be used to characterize data and control flows in the IoT. This model can be represented by a set of conceptual components (Zaborovsky et al., 2016), which is also known as the Observe-Orient-Decide-Act (OODA) loop originally invented by Boyd (1987):

1. *Observe*: Gathering information about the characteristics of the environment.
2. *Orient*: Analyzing the parameters of the current state of the controlled object.
3. *Decide*: Performing decision making processes to determine an appropriate course of action.

4. *Act*: Physical execution of decisions via actuation and observation of results, which restarts the loop.

In CPS, this feedback loop is implemented with embedded devices, sensors and actuators as well as computational capabilities: Sensors observe and communicate time-series observations to data centers for analysis. Analytical systems drive recommendations and perform decision making processes, whose results are looped back to sensor platform or actuators to improve the system or drive a physical process (Shukla & Simmhan, 2017). The four components use information exchange channels to interact with each other (Zaborovsky et al., 2016). In the IoT (as a manifestation of a CPS), these channels are established by the Internet protocol suite (see Section 2.3.2) and Machine-to-Machine (M2M) communication protocols on different layers in a communication stack (see Section 2.4).

WHAT IS DISRUPTIVE WITH THE IOT?

In the future, the IoT will be omnipresent and will impact our lives significantly (Lee et al., 2013). The evolution of the Internet towards the IoT will extend the number of endpoints of Internet communication enormously by embedded devices, sensors and actuators (Bonomi et al., 2014) and, thus, increase complexity. While in the traditional Internet of Computers (short IoC) with smartphones, PCs, laptops or tablets, a person is mostly behind the endpoints, in the IoT these are machines or complex systems. Therefore, Ibarra-Esquer et al. (2017) argue that IoC and IoT are two disjoint sets in an ecosystem of Internet-connected devices but they provide data and services for each other. For instance, devices of the IoC such as PCs use methods such as web services to access the devices in the IoT.

In IoT the endpoints are organized into systems embedded into large systems. Bonomi et al. (2014) illustrate this with the many sensors and actuators in a smart vehicle which communicate with each other, but also the entire vehicle communicating with other vehicles and, ultimately, with the Internet. The same holds for other domains such as smart cities or the industry 4.0. Thus, they call for a "system view" rather than an "individual view", which comprises a lot of consequences. Domain expert companies will appear in a new market that develop new systems and applications, which have specific requirements and characteristics.

IOT CHARACTERISTICS AND DRIVERS

The ensemble of all things connected to the Internet along with the underlying infrastructure of server form the IoT. Based on the mentioned concepts, we can derive the following fundamental characteristics of IoT (ITU-T, 2012; Lee et al., 2013):

1. *Interconnectivity*: Things are enabled to communicate with the IoT infrastructure and among each other.
2. *Things-related services*: The IoT infrastructure provides thing-related services such as sensor data request or actuating of a thing.
3. *Heterogeneity*: Devices in an IoT ranges from tiny sensors to mobile devices and large computers. They can interact with each other or with service platforms through communication networks.
4. *Dynamic change*: Like in the physical world, things connected to the IoT change their states dynamically. They can be in sleep mode or moving between places. Furthermore, the number of devices in an IoT also changes dynamically.
5. *Huge scale*: Depending on the point of view, the estimations of connected IoT devices to the Internet range from 20 billion (Gartner, 2017) to 50 billion (Ericsson, 2011) by 2020. Regardless of the true number, the IoT will influence humans' life with an enormous number of connected devices.

The evolution of the IoT is driven by different developments in several domains during the last decade. In the technical domain, these are the developments in M2M communication, sensor networks, respectively WSN or hardware innovations. However, other issues relate to the accessibility of measured data or devices. This can, for instance, be achieved by applying methods and services of the Sensor Web or the WoT (see Section 2.5.1). Innovations in the research area of Semantic Sensor Networks (SSN) facilitate the homogenization of sensors and sensor network (Wang et al., 2015). So, there is not a single discipline, which drives the evolution of the IoT.

2.1.1 IOT ACTORS AND SMART THINGS

Two key actors can be identified in IoT scenarios: *Users* interact with *physical entities* in the real-world (Serbanati et al., 2011). A *user* can be instantiated as a human person or a software agent. He has a specific goal whose completion depends on the interaction with entities in the physical world. The mediation is performed through the IoT infrastructure. A physical entity exists in the physical environment and can be of interest to users to satisfy a specific goal. It can be any object or any environment from living organisms to machines or from electronic appliances to closed or open environments.

HOW DO THE ACTORS COMMUNICATE IN IOT?

Since the actors can be human persons or different machines, the communication endpoints in the IoT are not homogeneous and the interaction mechanisms need to be adapted to the capabilities of the communicating entities. Therefore, depending on the instantiation of the actors, Lee et al. (2013) suggests distinguishing between two modes of communications for the IoT:

- Human-to-Object (or Human-to-Thing) Communication: Human beings communicate with objects (devices) to obtain information or to control the behavior of the objects.
- Object-to-Object (or Thing-to-Thing) Communication: Objects communicate with each other. An object or device delivers information to other objects with or without the involvement of humans. In the IoT this communication includes physical devices, logical content and resources. Thus, M2M is a subset of Object-to-Object communication.

WHAT IS A (SMART) THING?

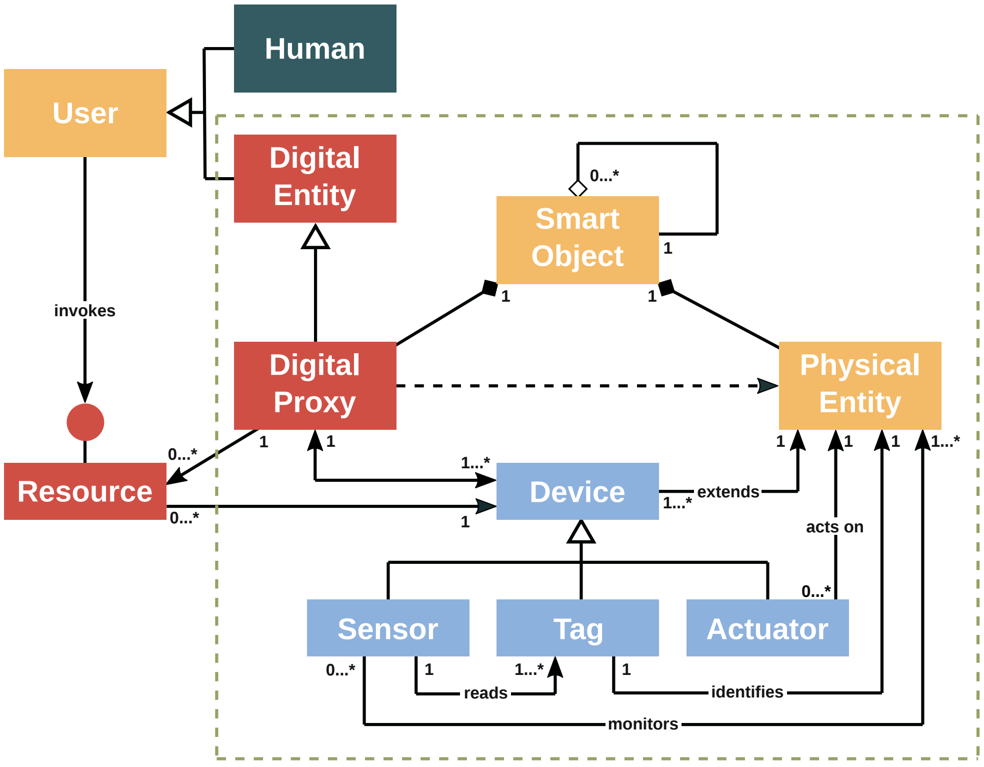
A *thing* in the IoT represents either an object of the physical world, a physical thing (or physical entity), or of the information world, a virtual thing. Whether physical, the thing is integrated into information and communication networks, is connected to the Internet and is identifiable. Van der Zee and Scholten (2014) point out that the "environment consists of physical objects (or things), the Earth's natural objects (trees, rocks, etc.) and man-made artificial objects, some of which are smart objects." These can be things connected to the IoT and helping to perform a certain goal. The term *smart thing* (or *smart object*) refers to things embedded with processors, sensors, software and connectivity functionalities and exchange data between the thing (itself) and its environments, manufacturers, users and other systems. *Smart things* have several features and capabilities. Basically, these are not based on single technologies, but multiple convergent technical lines of development contribute to novel functionalities. The capabilities, which can be found in *smart things*, are described in the following list (Mattern & Flörkemeier, 2010):

1. *Communication*: Things can communicate with the Internet or other things (Thing-to-Thing) in the IoT. They may update their state, send messages and use resources and services provided by servers.
2. *Addressability*: Things are detectable by discovery, lookup or name services and can be requested and influenced.
3. *Identification*: Objects are uniquely identifiable. This can be done actively or passively by using RFID technologies or bar codes in conjunction with a mediator such as a smartphone.
4. *Sensors*: Things are equipped with sensors to collect information about their state or environment. They forward measured data or react to these.

5. *Actuators*: Beside sensors, actuators can be connected to things to drive physical processes in their environment.
6. *Embedded processing*: Processors and microcontrollers may process measured data immediately.
7. *Localization*: Things in the IoT know their physical location, which is captured automatically using GNSSs or other positioning technologies.
8. *User interface*: Smart things provide mechanisms so that humans can interact with them (Human-to-Thing).

Although *smart things* (or *smart objects*) may embody these functionalities, it depends highly on the use case of the IoT application, which of the features are really implemented. For instance, smart things in an application aiming at monitoring the environment are probably equipped with sensors and use communication mechanisms for real-time monitoring and publishing of data, but do not need to implement user interfaces or actuators. Haller (2010) defines smart things as physical entities which are equipped with technical communication devices. According to Serbanati et al. (2011), a *smart object* is the extension of a *physical entity* with its associated *digital proxy*. Although smart objects depend on IoT devices with sensors, tags and actuators, Serbanati et al. chose a definition which is decoupled from the technology level. They define a reference model for smart objects by giving a relationship diagram between entities which is depictured in Figure 2.1.

The figure illustrates the IoT reference model by Serbanati et al. (2011) with the conceptual model of smart objects given in the dashed rectangle. Like mentioned, the actors in IoT are basically *users* on one side who interact with *physical entities* on the other side. The interaction happens through the smart object model. *Physical entities* are represented in the digital world with *digital proxies*, *digital entities* that are software entities such as agents, services or data entries. *Digital entities* can be viewed as *users* in the IoT context and may interact with other *users* including *humans*. In the model, *digital proxies* are bi-univocally associated to the represented *physical entities* and own a unique ID to identify it. Further, proxies are digital synchronized representations of a set of properties or aspects of the entity meaning that parameter are updated in the digital representation if changes happen to the *physical entity* and changes to the *digital proxy* could also manifest on the *physical entity* in the physical world. A *smart object* is the extension of a physical entity with its associated *digital proxy*. Since any changes in the properties of a smart object must be represented in the physical as well as in the digital world, one or more *devices* are embedded, attached or placed in vicinity to the associated *physical entity*. They can be tags to identify, sensors to monitor, actuators to act on the *physical entity* or a combination of these bundled into a platform. The *device* is necessary to mediate the interactions between *physical entities* and *digital proxies*.



Source: based on Serbanati et al. (2011)

Figure 2.1: *IoT reference model*

Users may use *resources* to interact with *smart objects*. These digital, identifiable components are associated to *digital proxies* and can be used to retrieve information of physical properties of the associated *physical entity* through sensors or modify them through actuators. Further, digital properties of the *digital proxy* can be retrieved and modified or, finally, complex hardware or software services offered by the *smart object* may be utilized. Actual access to *resources* is provided by *services*.

2.1.2 IoT Challenges and Opportunities

Before going into the different building blocks in detail, we will describe some challenges and opportunities, which arise with the concept of the IoT and its diffusion into everyday applications. The following list is based on Mattern & Flörkemeier (2010) and supplemented by a review of relevant challenges and aspects described in the literature.

Scalability

One of the main challenges in developing an IoT is the potentially vast number of physical entities and devices, which may be connected to the IoT. The current Internet consists of a comprehensible number of participants. But with the technical innovations such as miniaturization of computing power, the number of connected devices increases exponentially. Communication mechanisms in such a network with millions of participants must be efficiently implemented. The scalability of the network is a crucial requirement which must be met in IoT. Guinard et al. (2011) argues that for open and less constrained applications, massive scalability is necessary among other requirements.

Plug and play

Smart things should be connected to the IoT in a plug and play manner. Large configuration efforts inhibit the diffusion and development of smart things in an IoT, which is not desirable. Seamless integration and cooperation of interconnected things with applications work only, if plug and play capabilities are supported (ITU-T, 2012).

Interoperability

The physical world and its potentially connected objects in an IoT are extremely heterogeneous, from everyday analogous objects to specialized machinery. Consequently, the used hardware but also software technologies to establish connections to the Internet may vary from object to object. But, it must be ensured that the objects are able to intercommunicate with each other. Otherwise each IoT application evolves into isolated silos. Therefore, suitable communication standards need to be developed to ensure interoperability. Nastase (2017) states that interoperability should be kept in mind since the huge amount of IoT research in the last decade have led to large volume of new IoT products on the market. However, a fragmented marked consists of multiple vertical industries, which leads usually to domain or vendor specific closed systems. These "vertical silos" do not support interoperability and cause various problems e.g. in data exchange (Latvakoski et al., 2014).

DISCOVERY

The large number of things and their digital representations in the IoT require mechanisms to guarantee their identification and discovery. Interactions with objects are only possible, if they are known and findable. Mayer & Guinard (2011) point out that in the complete IoT vision millions or even billions of smart things will be linked to the Web. They expect that finding, selecting and using smart things in a fast, reliable and user-friendly way will become extremely difficult. That is why mechanisms are required to allow users or machines to discover smart things and understand their capabilities. For instance, (Jirka et al., 2009) suggest a Sensor Instance Registry (SIR) and a Sensor Observation Registry (SOR) enabling the discovery of sensor instances and sensor services as a Web service.

SOFTWARE COMPLEXITY

The heterogeneous hardware embedded in IoT things leads to a diversity of hardware and software products. For instance, embedded systems consist of restricted hardware resources, while servers that offer services and manage representations have access to basically unrestricted resources. We already saw that interoperability is a key issue in deploying open IoT systems. Because of the heterogeneity of the devices and servers, this issue becomes more complex. Standards for communications in IoT may be developed, however, the real challenge is to adopt these standards in every device.

DATA VOLUME AND INTERPRETATION

In the IoT the volume of unstructured and structured data grows exponentially with rising numbers of IoT devices and other entities. The data need to be handled in a sophisticated way: new approaches in storing, managing and processing are required. Big data analytics describe the process of searching a database, mining and analyzing large data sets. Marjani et al. (2017) investigate different analytic types and their usage in IoT applications. They conclude that the combination of IoT and big data analytics is compelling, but existing solutions of analytics are still in their early stages of development and need to be advanced for IoT data. For instance, simultaneously to the amount of data, the velocity of data e.g. measured by sensors increases tremendously as well. Tönjes et al. (2014) argue that the real-time demand of IoT applications challenges data interpretation. There is a gap in providing efficient and scalable methods that enable real-time processing and interpretation of streaming sensory data. Novel big data solutions such as the *Lambda architecture* (Marz & Warren, 2015) or the *Kappa architecture* by Kreps (2014) are required to analyze and interpret multiple high-frequency data streams.

SECURITY AND PRIVACY

Lin et al. (2017) define the security features as well as the security and privacy issues, which should be considered in IoT infrastructures. First, *confidentiality* ensures that only authorized entities have access to IoT devices and data. During communication, the transferred data must not be tampered by intended or unintended interference (*integrity*). *Availability* guarantees that data and devices are available for authorized entities (users and services) whenever they are requested. *Identification & authentication* is used to legitimate the data sent in networks and ensures that only authorized devices and applications can connect to the infrastructure. The concepts of *privacy and trust* must also be applied to IoT architectures. Furthermore, *security* mechanisms must protect systems from attacks such as node capture attacks performed on an IoT device or code injection attacks where malicious code is injected in a device or service to gain control, intercept communication or force an unorthodox behavior. These issues are already tackled in the conventional Internet and, thus, corresponding mechanisms are already developed. However, the high complexity of the IoT raises advanced security challenges.

FAULT TOLERANCE

The physical world is highly dynamic and changes occur frequently, sometimes unpredictably. Accounting this in IoT infrastructures, digital representations of physical entities and their connections among each other must be implemented reliably to guarantee a functioning system regardless of frequently changing contexts. Mobile things dynamically join and leave the IoT, generate and consume billions of parallel events, which increases the diversity of possible contexts. An IoT infrastructure must manage communication, storage and compute resources supporting important characteristics such as reliability, safety, survivability, and fault tolerance (Bonomi et al., 2014). Redundancies on different architectural levels should be introduced to meet the challenge.

ENERGY SUPPLY

Although many physical things are already electrified and tethered on an electrical grid, the majority of objects are unbounded to a location and lack of a steady power supply. Additionally, the integration of battery packs in very small objects is difficult. Autarkical power supply systems need to be developed to drive the IoT vision, so that potentially a huge number of physical objects may participate. Energy challenges can be tackled in different ways: First, energy consumption can be reduced e.g. by employing passive systems such as RFID (Gubbi et al., 2013), by using energy-efficient routing mechanisms between IoT devices (Kim, 2018) or by applying hardware and protocols which support sleep modes (Al-Fuqaha et al., 2015; Teklemariam, 2018). Secondly, concepts of energy harvesting can be utilized to collect energy from external sources such as solar power (Benoit et al., 2015) or kinetic energy (Gorlatova et al., 2015). For instance, the small amount of power provided by energy harvesters can be captured, stored and used by small, wireless autonomous devices based on low-power electronics.

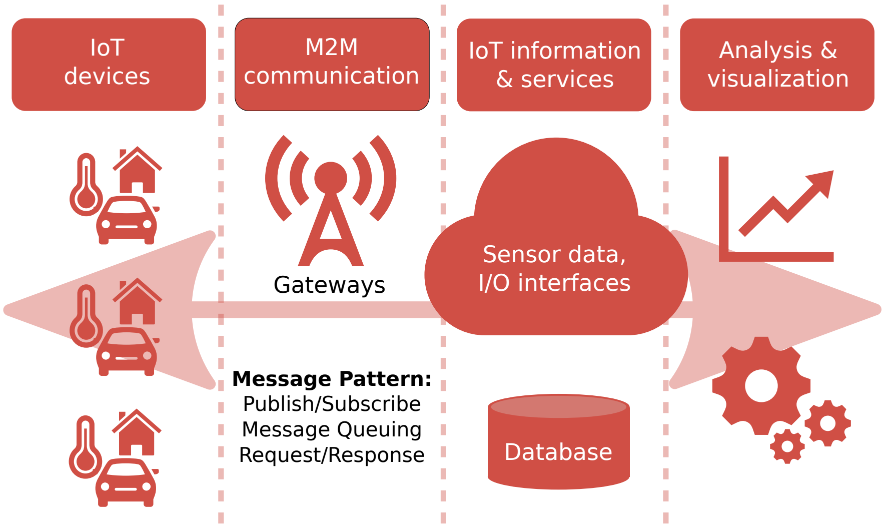
This list makes no claim to completeness but it shows some major challenges, opportunities and how to handle them. Beside of the mentioned ones, the cost factor also plays an important role. Since the IoT will have an enormous scale with billions of devices in the future, the technology should be affordable in purchase and operation.

2.1.3 IoT Architectures

Like already pointed out, an IoT architecture can be very diverse in its configuration based on the use case and the utilized technologies. Nevertheless, one can identify several building blocks in IoT architectures, so that a generic type of architecture can be derived. Basically, this generic architecture follows distributed systems which implement IoT applications. Based on Latvakoski et al. (2014), we subdivide the architecture into four central building blocks: (1) IoT devices, sensors and actuators, (2) M2M communication technologies, (3) IoT information and services and, finally, (4) IoT applications which include analysis and visualization. Figure 2.2 show the generic IoT architecture consisting of the four building blocks and their interactions.

The things are the basic components in the IoT and, thus, represent the first building block, which interacts with the physical world through sensors and actuators. We focus on this building block in Section 2.2 in detail. Things communicate with certain M2M communication mechanisms and protocols among each other but also with the servers in the Internet. Here, a so-called gateway may be necessary to translate between different protocols. For instance, in WSNs several connectionless protocols are used typically, since small sensor nodes might not be able to communicate with the Internet protocols directly. Section 2.3.1 and Section 2.4 deal M2M communications in detail. The data send by M2M communication mechanisms is stored, processed and made available through IoT information and services, which is covered in Section 2.5. Finally, Section 2.6 deals with analyses and visualization of IoT data, the application building block of IoT architectures.

The four building blocks interact with and depend on each other. Messages and data flow in both directions. For example, while measured data by sensors is send by IoT devices and stored in corresponding databases or visualized by applications, application may send remote commands through web services of the IoT service layer to devices to control an actuator and influence the physical world.



Source: based on Herle et al. (2018)

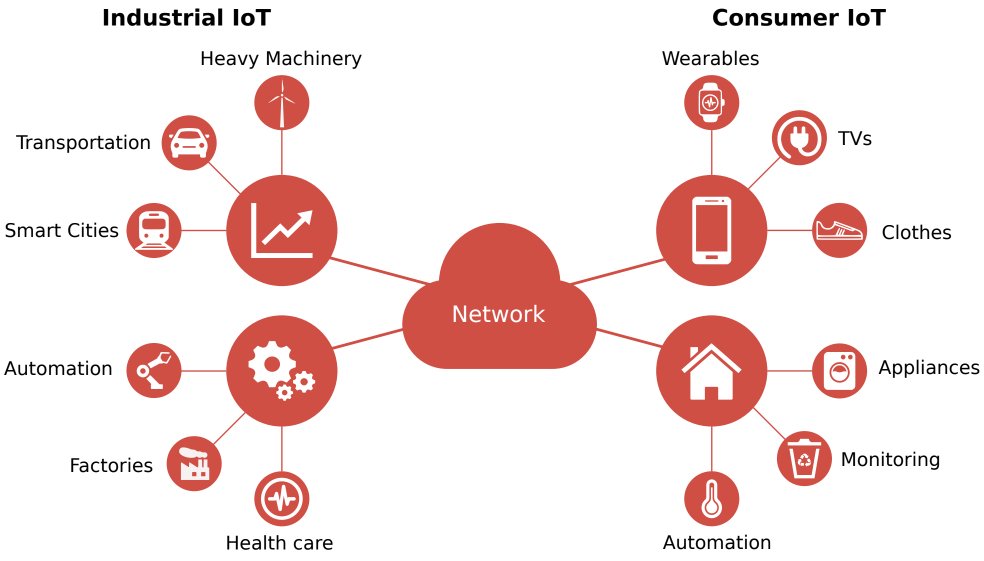
Figure 2.2: *Generic IoT Architecture*

2.2 IoT Devices, Sensors and Actuators

A thing in the IoT is of physical or virtual nature. Virtual things do not exist in the physical world but in the information world. These things can be stored, processed and accessed but do not exist physically. Application software, service representations or multimedia content belong to this group of things (Lee et al., 2013). Most of them are either already connected to the Internet or easily and quickly wired.

Physical things, on the other hand, are objects in the physical world and can be sensed, actuated upon and/or connected to. This means automatically that these things are connected to the Internet and, therefore, somehow electrified. Thus, electrical equipment such as industrial robots, consumer electronics, or most household appliances are by nature already prepared to be part of the IoT. Non-electric things need to be equipped with appropriate and embedded electronic systems to be able to participate in the IoT world. Conceptually, physical things can be grouped into the application fields of Industrial IoT (aka Industry 4.0) and consumer IoT, although there is naturally no sharp line between them (Palattella et al., 2016). Figure 2.3 shows this distinction broken down further into different domains.

Like implied, things can be huge in size and already electrified by nature such as cars or appliances, but can also be hitherto analogous objects such as sneakers or garbage containers. Some things are already connected to networks through



Source: Author's illustration

Figure 2.3: *Industrial and Consumer IoT*

M2M communication (e.g. heavy machineries), while others are newcomers in interconnecting with other systems. Depending on the type of IoT application (IIoT or Consumer IoT), requirements are very different. This holds for instance in terms of reliability and privacy of communication. Thus, the hardware and software configurations of things and their requirements vary as well.

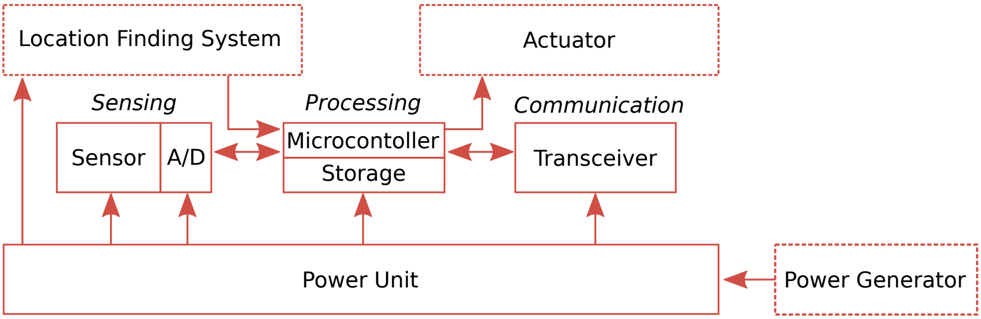
2.2.1 IoT Devices

The first application based on M2M communication and, thus, the first IoT applications were realized by the RFID technology. So technically, an IoT device may only consist of a tag which is attached to an object to provide its identity. In RFID a query signal is transmitted by a RFID reader towards the tag. The reader receives reflected signals which can then be used in conjunction with a database to identify the object. The RFID tag itself can either be passive, active or semi-passive/active (Al-Fuqaha et al., 2015).

Nevertheless, the IoT evolution is mainly driven by specific devices which can be embedded into everyday objects. These embedded computer systems are the fundamental technology for the IoT (Ibarra-Esquer et al., 2017), although it is not a new concept. Manley (1974) already introduced the term of embedded computers.

He describes embedded computers as information processors which are physically incorporated into electromechanical systems whose primary function is not data processing, but integral from a design, procurement and operations viewpoint. Embedded systems are implemented by using devices such as microcontrollers and single-board computers. Recently, prototyping platforms such as Arduino, Raspberry Pi or Beagle Bone, gained popularity to realize IoT projects in an affordable and easy to use way. These miniaturized computers and other technological developments such as 3D printers allow for rapid development which fosters also do-it-yourself communities (Makers clubs), Hackspaces and citizen science in general (Haklay et al., 2018).

Single-board computers are offered by several manufacturers and with different architectures. The main components of a single board computer are the microcontroller with a Central Processing Unit (CPU), a storage and a power unit. Figure 2.4 illustrates an architectural view of a single-board computer, which is used in a sensor network. Beside its main components, other possible units such as a sensing (with analog-digital converter (A/D)) and actuating unit, a communication unit and a location finding system are embedded.



Source: adapted from Akyildiz et al. (2002b)

Figure 2.4: *Single-board computer as a sensor node*

Regarding the processing unit, ARM-based systems are most prevalent and are e.g. developed by ARM Limited, Texas Instruments, Samsung or Ericsson. Competing architectures are, for instance, Microchip AVR with the ATmega series or x86 used by Intel's Atom or AMD's Geode. These available microcontrollers are built in popular prototyping platforms such as the Arduino family or several versions of the Raspberry Pi. Both platforms are shortly introduced in the following paragraphs.

ARDUINO

The Arduino is a physical computing platform involving Arduino boards and the Arduino IDE to program the boards in an easy way. At this, the term *physical computing* describes the activity of designing and creating interactive objects by using software and hardware. The systems can sense and control the physical world (O'Sullivan & Igoe, 2004). Physical computing includes using computing skills as well as crafting and using electronics (Przybylla & Romeike, 2014). Thus, it is used for rapid prototyping of interactive objects, which is e.g. applied in education or product design. Software and hardware are open source and freely available, so that other manufacturers may copy the design.

Table 2.1: *Physical-Computing Platforms*

	Arduino Uno	Arduino Due	Raspberry Pi	Raspberry PI 3 B+
Processor	ATmega 328P	ARM Cortex-M3	ARM11 1-core	ARM Cortex 4-core
Clock Speed (MHz)	16	84	700	1400
GPU speed	-	-	250 MHz	300/400 MHz
Bus width (bits)	8	32	32	64
System memory	2 kB	96 kB	256	1024
Flash memory	32 kB	512 kB	-	-
EEPROM	1 kB	-	-	-
Input/Output (I/O)	SPI, I2C, UART, GPIO	SPI, I2C, UART, GPIO	USB, SPI, I2C, DSI, UART, SDIO, CSI, GPIO	USB, SPI, I2C, DSI, UART, SDIO, CSI, GPIO

Source: Author's illustration

The Arduino boards are based on different microcontrollers. Most of the Arduino boards are equipped with ATmega microcontrollers, some newer boards use ARM-based microcontrollers. Table 2.1 shows the specifications of the Arduino Uno and the Arduino Due. While the Uno uses a ATmega328 with 16 MHz, the Due is based on an ARM Cortex with 84 MHz. Arduino Boards are extendable by so-called *shields*. These are attached to the boards and provide further interfaces. For example, several I/O shields offer wireless communication modules such as Bluetooth, WLAN or ZigBee, other extend the memory of the board or attach low cost GNSS modules for positioning.

RASPBERRY PI

With the market introduction of the first Raspberry Pi version in 2012, it becomes an immediate success in part due to the low price of approximately 35 USD. By connecting some peripherals (keyboard, mouse, SD card, monitor), a fully operating computer which runs on a Linux operating system can be obtained (Johnston & Cox, 2017). The Raspberry Pi is like the Arduino boards popular among prototype builders and hobbyists but also with the industry and benefits from the demand for embedded systems induced by IoT applications. Unlike the Arduino boards, the Raspberry Pi is solely based on multithreaded ARM technology. Since its potential operating systems are Unix-like, it supports also high-level programming languages such as Python.

In its first version, the Raspberry was equipped with an ARM11 1-core processor and 700 MHz clock speed (see Table 2.1). From there on, several successors were introduced to the market with the latest model "Raspberry PI 3 B+" in 2018. The newest version features an ARM Cortex 4-core processor and 1400 MHz clock speed which is a significant improvement in performance. Connecting sensors and actuators is provided by numerous I/O interfaces. Like for the Arduino boards, several shields can be mounted to increase the scope of interfaces.

Arduino boards and Raspberry Pi are utilized in a variety of projects, which interact with the physical world. By deploying also communication mechanisms to establish a connection to the Internet, the boards can be used to construct smart things. Johnston & Cox (2017) regard the Raspberry Pi and other low cost single-board computers as an enabler technology for the evolution of the IoT.

2.2.2 SENSORS AND ACTUATORS

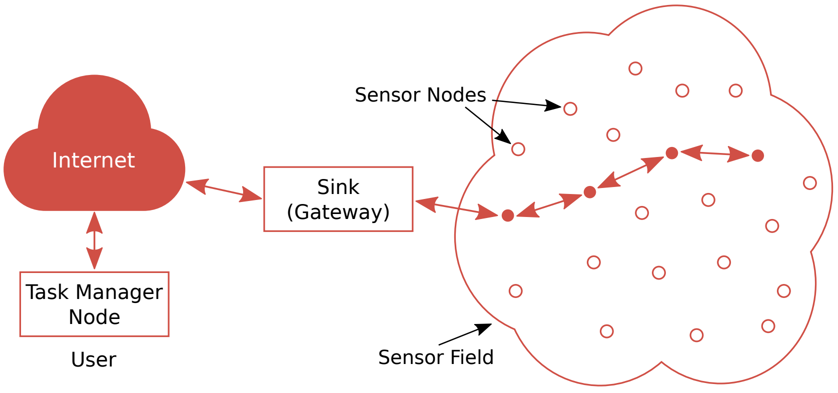
Although the first IoT application was solely designed for the identification of objects using RFID tags, nowadays sensors and actuators are connected to IoT devices to interact with the physical world. For this purpose, single-board computers offer digital and analog I/O ports (or pins). The product ranges of sensors and actuators are very wide. *Sensors* monitor physical entities and may provide information about them. They deliver information about the identity or measures of the physical state of entities and their surroundings. On the one hand, they can be attached or embedded in the physical structure of entities or, on the other hand, observe a specific environment, where they identify and monitor entities. The latter one holds e.g. in environmental monitoring. For instance, the PortoLivingLab project (Santos et al., 2018) deploys a city-wide platform for continuous environmental monitoring, which is named the UrbanSense platform. It is as a smart city application trying to measure and improve the quality of urban living. The sensors used in the environmental monitoring can be grouped into three classes: (1) Meteorological sensors: thermometer, hygrometer, wind vane, anemometer, rain gauge, luxmeter and solar radiation, (2) quality of life sensor: sound level meter and (3) air quality sensors: particulate matter detector, CO, NO_2, O_3, gaseous. The sensors are attached to a Raspberry Pi forming a data collection unit within the city of Porto. Basically, every imaginable sensor can be connected to IoT devices and, thus, be part of the Internet. E.g. the Spanish IoT company Libelium provides currently more than 120 sensors which can be connected to their Arduino-based sensor platform named Waspmote (Libelium, 2018). Available sensors range from the types of environmental sensors used in the PortoLivingLab, over water and soil sensors to sensors used in precision farming.

Actuators can drive physical processes in the physical world by acting. They modify the physical state of physical entities through moving them or activate/deactivate functionalities of more complex ones. For example in building automation systems, actuators control heating and cooling by maintaining the room temperature at a certain level. By connecting these systems to the Internet, smart buildings emerge whose actuators can be controlled by means of the Internet (Al-Fuqaha et al., 2015). Other actuators may help to drive a car by itself or help a door become open or shut (Isikdag & Pilouk, 2016). It is evident that actuators in industrial IoT applications (Industry 4.0) such as automation are likely to increase productivity (Wang et al., 2016).

2.2.3 SENSOR NETWORK (SN)

IoT devices can either directly communicate with the Internet, but often they are arranged in Sensor Networks (SNs), sometimes also called Sensor and Actuator Network (SAN) if actuators are involved (Serbanati et al., 2011; Tönjes et al., 2014). Typically, the term Sensor Network is used throughout the literature regardless of the deployment of actuators and/or sensors. We will stick to the common definition, so that SN describe a network that is "composed of a large number of sensor nodes that are densely deployed either inside the phenomenon or very close to it" (Akyildiz et al., 2002a). A *sensor node* can be equipped with sensors to monitor physical entities and events, and/or actuators to act on physical entities. The sensor nodes are usually scattered in a *sensor field* (Akyildiz et al., 2002b). Figure 2.5 shows sensor nodes deployed in a sensor field as well as the communication architecture of SNs.

Sensor nodes can be arranged in a sensor field with different topologies. For instance, nodes in a star topology may only communicate with a central controller, while a peer-to-peer topology allows sensor nodes to communicate directly with each other, so that ad-hoc and self-configuring networks can be formed (Yick et al., 2008). Depending on the applied topology, the sensor nodes in a sensor field may send and receive data from other sensor nodes, collect data and route data back to the *sink*. If the sink is connected to the Internet, then it is also known as the *gateway* to the Internet. It communicates with the *task manager node* and, finally, with a *user*. The communication between the sensor nodes and the sink depend on the applied technologies and use case. From a hardware point of view, a wired SN is sometimes reasonable in security-related applications, however, technological advances in wireless communications and electronics allow to develop Wireless Sensor Networks (WSNs), which are low-cost, low-power and can monitor a large area. Thus, WSNs have a significant improvement over traditional sensors and wired SNs (Akyildiz et al., 2002a). From a software point of view, several radio technologies are utilized, which are discussed in Section 2.4.1.



Source: based on Akyildiz et al. (2002b)

Figure 2.5: *Sensor nodes in a Wireless Sensor Network (WSN)*

A specialized application of WSNs are Geo Sensor Networks (GSNs) (or Wireless Geo Sensor Network (WGSN)), which can be loosely defined as a sensor network that detects, monitors and tracks environmental phenomena and processes in geographic space. Here, geographic space may range in scale from a room to highly complex dynamics of an ecosystem region (Nittel et al., 2004; Nittel, 2009). On multiple abstraction levels in GSN, the concepts of space, location, topology and spatiotemporal events are modeled. This ranges from the physical deployment of the sensors in a sensor field, its communication topologies, the nodes relationships to observed phenomena, to mobile sensors either being self-propelled or being attached to moving objects. In GSN, geospatial information is collected and analyzed. The sensor node can provide continuous streams of geospatially-rich information (Nittel et al., 2004). Based on these capabilities and characteristics, a variety of applications becomes possible. Examples are monitoring of structures (Glabsch et al., 2011; Herle et al., 2018; Alonso et al., 2018) or mass movements (Walter & Nash, 2009), environmental monitoring (Werner-Allen et al., 2006; Terhorst et al., 2008) or agricultural systems (Gutierrez et al., 2014).

2.3 INTERNET COMMUNICATION

Before taking a detailed look at the M2M protocols in the IoT, the conceptual communication models and patterns in distributed architectures are introduced in this section. M2M protocols implement some of these patterns with their advantages and disadvantages for communication in certain scenarios. Further, we give a brief introduction of the basic protocols of the Internet and, thus, the IoT. Therefore, this section can be regarded as a foundation for the following section, where we introduce the M2M protocols in the IoT.

2.3.1 MESSAGING PATTERNS IN DISTRIBUTED ARCHITECTURES

Basically, in distributed computing environments two interaction models dominate. First, the *synchronous interaction model* represents a pattern where a procedure (function or method) is invoked remotely from a caller which blocks and waits until the called code completes execution and returns a result. The calling system does not have the processing control, it depends on the return of control from the called system. In *asynchronous interaction models* the caller retains the processing control since he does not block and wait for the called code to return a result. This model often requires an intermediary layer to handle the exchange of requests such as a queue. In asynchronous interaction models, every participant retains its processing independence. Regardless of the state of other participants, they may continue processing (Curry, 2004).

Several synchronous and asynchronous messaging patterns exist but only a subset is suitable for applications in IoT. Besides the synchronization, patterns can be categorized further through the properties of *space and time decoupling*. *Space decoupling* means that the interacting parties do not need to know each other to communicate. In direct communication models without an intermediary layer, this cannot be achieved obviously. *Time decoupling* describes that both communicating parties do not need to be actively participating in the action at the same time.

Again, in direct communication, this is hardly applicable (Eugster et al., 2003). For instance, the pattern *message passing* represents the earliest and a low-level form of distributed communication where a receiver (or consumer) listens synchronously on a previously established network channel and a producer sends messages asynchronously through that channel. It is a direct communication between sender and receiver, thus, they are coupled in space and time. Since the sender does not wait for a response, it is decoupled from synchronization while the receiver is not (Eugster et al., 2003). On the other hand, indirect communication systems such as *message queuing* (see Section 2.3.1.2) require an intermediary layer and, thus, sender and receiver may be decoupled from space, time and/or synchronization.

The most important messaging patterns discussed for IoT systems are presented in the following. The protocols are classified in the dimensions of *space*, *time* and *synchronization decoupling* and their applicability in the IoT context.

2.3.1.1 Request/Response

Like described, the message passing pattern implements a direct communication mechanism between a sender and a receiver of a message. Hence, it realizes a one-way communication. However, communication between distributed entities often requires two-way interaction. This holds e.g. for a program which calls a function and expects a returning value or an application executing a query and receives a query result (Hohpe & Woolf, 2003).

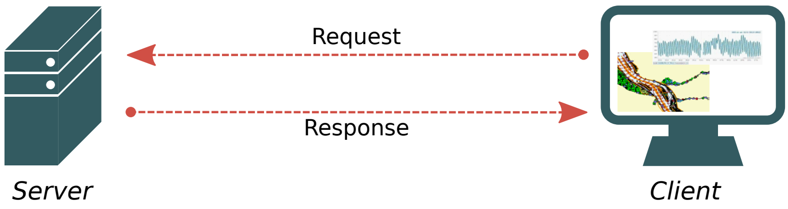
The *Request/Response* pattern (sometimes also *Request/Reply*) describes a two-way direct communication pattern between two participants. These are named

1. *Requester*: Sends a request message and waits for a response message.
2. *Responder (Replier)*: Receives the request message, processes it and replies with a response message.

Since it represents a direct communication, requester and responder are tightly coupled in space and time. The requester needs to know the location (responder) to send the message to (Eugster et al., 2003). Synchronization decoupling depends on the used methods and the desired behavior. The first applications for the request/response pattern were Remote Procedure Calls (RPCs) for procedural languages and Remote Method Invocations (RMIs) for object-oriented contexts to facilitate distributed programming. Originally, RPC and its derivatives describe synchronous interaction models. A requester, which calls a procedure or method by RPC (or RMI), blocks its execution, waits for the reply message and processes the response. Thus, requester and responder are not decoupled from synchronization. However, Eugster et al.

(2003) argue that several implementations avoid the tightly coupled synchronization by introducing either a one-way modifier in the request to indicate that the requester does not expect a response (which is basically a fallback to the message passing pattern), or a mechanism to process the response later in another thread so that the caller thread is not blocked. The latter one is known as *asynchronous callback* and decouples the two participants from synchronization. In this case, the sender sets up a callback for the response of the request which is executed when the reply message arrives. The sender thread is not blocked because a separated thread listens for reply messages and invokes the corresponding callbacks (Hohpe & Woolf, 2003).

The request/response pattern with asynchronous callbacks is heavily used in today's Internet and web applications. The standard protocol in the Web is HTTP (see Section 2.4.2.1) which implements a request/response pattern. Thus, it is also highly relevant for the IoT. Figure 2.6 illustrates the request/response pattern in the Internet.



Source: Author's illustration

Figure 2.6: *Request/Response mechanisms*

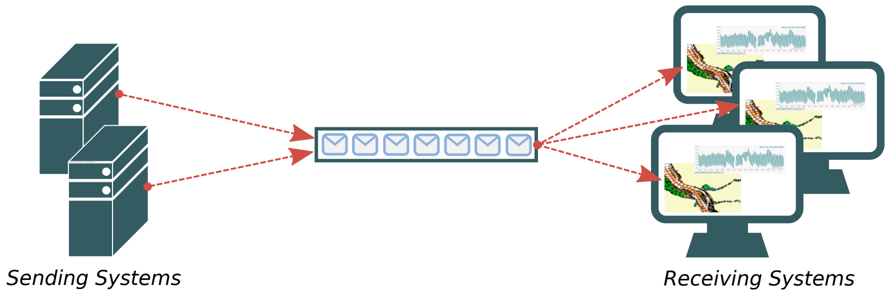
In the pattern the requester which sends the request is also called *client*, while the responder that receives the request, processes it and sends a response back is called the *server*. A server, which provides e.g. web pages, is known as a web server. Similarly, a client in the Web is called a web client. The web client pulls a website from a server, thus request/response represents a pull technology. Pulling is defined in Definition 2.1.

Definition 2.1 (Pull communication)
Pulling describes the consumer's mechanism to poll the provider to check for any messages or data.

2.3.1.2 Message Queuing

The *Message Queuing* pattern is an asynchronous interaction model which introduces an intermediary layer to exchange messages between participants in the system. A queue is used to store messages published by sending systems (see Figure 2.7). Receiving systems pull the message queue to check for messages. If a message is available the polling system removes it from the queue and processes the message. The standard queue, which can be found in messaging systems, is the First-In First-Out (FIFO) queue. It works like the name suggests: the first message sent to the queue is the first message which is retrieved from it (Curry, 2004). A queue has several attributes such as a name for addressing, a size or a sorting algorithm.

Like shown in the figure, the queue is located between the sending systems and the receiving systems. Therefore, the pattern realizes an indirect communication mechanism, which provides decoupling of space and time of the participating systems. Further, the producer-side is also decoupled from synchronization. But since the consumers synchronously pull messages, the message queue does not fully provide synchronization decoupling (Eugster et al., 2003).



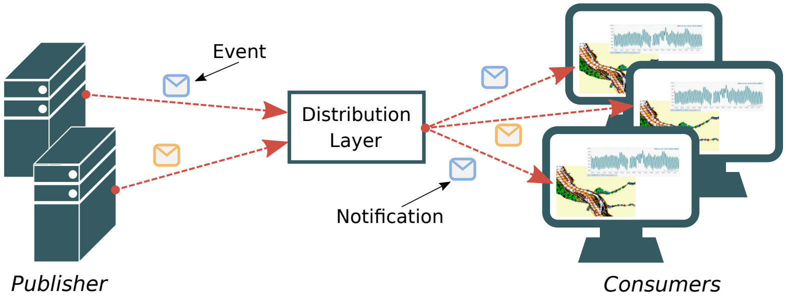
Source: Author's illustration

Figure 2.7: *Message Queuing*

Message-centric approaches such as message queuing can be used to implement Message-Oriented Middleware (MOM). A MOM provides distributed communication based on the asynchronous interaction model. Participants are not required to block execution and wait for a message. But unlike in request/response, sending participants have neither a guarantee that their messages are read by others nor about the time the message delivery will take (Curry, 2004). These are not determined by the producers but by the receiving applications. For instance, a reliable queue or a QoS mechanism can be used to ensure message delivery (Maheshwari & Pang, 2005; Sachs et al., 2009).

2.3.1.3 Publish/Subscribe

Another communication mechanism to implement MOMs is described by the *Publish/-Subscribe* pattern. It finds wide acceptance when a one-to-many interaction style is required in distributed systems (Oki et al., 1993). The pattern connects producers and consumers of messages through an intermediary layer. Consumers may subscribe to whatever kind of message they are interested in. At this, *subscribing* denotes the act of sending a *subscription*. Further, producers may *publish* messages anonymously and asynchronously. If the message published by the producers match a subscription, the intermediary distribution layer *pushes* the corresponding data to the consumers (Eugster, 2007). The publish/subscribe pattern is depicted in Figure 2.8.



Source: Author's illustration

Figure 2.8: *Publish/Subscribe interaction scheme*

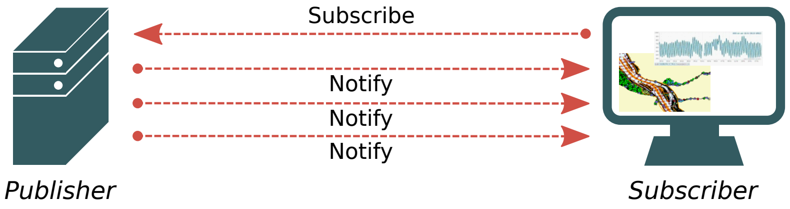
Like shown in the figure, while a message issued by a publisher is called an *event* in publish/subscribe systems, the act of delivering is denoted by the term *notification* (Eugster et al., 2003). Multiple consumers can be notified with the same event depending on their subscriptions, which is conceptually different from message queuing. In the figure the blue event is send simultaneously to two consumers. Consumers are also called *subscribers*, while publisher are also named *generators* of events. The intermediary distribution layer in MOMs are often referred to as *Message brokers* or simply *broker* (Curry, 2004).

Because of the indirect communication channel between producers and consumers of messages, they are decoupled in space and time. This means, on the one hand, that producers of events do not need to know the consumers. On the other hand, publishers might publish events even if the consumers are disconnected and, the other way around, consumers might receive notification even if the original producer of the event is disconnected (Eugster et al., 2003). Furthermore, consumers and producers are fully decoupled from synchronization. Unlike the message queuing pattern, notifications are *pushed* to consumers (see Definition 2.2), so that they are asynchronously notified, while performing some concurrent activity. Thus, both, production and consumption of events might happen in an asynchronous manner.

Definition 2.2 (Push communication)
Pushing describes the procedure, where the provider sends relevant messages to the consumer as soon as the provider receives them. Consumers instruct the providers in advance to push relevant messages to them.

Publish/subscribe mechanisms can be realized with different subscription mechanisms. According to Eugster (2007), in the first publish/subscribe systems, consumers could specify their interest by subscribing to a group. Since these groups appeared also under the term "topics", the underlying pattern is denoted *topic-based publish/-subscribe* (sometime also *subject-based publish/subscribe*). A topic T is a group T which consumers join by subscribing to it. If a producer publishes an event on topic T, the notification is broadcasted among the members of T. In these systems, participants can publish events as well as subscribe to individual topics which are keywords. Topics are mapped to distinct communication channels and are an event service offering the operations `publish()` and `subscribe()`. Usually, the topics are ordinary strings which are compared to the strings given in subscriptions but additional improvements extend this topic-based scheme. For instance, several systems use *hierarchical addressing* to subdivide and organize topics. A consumer subscribed to a specific node in the hierarchy, also receives events which are marked with subtopics of that node. For the M2M protocol MQTT this mechanism is described in detail in Section 4.2.

Besides topic-based, the pattern can also be implemented by content-based or type-based publish/subscribe. *Content-based publish/subscribe* describes a publish/subscribe pattern where subscribers may choose different filtering criteria along multiple properties. The information contained in each event is defined in an *event schema*. A content-based subscription is evaluated against that event schema and the content of the event. Hence, content-based publish/subscribe describes a more general approach and can be used to implement the original topic-based pattern (Banavar et al., 1999). In *type-based publish/subscribe* events are objects which can be of different application-defined types. Eugster et al. (2001) argue that by defining event objects and match them with types, a closer integration of object-oriented languages and the middleware is enabled. The type-based subscription respects type safety. Consumers may subscribe to a specific event type, which can also be hierarchical structured with subtypes. The subscriber receives all events of that particular type and its subtypes (Eugster et al., 2001). However, if the publish/subscribe pattern is integrated into object-oriented programming languages, passing objects to other languages is an interoperability issue.



Source: Author's illustration

Figure 2.9: *Notification scheme*

2.3.1.4 Notification (Observer Pattern)

The publish/subscribe pattern can also be realized without an intermediary layer, so that a direct communication between event publisher and subscriber is possible. This mechanism is called *Notification*, or *Observer* pattern, and provides a limited form of publish/subscribe without decoupling of space and time. However, synchronization decoupling still holds. The pattern is illustrated in Figure 2.9.

Subscribers register their interests such as a change in a resource directly with the publisher. The publisher manages the subscriptions and connections to the subscribers by itself. Whenever e.g. a resource a subscriber is interested in changes, the publisher notifies the corresponding subscriber with an event in a push-based manner.

2.3.1.5 Data Streams

In IoT, data streams are important communication mechanisms. A data stream itself is not communication pattern such as request/response or message queuing, but the previously described communication patterns can be used to initiate data streams. A data stream is defined as a sequence of data tuples (see Definition 2.3).

Definition 2.3 (Data Stream)
A data stream S can be defined as an unbound sequence of tuples p_j of the form $p_j = (a_1, a_2, ..., a_n, t)$ where a_i denotes attribute i, and t a common timestamp. The frequency and ordering of arriving tuples p_j of S is unpredictable and can change over time.

From the nature of the different communication patterns presented before, not every pattern is suitable for realizing data streams. Typically, patterns based on the synchronous interaction model and a pull communication cannot be used for data streams. Consider a producer of data that wants to initiate a data stream with a request/response pattern. After sending a tuple of a data stream to the server, the producer would block and wait until a response for his request arrives. So, basically producer and consumer of a data stream should be decoupled from synchronization. From the patterns introduced before, this holds especially for the publish/subscribe as well as the observer pattern. In parts, it also holds for message passing and message queuing. However, since the consumer is not decoupled from synchronization (e.g. in message queuing the consumer has to actively poll the queue), a fully push-based data stream from producer to consumer cannot be realized.

2.3.2 INTERNET PROTOCOL SUITE

Basically, IoT means connecting things to the Internet by using appropriate protocols and communication patterns. The protocol stack of the Internet is defined through the Internet protocol suite, which is also known as TCP/IP model since the foundational protocols in the suite are TCP and IP. IoT devices can connect to the Internet either by implementing a TCP/IP protocol stack on their own or through gateways like in SN. The TCP/IP model is illustrated in the middle column of Figure 2.10.

OSI model	TCP/IP model	Implementations
Application (7)	Application	HTTP, FTP, SMTP, etc.
Presentation (6)		
Session (5)		
Transport (4)	Transport	TCP, UDP
Network (3)	Internet	IPv4, IPv6
Data Link (2)	Link	Ethernet, Token Bus, etc.
Physical (1)		

Source: based on Gessler & Krause (2015)

Figure 2.10: *Layers of OSI reference model, TCP/IP model and protocol examples*

Figure 2.10 assigns the layers of the TCP/IP model to layers of the Open System Interconnection model (OSI model), which is a conceptual communication model for telecommunication or computing systems (see ISO standard in ISO/IEC JTC 1 (1994)). The OSI model consists of seven layers and was developed based on the less detailed TCP/IP model. The TCP/IP model's lowest component layer is the *link layer* which consists of a group of methods and communication protocols that operate on the link a host is physically connected to. It is responsible for transmitting data packets between two different hosts on the same link. Different wired or wireless link layer protocols can be used such as Ethernet or protocols of the IEEE 802 family (see Section 2.4.1).

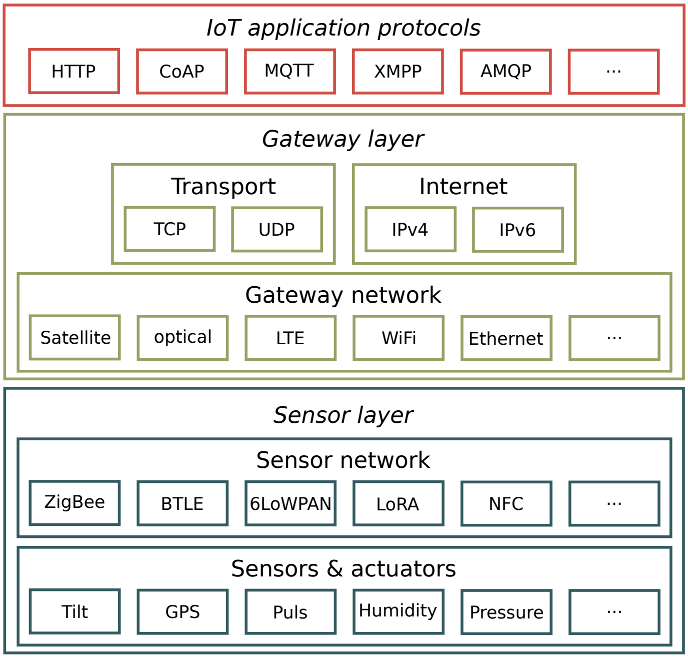
The *Internet layer* and the *transport layer* are the core components in the TCP/IP model. The Internet layer is responsible for addressing and routing of datagrams through the network. The layer determines and forwards the received packets to reach the next location in the network. Mainly the Internet Protocol (IP) is used in the layer. On the other hand, the transport layer performs host-to-host communication on the same or different hosts and on the same or remote networks. The Transmission Control Protocol (TCP) is used for reliable and connection-oriented data transmission, while the User Datagram Protocol (UDP) is the unreliable counterpart providing a connectionless datagram service for the benefit of reduced latency. It depends on the application and its requirements if TCP or UDP is applied. For a detailed view on the IP, the TCP and the UDP protocol, the reader may consult e.g. Tanenbaum & Wetherall (2013).

Finally, the *application layer* covers protocols, which work with application software and use the network infrastructure to exchange application specific data. Thus, it implements the so-called process-to-process communication. Protocols in the application layer are, for instance, HTTP for the Web or SMTP for exchanging emails. In the IoT M2M application layer protocols are applied, which are discussed in detail in Section 2.4.2.

2.4 M2M COMMUNICATION STACK

The direct communication between devices by wired or wireless communication channels is called M2M. It refers to different protocols and technologies on different layers on a communication stack. Since in the IoT every device can possibly speak to each other, M2M is a building block and essential in the IoT vision. It enables the flow of data between machines but also between machines and humans ultimately (Lee et al., 2013). But like the term suggests, M2M does not need human intervention, the transfer of information and commands between two machines is independent and automatically induced.

However, the domain of M2M communications is quite wide and not every standard or protocol is usable for applications in the IoT. Given the Internet protocol suite for basic interaction in the Internet (see Section 2.3.2), various protocols on the link layer and the application layer can be chosen. On the link layer, suitable wired and wireless communication technologies should be applied to ensure the connectivity. IoT devices, which are deployed in WSNs, have different requirements and, thus, use different mechanisms and protocols to communicate potentially through a gateway with the Internet. Other devices implement a TCP/IP stack by themselves and, therefore, have access to the Internet directly. So, IoT architectures might become quite complex with various layers. Figure 2.11 illustrates an architecture with three main layers: a sensor layer, a gateway layer and an application layer. Each layer includes corresponding M2M protocols.



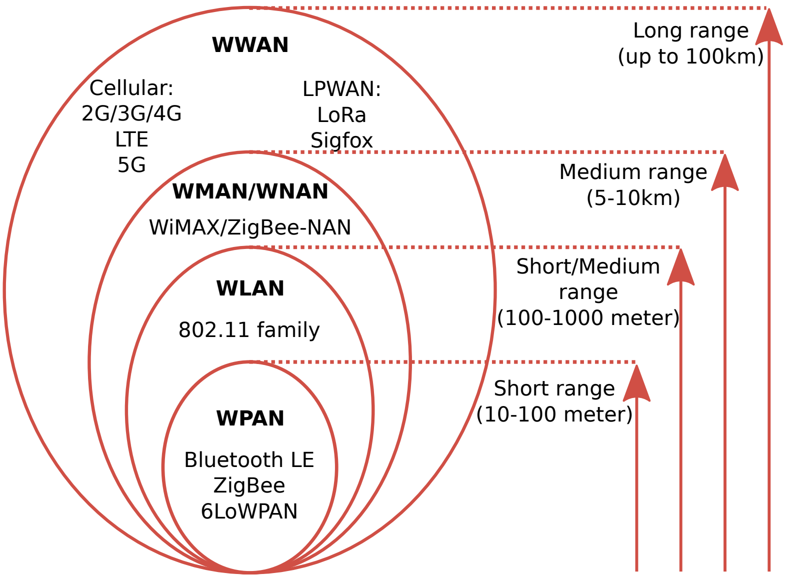
Source: adapted from Ray (2018)

Figure 2.11: *M2M protocol stack in IoT*

Wireless communication protocols are applied in the sensor layer. Some important ones are presented in detail in Section 2.4.1. The gateway layer consists of the used gateway network and the associated protocols to establish a connection to the Internet, which is mainly IP and TCP respectively UDP. If an IoT device provides a network connection and can use a TCP/IP stack, the gateway layer might be merged with the sensor layer. Finally, the IoT application protocols cover the high-level communication mechanisms (application layer) which are employed in the IoT to enable the communication between things. They are introduced in detail in Section 2.4.2.

2.4.1 WIRELESS COMMUNICATION IN THE IOT

The sensor and the gateway layer utilize several protocols to ensure Internet connectivity. Like described in Section 2.2.3, the connectivity of sensor nodes in WSNs rely on different radio protocols to transfer the data to a gateway which is directly connected to the Internet. The protocol overview in Figure 2.11 already introduced a sensor layer and associated protocols to communicate between sensor nodes in a sensor field as well as a gateway. Further, the gateway layer utilizes various protocols to establish an Internet connection. Like mentioned, sensor nodes can also directly communicate with the Internet omitting the gateway layer if suitable hardware is applied and an Internet protocol stack is in place.


WWAN
Cellular: 2G/3G/4G LTE 5G
LPWAN: LoRa Sigfox
WMAN/WNAN
WiMAX/ZigBee-NAN
WLAN
802.11 family
WPAN
Bluetooth LE ZigBee 6LoWPAN
Long range (up to 100km)
Medium range (5-10km)
Short/Medium range (100-1000 meter)
Short range (10-100 meter)

Source: Author's illustration

Figure 2.12: *Key wireless M2M technologies for the IoT*

Nodes in sensor networks as well as things in the IoT often require wireless technologies that provide connectivity for IoT applications. Wireless communication utilizes different radio frequency (RF) signals between 9 kHz and 300 GHz. RF are electromagnetic waves, which consist of an electric and a perpendicular magnetic field traveling at the speed of light. The oscillation of the waves determines their frequency f which is measured in Hertz (Hz). The wavelength λ is the distances, which is traveled by one complete cycle and, thus, inversely proportional to the frequency. Hence, lower frequencies have longer wavelengths. The amplitude indicates the strength of the RF signal. Amplitude or frequency can be modulated to transmit analogous data: amplitude modulation (AM) or frequency modulation (FM). Digital signals are transmitted by modulation techniques such as Frequency Shift Keying (FSK) or Phase Shift Keying (PSK) (Gessler & Krause, 2015). RF transmitters have a specific Transmitter Power Output (TPO), which is the actual amount of power measured in watts or decibel-milliwatts (dBm). Depending on the radio waves (frequency and power) and modulation, the applied protocols vary in power consumption, physical antenna size, travel distance, data size and so on. So the needs and requirements to the wireless communication technology are determined by the IoT use case. For instance, if massive amounts of data have to be transferred, a high-bandwidth solution is needed and, thus, a higher frequency and power. If, on the other hand, long distances have to be traveled, a lower frequency must be chosen.

According to Medina et al. (2017), the coverage of wireless technologies is the most important designing parameter of a communication network for IoT solution. However, parameters such as error rate, transmission rate or energy consumptions are also important depending on the use case and differ from protocol to protocol. Thus, different wireless M2M protocols and standards on the link layer have been developed to provide connectivity. Figure 2.12 orders and compares some standards and functional protocols by the working range. Additionally, the protocols are grouped into the wireless network categories Wireless Personal Area Network (WPAN), WLAN, Wireless Metropolitan Area Network (WMAN)/Wireless Neighborhood Area Network (WNAN) and Wireless Wide Area Network (WWAN).

2.4.1.1 Cellular Networks

Cellular networks (or mobile networks) are WWANs communication networks whose cells are distributed over the land area. Each cell is supplied by a cellular radio tower equipped with transceivers to transmit voice, data or other contents. The cells are joined together to form a cellular network to provide radio coverage over wide geographic areas. Transceivers such as mobile phones, laptops or IoT devices may communicate with each other through this network. Telecommunication providers have established and operate cellular network for transmitting voice and data in most countries of the world. Different standards have been developed over time increasing the capacity and speed of cellular networks:

The GSM[1] standard was introduced in 1990 as the first digital standard, which is used for telephony and packet switching data transmission. It is also called "2G" since it describes the second generation after analogous cellular networks. It operates in different frequency bands: for instance, GSM 900 uses the 900 MHz frequency band,

[1]Global System for Mobile Communications

which results in a data rate of 9.6 kbit/s. GPRS[2] and EDGE[3] (2.5G) were introduced based on the GSM standard to increase data rates to 57.6 kbit/s, respectively 384 kbit/s. The 3G standards (UMTS[4] and HSPA/HSDPA/HSUPA[5]) are the next generation of cellular networks. They provide data services for videotelephony and mobile broadband Internet access with a data rate of up to 7,2 Mbit/s (Gessler & Krause, 2015). Subsequently, LTE[6] together with LTE-Advanced and LTE-Advanced Pro are the fourth generation (4G) of mobile telephony. It uses a different radio interface and other improvements to increase data rate up to 600 Mbit/s (Sauter, 2015). Finally, the fifth generation of cellular mobile communications (5G) will succeed 4G, 3G and 2G in the future. These coming cellular standards target at high data rates, reduced latency, energy saving, cost reduction and massive device connectivity (Palattella et al., 2016).

Cellular networks can be used to directly connect IoT devices to the Internet (Karagiannis et al., 2015). But due to their inherent complexity and energy consumption, they are currently less suitable for peripheral IoT nodes (Zanella et al., 2014). Furthermore, the number of things in the IoT is expected to be massive, which limits the applicability of traditional cellular networks (Augustin et al., 2016). However, Palattella et al. (2016) argue that by the design principles of the coming cellular network generation, 5G is ready to enable the vision of a truly global IoT. They conclude that 5G will constitute an essential enabler of an IoT roll-out shaping new business models by transforming the cellular value chain.

2.4.1.2 WLAN

For the different wireless network categories, the standardization body *802* of the IEEE standards association (D'Ambrosia, 2018) develops standards for the physical (layer 1) and the data link layer (layer 2) in the ISO/OSI reference model (see Figure 2.13). The data link layer is further subdivided into the media access control (MAC) layer responsible for accessing the transmission medium and the logical link control (LLC) layer that multiplexes various network protocols for transmission by the MAC layer.

The wireless variants of the Ethernet protocols (802.X standards) are developed and specified in the 802.11 standard family. The original Ethernet or LAN standards describe a complete Internet protocol stack by wire with TCP/IP (see Section 2.3.2).

[2]General Packet Radio Service

[3]Enhanced Data Rates for GSM Evolution

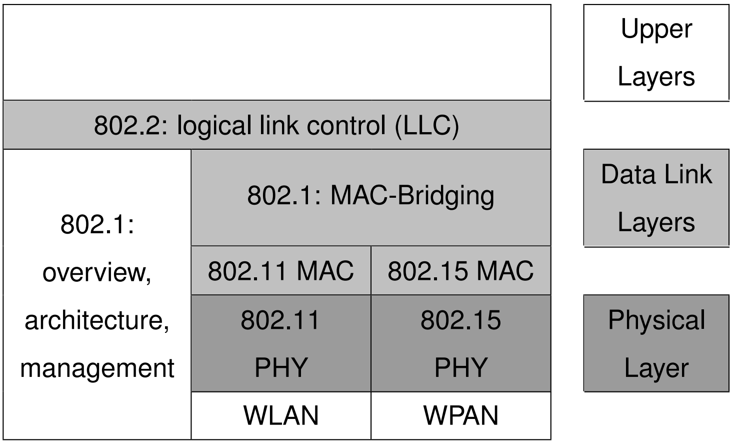
[4]Universal Mobile Telecommunication System

[5]High Speed Packet Access/High Speed Downlink Packet Access/High Speed Uplink Packet Access

[6]Long Term Evolution

In comparison to the wired LAN standards, the main derivations for the WLAN protocols were made in some management operations in the MAC layer and a complete redevelopment of the physical layer. Since over the course of time advancements were made in RF technologies, multiple physical layers are defined in sub specifications. Although the first standard 802.11 was specified in 1997, the breakthrough and wide application of the WLAN technology was achieved first with the introduction of the extension 802.11b in 1999 (Sauter, 2015). The extension exploits the 2.4 GHz frequency band, known as Industrial, Scientific and Medical Band (ISM) which is public domain in most countries, and a maximal TPO of 100 mW. Theoretically, data rates of 11 Mbit/s are reachable between sender and receiver traveling short distances from 10 to 20 meters, however, bad conditions reduce the rate to 1 Mbit/s since redundancies have to be integrated. Further versions of the 802.11 standard improved the data rate by different modifications. The most recent version 802.11ac occupies the 5 GHz ISM band and with improved channel bonding, modulation and other approaches, data rates of 6.93 Gbit/s are possible (Sauter, 2015). Depending on the deployment site (e.g. outdoors or indoors), the 802.11ac protocol offers a range of about 100 meters under ideal conditions (Gessler & Krause, 2015).

The WLAN standard and its different versions define the physical and the data link layer in the OSI reference model. Because of the potentially high data rates, the upper layers may directly implement the Internet protocol suite. However, the high power consumption disqualifies the WLAN standard for WSNs consisting of small and resource-restricted devices. The following protocols such as protocols based on the 802.15.4 standard are better suited for these setups.



Source: based on Gessler & Krause (2015)

Figure 2.13: *IEEE 802 standards in the ISO/OSI model for WPAN and WLAN*

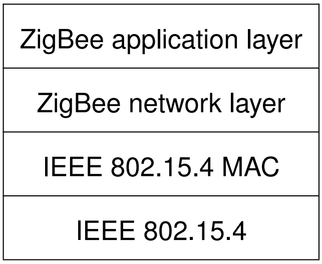
2.4.1.3 WSN Protocols

802.15.4

In 802.15 several standards for WPANs are specified. The IEEE 802.15.4 is a widely used protocol by WSNs. It specifies the physical and data link layers for small devices with limited communication capabilities (Alonso et al., 2018). Unlike 802.11 it focuses on exchanging small packets with simultaneously a low energy consumption. Thus, it can transmit data with 250 kbit/s in the 2.4 GHz ISM band or 20 kbit/s in the 868 MHz ISM for Europe, respectively 40 kbit/s in the 915 MHz band for North America. As will we see later, 802.15.4 has been established as the main physical and link protocol for WSNs mainly because of the low power consumption. Hence, several WSN protocols are based on IEEE 802.15.4. standard such as ZigBee or 6LowPan.

ZIGBEE

The ZigBee standard was developed and released by the ZigBee Alliance in 2004 aiming at cost-effective home automation applications. In 2007 for industrial environments, ZigBee PRO was released focusing on application with low data rate and low power consumption. Since it uses the IEEE 802.15.4 in the physical and the data link layers with the mentioned frequencies and features, it is designated to be applied in WSNs of limited size. Thus, it allows a maximum communication speed of 250 kbit/s (in the 2.4 GHz band) and maximal distances between sender and receiver up to 50 meters (Alonso et al., 2018). The protocol stack of ZigBee is illustrated in Figure 2.14.



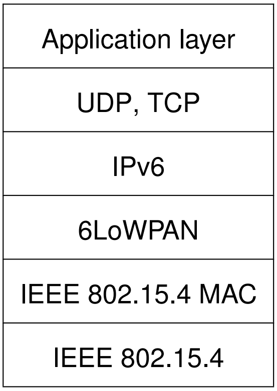
Source: based on Alonso et al. (2018)

Figure 2.14: *ZigBee stack*

Like mentioned, the IEEE 802.15.4 is used in the lower layers. The ZigBee network layer is responsible for the formation of the network through node discovery mechanisms, the allocation of addresses and the routing of the messages (Alonso et al., 2018). With this layer, ZigBee supports decentralized network topologies such as meshes for failsafe implementations. If sensor nodes break down, alternative paths through the network can in case be found. The application layer implements an application framework consisting of an Application Support Sublayer (APS), ZigBee Device Objects (ZDOs) and manufacturer-specific "application objects". Each device in the network has a digital ZDO which determines the role in the network (coordinator, router or end device) and initiates the sublayers. The application objects are the actual use-case specific objects, while the APS represent the interface between the network layer and the object (Gessler & Krause, 2015).

6LoWPAN

IPv6 over Low-power WPAN (6LoWPAN) is an open standard which is defined in RFC 6282 by the Internet Engineering Task Force (IETF) (Hui & Thubert, 2011). The standard enables the use of IPv6 over IEEE 802.15.4 in the 2.4 GHz band (Olsson, 2014) and, thus, allows the direct interaction between the Internet protocol and small devices, which was the initial idea behind the development (Alonso et al., 2018). Therefore, 6LoWPAN is an adaption layer between the network layer and the data link layer. The stack is shown in Figure 2.15.



Source: based on Olsson (2014)

Figure 2.15: *6LoWPAN stack example*

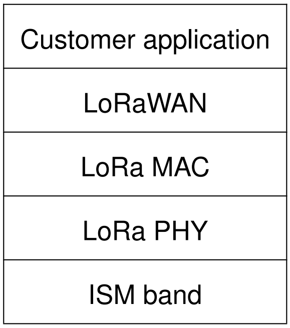
This adaption layer is integrated to optimize IPv6 performance over the IEEE 802.15.4 standard. A compression format for IPv6 headers and payload is established to support low-power constrained networks. A so-called border router is attached to the WSN and transparently performs the conversion between IPv6 and 6LoWPAN packets. This way, IoT nodes in a 6LoWPAN network can interact with any IPv6 host in the Internet through the border router (Zanella et al., 2014).

BLUETOOTH & BLE

Bluetooth was first proposed by Ericsson in 1994 for short distance wireless communications to substitute wired connections between portable and/or stationary devices (Gessler & Krause, 2015; Alonso et al., 2018). Currently, the IEEE 802.15.1 task group addresses the Bluetooth technology. The specification ensures high security, but simultaneously low power consumptions and low cost. It operates in the 2.4 GHz ISM band and uses different modulation techniques to reduce interferences with other protocols. Since version 1.0, the standard is consequently being evolved up to the most recent version 5.0. The advancements of the newer versions cover interesting features for IoT application such as improvements in energy consumptions. With Bluetooth Low Energy (BLE) an especially energy-saving variant was developed. However, Bluetooth is designed for small distances in WPANs up to 10 meters in early version, and theoretically up to 100 meters in version 5.0.

LORA & LORAWAN

Long Range (LoRa) is a patented physical layer protocol, which is developed by the LoRa Alliance. Manufacturer of the chips and patent owner is Semtech USA. It operates in the license-free ISM bands, especially 433 MHz, 868 MHz (Europe) and 915 MHz (North America). It allows for very long range transmissions (> 10 km) with low power consumption, thus, it is a Low Power Wide Area Network (LPWAN) protocol. The stack of LoRa is shown in Figure 2.16.



Source: derived from Bouguera et al. (2018)

Figure 2.16: *LoRa & LoRaWAN layer stack*

LoRa PHY is the layer which provides the physical transmission between end devices and the gateway. It uses a proprietary spread spectrum modulation scheme based on Chirp Spread Spectrum (CSS) modulation (Semtech Corporation, 2013). Resistance to inferences is the main advantage of this modulation technique. In spread spectrum techniques the signal is spread in the frequency domain, meaning that the signal gets a wider bandwidth. A *chirp* impulse is a sinusoidal signal which increases (up-chirp) or decreases (down-chirp) in time with respect to the bandwidth. For instance, an up-chirp signal consists of up-chirps that start at the lowest frequency and increases to the highest frequency of the frequency band before the next up-chirp continues the signal. In LoRa PHY these chirp signals are the carrier signals where a message is encoded on. For transmitting data, the chirps are cyclically-shifted resulting in frequency jumps that defines how information is encoded (a LoRa symbol). A LoRa symbol consists of 2^{SF} *chips*, where SF is the Spreading Factor ranging from 7 to 12, and cover the entire frequency band (Augustin et al., 2016). Since LoRa allows for error correction, some of the SF bits are used for redundant information (Casals et al., 2017). Depending on the SF, the physical bit rate, the transmission range and the power consumption differ. Higher SFs have a longer range but a lower physical bit rate. In the physical message, a LoRa MAC message is transmitted. The layer implements some useful mechanisms such as acknowledgements, receive windows or retransmissions (Casals et al., 2017).

LoRa is the related modulation in the physical layer for LoRaWAN (Long Range Wide Area Network), which is a wireless communication technology for connecting end devices with a network server. End devices and gateways form WSNs in star-of-stars topology (Augustin et al., 2016). LoRa PHY is used to communicate between gateways and end devices with very long ranges, while a LoRaWAN message is encapsulated in the message. The gateways receive the messages and forward the LoRaWAN message with a TCP/IP stack to the network server. Thus, LoRaWAN describes a system architecture rather than a single protocol.

OTHERS WSN PROTOCOLS

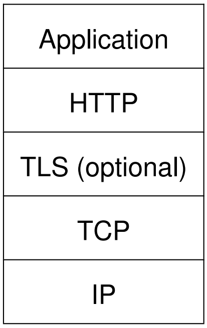
Besides the introduced protocols for WSNs, a variety of other protocols exists. For very short ranges, *Near Field Communication (NFC)* is a communication technology which enables interaction between two NFC devices in few centimeter ranges. It is specified in ISO norm 18092:2013 (ISO/IEC JTC, 2013). In home automation, protocols such as KNX or Z-wave are used by different manufacturers. For instance, *Z-wave* operates also in the 868 MHz ISM band with FSK modulation, which provides a communication range of around 30 m in buildings but with a data rate of about 10 kbit/s. Typically, it can be used to remotely control heating, ventilation, air conditioning or lightning (Gessler & Krause, 2015). Another LPWAN protocol and main contender to Long Range Wide Area Network (LoRaWAN) is *SigFox* also operating in the 868 MHz ISM band with an average coverage of 30-50 km in rural and 3-10 km in urban areas including obstacles. Its data rate is 100 bit/s and, therefore, targets at applications that have low data flow requirements such a smart parking, smart meter or environmental monitoring (Teklemariam, 2018).

2.4.2 IoT Application Layer Protocols

The IoT application layer connects sensor nodes and/or gateways utilizing the Internet with applications. Application layer protocols update servers and end-points with the latest IoT device values (e.g. sensor measurements), but also provide capabilities to carry commands from applications to e.g. actuators deployed in devices (Karagiannis et al., 2015). These protocols of the fourth layer in the TCP/IP model sit on top of the Internet protocol suite for transport and network (see Section 2.3.2). In the following, we discuss some communication protocols, which are seen in the literature as potential candidates for the IoT application layer (Karagiannis et al., 2015; Chen & Kunz, 2016; Nastase, 2017; Yokotani & Sasaki, 2017). They have various features, layer stacks and utilize different communication patterns, whose concepts were already introduced in Section 2.3.1.

2.4.2.1 Hypertext Transfer Protocol (HTTP)/HTTPS

One of the application protocols is the well-established and -known Hypertext Transfer Protocol (HTTP), which was initially standardized by the IETF and the W3C in 1996 (Berners-Lee et al., 1996), with its latest version HTTP/2 in 2015 (Balshe et al., 2015). Besides other protocols (e.g. Simple Mail Transfer Protocol (SMTP) or File Transfer Protocol (FTP)), it has been applied for communicating in the Internet and is the foundation of data communication in the WWW. It can also be used for IoT communications.

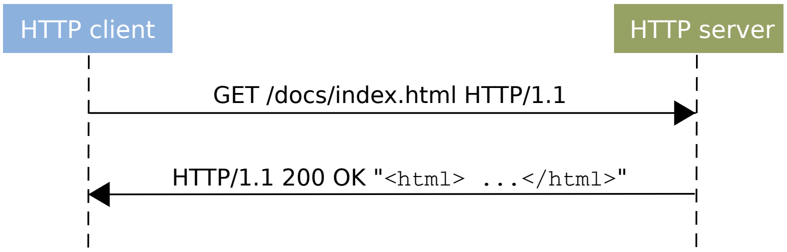


Source: Author's illustration

Figure 2.17: *HTTP protocol stack*

HTTP communication presumes an underlying and reliable transport layer protocol. Therefore, it generally takes place over TCP/IP connections with a default port of 80. However, it can also be implemented on top of any other protocol in the Internet or on any other networks (Berners-Lee et al., 1996). The typical layer stack for HTTP in the WWW is shown in Figure 2.17. Using an optional Transport Layer Security (TLS) layer for encryption, which constitutes as Hypertext Transfer Protocol Secure (HTTPS), the default port changes to 443.

HTTP implements a request/response protocol for the client-server computing model. Traditionally, the server provides a resource (e.g. a website), which is identified and located with a Uniform Resource Locator (URL). The client submits a request message to the server indicating the requested resource. The server returns a response message with completion status, information about the request and the requested content in its message body.


HTTP client
HTTP server
GET /docs/index.html HTTP/1.1
HTTP/1.1 200 OK "<html> ...</html>"

Source: Author's illustration

Figure 2.18: *HTTP request/response interaction scheme*

Figure 2.18 shows a HTTP request performed by a client. The depicted code shows the request line with the used method (*GET*), the requested resource (*/docs/index.html*) and the used version of the protocol. Header fields, which contain more information such as the host address or the user-agent, are omitted in the figure. Since the client uses the GET method for the request, a body is not necessary. The server handles the request and returns the requested resource, in this example a HTML website. The response status line consists of the used protocol version, a status code (*200* for OK) and a reason phrase (*OK*). Headers (omitted here) include e.g. the content type and content length of the response body. The body itself carries the requested resource, here the requested website (*<html>...</html>*).

Besides the GET method, HTTP defines also other methods. This includes the POST method, which encloses an entity in the request body as a subordinate of the resource, or the HEAD method which is similar to the GET method but only returns the header of the requested resource. Further methods are: PUT to store the enclosed entity under the supplied request URL and DELETE to request the server to delete the resource identified by the URL.

Since HTTP/2, the protocol is extended with a server push mechanism which allows server to send resources to a client before the client requests them. This is useful to reduce traffic for unavoidable request, for instance, if a website consists of multiple files (e.g. style sheets). However, HTTP/2 Server Push is not a notification mechanism like described in Section 2.3.1.4 and, thus, cannot be used for real-time messaging.

By using HTTP, a so-called RESTful architecture (Representational State Transfer (REST)) based on the Web can be implemented. Fielding (2000) developed this architecture style in his doctoral thesis. The style embodies various principles to access a resource in a distributed application, which is uniquely identified and linked to by an URL.

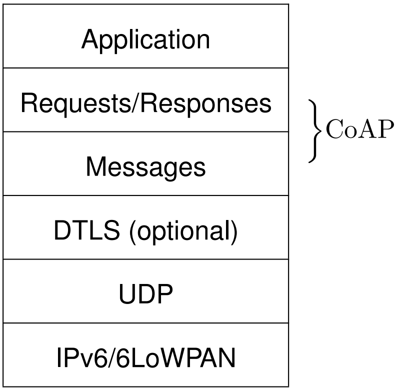
HTTP & IoT

The REST style is highly applicable in the IoT vision to access smart things, as we discuss later (see Section 2.5.2). However, although HTTP is widely used in the Internet, for M2M communications in the IoT, it has crucial disadvantages. Yokotani & Sasaki (2017) state that in IoT, HTTP must transfer large number of tiny packets. But because of the protocol's overhead, it forces high consumption of network resources and large delays. Further, Naik (2017) points out that compared to other IoT protocols the message size of HTTP is quite large and it requires high power and resource consumption. Further, with the request/response mechanism, HTTP is not suitable for real-time messaging and data stream initiation.

2.4.2.2 Constrained Application Protocol (CoAP)

Constrained Application Protocol (CoAP) is a proposed standard by the Internet Engineering Task Force (IETF), which was published in June 2014. It is a specialized web transfer protocol for use with constrained nodes and constrained networks. It is designed for M2M applications and suitable for IoT environments (Shelby et al., 2014).

Like HTTP, CoAP supports a request/response interaction model for application endpoints. It includes key-concepts of the Web such as URLs and Internet media types, so that it can be interfaced with HTTP easily. Still, it meets specialized requirements such as a low overhead and, thus, adaption to constrained environments is facilitated. The goal of CoAP is to realize a subset of REST communication common with HTTP but optimized for M2M applications. Unlike HTTP, CoAP is mainly developed for applications based on UDP as transport layer protocol, but can also be used with TCP, SMS and so on. Datagram Transport Layer Security (DTLS) may be used as a security layer (see Figure 2.19).



Source: based on Shelby et al. (2014)

Figure 2.19: *Abstract protocol stack of CoAP*

RESOURCE IDENTIFIER

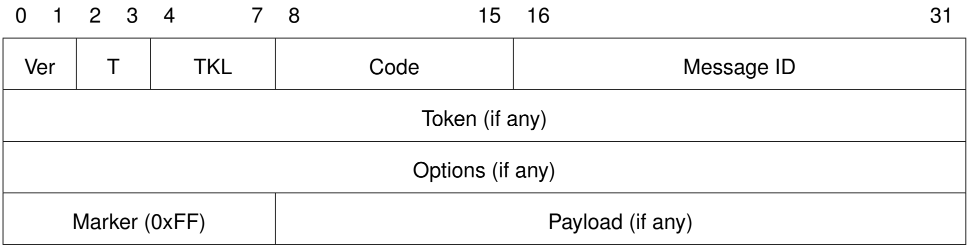
Since CoAP is mainly developed to interoperate with HTTP through proxies, CoAP uses "coap" and "coaps" Uniform Resource Identifier (URI) schemes for identifying resources. These schemes can be compared to HTTP URI schemes. The CoAP URI scheme looks like the following.

```
"coap:" "//" host [ ":" port ] path-abempty [ "?" query ]
```

Like in HTTP, the `host` component must be provided as an IP-literal or a registered name whose address can be found by using a name resolution service (e.g. DNS). The default `port` 5683 can be changed by indicating a custom UDP port. A resource is identified by the `path` associated with the host and the port. The path may consist of a sequence of path segments which are separated by a forward slash. Optionally, the resource can be further parameterized by using arguments in a `query` section. The arguments are Key-Value Pairs (KVPs) encoded and separated by an ampersand character. The "coaps" URI scheme is similar but uses a `coaps` as scheme name and has a default `port` of 5684 (Shelby et al., 2014).

MESSAGE FORMAT

A CoAP message consists of a fixed length binary header (4 bytes), followed by compact binary token, options and a payload.



Source: based on Shelby et al. (2014)

Figure 2.20: *CoAP message format*

Ver is the version number of the used CoAP protocol.

T specifies the message type, which can be *Confirmable (0b00)*, *Non-confirmable (0b01)*, *Acknowledgement (0b10)* and *Reset (0b11)*.

TKL determines the length of the token field in bytes if there are any tokens in the message.

Code is an 8-bit unsigned integer, which is split into 3-bit class and 5-bit detail. The class indicates a *request (0)*, a *success response (2)*, a *client error response (4)* or a *server error response (5)*.

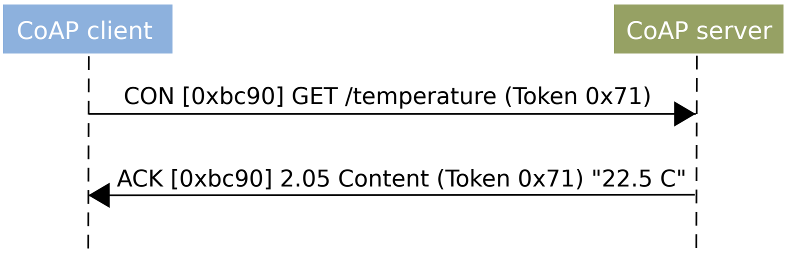
Message ID is a 16-bit unsigned integer and is used to detect message duplication and to match messages when they need acknowledgement.

The **Token** value may be 0 to 8 bytes, as indicated in the TKL field. This value is used to correlate requests and response. The token is followed by zero or more **Options**. Subsequently, the **Payload** joins, which is prefixed by a payload marker (`0xFF`).

MESSAGING MODEL

The CoAP messaging model is divided into two layers: (i) a messaging layer and (ii) a request/response layer (see Figure 2.19). The messaging layer is responsible for the exchange between two or more endpoints. It sits on top of the UDP protocol and must compensate its reliability drawbacks. This includes reliability and coordination mechanism, congestion control as well as duplication detection. Still, the 8 byte header of UDP is much more efficient than using TCP. The request/response layer is similar to a RESTful communication with HTTP. A client may request a resource on a server with different methods, which is identified by a URL. The header of the messages carries the REST information such as the method (GET, POST, PUT, DELETE), the response code and other options (e.g. content-type). Since it is similar to HTTP, a simple proxy can be implemented which translates between a HTTP REST and a CoAP REST system.

The messaging model of CoAP is simple. For instance, a client can send a confirmable (CON) GET request to a URL such as *temperature* to request a resource (see Figure 2.21). The message also includes a message id (*0xbc90*) and a token (*0x71*).

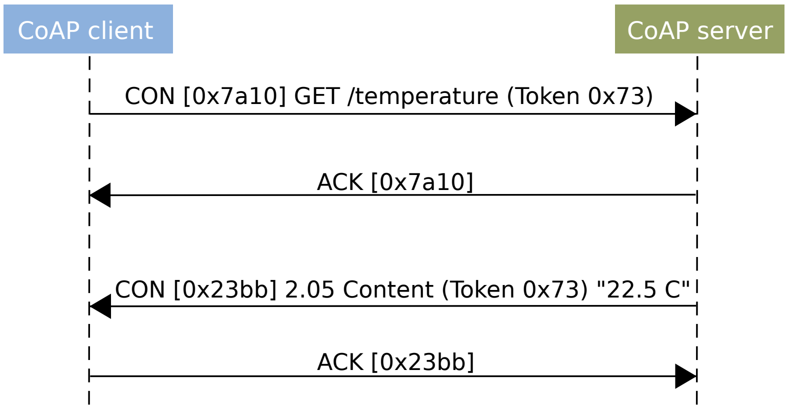

CoAP client
CoAP server
CON [0xbc90] GET /temperature (Token 0x71)
ACK [0xbc90] 2.05 Content (Token 0x71) "22.5 C"

Source: adapted from Shelby et al. (2014)

Figure 2.21: *CoAP GET request with piggybacked response*

Since a CON message was send, the client expects an acknowledgement (ACK) or a reset (RST) message. In the example, the CoAP server responds with an ACK message which the same message id, a status code (*2.05 content*) and the payload as a resource representation. If packet loss occurs - for instance, the server never receives the CON message - the client might repeat the CON message with the same message id after a timeout triggers. The example in Figure 2.21 shows a so-called piggybacked response since the ACK message is expected immediately. However, if the request requires more time to e.g. calculate the response, the server might send an empty ACK message without a payload. Later, the server can send a separate response which is a CON message that matches the token of the original request. The client confirms with an ACK message (see Figure 2.22).

An interesting option in CoAP is the *observe* feature which introduces a notification mechanism (see Section 2.3.1.4) in the REST architecture (see Figure 2.23). The client requests a resource by its URL with a CON GET request. By setting the *Observe* option to 0 (register), the client signals the server its interest in every update to the resource's representation. The server acknowledges the request immediately with a message including the current state of the resource and an observe flag. Whenever the resource changes its representation, the server notifies the client with


CoAP client
CoAP server
CON [0x7a10] GET /temperature (Token 0x73)
ACK [0x7a10]
CON [0x23bb] 2.05 Content (Token 0x73) "22.5 C"
ACK [0x23bb]

Source: adapted from Shelby et al. (2014)

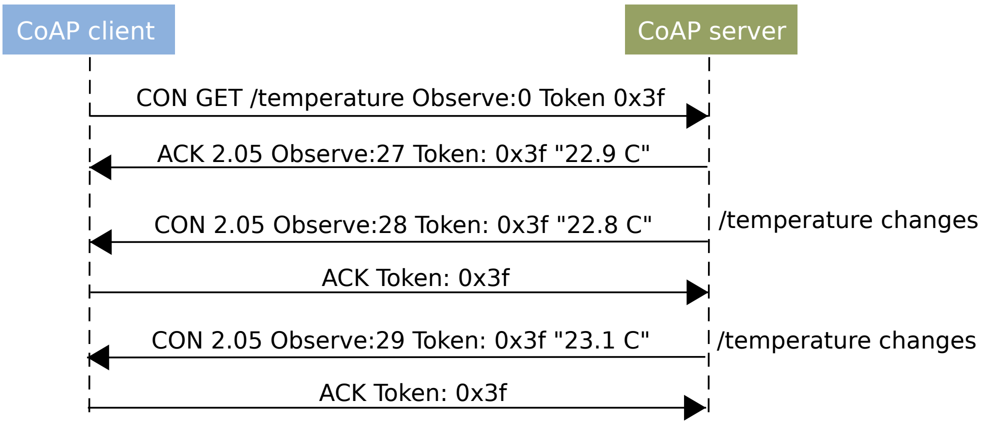
Figure 2.22: *CoAP GET request with separate response*

the updated representation, who replies with an ACK message. Important to notice is, that the token which was originally send in the CON GET request is unchanged for each consecutive message. This way, client and server can match with the original subscription.

CoAP has also other features which are described in the specification (Shelby et al., 2014). This includes e.g. HTTP-proxying, service and resource discovery, multicasting and several security mechanisms.

CoAP & IoT

CoAP is designed for the IoT and for M2M communication. Since it uses UDP for transport, the protocol has a low overhead with connectionless properties which is in favor of TCP-based protocols (Hakiri et al., 2015). But like HTTP, it supports mainly a synchronous request/response mechanism which is less suitable than protocols with publish/subscribe functionalities (Wang et al., 2017). Further, Karagiannis et al. (2015) argue that although CoAP is created for the IoT, no security-features are built-in. Securing messaging with DTLS, like suggested by the standard, is not adjusted to IoT requirements and, thus, its suitability can be argued.


CoAP client
CoAP server
CON GET /temperature Observe:0 Token 0x3f
ACK 2.05 Observe:27 Token: 0x3f "22.9 C"
CON 2.05 Observe:28 Token: 0x3f "22.8 C"
/temperature changes
ACK Token: 0x3f
CON 2.05 Observe:29 Token: 0x3f "23.1 C"
/temperature changes
ACK Token: 0x3f

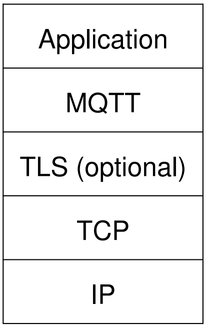
Source: adapted from Hartke (2015)

Figure 2.23: *Observing a resource in CoAP*

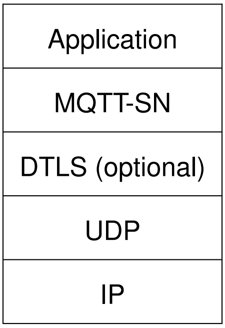
2.4.2.3 Message Queuing Telemetry Transport (MQTT)

Message Queuing Telemetry Transport (MQTT) is an open messaging protocol for M2M communication, which implements the publish/subscribe messaging pattern. The protocol was designed by IBM in 1999 and initially developed for unreliable networks with restricted resources, especially low bandwidth and high-latency (O'Leary & Piper, 2019). Since 2013 it is standardized by the Organization for the Advancement of Structured Information Standards (OASIS) and is promoted as an IoT application protocol. The first standard, which is based on IBM's specification v3.1 (IBM & Eurotech, 2010), was published in 2014 with the version number 3.1.1 (Banks & Gupta, 2014). In 2016 the same version became the ISO standard ISO/IEC 20922:2016 (ISO/IEC JTC 1, 2016). In March 2019, OASIS finalized the new version MQTT Version 5.0 (Banks et al., 2019), skipping over version number 4.0.

Specifically, the protocol implements a topic-based publish/subscribe messaging pattern (see Section 2.3.1.3), whereby clients can register their interests in messages with a topic filter at a message broker, known as the subscribing process. A client publishes a message tagged with a topic name by sending it to the broker (a.k.a. the publishing process). The broker validates the topic with all registered topic filters and notifies the subscribers with the message accordingly. The sender of the message is not aware of the possibly multiple receivers. Sender and receivers are decoupled in space, time and synchronization.



(a) *MQTT*



(b) *MQTT-SN*

Source: Author's illustration

Figure 2.24: *Layering of MQTT and MQTT-SN*

MQTT is based on a TCP/IP communication stack with the standard port 1883 (see Figure 2.24a). Using TLS, port 8883 is exclusively reserved for MQTT over TLS. However, with its extension MQTT-SN it can also be implemented for connectionless protocols (e.g. UDP or ZigBee; see Figure 2.24b) (Hunkeler et al., 2008; Stanford-Clark & Truong, 2013). The protocols and their features are discussed in more detail in Section 4.2, since they are used heavily in the implementation part of this thesis.

MQTT & IoT

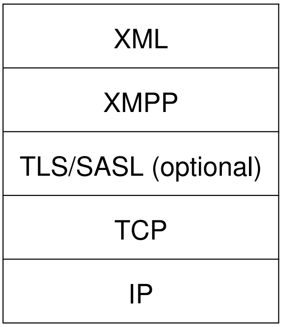
MQTT realizes the favored publish/subscribe messaging pattern, which is implemented in a very lightweight way and can be used to initiate data streams. Studies show that it has a better performance in terms of latency than HTTP (Yokotani & Sasaki, 2017), but performs worse compared to UDP-based protocols such as CoAP (Chen & Kunz, 2016). With MQTT-SN it may also connect IoT devices deployed in WSNs and build on top of unreliable protocols.

2.4.2.4 Extensible Messaging and Presence Protocol (XMPP)

The Extensible Messaging and Presence Protocol (XMPP) enables the near real-time exchange of structured extensible data between any two or more network entities. For this, it is based on Extensible Markup Language (XML) encoding to setup XML streams. XMPP, originally named Jabber, was developed by the Jabber community in 1999 for near real-time Instant Messaging (IM), presence information and contact list maintenance. In 2004 it was approved as a proposed standard by the IETF (Saint-Andre, 2004), and in 2011 the XMPP core standard was updated (Saint-Andre, 2011). Further, the XMPP Standards Foundation (XSF) (former Jabber Software

Foundation (JSF)) was founded to develop and publish XMPP extensions using a standards process. These XMPP Extension Protocols (XEPs) define additional features for the basic protocol (Saint-Andre & Cridland, 2016). Extensions are, for instance, Service Discovery (XEP-0030) or Multi-User Chat (XEP-0045).

XMPP uses a decentralized client-server architecture and is designed to support instant pushing of messages over an established TCP link. Therefore, the stack constitutes like shown in Figure 2.25. XMPP acts on top of a TCP/IP connection with a default port of 5222. An encryption layer such as TLS can be used optionally.



Source: based on Saint-Andre (2011)

Figure 2.25: *Layering of XMPP*

ADDRESSING

Every entity in an XMPP network is addressable with a so-called Jabber Identifier (JID) and can communicate with every other entity in the network. The JID is a string consisting of a localpart, domainpart and a resourcepart. The first two parts are separated by an '@', whilst the domainpart and resourcepart are demarcated with a '/'. For instance, a typical JID is *alice@jabber.org/mobile*, where *alice* is the user name (localpart), *jabber.org* is the server name (domainpart) and *mobile* is the resource name (resourcepart). The resource is an arbitrary string which is used to distinguish between multiple connections of the same user from different locations or multiple clients. So each entity is uniquely identifiable and addressable in an XMPP network.

COMMUNICATION PRIMITIVES

An XMPP client can connect to a XMPP server and initiate communication with other clients connected to the same server or network. The server is responsible to manage connections and sessions for other entities which have the form of an XML stream to and from authorized clients, servers and other entities. An *XML stream* is a container for the exchange of XML elements between any two entities. It starts with an XML <stream> tag, when a client connects, and ends with a </stream> tag if the client disconnects. The client (or server) can send any number of XML elements (so-called XML stanzas) over the stream if the stream is active.

Furthermore, the server has to route the *XML stanzas* among the entities over the XML stream. XML stanzas are the basic protocol data units, which are fragments of XMLs that is sent over a stream. Three different types of stanzas exist in the XMPP core: `<presence/>` announces if a contact is on- or offline, `<iq/>` allows a request/response interaction between clients or server and clients, and `<message/>` is used to send and receive messages.

```xml
<stream>
  ...
  <presence>
    <show>chat</show>
  </presence>
  <iq type="get" id="bv1bs71f">
    <query xmlns="jabber:iq:roster"/>
  </iq>
  ...
  <message from="alice@jabber.org"
    to="bob@jabber.org">
    <body>Hello Bob!</body>
  </message>
  ...
  <presence type="unavailable"/>
</stream>
```

Listing 2.1: *Client to server stream*

```xml
<stream>
  ...

  <iq id="bv1bs71f" type="result">
    <query xmlns="jabber:iq:roaster">
      <item jid="bob@jabber.org"/>
      <item jid="carl@jabber.org"/>
    </query>
  </iq>
  ...
  <message from="bob@jabber.org"
       to="alice@jabber.org">
    <body>Hi Alice, how are you?</body>
  </message>
  ...
</stream>
```

Listing 2.2: *Server to client stream*

The Listings 2.1 & 2.2 show the communication between a client and a server by using the XML stream and different stanzas. First, the client sends an initial `<presence/>`-tag to signalize that it is online. An optional `<show/>` element specifies the availability (see Listing 2.1:3-5). Next, the client requests its roster from the server which contains any number of specific contacts (known JIDs) and is stored on the server on behalf of the client (see Listing 2.1:6-8). After receiving its contacts (see Listing 2.2:4-9), the client sends a `<message/>` with a `to` attribute assigning the receiver (here: bob@jabber.org, see Listing 2.1:10-13). Bob answers by sending a `<message/>` to the server, which is forwarded to Alice (see Listing 2.2:11-14). Finally, the client disconnects from the server by sending an "unavailable"-`<presence/>` and closes the XML stream (Listing 2.1:15).

The mechanism behind the three different types of stanzas can be conceived by this example. The `<presence/>` advertises the availability of other entities in the network. Other entities know if a specific entity is online and available for communication. But Alice (entity A) only knows if Bob (entity B) is online, if he authorizes her by a so-called *presence subscription*. To see Bob's presence, Alice must send a subscription request, which he has to approve. Once approved by Bob, Alice receives regular notifications about his network availability. Basically, this is a simple, specialized publish/subscribe method about the presence of entities (Saint-Andre et al., 2009). IM applications use the presence information in the roster which is the contact list of an entity containing JIDs and their presence information.

The `<iq/>` (Info/Query) stanza implements a request/response interaction scheme, similar to HTTP GET, POST and PUT methods. The payload defines the request to be processed or the action to be taken by the receiver. A reply from the requested entity is mandatory. The request includes an `id`, which is also enclosed in the response to match with the request, and a `type` attribute. The `type` attribute in an `<iq>` stanza can take four values: `get` is similar to HTTP GET and allows the requesting entity to ask for information, `set` can be compared to HTTP POST or PUT and provides some information in the request. `result` is enclosed in a response to deliver the result of a `get` or acknowledge a `set` request, and `error` is sent to inform the requesting entity that the `get` or `set` request was not processed.

Finally, the `<message/>` stanza implements a basic push method. The stanza is used to transfer information from one entity to another, but since messages are not acknowledged, it is a kind of "fire-and-forget" mechanism. They contain some attributes: a `type` for differentiating the flavor of the message, a `to` and a `from` address attribute as well as optionally an `id` for tracking purposes. `to` and `from` addresses are the JIDs of the intended recipient respectively of the sender. `<message/>` stanzas also contain payload elements, e.g. `<body>` or `<subject>` for person-to-person chat messages. Some basic payloads are defined in the core XMPP specifications, however, XEPs can also define other payload elements (Saint-Andre et al., 2009).

XMPP EXTENSION PROTOCOLS (XEPS)

XMPP is extensible. With XEPs developed by the XMPP community, the core protocol and communication primitives became extended over the years. Extensions exist for e.g. Publish-Subscribe (XEP-0060), Multi-User Chat (MUC) (XEP-0045) or Jingle for voice calls (XEP-0166) (XMPP Software Foundation, 2018). How XEPs are integrated by extending the three different types of stanzas is shown with the help of a MUC example in the following:

The XEP-0045 is the standardized extension for defining Multi-User Chats (MUCs). The basic idea is to enable people to join a chat room and send messages that are delivered to all other participants. A chat room owns a JID such as `teaparty@conference.jabber.org` and has a "room roster" of all the participants. A client joins a chat room by sending a presence to the room, including a preferred nickname as resource identifier (Listing 2.3)

```xml
<presence from="alice@jabber.org/mobile"
    to="teaparty@conference.jabber.org/Alice"/>
```

Listing 2.3: *Client joins a MUC*

The `room@domain.tld/nick` in the `to`-attribute is the client's *room JID*. After the connection is established, the room sends a join notification from Alice's room JID to all other participants. Simultaneously, Alice receives presence from the room JIDs of all other participants. Depending on the configuration, the room sends some of the most recent messages exchanged in the room. Now, Alice can send messages to the room by using the JID of the room (see Listing 2.4). The room forwards the message to all other participants but sets the `from`-attribute to Alice's room JID.

```xml
<message from="alice@jabber.org/mobile"
    to="teaparty@conference.jabber.org"
    type="groupchat">
    <body>Hello all!</body>
</message>
```

Listing 2.4: *Client sends a message to a groupchat*

MUCs have further additional features such as private messages, user roles (visitor, moderator, admin, etc.) and security features. For more information, the reader may consult the specification (Saint-Andre, 2018).

XMPP & IOT

XMPP was initially designed for chatting and message exchange based on XML documents. The parsing and creation of XML documents needs additional computational capabilities and increases power consumption, both may be not sufficiently available in IoT devices (Karagiannis et al., 2015). Some efforts have been made to provide lightweight implementation. For instance, Hornsby & Bail (2009) implemented µXMPP, a lightweight client for the low power operating system Contiki to run on highly constrained devices. Further with its supported messaging patterns (publish/-subscribe and request/response), it is also adjustable to the requirements of IoT applications. Wang et al. (2017) implemented a lightweight XMPP publish/subscribe scheme for resource-constrained IoT devices based on UDP. They incorporated features such as sleeping clients and evaluation results on performance were promising. Furthermore, XMPP has great scalability, addressing, and security capabilities, which can be useful in IoT applications (Lin et al., 2017).

Besides that, different IoT related XEPs were proposed such as XEP-0323 (IoT - Sensor Data) (Waher, 2017). The extension provides clients a mechanism, for instance to request data from IoT devices through an introduced set of XML stanza. However, along with the other proposed XEPs for IoT, they were retracted.

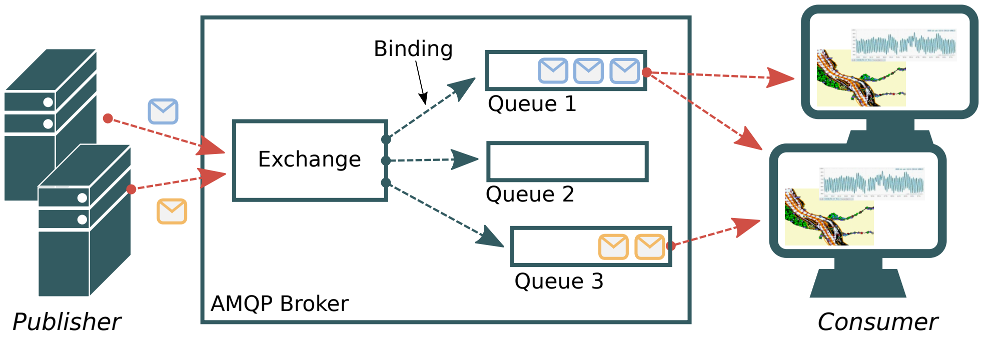
2.4.2.5 Advanced Message Queuing Protocol (AMQP)

The Advanced Message Queuing Protocol (AMQP) is a lightweight M2M protocol whose development was originally started by John O'Hara at JPMorgan Chase in 2003 (O'Hara, 2007). In 2006, the AMQP working group was set up by JPMorgan to begin a partnership in the industry and develop a standardized MOM for most commercial computing systems, especially in the financial service industry. The working group's efforts resulted 2012 in an OASIS standard AMQP version 1.0 (Godfrey et al., 2012), which then became the ISO Standard ISO/IEC 19464:2014 (ISO/IEC JTC 1, 2014). However, AMQP 1.0 is mainly a novel wire-level protocol with only abstract broker requirements. Much more interesting in the context of IoT messaging is the previous version AMQP 0-9-1 released in 2008 (Trieloff et al., 2008), which differ significantly from version 1.0. It is also a wire-level protocol but defines a broker model for various message exchange mechanisms as well. Therefore, this thesis focuses on version 0-9-1.

MESSAGE MODEL

AMQP 0-9-1 uses the Advanced Message Queuing model (AMQ) for message exchange, which consists of a set of components that route and store messages within a server and a set of rules for wiring these components. In the server, three main types of components are connected into processing chains to create a desired functionality. These components are depicted in Figure 2.26.

The server in the AMQ model is called an AMQP broker and has three components: the *exchange* (1) receives the messages from publishers and routes them based on



Source: adapted from Al-Fuqaha et al. (2015)

Figure 2.26: *Architecture of AMQP*

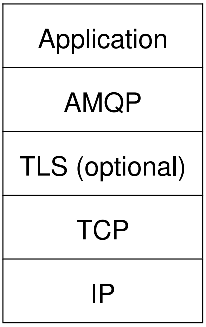
certain criteria. The message routing criteria are provided by *bindings* (2), which define the relationships between the exchange and *message queues* (3). These store the messages until they can be safely processed by consumers.

With this Exchange-Binding-Queue architecture, a flexible message distribution can be configured. This flexibility is achieved through different variable settings for the components: the exchange has a name and implements one of several exchange types, the binding binds a queue to an exchange by a routing key, which is usually equivalent to the name of the queue or routing pattern. Now, let a publisher send a message with a routing key R to an exchange identified by a name and running on a broker. In the trivial case, the exchange is configured to be of a *fanout* type, so that it passes the message to every message queue connected to the exchange unconditionally. If it is a *direct* exchange, the routing key of the message R is compared to the routing key of the binding K and passed to the queue if $R = K$. Finally, a *topic* exchange is similar to the direct type but compares the routing key of the message P with the routing pattern of the binding R. The message is passed to the message queue if R matches P. In this case, the routing key must consist of zero or more words (alphanumeric) delimited by dots. Similarly, the routing key follows the same rules but can also use * to match a single and # to match zero or more words. For instance, the routing pattern `*.stock.#` matches the routing keys `usd.stock` or `eu.stock.db` but not `stock.dax`. This represents a similar topic-based publish/subscribe mechanism like in MQTT (see Section 4.2 for details). Other less important exchange types are also possible. For this, the reader may consult the specification in Trieloff et al. (2008).

The message queues are created by consumers. For creation, a consumer declares an exchange by name, a queue name and, if necessary, a routing pattern at the broker. The broker establishes the binding between exchange and queue. Then, a message, that is passed to the queue by the exchange, is either delivered directly to the consumer if he subscribed to the queue (push) or is stored in the queue until the consumer pulls it on demand. Further, queues can be private to a single consumer or of a shared type. Shared queues may have many consumers and distributes the messages between these on a round-robin scheduling. This means, that a message of a queue is never sent to more than one consumer unless a failure occurs.

ARCHITECTURE

AMQP 0-9-1 is a binary protocol. Its messages are organized into frames with a header, a payload and a frame end. Its messages require a fixed header of eight bytes and a variable content header that carries the delivery parameters such as type of the message and content parameters such as the size of the content. The payload of a frame can be of arbitrary size. AMQP supports reliable communication focusing on message-oriented environments. This requires a reliable transport protocol to exchange messages, thus it is typically used as an application layer protocol on a TCP/IP protocol stack (see Figure 2.27).



Source: Author's illustration

Figure 2.27: *Layer stack of AMQP*

AMQP can be used with TLS or SASL, which provide authentication and encryption. However, the specification does not stipulate any security features or requirements. Thus, the security depends on the actual implementation.

AMQP & IoT

Through its flexible message-oriented approach, AMQP is an interesting protocol for the IoT. According to Naik (2017) with its wide range of services related to messaging, it is the preferred choice for businesses. A publish/subscribe mechanism like in MQTT can be applied, as well as ordinary message queues, which can be used to implement worker queues. Compared to MQTT, AMQP offers more aspects related to security but it is less energy efficient (Luzuriaga et al., 2015). Hence, Luzuriaga et al. (2015) would prefer AMQP with ideal WLAN but MQTT under constrained environments. Furthermore, the services of AMQP demand higher bandwidth and latency. Since it relies on the TCP/IP suite, devices that use AMQP cannot interoperate easily with devices that do not have the required resources to support TCP/IP (Al-Fuqaha et al., 2015).

2.5 IoT Information and Services

The third building block of IoT architectures is named "IoT Information and Services" (see Figure 2.2). Based on Latvakoski et al. (2014), the block can be subdivided into an information and a service level. The *information level* consists of information management services and exchange transactions between the different stakeholders of the IoT architecture. Generally, a standardized common information model facilitates a smooth and interoperable exchange of information and business interaction between the stakeholders of a system. Thus, it is favorable to use an agreed standardized information model in IoT architectures. Information models are used by the *service level*, which are also named IoT service platform. They include service solutions and frameworks. Depending on the use case and implementation, they may offer generic service elements such as service discovery and delivery, access control, generic storage and device management as well as environment monitoring and event notification (Pakkala & Latvakoski, 2007).

IoT service platforms can be implemented using a variety of architecture patterns. Several solutions in recent years incorporated Service-oriented Architecture (SOA) technology into IoT service delivery systems to ensure interoperability and to solve the heterogeneity of services and physical entities (Cheng et al., 2016). The Sensor Web services of the OGC form such an SOA-based architecture, which is described in detail in Section 2.5.1. However, researchers argue that traditional SOA standards and technology are not designed for the IoT but rather for the integration of enterprise-class heavyweight services (Cheng et al., 2016). That is why, some researchers argue for IoT service platforms based on Resource-oriented Architectures (ROAs) (Guinard et al., 2010b; Mayer & Guinard, 2011; Zanella et al., 2014), which adopt the RESTful style introduced by Fielding (2000). ROA are used in the IoT to support the idea of the WoT (Traversat et al., 2003). Other researchers favor Event-driven Architectures (EDAs) for implementing IoT service platforms, which provide on-demand distribution of sensed information and event-driven service coordination (Zhang et al., 2014; Cheng et al., 2016; Rieke et al., 2018).

In the following, we will look at IoT service platforms that are implemented using SOA, ROA and EDA patterns and describe example implementations. Since interconnection and interoperability is a crucial requirement for information models and services in the IoT, we focus on established or promising standards and interfaces for IoT service platforms. Furthermore, for the sake of suitability in a geospatial IoT, platforms are chosen, which internalize a geospatial nature.

2.5.1 OGC's SWE: a SOA for the Sensor Web

SOAs and the IoT

The SOA approach is the dominating middleware layer designed to support communication and information sharing in the contemporary WWW. Services, which embodies the key concept of SOAs, are autonomous, loosely coupled and platform independent entities. They can be described, published, used and combined (Lan et al., 2015). SOAs follow the "publish, find and bind" paradigm: a service provider *publishes* a service description to a service registry, which can be searched by service requesters to *find* the service. Finally, the service requester may invoke the service by using the *bind* operation (Bukhsh et al., 2015).

Since the SOA style is heavily used in distributed architectures in the Internet, it also plays an important role in the IoT. Razzaque et al. (2016) argue that a middleware architecture for the IoT should be service-based to offer high flexibility and abstract services for the complex underlying hardware for e.g. data management, reliability, security and so on. They include several advantages, for instance, ensure interoperability through web service standards such as the Simple Object Access Protocol (SOAP). But embedding these concepts at device level requires significant simplification, optimization and adaption of SOA tools and standards (Guinard et al., 2010a). Furthermore through their messaging pattern (request/response), they only provide limited real-time capabilities of sharing information (Razzaque et al., 2016).

Several sensing and IoT systems use a SOA-based approach (e.g. Bendel et al., 2013; Fazio & Puliafito, 2015; Kotsev et al., 2015). The Sensor Web and its associated services are one example of a SOA for sensing infrastructures. Thus, they are explained exemplary in the following.

Sensor Web

The concept of the Sensor Web describes a collaborative, coherent, consistent and consolidated sensor data collection, fusion and distribution system. Originally, it was not a dedicated platform for the IoT but rather was proposed as a meta-platform that integrates arbitrary sensors and sensor networks operated by institutions such as the European Environment Information and Observation Network (EIONET) (Simonis, 2008). However, in recent years, Sensor Web infrastructures became associated with the IoT as a middleware layer between IoT devices and applications (Bröring et al., 2011; Latvakoski et al., 2014). The Sensor Web aims at providing access to sensor information and data through standardized web services interfaces (Klopfer & Simonis, 2009). Thus, it can be viewed as a web overlay for the underlying sensor-based systems (Latvakoski et al., 2014). Services in the Sensor Web enable access to sensor and sensor data using web technologies. They provide a facade to the complexity of the communication in the underlying layers. Botts et al. (2007) provides a general definition, which is adopted for this thesis (see Definition 2.4).

Definition 2.4 (Sensor Web)
A Sensor Web refers to web accessible sensor networks and archived data that can be discovered and accessed using standard protocols and Application Program Interfaces (APIs).

The functionalities within a Sensor Web infrastructure are according to Botts et al. (2007):

- The discovery of sensor systems, observations, and observation processes which are met by the immediate needs of applications or users.
- The sensor's capabilities and quality of measurements must be determined and retrievable.
- Software can access sensor parameters which allow them to process and geo-locate observations automatically.
- Real-time or time-series observations and coverages in standard encodings are retrievable.
- Sensors can be assigned tasks to acquire observations of interest.
- Software may specify certain criteria to subscribe to alerts issued by sensor or sensor services.

Since it can be implemented as a SOA in the Web, a Sensor Web infrastructure builds on the WWW and uses a variety of standards recommended by the W3C, such as XML, XML schema or SOAP for data encodings and interface specifications (Klopfer & Simonis, 2009). A driving institution of the Sensor Web concept is the Open Geospatial Consortium (OGC) and, thus not surprisingly, the view of Sensor Web has been largely shaped by the architecture developed by OGC's SWE working group (Latvakoski et al., 2014). As a consequence, we will focus on their work for an implementation of the Sensor Web in the following.

SENSOR WEB ENABLEMENT (SWE)

The OGC SWE working group started their work in 2003 focusing on standards to implement the concept of the Sensor Web. The group issued a SOA-based framework containing of a number of standards, which define formats for sensor data and metadata as well as sensor service interfaces. It is accordingly called the "SWE framework". The standards allow the integration of sensor networks in the Geospatial Web. The SWE architecture has been advanced to a mature state and some of the SWE standards have also been adopted to official OGC standards, but the efforts of the working group are still on-going, so that new proposed standards are issued or adopted standards are reviewed (Jirka et al., 2009). Between 2006 and 2007 the first framework (SWE 1.0 specifications) have been approved as standards, followed by the second generation (SWE 2.0) since 2011 (Bröring et al., 2011). Here, only the standards of SWE 2.0 are described briefly. The reader may consult the given literature for further information about SWE 1.0 and their differences.

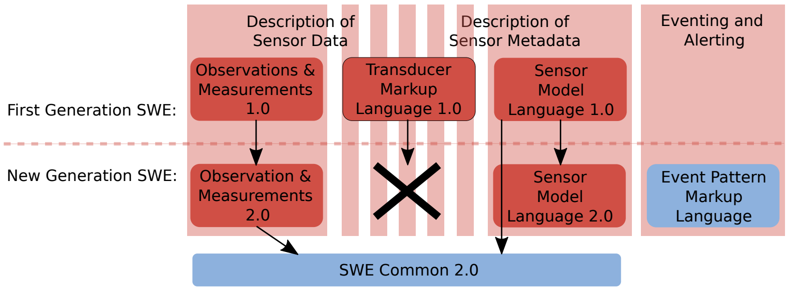
As mentioned in the introduction to this chapter, the IoT Information and Services block consists of an information and service layer. Accordingly, the SWE framework is structured into an information model and an interface model, which are discussed in the following sections.

SWE INFORMATION MODEL

The SWE information model defines data models mainly for the encoding of sensor observation and sensor metadata. In SWE 1.0 the set of standards covers three specifications: Observations and Measurements (O&M), Sensor Model Language (SensorML) and the Transducer Markup Language (TML). The SWE working grouped reviewed the information model for the second generation resulting in the advancements of O&M 2.0 and SensorML 2.0. These two form the proposed SWE Common 2.0, which defines data types shared by multiple specifications (see Figure 2.28). Further, the Event Pattern Markup Language (EML) encodes event patterns as processing rules for Complex Event Processing (CEP). O&M 2.0 and SensorML 2.0 were adopted as OGC standards and described further.

SensorML 2.0 is a standard model and XML schema for describing sensors, sensor systems and their associated processes for inventory management. According to Botts et al. (2007) it provides information needed for discovery of sensors, location of sensor observations, processing of low-level sensor observations and listing of taskable properties. In version 2.0 some key features were added such as explicit support for data streaming or its improved derivation and association with GML 3.2 (Botts & Robin, 2014).

O&M 2.0 provides a conceptual schema for observations as well as for features involved in sampling when making observations. Models for exchanging information,


Description of Sensor Data
Description of Sensor Metadata
Eventing and Alerting
First Generation SWE:
Observations & Measurements 1.0
Transducer Markup Language 1.0
Sensor Model Language 1.0
New Generation SWE:
Observation & Measurements 2.0
Sensor Model Language 2.0
Event Pattern Markup Language
SWE Common 2.0

Source: adapted from Bröring et al. (2011)

Figure 2.28: *SWE interface model*

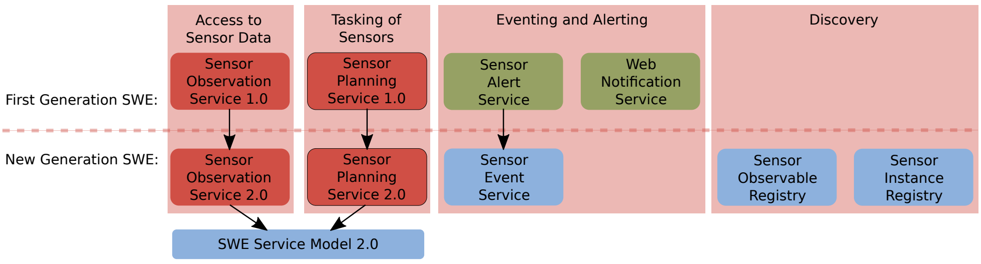
which describe observation acts and their results are specified in the standard (Cox, 2013). In O&M 2.0 the conceptual model and its implementation are separated from each other. Further, it introduces new spatial profiles and new observation properties (Bröring et al., 2011). While the model became an ISO standard, the OGC provides encoding standard for the implementation. In Cox (2011), an XML schema of O&M 2.0 is standardized.

SWE INTERFACE MODEL

The SWE interface model uses the encoding standards given in the information model to provide services for the Sensor Web. In its first generation, four service interfaces were defined (Bröring et al., 2011):

- The *Sensor Observation Service (SOS)* provides access to sensor measurements as well as sensor metadata based on HTTP requests.
- With the *Sensor Planning Service (SPS)* tasking of sensors and setting their parameters can be accomplished.
- The *Sensor Alert Service (SAS)* allows clients to subscribe for alerts based on certain criteria, for instance if a sensor measurement exceeds a threshold.
- Finally, the *Web Notification Service (WNS)* is used to provide asynchronous notification mechanisms between SWE services or other clients. It is not directly sensor related.

SOS and SPS were adopted as standards by the OGC. In the second generation of the interface model, both were advanced to version 2.0. The SAS never reached maturity and was replaced by the *Sensor Event Service (SES)* draft in the second generation. Finally, two services for sensor discovery were proposed as discussion papers but not yet standardized. In the second generation, solely the SOS 2.0 and the SPS 2.0 are in a solid state and became official OGC standards. Both depend on the common SWE Service Model 2.0 (see Figure 2.29).



Source: adapted from Bröring et al. (2011)

Figure 2.29: *SWE interface model*

The SOS allows requesting sensor metadata and sensor measurements by means of the WWW, namely HTTP. SOS 2.0 provides methods to retrieve service metadata (*GetCapabilities*), query sensor descriptions (*DescribeSensor*) or access observations (*GetObservation*) (Bröring et al., 2012). Thereby, it utilizes the SWE information model 2.0 and its encoding standards to encode requests and response data. The *GetObservation* operation of the SOS 2.0 uses particularly the O&M 2.0 model for the creation of a response document, whilst the DescribeSensor method relies heavily on the SensorML 2.0 encodings. The SOS 2.0 may be configured with a transactional extension which contains operations to add new or delete old sensor descriptions (*InsertSensor* and *DeleteSensor*) and to insert new sensor observations (*InsertObservation*). Natively, they also make use of the standards defined in the SWE information model.

The SPS is a web service interface, which allows clients to request user-driven acquisitions and observations by tasking of sensors (e.g. setting a sampling rate). Furthermore, it provides information about the capabilities of a sensor as well as how to task sensors (Botts et al., 2008). Version 2.0 offers some important methods to steer the tasking of sensors. Besides the *GetCapabilities* operation to retrieve service metadata from a server, other important operations are the *GetFeasibility* operation to check whether a task is feasible for a sensor, *Submit* to submit tasks or the *GetStatus* method to track the status of submitted task. For retrieving sufficient information to formulate tasking requests, the *DescribeTasking* can be used by clients to request the syntax for describing a task. Further operation can be obtained from the standard in Simonis & Echterhoff (2011).

2.5.2 SENSORTHINGS API: A RESOURCE-ORIENTED ARCHITECTURE FOR THE IOT

ROA AND THE IOT

In Resource-oriented Architectures (ROAs) a RESTful style is adopted, which can also be used to implement the information and service block of IoT architectures. Some researchers argue that a RESTful architecture is more suitable for IoT applications. Zanella et al. (2014) state that is has very strong similarity with traditional web services and, thus, facilitates the adoption and use of IoT by both, end users and service developers. Furthermore, the resources of particular IoT devices can directly be mapped to resources in the Web, the basic components of ROAs (Guinard et al., 2011). Like mentioned, the WoT initiative, which envisions a world where IoT devices are exposed using WWW technologies, is built upon REST principles: smart things are accessible using URIs that can be exchanged, referenced and bookmarked (Collina et al., 2012). Devices of the IoT can either serve content directly by deploying web servers, or the connectivity can be achieved indirectly through a proxy (Guinard et al., 2011). We already learned about the CoAP protocol, which offers a RESTful style for resource-constrained devices. For instance, by implementing a CoAP-HTTP proxy, the connectivity to the Web can be easily established.

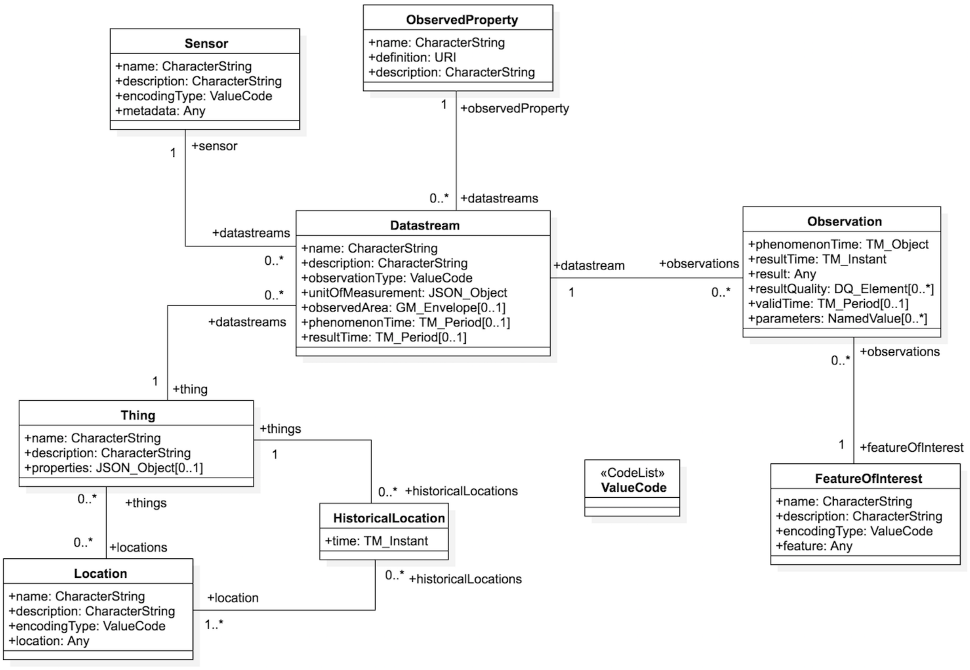
OGC SENSORTHINGS API

In 2016, the SWE suite was extended by the OGC SensorThings API, a standard that is designed to be RESTful and is based on the already established OGC SWE information model standards including the O&M 2.0 data model. However, the SensorThings standard focuses on interconnecting IoT devices, data, and applications over the Web (Liang et al., 2016b). Therefore, it uses a more lightweight code-base and JavaScript Object Notation (JSON) as data exchange format (Kamilaris & Ostermann, 2018). Jazayeri et al. (2015) showed that the standard can be used on resource-constrained devices and is more suitable for the IoT than a SOS-based architecture due to efficiency. The standard provides two main functionalities whereby each function is handled and standardized by a separate profile: *Sensing Profile* (Part I) and *Tasking Profile* (Part II).

The *Sensing Profile* (Part I) has been adopted as an OGC standard in 2016 (Liang et al., 2016a). It offers managing and retrieval of observations and metadata of heterogeneous IoT sensor systems in a standardized way. Thus, it is comparable to the SOS introduced before, but designed for resource-constrained devices. It adopts the REST principle to represent each resource by a unique URI. This allows to perform CRUD[7] operations by using methods of HTTP. In addition to HTTP,

[7] Create, Read, Update, Delete

a SensorThings service may support the MQTT protocol to enhance the API with publish and subscribe capabilities. The entities of the SensorThings API are illustrated in the UML diagram in Figure 2.30. The model is based on the O&M 2.0 concept. While *observation*, *observed property* and *feature of interest* exist in both models, the *sensor* entity corresponds to the *procedure* of O&M. *Datastream* for grouping observations and the *thing* entities are further added to the model. Hence, the SensorThings model is much more aligned to objects in the IoT than the SWE standards.



Source: Liang et al. (2016b)

Figure 2.30: *Sensing entities in the SensorThings API*

Part II of the SensorThings API, the *Tasking Profile*, was adopted recently (Liang & Khalafbeigi, 2019). The profile introduces interoperable methods for submitting tasks to control sensors and actuators. Therefore, it is similar to the SPS but again is specifically designed to work with resource-constrained devices. The profile extends the thing entity of the model by *tasking capabilities* and *tasks*. These can also be accessed by CRUD operations using HTTP methods.

2.5.3 EVENT-DRIVEN ARCHITECTURES & OGC'S EVENTING WORK

EDA BASICS

In 2003, Gartner coined the terminology Event-driven Architecture (EDA) by describing a conceptual architecture pattern based on events (Schulte & Natis, 2003). The pattern defines a software design for implementing applications and systems with messages (events) that determine the flow between decoupled software components and services (Maréchaux, 2006). This is fundamentally different to the SOA and ROA patterns; while these use directed, bidirectional request/response communication to invoke procedures, EDA utilizes unidirectional messaging to communicate between multiple independent software systems (Schulte & Natis, 2003). It is extremely loosed coupled and highly distributed. According to (Maréchaux, 2006), the fundamental characteristics of EDA are

1. *Decoupled interactions*: Software systems in EDA are unaware of each other. The emitter of an event does not know necessarily the receiver.
2. *Many-to-many communications*: Publish/subscribe messaging is used, where a specific event might impact many subscribers.
3. *Event-based trigger*: The flow of control is determined by the event's receivers.
4. *Asynchronous*: Event messaging enables asynchronous operations.

In EDA an event is defined as "a notable thing that happens inside or outside your business. ... [It] may signify a problem or impending problem, an opportunity, a threshold or a deviation." (Michelson, 2006). Events in EDA describe detectable conditions inside a computer system. Thus, the detection and processing of events determines how a system reacts to a specific situation (Morales & Garcia, 2015). The event header may describe the event occurrence such as an ID, the type, the name, the timestamp, the occurrence number or the generator. The body consists in general of a description of what happened. Since it must be understood by all consumers participating in the system, a business lexicon or ontology should be used (Michelson, 2006).

EDAs are based on four logical levels:

1. *Event generator*: A source, e.g. an application, a database or a service, that generate events.
2. *Event channels* are the messaging backbone of EDA. It is used to transport events between the generators, event processors and the downstream subscribers.

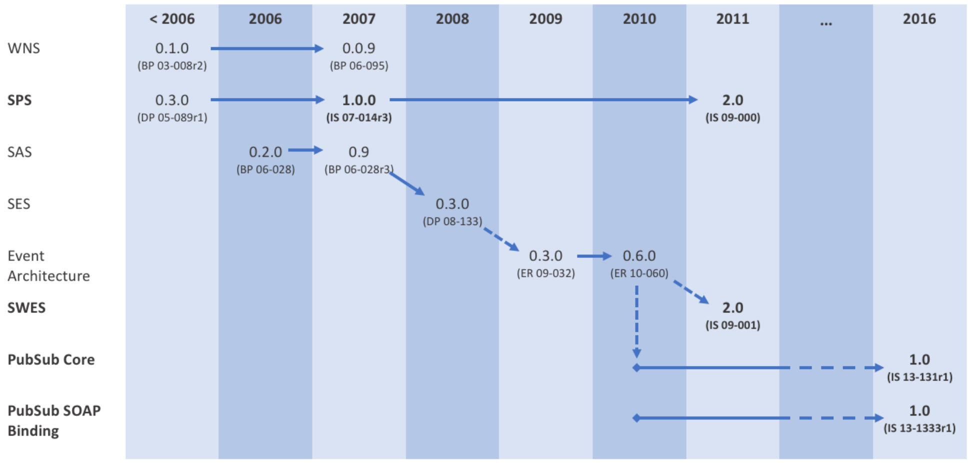
3. *Event Processing* describes the layer which is responsible for evaluating events against event processing rules and performing actions accordingly. There are four types of event processing:
   (a) *Simple event processing* involves filtering or threshold detection.
   (b) In *Event Stream Processing (ESP)*, a stream of events is analyzed continuously by e.g. pattern matching in real-time.
   (c) *Distributed Stream Processing (DSP)* is related to ESP. These systems dynamically distribute the work on multiple computing systems which allows for a highly scalable processing infrastructure.
   (d) *Complex Stream Processing (CSP)* involves evaluating a confluence of events and, subsequently, take action according to the result. The observed events might be of different types collected over a period of time and can be correlated causally, temporally or spatially.
4. *Downstream event-driven activity* are initiated by single events or event correlation. Technically, this can be accomplished by a push notification issued through the event processor but also pulling events by the subscribers is possible.

EDA AND THE IOT

EDA are widely used in designing IoT systems (e.g. Filipponi et al., 2010; Wan et al., 2012). However, the EDA approach mostly complement systems based on SOAs, known as event-driven SOA, advanced SOA or SOA 2.0 (Theorin et al., 2017). Several event-driven SOA approaches for the IoT, which supports e.g. real-time, event-driven and concurrent service execution, can be found in the literature (e.g. Zhang et al., 2014; Lan et al., 2015; Cheng et al., 2016). Van der Zee and Scholten (2014) propose a combination of SOA-EDA, which can be utilized to realize real-time spatial analysis of event streams by integrating spatial algorithms in IoT. Their architecture consists of an event pre-processing for e.g. (geo)filtering or (geo)transformation and a Geo-enabled CEP engine, which applies predefined geo event rules. Deployed web services can be used to access the event stream. Rieke et al. (2018) call on integrating event-based communication in traditional service-oriented SDIs to integrate IoT devices and their data flows for a Geospatial IoT.

OGC WORK ON EVENTING

Event-driven models and architectures has been conducted also by the OGC (see timeline in Figure 2.31). Starting with the SAS in the first generation of SWE standards, the successive SWE generation brought out a draft for the SES. While the SAS is a push-based counterpart of the SOS based on the XMPP protocol, the SES is not tightly coupled with a specific transport protocol and provides advanced filtering



Source: Rieke et al. (2018)

Figure 2.31: *Seminal OGC work on eventing-related specifications*

capabilities. Westerholt & Resch (2015) argue that the SES has a quite complex specification and the approach goes far beyond the basic purpose of notification. Both proposals never reached the status of an official standard. Further, the OGC tried to introduce a more general approach for notification by developing the WNS, which is a standalone message broker that offers notification capabilities to other OGC Web Services (OWSs) by a range of possible protocols such as XMPP or SMS (Simonis & Echterhoff, 2006). However, it is outdated and not under proactive development (Westerholt & Resch, 2015). Under the working title OGC Event Service (Echterhoff & Everding, 2011), the OGC incorporated their experience from SES and their Event Architecture (Echterhoff, 2010) by applying OASIS Web Services Notification (WS-N) and a SOAP binding. Finally, the efforts resulted in the OGC Publish/Subscribe standard in 2016, which is presented briefly in the following.

OGC PUBLISH/SUBSCRIBE

The OGC Publish/Subscribe Interface Standard (OGC PubSub in short), officially approved in 2016, was developed by the PubSub Working Group, which was initiated in 2010. It describes a mechanism to support publish/subscribe requirements across OGC data types and service interface. Thus, it is destined for the Sensor Web standards such as the SOS, but also for other OGC services like the Web Feature Service (WFS) (Bigagli & Rieke, 2017). According to Rieke et al. (2018), the standard is a fundamental enabler towards event-driven SDIs. It extends the request/response mechanism of most mature OWSs by a publish/subscribe mechanism and, thus, enables clients to subscribe to information already provided by any these services. Furthermore, OGC PubSub may use specific filtering capabilities.

Currently, the OGC PubSub consists of two standard documents (see Figure 2.31): (1) the core document, which is independent of the underlying technology and covers the basic mandatory functionalities and optional extensions abstractly (Braeckel et al., 2016), and (2) a SOAP binding document defining the implementation in SOAP services, which relies on WS-N set of standards (Braeckel & Bigagli, 2016). Furthermore, a binding for PubSub functionality in REST/JSON services is in draft status (Rieke et al., 2018).

The core document of OGC PubSub adopts the terminology and concepts of the publish/subscribe pattern like discussed in Section 2.3.1.3. Further, it introduces two additional roles: a *receiver* and a *sender*, which often coincides with subscriber respectively publisher, but can also be decoupled from another. A subscriber generates a subscription on behalf of the receiver by using the `subscribe()` operation on a publisher. The subscription may include filter expressions, e.g. based on OGC Filter Encoding 2.0 (Vretanos, 2014), to apply on the message content, metadata or other aspects. The publisher responses with a *SubscribeResponse* message for acknowledging or declining the subscription. In case of a successful subscription, the receiver is notified (via the *Notify* operation) by the sender asynchronously, if the new message matches the subscription. Subscriber may terminate subscriptions with the *Unsubscribe* operation send to the publisher, who answers with an *UnsubscribeResponse*. The core standard can be extended by further optional functionalities. The functionalities a publisher provides can be requested by clients using the *GetCapabiltities* operation. It delivers a capabilities document consisting of metadata such as supported filter functionalities, requirements for use or content information. Furthermore, a subscriber can retrieve information on one or more subscriptions sending a *GetSubscription* request to the publisher.

2.6 IoT Data Processing and Visualization

Finally, IoT data analysis and visualization is the fourth building block depicted in Figure 2.2. It is also called "the application level" (Latvakoski et al., 2014), since provided services and the collected data are used to manage the IoT infrastructure or to analyze and visualize in applications. Since the emerging IoT produces volumes of data in a short time, it has similar challenges and opportunities in analyzing and visualizing as the big data industry (Marjani et al., 2017). This includes the three characteristic "Vs": growth in volume, velocity and variety of data (Rios & Diguez, 2014). Of course, solutions to meet these problems are wide-ranging. Hence, we only will give a brief and surficial overview about data processing and visualization. For more information, the reader may consult the common literature on big data.

2.6.1 IoT Data Processing

The data collected by sensors in the IoT is massive in volume and velocity: millions of IoT devices produce new observations with a high frequency. Thus, the need to adopt big data analytics in IoT applications is compelling (Marjani et al., 2017). This can be subdivided into *batch processing* for analyzing massive quantities of stored data and *stream processing* to analyze data streams. Both can be used to process the data produced in IoT, but stream processing also handles newly arriving data tuples in data streams.

Batch Processing

Traditionally, batch processing describes a computer program which read data from a file, process it and writes the result back to another file. For IoT data this means that a process takes data as an input, e.g. a set of sensor observations, applies algorithms to the set and delivers an analysis. Depending on the size of the input data, this might become infeasible in terms of limited memory or computation time for a single computing system. Thus, big data analytics offers tools and software to distribute batch processing jobs to multiple machines balancing the workload. For instance, Apache Hadoop[8] provides an open-source implementation of the MapReduce model that is a popular parallel processing model to handle data processing in distributed systems (Yue & Jiang, 2014). It especially allows for horizontal scaling. These systems can be used to process sensor data on a large scale like shown by e.g. Rios & Diguez (2014). However, it still performs a batch process whose input is static and, once invoked, cannot be updated.

Stream Processing

The growing number of IoT devices result in an exponentially increasing number of (spatiotemporal) events and data streams. Processing these data streams in a real-time manner and finally supporting fast decision-making processes drives the development of suitable tools and algorithms for stream processing (Rieke et al., 2018). Definition 2.3 introduced data streams as an unbound sequence of tuples. In particular, these can also be seen as a sequence of events. Incorporating real-time analysis on events, we already introduced different types of event processing in EDA (see Section 2.5.3). While simple event processing is applied to single events in a stream, ESP (and its distributed variant DSP) as well as CEP can be used to analyze (multiple) event streams.

CEP solutions process concurrent events and derive new events based on their combination. Fusing and synthesizing events from multiple source and data streams

[8] https://hadoop.apache.org/

is the complex part of these systems. The joint events are then checked against specified query patterns and analyzing rules. CEP systems such as Esper[9] are traditionally centralized architectures and do not provide any scalability.

Since key challenges for streaming analysis are (1) one-pass processing, (2) limited amounts of memory and (3) limited time to process (Bifet & Kirkby, 2009), ESP systems to analyze (spatiotemporal) data streams in real-time are limited if the number of IoT devices and streams grow. The distributed variant, DSP, can be then used to scale horizontally to balance workload on multiple computing systems. Several DSP frameworks exist such as Apache Storm[10], Apache Flink[11] or Apache Spark Streaming[12]. They support various features and have different performances on data streams (Lopez et al., 2016). Whole architectures such as the Lambda (Marz & Warren, 2015) or the Kappa architecture (Kreps, 2014) are built around these technologies to tackle the mentioned challenges.

2.6.2 IoT Data Visualization

IoT requires sophisticated visualization techniques because of the large size and high dimensions of the data. Marjani et al. (2017) argues that big data analytics and visualization should work seamlessly to obtain the best results for IoT applications. The design of presenting data as well as the response time are important factors in IoT data visualization. IoT devices provide the raw data, but raw data remain useless for a user. Only accurate visualization interfaces allow humans to perceive and interpret the data (Logre et al., 2014). Depending on the data and use case, requirements change for visualization and for providing a suitable user experience: time series of historical sensor data requires different views and functionalities for exploring than real-time data updating frequently.

Historical sensor data, obtained by services such as the SOS of the SWE standard suite (see Section 2.5.1), can be visualized, for instance in form of graphs, in web portals. This has been done in several research projects (Cannata et al., 2014; Vitolo et al., 2015; Herle et al., 2018). For instance, in the AirSensEUR Platform the 52° North Helgoland client[13] is utilized to access the stored sensor data (Rieke et al., 2018). It is a lightweight web application which can access the SOS and provide diagram views of multiple time series, temporal zooming, panning and so on. Additionally, the project provides a smartphone app, which enables mobile users to retrieve and visualize sensor data.

[9] http://www.espertech.com/esper/

[10] http://storm.apache.org/

[11] https://flink.apache.org/

[12] https://spark.apache.org/streaming/

[13] https://github.com/52North/helgoland

Since managing and analyzing IoT data effectively is challenging at scale and at near real-time, real-time analytics and visualization require different features (Winters et al., 2016). Nevertheless, the information should be perceivable in a fast way, that is why dashboards are often applied to visualize real-time IoT data. Dashboards are visual displays of the most important information to achieve objectives. The information is arranged on a single screen to make the information perceivable at a glance (Few, 2006). For instance, they are extensively used in smart city applications (e.g. Kitchin & McArdle, 2017; Zdraveski et al., 2017; Matheus et al., 2018).

CHAPTER 3

GEOSPATIAL INTERNET-OF-THINGS

Devices in the IoT observe occurrences as well as things or drive processes in the physical world. In doing so, they occupy also a certain location in space, which is fixed or varies over the course of time. Thus, their emitted data can be annotated geospatially. An IoT, which measures, sends and processes geospatial data can be considered as a Geospatial IoT, since the main part of the data and control flow involves geospatial data, often in real-time.

In this chapter, the characteristics of a Geospatial IoT are discussed to derive an appropriate architecture. We introduce the spatial nature of things and events and their integration in IoT systems first. Accordingly, the temporal and spatial components of events are investigated further. Finally, we derive a model for an architecture for a Geospatial IoT based on geospatially annotated events and analyze the requirements for a communication mechanism and protocol, which can be used to implement the proposed architecture.

3.1 CHARACTERISTICS OF A GEOSPATIAL IOT

3.1.1 SPATIAL NATURE AND MODELING OF THINGS

We already discussed in Chapter 2 that essentially every object of the physical world can be considered to be part of the IoT. These can be already electrified by nature or may be analogous but upgraded to smart objects by embedded technology. However, a thing in the physical world also always has naturally spatial properties. Besides location ("where"), these are its shape ("what form"), its size ("how big") and its orientation ("facing in which direction") (Huisman & de By, 2009). Further, van der Zee & Scholten (2014) consider the sphere of influence or range of an object as the fifth spatial property of a physical thing. Each of these spatial properties may undergo changes over time. For example, a car changes its location and orientation as well as its influence (e.g. noise of the engine) while driving along a road. Thereby, its form and size stay probably unchanged. Other spatial things such as cyclone might undergo changes in each spatial property while moving. Here, the location, the shape and size, the orientation and the influencing area are dynamic attributes which are functions of time (Venkateswara Rao et al., 2012).

The question that arises is how these spatial properties of things can be modeled in a Geospatial IoT within a digital representation? Do we have to consider all spatial properties, so do we need an identical digital twin of the real-world object? Or is a digital representation which include the location and e.g. orientation of the object sufficient? This obviously depends on the desired Geospatial IoT application and use case. If we consider a smart parking guidance system in a smart city application, it is probably sufficient to know the current location of the car and its destination. But this only works, if a specific assumption of the car's size holds. Van der Zee and Scholten (2014) argues that in most current cases the location is only needed, but specific use cases also demand for size, shape, orientation or influence sphere of the spatial object.

The geospatial location is probably the most important characteristic of measurements by Internet-enabled things. For one thing, the exact location while measuring the physical world is highly relevant for understanding local environmental conditions. But also powerful, context-aware location-based services and Geospatial IoT applications can only be developed and operated by knowing the accurate location of humans, animals or objects (Kamilaris & Ostermann, 2018). In recent years, evolutions in various outdoor and indoor positioning technologies and techniques have facilitated the location sensing process. According to Zafari et al. (2017) seamless and ubiquitous indoor/outdoor positioning and/or navigation of both, static and mobile devices, will be required by many IoT applications. While outdoor positioning techniques such as GNSS (e.g. Global Positioning System (GPS) or Galileo) or cell identification in cellular networks are well-established and cost-efficient, these technologies are less suitable for indoor environments since they cannot penetrate well indoors. Other signals have to be utilized for indoor localization purposes (He & Chan, 2016). However, techniques for indoor positioning such as radio-based positioning (based on WLAN, BLE or Ultra Wide Band (UWB)) are still in a developing phase (Goodchild, 2010). A detailed technological view on different indoor and outdoor positioning technologies and their (dis-)advantages can be found e.g. in Bensky (2016) or Brand et al. (2017). Beside the applicable environment, the accuracy as well as the operation scale of the different technology is interesting for Geospatial IoT applications. Figure 3.1 orders several indoor and outdoor positioning technologies according to their accuracy and operation scale levels.

Beside the geographic location, the orientation of (moving) objects can be useful in applications. This holds, for instance, for vehicles in smart traffic applications or for surveillance cameras. The orientation can be measured by sensors such as gyroscope or accelerometer, which refers to direct determination of orientation. Or it can be determined indirectly by e.g. 3D object recognition technologies (van der Zee & Scholten, 2014).

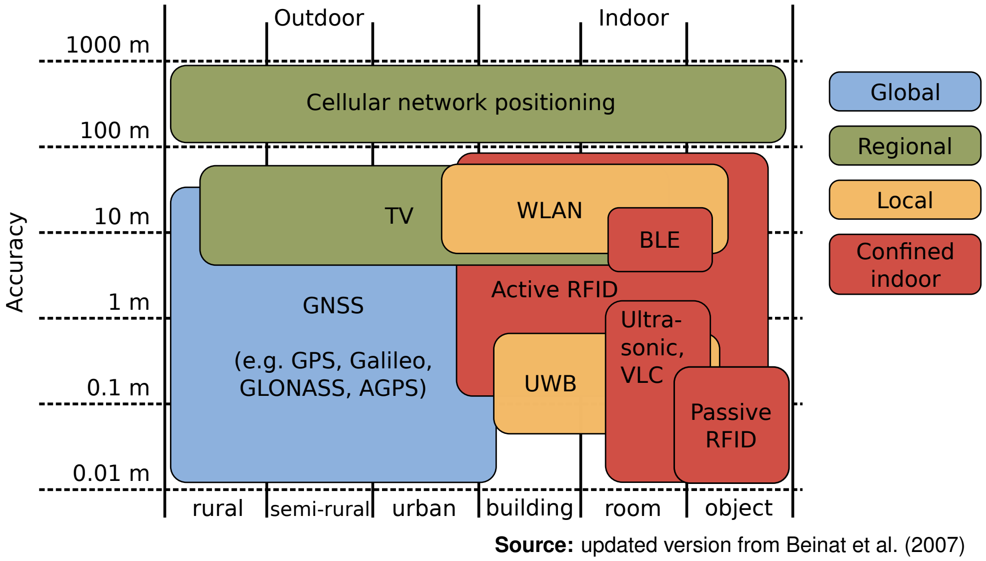


Figure 3.1: *Positioning technologies: accuracy and operation scales*

Positioning and orientation play especially an important role in IoT applications dealing with *moving objects*. Moving objects vary their position in the course of time such as humans, animals or vehicles. Thus, knowing the exact location at any time can be mandatory. Moving objects undergo location changes meaning that within a certain period of time, they travel a so-called spatial trajectory, which is a projection of a moving point into the plane yielding a polyline. A spatial trajectory can be defined as in Definition 3.1.

Definition 3.1 (Spatial Trajectory)
A spatial trajectory is a trace generated by a moving object in geographical spaces, usually represented by a series of chronological ordered points $p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_n$*, where each point consists of a geospatial coordinate and a timestamp, such that* $p = (x, y, t)$ *(Zheng, 2015). If the height is included, each point becomes a 4-tuple* $p = (x, y, h, t)$.

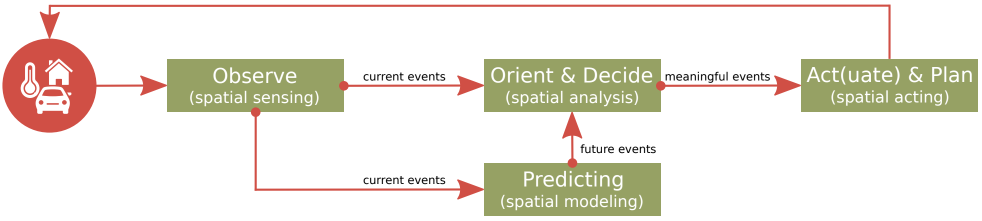
Güting & Schneider (2005) distinguish moving objects further into points and regions: in most cases the position in space is the only relevant parameter, which characterizes the objects as *moving points*. However, if the moving object has also an extent, which is essential for some applications (e.g. a cyclone or an epidemic disease), they can be characterized as *moving regions*.

Summarizing, spatial modeling of things may include five spatial properties, namely the location, the shape and size, the orientation as well as sphere of influence or range. As we have seen, the object can be modeled by different spatial data types in 2D such as point, line (e.g. trajectory) or polygon (e.g. extent) depending on the envisaged use case. In Section 3.4, we discuss these spatial data types further. Of course, it might be necessary to extend the modeling by 3D rigid or even non-rigid bodies.

3.1.2 SPATIAL INTEGRATION IN IOT SYSTEMS

IoT systems are a manifestation of the CPS idea like introduced in Section 2.1. These systems consist of devices equipped with sensors and actuators as well as computational capabilities on the one hand, but, on the other hand, embody the cyber-physical model, which is driven by a feedback loop oriented on the OODA loop concept. This feedback model can also be applied to Geospatial IoT application like already shown in different research projects (e.g. Terhorst et al., 2008; Yavari et al., 2016; Curry et al., 2019).

For van der Zee & Scholten (2013), the integration of spatial data, concepts and technologies in every phase of the OODA loop is increasing efficiency in Geospatial IoT applications. In their opinion, smart cities benefit from systems implemented in accordance with the feedback loop. Their adjusted OODA loop for smart cities is illustrated in Figure 3.2.



Source: adapted from van der Zee & Scholten (2013)

Figure 3.2: *Continuous loop of sensing, analyzing, predicting, and act(uat)ing in a smart city*

According to the ISO/TC 268 (2018), the term *Smart City* describes a city, which "dramatically increases the pace at which it improves its sustainability and resilience". Hence, it is an umbrella term for different intelligent developments within a city and its community. One aspect is the use of sensors, actuators and communication networks to monitor and control city systems and processes, for instance, infrastructural facilities. It enables applications in a city such as smart mobility, smart environment, smart energy or smart citizen, which highly integrate spatial data. Thus, the feedback loop provided by Figure 3.2 is a data and control flow model for these applications.

In such a feedback system, sensors or sensing ranges are selected and activated based on their location. They observe their environment, which refers to *spatial sensing*, and emit current spatial events. These are the basic input parameters for *spatial modeling* and *spatial analysis* in the orientation phase. Different algorithms are applied such as simple filtering and functions on single events or CEP methods to analyze relationships between multiple events. Based on the analysis of spatial observations and predictions, (spatial) decision can be made in real-time. Meaningful events are generated, which select and activate appropriate actuators, so that actuating ranges spatially (*spatial acting*). Processes in the physical world are finally invoked and driven. Subsequently, their influences can be observed by sensors starting the feedback loop all over again.

The control model based on the feedback loop represents a cognitive-aware approach to the IoT incorporating response-mechanisms to observations, analyses and decisions. Although proposed for smart cities, it can be obviously applied to other Geospatial IoT applications as well. The driver of this loop are IoT devices and their emitted spatiotemporal events e.g. measurements of a real-world process or a state. The control depends on the flow of events like Figure 3.2 suggests. Thus, we need to define spatiotemporal events in this context. Since the proposed model starts by observing events in the physical world, we start investigating the concept of spatiotemporal events in the real-world in the following. Then, we introduce the temporal and the spatial component (Sections 3.3 & 3.4), before defining an architecture for a Geospatial IoT based on spatiotemporal events in Section 3.5.

3.2 MODELING REAL-WORLD EVENTS

In a Geospatial IoT, real-world events are observed by sensors or are driven by actuators. Intrinsically, they implicate a temporal, but sometimes also a spatial component. In a Geospatial IoT both components are relevant. Therefore, we will call them in the following spatiotemporal events. An architecture based on spatiotemporal events must define a specific data type. However, before doing so, we need to clarify the character of the events we observe. First, we conduct a brief literature review, how the term "spatiotemporal event" is defined in other IoT applications and especially in the field of GIS software. Subsequently, we approach the notion of events and its related concepts and finally define geospatial processes, events and states from an ontological point of view (see Section 3.2.3).

3.2.1 SPATIOTEMPORAL EVENTS IN IOT APPLICATIONS

How can spatiotemporal events be defined? In the literature, spatiotemporal events in IoT applications are not consistently defined and often related to the specific use case or application. In the following, different views on spatiotemporal events are described. They range from modeling real-world occurrences to definitions of spatiotemporal events in system architectures and databases.

For Hornsby & Cole (2007) a spatiotemporal event in a vessel tracking setting is a 4-tuple:

$$ {}^{v}_{id}e^{zone}_{occurTime} \tag{3.1}$$

It has an *id*, which is the identifier of the associated object, a *zone*, which is the spatial region where the event happens and an *occurTime*, which is a timestamp reflecting the event's occurrence time. They also added a term v for capturing important semantics about the nature of the event in *attribute name:value* pairs.

Contrarily, Jin & Chen (2010) adopt an interval-based method to denote the duration of an event occurrence, with a starting and ending time. Further, they use spatial data types such as point, line and region to represent spatial objects.

Morales & Garcia (2015) define a so-called *g-event* for the implementation of GeoSmart City architectures. A g-event is an occurrence of a "change of state associated to a phenomenon of interest, and which is related to a geographic location and a specific time." They use this definition for so-called *low level g-events* while *high level g-events* also include the characteristics of recurrence and causality.

From a stream processing point of view in an EDA, Liebig & Morik (2013) argue that three type of spatiotemporal data streams exist whereby an event is a central concept:

1. A $spatial\ time\ series$ consists of tuples $(attribute, object, time, location)$.
2. An $event_i$ is triggered from a spatial time series under certain conditions and contains the tuples verifying these conditions $(event_i, object_n, time_n, location_n)$.
3. A $trajectory$ is a partial time series for a particular $object_i$. A trajectory is a series of tuples $(object_i, time_n, location_n)$.

Güting & Schneider (2005) characterize application data for spatiotemporal databases. They distinguish between ten different categories of spatiotemporal events and give example applications:

1. *Events in space and time - (point, instant): plane crash, volcano eruptions.*
2. *Locations valid for a certain period of time – (point, period): construction sites, coal mines.*
3. *Set of location events – sequence of (point, instant): collections of (1), volcano eruptions of the last year*.
4. *Stepwise constant locations – sequence of (point, period): the capital of a country, the headquarter of a company.*
5. *Moving entities – moving point: people, planes, cars*
6. *Region events in space and time – (region, instant): forest fire*
7. *Regions valid for some period of time – (region, period): closed area for a certain time after a traffic accident*
8. *Set of region events – sequence of (region, instant): Olympic games viewed collectively*
9. *Stepwise constant regions – sequence of (region, period): countries, agricultural land use*
10. *Moving entities with extent – moving region: growth of forests, people in history*

In a more generic approach, Tan et al. (2009) tries to define a spatiotemporal event for CPSs. According to them a spatiotemporal event is an occurrence of interest, which describes the state of one or more objects either in the cyber-world or the physical world according to attributes, time and location. This is specifically denoted as:

$$\mathcal{E}id \quad \{t^o_{\mathcal{E}id}, l^o_{\mathcal{E}id}, V_{\mathcal{E}id}\} \tag{3.2}$$

where $\mathcal{E}$ is the event type identifier, id is the event ID, $t^o_{\mathcal{E}id}$ is the event occurrence time, $l^o_{\mathcal{E}id}$ is the event occurrence location and $V_{\mathcal{E}id}$ is a set of event occurrence attributes. The occurrence time can be either of punctual type or interval type, while the occurrence location may be a point or a field. In their model, they classify events into different classes. While a *physical event* models the occurrence of the end-user's interest in the physical world, a *sensor event* is the actual digital representation of a physical observation. Moreover, *cyber-physical events* are generated from sensor events based on cyber-physical event conditions.

3.2.2 SPATIOTEMPORAL MODELING IN GIS

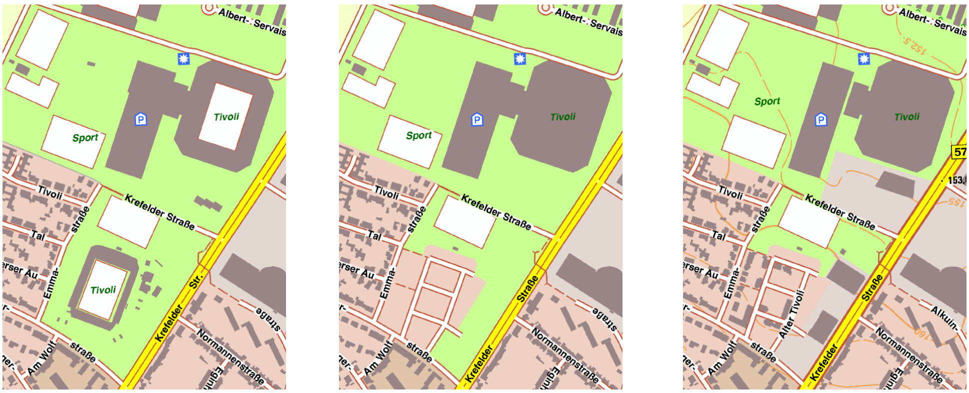
Also from a Geographic Information Science (GIScience) point of view, some efforts have been made to introduce various concepts of time in the traditional model of spatial data. From a modeling perspective, Worboys (2005) and Peuquet (2005) investigated the evolution of spatiotemporal modeling in GIS. They identified three stages of evolution phases from snapshot to event-based modeling.

However, Worboys (2005) also introduces a *stage zero*, which defines a static GIS with only a single state of knowledge, a single moment about the application domain. Basically, traditional GIS technology can handle this stage only. Stage zero technology allows to represent past or future but only in a single moment. So, no comparisons between the states of affairs at different times are possible. Therefore, this stage actually does not include spatiotemporal modeling.

3.2.2.1 Stage One: The Snapshot Model

Stage one introduces temporal snapshots which represent a specific state in a domain at a single moment in time. A temporal sequence describes a collection of snapshots of spatial configurations of objects, mostly of the same spatial region, indexed by a temporal variable. In Figure 3.3 three temporal snapshot of the Soers area in Aachen are depicted. The temporal sequence shows topographic maps (Digitale Topographische Karte 1:10000, DTK10) from the years 2011, 2014 and 2017. Over the time, different changes can be observed, like the deconstruction of the old tivoli stadium and the following erection of the residential area in the same place.

The snapshot model is by far the most common method for spatiotemporal modeling of the world in databases. Usually the model employs a grid data model like also used in Figure 3.3, however, also vector models may be utilized (Peuquet, 1999). Additionally, the layers include information about a single thematic domain at selective timestamps, whose temporal distances do not have to be necessarily uniform. It is also possible to represent past or future states of objects but only of single instants in time. According to Worboys (2005), it depends on the nature of the geographic phenomena under consideration how the model is structured. If a continuous event such as the movement of a glacier should be modeled, the model and its time domain should allow for interpolation between the snapshots. Otherwise, for discrete events - for instance a change in administrative boundaries - the temporal domain of the model should reflect this discrete nature. Peuquet (1999) state that with this straightforward approach, "the state of any location or entity at a given time can be easily retrieved." But, she also describes three drawbacks, which are inherent for the snapshot model:



(a) *DTK10 2011* **(b)** *DTK10 2014* **(c)** *DTK10 2017*

Source: Author's illustration

Figure 3.3: *History of the Soers area in Aachen.*

1. Each snapshot represents a complete map of the entire region, which increases the data volume enormously when the number of snapshots increases. But in most cases the spatial changes in two adjacent snapshots are marginal so that the amount of redundant data also increases.

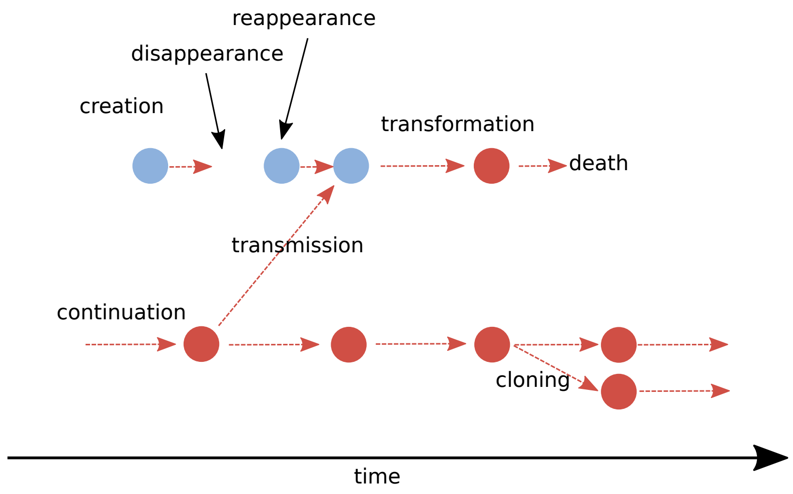
2. Detecting changes of spatial entities in two consecutive snapshots is a time consuming and computing expensive process, since the changes are stored implicitly and can only be retrieved by cell-by-cell comparisons. Furthermore, it may happen that short-lived changes at some location are not represented at all by two adjacent snapshots.

3. The timestamp of a change to a specific spatial entity cannot be determined exactly. From the snapshots in the example of the Soers area (see Figure 3.3), you can only tell that the demolition of the old tivoli stadium must have happened between 2011 and 2014, but the exact date cannot be retrieved.

Meeting these drawbacks, Langran (1990) and Peuquet & Qian (1996) modified the grid model in a way, that each pixel in the grid owns a list of variable length, which stores each change at that specific location with a new entry. This way, the event history for each cell location is maintained by an associated list in temporal order.

3.2.2.2 Stage Two: The Object Model

Stage two represents the change of objects and their spatial configurations. In this stage the focus is shifted from the temporal sequence of objects to changes of objects, thus, Worboys (2005) terms this the object-oriented view. For instance, in Figure 3.3 the old tivoli stadium is demolished between the temporal snapshots of 2011 and 2014, but this information is solely implicitly shown in the maps. The actual event of the deconstruction cannot be represented with snapshots like in stage one. Thus, Hornsby & Egenhofer (2000) introduced a change-based model with the two primitives *identity states of objects* and *transitions*. An *object* represents a real-world phenomenon in an information system, either a physical entity, such as a building or lake, or a conceptional object such as states or borders. The *object identity* is a key concept since it provides information about the existence or non-existence as well as the capability of tracking changes or differences of a spatial object. *Identity states* associated with objects allow assigning a possibly changing state to objects. Possible states of objects are, for instance, existing or non-existing. Finally, the *transition* primitive models the progression between object states. Some possible transitions are depicted in Figure 3.4.



Source: adapted from Worboys (2005)

Figure 3.4: *Object-change history*

Creation, disappearance, reappearance, transformation and death can be applied to a single object, while cloning is an operation performed by an object to replicate itself and the transmission operation is performed by one object on another. Besides change primitives such as creation or destruction, movement occurs when a physical object changes its position. This can be either continuous like a moving vehicle along the highway or discontinuous like a relocation of a boundary. Polous (2016) states that these primitives are in fact events which happen to objects. However, for explaining complex changes, an event model is essential, which is the third stage of spatiotemporal modeling.

3.2.2.3 Stage Three: The Event Model

Stage three is a full treatment of change in terms of events, actions (or activities) and processes. These terms describe slightly different meanings of occurrents in the real world but also throughout the literature: For Claramunt & Theriault (1995) events are things that happen, according to Peuquet (1994) an event denotes some change in some location(s) to some object(s), while Peuquet & Duan (1995) define an event as the representation of the spatiotemporal manifestation of some process. Further, Worboys (2005) sees an action as an event which is initiated and sometimes terminated by humans or non-human agents. He gives further the example of photosynthesis as a process but the typing of a sentence by an author as an event. However, all of these occurrents represent happenings in the real world. Complex occurrents are modeled by the ways how objects are participating in them and how the involved occurrents are related to each other. So far, many researchers proposed a range of event-based models in contrast to object-based models (e.g. Claramunt & Theriault, 1995; Peuquet & Duan, 1995; Worboys & Hornsy, 2004; Worboys, 2005). In an event-based model the sequence of events is essential, thus, the temporal component dominates the spatial component (Beard, 2006). While object-based models (stage two) keep track of the changes of properties and attributes belonging to geographic objects, event-based models (stage three) focus on the analysis of the changed object and record particular attributes such as start time, period or the method of accomplishment. Therefore, Galton (2004) distinguishes between *histories*, which are functions from a temporal domain to attribute values or properties of objects, and *chronicles*, that treat dynamic phenomena as collections of occurrents. Grenon & Smith (2004) make the same differentiation; for temporal sequences of object configuration, they propose the SNAP ontology while for the event/action/process view they formulate the SPAN ontology. SNAP is the ontology of what exists (at a moment in time - a SNAPshot), whilst SPAN describes the ontology of what happens (SPANning a period of time). More on these definitions and distinctions are described in Section 3.2.3, where we develop a definition of events and processes in a Geospatial IoT.

However, storing data in an event-view fashion can be useful for understanding processes since event-based approaches allow the representation of dynamic behavior, hypothesis generation, scientific investigation of complex relationships, an ability to investigate causal linkages and associate entities with influences and underlying processes. There have been early calls for keeping track of events and processes. However, according to Beard (2006) the realization is difficult because it owes much to new technologies. But she argues that technology improvements in e.g. environmental monitoring and initiated fine temporal resolved sensor data streams support the analysis of change and provide a picture of how processes operate.

3.2.3 GEOSPATIAL PROCESSES, EVENTS AND STATES

Before defining events and processes in a Geospatial IoT, both terms has to be investigated further. Throughout the literature events and processes are not homogeneously defined. Its characterization depends mainly on the discipline background but also the application domain. We already introduced events in EDA (see Section 2.5.3), how spatiotemporal data is modeled in GIS (Section 3.2.2) and, exemplarily, how other applications model events (Section 3.2.1). Since the IoT connects physical object and processes to the Internet, we focus on the events and processes, which happen in the real-world first. Therefore, e.g. an event definition such as in EDA is not sufficient.

Omitting the spatial component at first, event and process models store temporal information in terms of *events* or *processes* intrinsically. Like already stated, both notions share common attributes and an explicit distinction is hard to accomplish. Galton (2015) argues that both terms are not defined homogeneously in the literature and may imply sometimes the same thing:

> 'One person's process is another's event, and vice versa' (Worboys, 2005)

A review of the relevant literature reveals various definitions and relations between both concepts, for instance:

- Mourelatos (1978) describes *situations* as the overall concept, which can be divided into *states* and *occurrences*. The latter one can be subdivided further into *processes* and *events*.
- Pustejovsky (1991), on the other hand, regards an *event* as a base type with its manifestations *state*, *process* and *transition*.
- For Allen (1984), state, process and event have the same base.
- Campelo & Bennett (2013) distinguish *events* from *processes* by means of the temporal dimension. Whilst events are entities whose properties are not subject to change over time, processes are entities that are subject to change over time such as the processes *accelerating* or *slowing down*.

- Sowa (2000) defines *continuous processes* and *discrete processes*. Continuous processes are subdivided into *initiation*, *continuation* and *cessation*, while discrete processes are *states* or *events*

That is why Galton (2015) calls for a theoretical framework of processes and events. Several ontologies are defined to face this definition problem such as the Basic Formal Ontology (BFO) (Smith et al., 2015) or the Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) (Masolo et al., 2003). As an example, we give an introduction into the BFO in the following.

BASIC FORMAL ONTOLOGY (BFO)

The BFO makes use of the SNAP and the SPAN ontologies (Grenon & Smith, 2004) already mentioned in Section 3.2.2. The components of the two ontologies are called *continuants* and *occurrents*:

- A *continuant* endures through time, can lose or gain parts and its properties may change. It has spatial parts but no temporal parts.
- An *occurrent* exists over a span of time. It cannot undergo change but its occurrence may result in changes in various continuants. An occurrent has temporal and possibly also spatial parts.

Galton (2016) gives an example to illustrate the difference between SNAP and SPAN and, therefore, for continuants and occurrents. He considers a person as a continuant. The portrait of this person can be captured through snapshots at particular dates, e.g. every year. This describes a SNAP ontology since the person in the snapshots is always the same continuant. The corresponding SPAN entity of the sequence of portraits is the person's life which is an occurrent. This entity consists of multiple temporal parts which are by itself occurrents. This can be, for example, the person's schooldays or the person's wedding. However, each of these occurrents depends on the continuant, the person. Projecting these concepts to an IoT application, consider a smart mobility solution with buses in a city. In this, each bus embodies a continuant while the corresponding rides between the stations are occurrents.

The BFO does not distinguish any further between events and processes. It defines a process p as "an occurrent that has temporal proper parts and for some time t, p $s\text{-}depends\_on$ some material entity at t" , where "a material entity is an independent continuant" (Smith et al., 2015). $s\text{-}depends\_on$, which stands for *specifically depends on*, means that if an entity b $s\text{-}depends\_on$ c, b can only exists if c exists too. In the previous smart mobility example, the bus ride $s\text{-}depends\_on$ the bus. So, a bus ride is a process for some time t and the material entity is the bus itself but also e.g. each passenger.

The process notion in the BFO includes that processes do not have qualities, which means that corresponding occurrents cannot change for the time they exist (Smith et al., 2015). This implies for the process "bus ride" that the speed of motion must either be constant - which is unrealistic - or the process must be subdivided into multiple processes with temporal proper parts. Galton (2016) states that instantaneous speed at a timestamp cannot be snapped in BFO. The only possibility is to reduce the measurement of speed to "average speed over an interval" which supports scientific correctness but "is not so friendly from the point of view of more everyday human purposes".

PROCESS AS PATTERNS OF OCCURRENCE

Processes can also be understood as abstract entities whose occurrences in the physical world cause continuants to undergo change (Özgövde & Grüninger, 2010). This view uncouples the process notion from the concepts of continuants and occurrents. Özgövde & Grüninger (2010) further define *process occurrences* as occurrents which instantiate or realize the corresponding abstract processes. They may have multiple temporal parts (sub-occurrences) and have a beginning as well as an end point in time. Galton (2016) gives the example of "walking" as an abstract process, which he calls a pattern of activity. This process can be realized in the physical world by process occurrences, which can be initiated by continuants. For instance, the following two occurrences describe the process "walking" in the real world realized by a person named Mary:

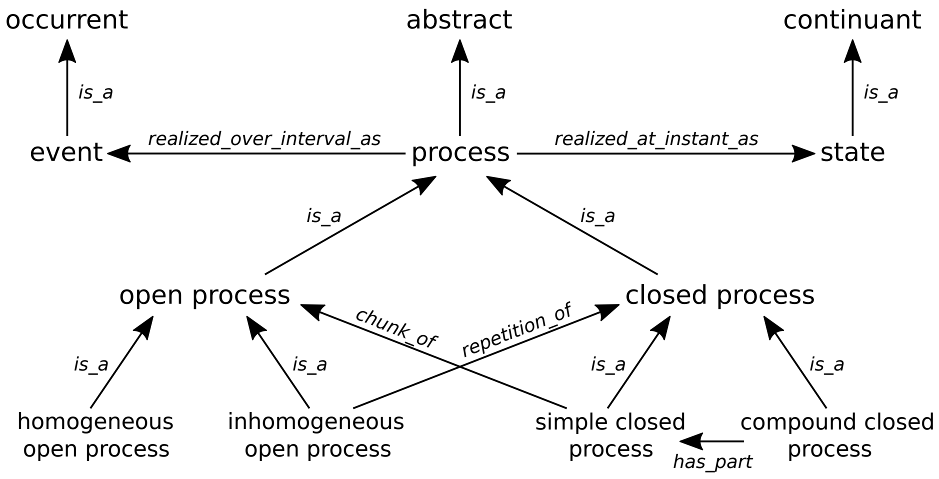
1. *Mary is walking*
2. *Mary is walking along the path from 10 am to 10.30 am*

However, Galton (2016) argues that although both process occurrences are instances of the abstract process "walking", the occurrences describe different patterns. While *Mary is walking* characterizes a state in which Mary is in, *Mary is walking along the path from 10 am to 10.30 am* has an event character. He concludes that two types of processes exist: open and closed processes:

- An *open process* describe an open-ended activity which can be repeated in the same way endlessly, such as human activities (walking, driving) or non-human activities (raining, coastal erosion). Although the process itself does not determine when it comes to an end, any actual occurrence - e.g. of walking - must come to an end, which is imposed by external factors.
- A *closed process*, on the other hand, is a finite routine with an intrinsic start and end point. If the process occurrence is completed, this realization cannot be

continued. But another instantiation of the same process can start. Examples for closed processes are a specific walking tour or a bus ride, which involve human activities, or an explicit rain shower, which is a natural event.

Galton (2016) derives a fragment of an ontology from this view on processes which is illustrated in Figure 3.5. For closed processes, the additional distinction between simple and compound closed processes holds. A simple closed process is a chunk of an open process such as *Mary is walking from A to B* is made up of *Mary is walking*. A compound closed process is an assembly of simple closed processes such as a flight from New York to Paris with the closed processes *take off*, *flying phase* and *landing* at particular points in time. Open processes, on the other hand, are divided into homogeneous with repeating patterns such as *walking* and inhomogeneous open processes. The latter one provides the capability to define an open process in terms of a closed process by a simple repetition operation.



Source: adapted from Galton (2016)

Figure 3.5: *Process as patterns of activity*

Unlike the BFO, the advantage of this ontology is that real-world processes (process occurrences) can also be portrayed at single points in time. Seeing processes as patterns of activity allows to model open and closed processes as events and states. At this, an *event* represents an occurrent, an instantiation of an abstract closed process in the physical world, which is realized over a time interval. Furthermore, an abstract open process can be realized and described at an instant as a *state* of a continuant in the physical world. Galton (2018) summarizes that "processes are radically different in kind from states and events: processes are higher-level, abstract patterns that are realized concretely as states or events".

DERIVING GEOSPATIAL PROCESS, EVENT AND STATE

Regarding the previous sections, it becomes pretty clear that a universal definition of the concepts of processes and events is not available. So, what about spatiotemporal events or processes in a Geospatial IoT? The various event definitions for IoT applications (see Section 3.2.1) and the third stage of spatiotemporal modeling (see Section 3.2.2), show that the basic unit of spatiotemporal information is a combination of place (where), time (when) and theme (what):

In place p at time t there is X.

This is referred as Peuquet's *Triad Framework* (Peuquet, 1994) and is generally accepted from a data-modeling point of view. In this generalized view, p can be point, a region or a grid square; t can be an instant, standard interval like a month or an arbitrary interval; and X can be an object, a value, an event or process (Galton, 2015).

Combining Peuquet's Triad Framework with the processes as patterns of occurrence ontology by Galton (2018), we can now derive the spatiotemporal versions of the different concepts for a Geospatial IoT.

Definition 3.2 (Geospatial Process)
A geospatial process is an abstract process with a spatial and a temporal component.

A geospatial process describes an open or a closed process according to Galton's processes as patterns of occurrence ontology but necessarily with a spatial component. Like in Figure 3.5, a geospatial process can be instantiated or realized through a geospatial process occurrence in the physical world. This is either a geospatial event or a geospatial state of a continuant. The Definitions 3.3 & 3.4 can be derived.

A geospatial event can be composed of a sequence of smaller geospatial event. In this case, "smaller" means that both components (spatial and temporal) are a subspace of the compound geospatial event. The flight example introduced early illustrates this concept. The different phases in a flight from New York to Paris, e.g. taking off in New York, cruising and landing in Paris, have their own trajectory and temporal spreading and can each be realized by geospatial events. By composing the flight phases the compound geospatial event "the flight from New York to Paris" can be modeled. On the other hand, the open process *Plane A is flying* can be realized as a geospatial state of the continuant "Plane A". The plane is in this specific state of flying at a certain instant in time and at a certain location.

Definition 3.3 (Geospatial Event)
A geospatial event is an instantiation of a closed geospatial process in the physical world, which is characterized over a time interval (with defined start and end point) and a spatial component. A geospatial event can be described by a 3-tuple containing a spatial part, an interval and a theme (event name).

Definition 3.4 (Geospatial State)
A geospatial state is an instantiation of an open geospatial process in the physical world which is characterized by a 3-tuple consisting of a time instant, a spatial component and a theme. It describes a property or activity of a continuant (theme) at a timestamp. The spatial part specifies the spatial occupation of the continuant or its property.

3.3 TEMPORAL COMPONENT OF GEOSPATIAL PROCESSES

Time can be denoted in two ways: a point-based method and an interval-based method. While some models consider the occurrence time as a time point, other model it as a time interval. Tan et al. (2009) argue that the event "a light is on for the last 30 minutes" can only be defined by considering the temporal component of an event by using time intervals. Sometimes events are modeled instantaneous so that they have no duration but mark a change in state of e.g. an object (Hornsby & Cole, 2007). In other projects, the event occurrence is modeled by an interval to denote the duration (Jin et al., 2013).

With the definitions of geospatial events and geospatial states in the previous section, we meet Tan et al.'s requirement that both temporal events types - punctual and interval events - are required in an event model for CPS (Tan et al., 2009). The concept of the abstract geospatial process can model both types either as an open geospatial process, which is an instantaneous geospatial state of a continuant, or a closed geospatial process, which instantiates as a geospatial event over an interval with a start and an end point on the timeline.

3.3.1 TIME DOMAIN

Time describes a one-dimensional space extending from the past to the future, which is referred as the timeline or time axis. The *density* of a timeline can be describes as *discrete*, *dense* or *continuous*. Discrete models are isomorphic to natural numbers, which implies that each instant has a single successor. Dense and continuous models are isomorphic to the rational respectively to real number (Özsoyoglu & Snodgrass, 1995). Time itself is continuous but it is conventionally broken into discrete units of uniform or variable length (Peuquet, 1999). In the discrete model, a so-called *chronon* describes an "atomic" time interval, the smallest unit of recorded time depending on the application. A particular chronon is represented by a natural number on the timeline. In the dense and the continuous model, on the other hand, another chronon exists between any two chronons. Güting & Schneider (2005) argues that most people perceive time as continuous, but discrete representation are often used for practical reasons. Therefore, the discrete model is used in this thesis.

Temporal locations can be located relatively to other temporal locations on the timeline. Specifying a temporal location in an absolute system (or anchored time) requires an agreed standard reference location. In the Gregorian calendar this reference point in time is the assumed time of Christ's birth, whilst in cosmological contexts, temporal locations may be referred to the Big Bang (Galton, 2009). For instance, "2018-11-12 11:00:14" is an absolute time, but "nine days" is relative (or unanchored time).

Several data types can be modeled on the timeline. An *instant* is named a point on the timeline, which is represented by a *timestamp*. In a discrete model, it is actually not a point but a line segment on the timeline (Özsoyoglu & Snodgrass, 1995). For instance, assuming a second as the chronon in a discrete model, the timestamp "2018-11-12 11:00:14" represents the second which lasts from "2018-11-12 11:00:14" to "2018-11-12 11:00:15". This implies that the exact instant is never precisely known. A special instant represents the concept of *now*, which terms the current time as an instant and separates the timeline into past and future. Further, a *period* is a duration of time, which is anchored between two instants. A period can be modeled by a *period timestamp*. They can be closed, half-open or open. In the literature, a set of disjoint anchored periods are called *temporal elements* (Güting & Schneider, 2005). For modeling an unanchored duration, an *interval* is used.

3.3.2 GRANULARITY OF TIME

Granularities are introduced for purposes of convenience (e.g. we say "2 years" instead of "730 days") and for modeling vagueness, e.g. when the exact instant is not known. Examples are birthdates, which are usually specified at granularity of days, or train schedules at granularity of minutes. In discrete models, the smallest possible granularity is that of a chronon, in the previous example a second. *Granules* are larger units of grouped consecutive chronons such as minutes, hours, weeks or years. The largest granularity is the entire timeline. The length of a granule can be fixed or variable. For instance, a single year has either 365 or 366 days depending on whether it is a leap year.

3.3.3 ENCODINGS OF TEMPORAL DATA TYPES

In computer systems the temporal data types *instant*, *period*, *interval* and *temporal element* are encoded in different standardized formats. Some of them are presented in this section.

UNIX TIME

A simple representation of instants in computer systems, which uses seconds as the chronon, is the Unix time (also known as POSIX time or UNIX Epoch time). It does not support any granularities but simply stores the seconds that have elapsed since the 1st January 1970 at 00:00:00 UTC time minus the leap seconds. It is an absolute system with the 1st January 1970 as a standard reference location on the timeline. In Unix time, each day has exactly 86400 seconds, thus, leap seconds are to be subtracted since the Epoch. Since a lot of computer systems store the Unix time as a signed 32-bit number, only $2^{31} - 1$ seconds are possible to represent. The so-called "year 2038 problem" refers to the overflow, which will occur in 2038 leading possibly to wrong behaviors in computer systems.

ISO 8601:2004

The most common format to encode date and time is the ISO 8601:2004 entitled *Data elements and interchange formats – Information interchange – Representation of dates and times*. The following descriptions are taken from the German adaption of the standard, which can be found in ISO/TC 154 (2004).

The ISO 8601 covers multiple data and time formats, especially it defines rules to express dates, time instants, time intervals and repeating time intervals in the Gregorian calendar. However, all encodings follow the following principles:

- Both data and time formats are ordered descending from left to right, starting with the largest to the smallest unit of time. This include: year (Y), month (M), day (D), hour (h), minute (m), second (s) and fraction of second.
- The units have a fixed number of digits in each format. Leading zeros must be introduced if necessary.
- Two formats, a basic and an extended format, are valid. The extended version facilitates human readability. It uses the hyphen "-" to separate date values, while the colon ":" separates the time values. In the basic format theses separators are omitted.
- Granules can be dropped to reduce accuracy. This is however only valid if values are dropped in the order from the least to the most significant.

- The standard allows adding decimal fraction to the smallest time value depending on the particular specification.

Based on these rules, a date can be represented by [YYYY][MM][DD] in basic or [YYYY]-[MM]-[DD] in extended format with [YYYY] for year, [MM] for month and [DD] for day. For instance, the corresponding date representation for November 2nd, 2018 in the extended format is "2018-11-02" respectively "20181102" in basic format. Similarly for local time, the basic format is [hh][mm][ss] and the extended format is [hh]:[mm]:[ss] with [hh] for hour, [mm] for minutes and [ss] for seconds. Since the ISO 8601 uses a 24-hour clock system, the string "15:43:27" (or in basic format "154327") represents 43 minutes and 27 seconds past 3 pm. For UTC time, the letter 'Z' is added to the time string, while for a difference to local time, a $\pm$[hh]:[mm] (in basic format $\pm$hmm) can be added. For example, let the previously used time be in UTC time "15:43:27Z", the corresponding string in Central European Time (CET) is "16:43:27+01:00".

Date and time formats can be combined to represent an instant in time with a timestamp including the date. Basically, the date string is concatenated with a time string separated by the letter 'T'. For the extended format, this compiles to [YYYY]-[MM]-[DD]T[hh]:[mm]:[ss]. Depending on the time zone, a 'Z' for UTC or a difference to UTC $\pm$[hh]:[mm] can be added. For example, the string "2018-11-02T16:43:27+01:00" denotes November 2nd, 2018 at 43 minutes and 27 seconds past 4 pm in CET time.

Beside timestamps to define instants in time, the standard allows to represent time intervals. The simplest representation of an anchored time interval (a period) is through the concatenation of a start and an end timestamp separated by a slash '/'. For instance, the string "2018-11-02T16:43:27+01:00/2018-11-02T17:03:50+01:00" denotes an anchored time interval in extended format that starts at November 2nd, 2018 at 43 minutes and 27 seconds past 4 pm (CET) and ends November 2nd, 2018 at 3 minutes and 50 seconds past 5 pm (CET).

With the duration format, an unanchored time interval can also be encoded. A duration expression starts with the letter 'P' (for 'period') to indicate that the following must be interpreted as a duration. The entire expression P[Y]Y[M]M[D]DT[h]H[m]M[s]S consists of the amount of each unit, where [Y] stands for years, [M] for minutes, [D] for days, [h] for hour, [m] for minutes and [s] for seconds. Each amount can have one or more digits. For example, the string "P1Y2M10DT2H30M" defines a duration of one year, two month, ten days, two hours and 30 minutes. However, it is also possible to use either a start timestamp and a duration string, or a duration expression and an end timestamp separated by slash to form an anchored time interval. Table 3.1 illustrates the same anchored time interval in multiple ways.

Table 3.1: *Interval definition in IS0 8601*

Name	Expression
Start-end	2017-08-01T15:43:27/2018-11-02T17:03:50
Start-duration	2017-08-01T15:43:27/P1Y3M1DT2H20M23S
Duration-end	P1Y3M1DT2H20M23S/2018-11-02T17:03:50

Source: Author's illustration

Time intervals can also be recurrent, which is considered in the ISO 8601 standard as well. Repeating intervals are formed by prefixing "R[n]/" to the interval string, where 'R' is the letter and [n] is replaced by a number indicating the amount of repetitions. If [n] is omitted, an unbounded number of repetitions are realized. E.g. the string "R5/2017-08-01T15:43:27/P1Y3M1DT2H20M23S" means that the interval is repeated five times starting from "2017-08-01T15:43:27".

The ISO 8601 is related to various national standards such as the German DIN ISO 8601:2006-09. Also, the W3C defines in the RFC 3339 (Klyne & Newman, 2002) a profile of the ISO 8601, which restricts the supported date and time formats for the Internet.

OTHER ENCODINGS

In the history of multimedia systems, there have been various proposed and established standards for the notation of times and dates. The RFC 822 (Crocker, 1982) and its updates define the format of E-mail messages starting from 1982. The most common encoding was something like "Fri, 2 November 2018 16:43:27 +0100 CET". But, the valid variations made it difficult to process the date string. Nevertheless, it remained valid for a very long time, but was finally replaced by its successor RFC 2822 (Resnick, 2001) in 2001, which restricts all formats to the exemplified above.

Other applications used similar but slightly different encodings. In RFC 1036 (Horton & Adams, 1987), the USENET message type is defined. An example for a date in this RFC is "Friday, 02-Nov-18 16:43:27 CET". It is also still in common use (e.g. in HTTP) but lacks a four-digit year. Further, the iCalendar standard (RFC 2445) (Dawson & Stenerson, 1998) generally adopts the date and time basic format of the ISO 8601 but introduces a different notation for time zones. The timestamp used above in the iCalendar format depict as "TZID=CET:20181102T164327".

A time interval is similarly defined by using the parameters DTSTART and DTEND:

DTSTART;TZID=CET:20181102T164327
DTEND;TZID=CET:20181102T170350

3.3.4 TEMPORAL RELATIONS OF GEOSPATIAL PROCESSES

Temporal locations on a timeline can be related to other temporal locations. Like in the absolute system of the Gregorian calendar, specific instants are numbered by a timestamp before or after Christ's birth. Considering temporal relationships between geospatial event and states in a Geospatial IoT is essential to order, analyze and link them on a timeline. Tan et al. (2009) argue that for the sake of completion, the relationships between punctual and interval events should be considered in a CPS model.

Therefore, the following sections discuss the possible relationships between the different temporal data types. This includes the relations between two instants, which is named *point-point relations* in the literature, between an instant and a period and vice versa (*interval-point relations*) and, finally, between two periods (*interval-interval relations*) (Vilain, 1982).

3.3.4.1 Point-Point Relations

The interrelations between points on a timeline can be described by primitives in a logic. Basically, three primitives exist between two points: *before*, *equals*, *after*. Let a point p and a point q be defined by variables over the set of real numbers $\mathbb{R}$ representing each an instant on a timeline. The relations in Table 3.2 hold.

Although p and q are real numbers here, the same relations hold for instants in a discrete time model. Further, the temporal relations between two instants on a timeline correspond to the temporal relationship between two geospatial states of one or more continuants.

Table 3.2: *Point-Point-Relations*

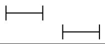
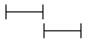
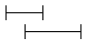
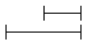
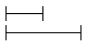
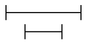
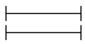
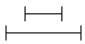
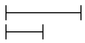
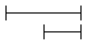
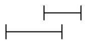
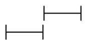
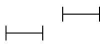
Name	Depiction	Relation
p equals q	⊗	$p = q$
p before q	● ×	$p < q$
p after q	× ●	$p > q$

Source: based on Vilain (1982)

3.3.4.2 Interval-Interval Relations

The relationship between the temporal components of two geospatial events can be expressed by an interval-interval relation. Allen's temporal interval algebra (Allen, 1983) introduced in 1983 is a fundamental building block in temporal reasoning and provides a set of relations between two intervals I and J and corresponding operations. From observations, Allen concluded that the amount of topological relations between these two intervals is limited. In his paper he proposed 13 basic relations between time intervals which are distinct, exhaustive and qualitative (Alspaugh, 2019).

Table 3.3: *Interval-Interval-Relations according to Allen's interval algebra*

Name	Depiction	Endpoint relation
I precedes J		$I^+ < J^-$
I meets J		$I^+ = J^-$
I overlaps J		$I^- < J^- < I^+ < J^+$
I finishes J		$I^- < J^- \leq J^+ = I^+$
I starts J		$I^- = J^- \leq I^+ < J^+$
I encloses J		$I^- < J^- \leq J^+ < I^+$
I equivalent to J		$I^- = J^- \leq J^+ = I^+$
I enclosed by J		$J^- < I^- \leq I^+ < J^+$
I started by J		$J^- = I^- \leq J^+ < I^+$
I finished by J		$J^- < I^- \leq J^+ = I^+$
I overlapped by J		$J^- < I^- < J^+ < I^+$
I met by J		$I^- = J^+$
I preceded by J		$I^- > J^+$

Source: based on Allen (1983)

The intervals I and J are represented by pairs $\langle I^-, I^+\rangle$, where $I^- \leq I^+$, respectively $\langle J^-, J^+\rangle$, where $J^- \leq J^+$. The superscript $^-$ indicates the begin instant of the interval, while $^+$ denotes the end instant of the interval. Like in the point-point-relation, begin and end points are interpreted over $\mathbb{R}$. Table 3.3 illustrates Allen's 13 basic relations between the two intervals I and J.

The first column is the name of the relation, the second column defines the relation graphically by a diagram relating I and J with running time from left to right (I is the upper interval) and, finally, the third column expresses the relation in terms of the intervals' boundaries. For instance, the relation I *meets* J means that I ends when J begins. They meet in exactly one instant, which is I^+, respectively J^-.

From the 13 relations, six pairs are converses; the six relations below *equivalent to* are the inverse of the above ones. This means e.g. that if I *meets* J is true, then also J *met by* I holds. The *equivalent to* relation is its own converse. The predicates for the relations used in the table are not homogeneously in the relevant literature. For example, in his original paper, Allen (1983) uses the expression *before* for the here illustrated *precedes* relation. In the following, we use the expressions for the relations like stated in the table.

Allen's 13 relations are pairwise disjoint and jointly exhaustive, meaning that given any two intervals, exactly one of the relations must hold (Galton, 2009). But based on these 13 interval-interval relations, other relations can be derived through disjunctions. For instance, the relation, which we call *disjoint*, describes two intervals that do not share one common instant. This is achieved by linking the *precedes* and the *preceded by* relationships logically by an "or" function, or in terms of endpoint relations $I^+ < J^- \vee I^- > J^+$.

For convenience reasons, Allen (1983) suggests collapsing the three *during* relations (*encloses*, *starts*, *finishes*) to derive the *contains* relation. Basically, it is identical with the *encloses* relation but includes the boundary points. The derived relations and their endpoint relations are shown in Table 3.4. Since the relations are disjunctions of the 13 basic relations, a distinct graphical representation is not possible.

Table 3.4: *Derived Interval-Interval-Relation*

Name	Endpoint relation
I disjoint J	$I^+ < J^- \vee I^- > J^+$
I contains J	$I^- \leq J^- \leq J^+ \leq I^+$
I contained by J	$J^- \leq I^- \leq I^+ \leq J^+$

Source: Author's illustration

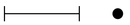
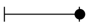
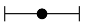
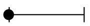
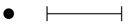
Like mentioned, derived relations are disjunctions of the 13 basic relations, which can be created for convenience purposes. Overall, $2^{13} = 8192$ relations can be theoretically derived, which is known as the full interval algebra (Galton, 2009). However, we only consider the one stated in the table.

3.3.4.3 Interval-Point Relations

Finally, to analyze the temporal relation between a geospatial state and an event, a period must be related to an instant. In the literature, this is called *interval-point relations*, which were introduced by Vilain (1982). He expands Allen's interval algebra for reasoning about time intervals by adding new primitive relations to deal with time points.

Let I be an interval represented by a pair $\langle I^-, I^+\rangle$, where $I^- \leq I^+$. Again, the superscript $^-$ indicates the beginning of the interval, while $^+$ denotes the ending of the interval. Begin and end points are interpreted over $\mathbb{R}$. Further, let a point p be defined by a variable over the set of real numbers $\mathbb{R}$ representing an instant (point) on a timeline. The following five relations between I and p can be compiled (see Table 3.5).

Table 3.5: *Interval-Point-Relation*

Name	Depiction	Endpoint-point relation	Inverse
I precedes p		$p > I^+$	p preceded by I
I finished by p		$p = I^+$	p finishes I
I encloses p		$I^- < p < I^+$	p enclosed by I
I started by p		$p = I^-$	p starts I
I preceded by p		$p < I^-$	p precedes I

Source: based on Vilain (1982)

The first column represents the name of the relation, the second column depicts a graphical representation with a timeline running from left to right, while the third column states the endpoint-point relation. The last column shows the inverse of the relations, which are named *point-interval relations*.

Like in Allen's interval algebra, several relations can be derived from these five basic relations. For reasons of convenience, we derive an *equivalent to* and a *contains* relation. The *equivalent to* relation holds if both, the interval's start and end point, are equal to the point; the *contains* relation is basically a *encloses* relation including the boundary points of the interval (see Table 3.6). Note that originally Vilain (1982) calls the *encloses* relation "contains" and, therefore, is not equivalent to our derived *contains* relation.

Table 3.6: *Derived Interval-Point-Relation*

Name	Endpoint-point relation	Inverse
I contains p	$I^- \leq p \leq I^+$	p contained by I
I equivalent to p	$I^- = p = I^+$	p equivalent to I

Source: Author's illustration

3.4 SPATIAL COMPONENT OF GEOSPATIAL PROCESSES

Continuants and occurrents in our model have a spatial component like defined in the geospatial state and event type. Spatial components are used to specify the spatial location (e.g., longitude and latitude), spatial extent (e.g. area, perimeter), shape, as well as elevation defined in a spatial reference frame (Shekhar et al., 2015). Usually, representations of the geographical reality in GIS are distinguished into either *geographical fields* or *geographical objects*. Geographical fields represent a continuous geographical variable over some region of the Earth. This can be for instance remote sensing images obtained by sensor systems. Geographical objects, on the other hand, represent individualizable entities of the geographic realm, whose locations are described by georeferenced sets of coordinates. The spatial component of geospatial states of continuants or events embody the character of geographical objects. Hence, we focus here on the second type of geographical representation. The section describes how to establish a spatial reference for geospatial processes, how to represent this reference in a geometry model, different formats to decode geometries and, finally, the possible relations between geometries.

3.4.1 SPATIAL REFERENCING

Spatial referencing or georeferencing describes the procedure of assigning locations to atoms of information. It is especially mandatory in GIS because all information must be mapped to the surface of the Earth. Thus, spatial processes, events and states must be georeferenced to a unique location. Understanding the unique location means to be familiar with the used domain and the applied system. For instance, a georeference called "London" is not globally unique if the domain is unknown, since there might be several cities called London. If the embedded domain of e.g. "Great Britain" is unknown, the actual location cannot be perceived. Further, the system of "city names" is a mandatory requirement to understand the georeferenced location. So sharing locations is only useful and valid, if the receiver knows the domain of the geospatial information and the system that is used.

Geographical locations can be described in many ways, but there are two basic types of georeferencing. The place name type of georeferencing like in the London example refers to *indirect* or *informal georeferencing*, while georeferencing based on longitude and latitude or other numerical spatial referencing systems is called *direct* or *formal georeferencing* (Hill, 2006). Both types of georeferencing are described briefly in the next sections.

3.4.1.1 Indirect Georeferencing System

In an indirect georeferencing system, geographical locations are determined by names and not by including explicit coordinates. Names cover geographical codes or addresses, which are related to known locations. The following compilation shows some indirect georeferencing systems, how and why they are used:

- The earliest and most commonly used georeference in everyday activities are *place names*, which work on different scales. While a lot of names are universally understood (such as "London, capital of Great Britain"), other are only recognized by locals. Sometimes place names indicate large bodies such as continent names, at other times exact locations (e.g. "Brandenburg Gate in Berlin, Germany").
- *Postal addresses and postcodes* work similarly to place names but use a hierarchical system with street addresses on the lowest level. Since mostly street names are only unique in local areas, the postal address includes often higher levels such as city or country names.
- In the EU, *NUTS* denotes a hierarchical classification for addressing spatial areas used in official statistics. They are especially applied to map different statistical parameters in several spatial resolutions.
- The 2013-founded *what3words*[1] is a global indirect georeferencing system of locations with a 3x3 meter resolution. Each cell is encoded by three semantically incoherent words separated by dots. For instance, the string "///dolphin.dressy.stormy" is assigned to the Quadriga atop the Brandenburg Gate in Berlin. The combination of these words is globally unique and can be used to describe locations based on natural language which is more memorable for humans than numbers.

Indirect georeferencing systems have advantages especially in everyday use or for humans to perceive and remember georeferenced locations. Place names e.g. are

[1] https://what3words.com/de

used in conversations, correspondence, reporting and documentation. However, additional steps are required to identify the corresponding location to a place name on a map. For instance, *gazetteers*, which are dictionaries of place names, must be consulted to translate between indirect and direct georeferencing to obtain the geospatial location on a map (Hill, 2006).

3.4.1.2 Direct Georeferencing System

In direct georeferencing systems, coordinates in some Coordinate Reference System (CRS) are utilized to define any point in space uniquely. The CRS defines a coordinate system that is related to the Earth using a so-called geodetic datum. Together with a specific coordinate system, a direct position can be indicated by a set of coordinates (Seeger, 1999). For doing so, reference surfaces of the Earth must be defined.

GEOID

The size and shape of the Earth is not uniform. Thus, reference surfaces of the Earth must be established to define points in space uniquely. For instance, the *geoid* represents the equipotential surface of the Earth's gravity field that approximates best to mean sea level. This means that any point on that surface has the same effective potential with force of gravity acting perpendicular to the surface. With this conceptual surface defined, heights can be measured in accordance to a stable reference surface (Huisman & de By, 2009).

ELLIPSOID

For defining position coordinates, a reference surface is also mandatory. The *ellipsoid* is described by an ellipse which is rotated around the polar axis. It approximates the geoid but is defined by mathematics rather than by physics. This allows to project position coordinates onto a mapping plane. The relationship between the geoid and an ellipsoid is part of the geodetic datum.

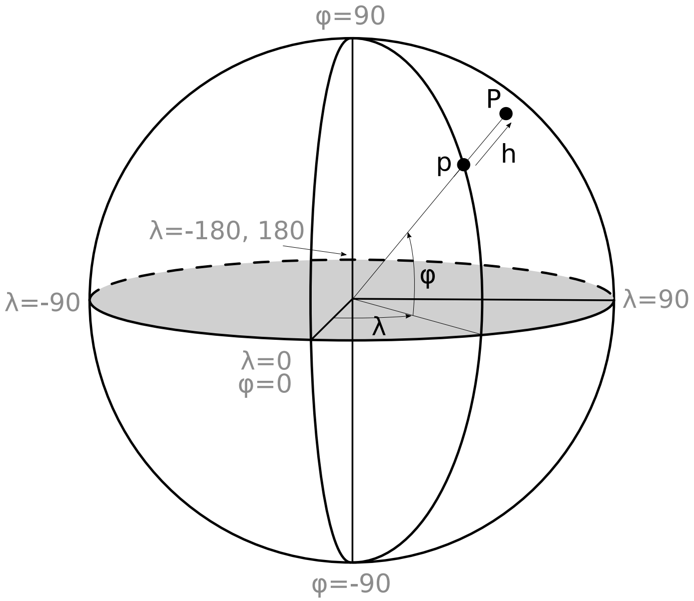
Ellipsoids can be defined locally or globally depending on the application. Local ellipsoids try to fit the geoid best for the area of interest such as a country or a continent. For Europe and especially the land surveying in Germany, the Bessel ellipsoid was an important reference surface, before the European Terrestrial Reference System 1989 (ETRS89) with its reference ellipsoid (GRS80) was introduced (Bill, 2016). Global ellipsoids on the other hand try to approximate the geoid globally. Over the course of geodetic history, several global reference ellipsoids had been defined. In 1909 the US geodesist Hayford introduced an ellipsoid, which was acknowledged as the international ellipsoid in 1924 by the International Union for Geodesy and

Geophysics (IUGG). Since the approximation of the ellipsoid was regarded as insufficient in 1967, the Hayford ellipsoid was replaced by the Geodetic Reference System 1967 (GRS 1967). Subsequently, the World Geodetic System 1984 (WGS84) was acknowledged as the international ellipsoid in 1984, which is still generally valid (Huisman & de By, 2009).

The *geodetic datum* defines a set of constants that specify the position and orientation of the ellipsoid to the Earth. Besides the geodetic ellipsoid, the datum consists of a so-called fundamental point with a latitude (φ), longitude (λ), a height (h) and a geodetic azimuth. The fundamental point is located best in the center of the desired area of the Earth's surface. For example, the Potsdam Datum is based on the Bessel ellipsoid and the fundamental point Rauenberg in Berlin. It has a sufficient approximation for the surface of Germany.

GLOBAL COORDINATE SYSTEMS

Various kinds of CRSs are used to position data in space. A general distinction is between global and planar coordinate systems. Global coordinate systems can be used to locate information on the Earth's surface in 3D space or in 2D space on the reference surface. These can be geographic or geocentric coordinate systems.



Source: Author's illustration

Figure 3.6: *Geographic coordinates*

Geographic coordinates are the most widely used approach for a global coordinate system. The Earth is partitioned into lines of geographic latitude (φ) and longitude (λ). While lines of equal latitude (called parallels) form circles on the surface of the ellipsoid, lines of equal longitudes (or meridians) are ellipses. Figure 3.6 shows the difference between latitude and longitude. Lines of equal latitude are parallel to the equatorial plane (gray). On the equator the value for latitude is zero degree ($\varphi = 0°$) and increases northwards to $90°$ at the North Pole and southwards to $-90°$ at the South Pole depending on the angle. For longitude, the zero meridian ($\lambda = 0°$) is defined as the one passing Greenwich. Going eastwards halfway around the ellipsoid increases the longitude to $180°$, while in the opposite direction (westwards) it decreases to $-180°$.

The point p can be represented by the 2D geographic coordinates (φ, λ) given a reference surface (ellipsoid or sphere). In this case, both coordinates are described by angular units. 3D geographic coordinates (φ, λ, h) introduce the ellipsoidal height h to the system. The point P in Figure 3.6 has the same 2D geographic coordinates like point p but has also a height h, which is the vertical distance from p to P above the ellipsoid. The unit of the height is not an angle but a distance unit.

Geocentric coordinates, on the other hand, define points on the surface of the Earth in terms of (X, Y, Z), where the origin is the mass-center of the Earth and the axes X and Y are in the equatorial plane. The X-axis passes the meridian in Greenwich, while the Y-axis is perpendicular to it. The Z-axis corresponds with the Earth's axis of rotation. The three axes are orthogonal to one another and form a right-handed system.

PLANAR COORDINATE SYSTEMS

To locate data on the flat surface of a 2D map, planar coordinate systems are used such as 2D Cartesian coordinates or 2D polar coordinates. However, this requires transforming the three-dimensional Earth into a two-dimensional map by methods of projecting. Map projections are mathematical representations of a geodetic ellipsoid (or sphere) as a plane (Seeger, 1999). They can be cylindrical, conical or azimuthal (planar) and always include a distortion in area, distance and/or angular. For instance, the Universal Transverse Mercator (UTM) is a widely used and famous cylindrical map projection, whose areas and distances are distorted but it is conformal. The fundamentals in map projection are omitted here, but can be found in many textbooks (e.g. Hill, 2006).

In *2D Cartesian coordinate systems* two values $width$ (or east E) and $length$ (or north N) specify a point in 2D. So, given a geographical coordinate in latitude and longitude (φ, λ) two projection functions $E = f(\varphi, \lambda)$ and $N = g(\varphi, \lambda)$ have to be applied to obtain the Cartesian coordinates (E, N). 2D Cartesian coordinate systems have two principal axes (X- and Y-axis), which intersect perpendicular in a point called *origin*. Usually, the X-axis, sometimes called $Easting$, forms the horizontal, while the Y-axis (sometimes $Northing$) denotes the vertical axis (Huisman & de By, 2009). The *map grid* terms the plane which is partitioned by equally spaced coordinate lines in X and Y. Given this, a point p can be defined unambiguously by its coordinates.

2D Polar coordinate systems use polar coordinates to define a point in a plane. Instead of two axes values, a distance d in length units from the origin and an angle α denotes a point uniquely. The angle α (or $bearing$) is given in angular units and is related in clockwise direction from a fixed direction, called the initial bearing. The reference direction can be chosen freely, but it often corresponds to true or grid north.

SPATIAL REFERENCE SYSTEM IDENTIFIER (SRID) & WELL-KNOWN TEXT (WKT)

All kind of CRS can be referred to by using a so-called Spatial Reference System Identifier (SRID) integer, which is a unique value to identify projected, unprojected and local CRS definitions. GIS and geospatial data vendors use their own SRID classification or refer to implemented systems of third parties. An important and widely used classification is issued by the European Petroleum Survey Group Geodesy (EPSG), whose SRIDs are labeled EPSG-Codes. These codes consist of 4 to 5 digits pointing to the definition of the CRS. For example, the EPSG-code for the geographical CRS WGS84 is 4326. The OGC manages the definitions of SRID, while the ISO standard 19111:2007 (ISO/TC 211, 2007a) adopts it.

The CRS definition itself is typically given as a WKT string, which is standardized in OGC's Simple Feature Access (Herring, 2011) and published conjointly by the ISO in ISO/TC 211 (2015). The WKT representation of a CRS includes the geodetic datum, the geoid, the coordinate system as well as the map projection of spatial objects. For the WGS84 coordinate system the WKT representation is given in Listing 3.1.

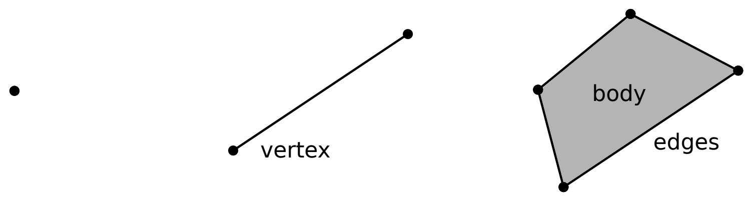
```
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.01745329251994328,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
```

Listing 3.1: *WKT representation of WGS84 (EPSG:4326)*

3.4.2 FUNDAMENTALS OF GEOSPATIAL DATA

Geospatial data represent and model real-world geographic phenomena in a digital fashion. Huisman & de By (2009) define a geographic phenomenon as a manifestation of an entity or process, which (1) can be *named* or *described*, (2) can be georeferenced and (3) can be assigned a *time (interval)* at which it is/was present. This definition complies pretty close with our definitions for geospatial events and states. GIS often represents these phenomena in a two- or three-dimensional *Euclidean space*, meaning locations are directly georeferenced and represented by coordinates (x, y) or (x, y, z) respectively. Accordingly, distances and directions can be specified by geometric formulas. In 2D the Euclidean space is known as the *Euclidean plane*, which is most used in GISs.

An essential distinction of geographic phenomena refers to the extent of the phenomenon in the real-world. For instance, if the study observes the Direct Normal Irradiance (DNI), this value can be measured anywhere on the surface of the Earth. It is named a *geographic field*, since a value can be determined for every location in the study area. A geographic field can be of *discrete* or *continuous* nature. The DNI is a continuous field measured in watt per square meter (W/m^2). The changes in field values are gradual and, thus, the field can be differentiable. Whereas in a discrete field, a discrete value is assigned to every location, such as a soil type or a land-use classification. Geographic fields are usually represented by *raster*, which is a set of regularly spaced and continuous cells with associated values. A value (continuous or discrete) is assigned to each *raster cell*. It holds that the value is valid for all locations within the cell (Huisman & de By, 2009). The raster's *resolution* determines the size of the area for a single raster cell.



(a) *Point feature* **(b)** *Line feature* **(c)** *Polygon feature*

Source: Author's illustration

Figure 3.7: *Different type of features in the Euclidean plane*

If the phenomenon solely occurs in certain locations, we call them *geographic objects* and are well-distinguished, discrete and bounded entities with undetermined space between them (Huisman & de By, 2009). Weather stations, e.g., are geographic objects with well-defined locations, which populate a study area and are erected by humans. Besides the location, shape, size and orientation can also be stored for these objects. The *shape* of a geographic object designates the *dimensionality* of the representation. One can distinguish between point, linear and polygon features, depending on the perception of the object in a certain application. In GIS, this is called the *vector* model. Figure 3.7 shows the three different basic data types for geospatial features in the Euclidean plane.

In the Euclidean plane, a *point* (a) is defined by a coordinate pair (x, y) in 2D, respectively (x, y, z) in 3D. The coordinates depend on the chosen CRS. Objects that are represented by points are zero-dimensional features with no shape and size. In fact, it depends on the object's spatial extent in conjunction with the scale in the specific application. For instance, petrol stations on a navigation map are usually depictured as points, while in cadastre applications they are probably modeled as polygons.

A *line* (or polyline, linestring, arc, edge) (b) represents one-dimensional objects, which are typically roads, rivers etc. GIS represents a line as a list of *vertices* (or nodes). Figure 3.7 shows a simple line with two *end vertices* $line = ((x_1, y_1), (x_2, y_2))$. *Internal vertices* can be added to the list to obtain a more complex line. A vertex itself is defined like a point. If a line consists of more than two vertices, the straight parts of that line between two consecutive vertices are called the *line segments*.

Polygons (c) are modeled by a linestring, whose last vertex is the same point as the first vertex in the list, a so-called *linear ring*: $polygon = ((x_1, y_1), ..., (x_n, y_n))$, where $x_1 = x_n, y_1 = y_n$. A line segment in a polygon is defined as an *edge*, while the inner area is named the *body*. Polygons may also have holes, which are also defined by linear rings. The actual implementation depends on the chosen encoding, e.g. in the WKT standard (see Section 3.4.3), the exterior linear ring is defined in counter clockwise direction, whilst interior linear rings for holes are defined in clockwise direction (Herring, 2011). Like mentioned, it depends on the application if objects are represented with polygons, points or lines.

3.4.3 Encodings of Geospatial Objects

In GIS and other systems, different formats are used to encode geometries. Some confine themselves to the three basic vector types like defined in the previous section, others also introduce more complex geometry features. In the following the common encodings for geographical objects are introduced.

WKT AND EWKT

The textual markup language WKT can be utilized to encode vector geometry objects, CRSs of spatial objects and transformations between CRSs. Furthermore, the binary equivalent Well-Known Binary (WKB) is used in databases. The encodings of geometric objects are defined in Herring (2011) and cover the three basic spatial data types (*Point*, *LineString*, *Polygon*) as well as some extended version such as *Curve* or *Surface*.

The geometries are encoded with coordinates in 2D or 3D, and, in addition, may have an *m* value. 2D Examples for geometries are given in the Listing 3.2.

```
1 POINT (6.06799 50.77906)
2 LINESTRING (293126.66164 5629309.72698,293125.14552 5629308.42261)
3 POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))
4 MULTIPOINT ((6.06799 50.77906), (6.06567 50.7785))
```

Listing 3.2: *WKT for basic geometries*

Line 1 to 3 shows the three basic geometry types. Essentially, the string starts with a keyword determining the type and completes with a list of points. In line 4 a *MultiPoint* is depicted, which represents a set of points. Taking a closer look at the coordinates of the geometries, they obviously do not share a common CRS. While the point is specified with longitude and latitude in WGS84, the linestring's coordinates are stated in UTM zone 32N. In WKT, a CRS is not specified in the encoding of the geometries itself. Applications must implicitly know which CRS is used for the coordinates.

Overcoming this issue, the PostGIS Development Group (2018) introduces the Extended Well-Known Text (EWKT) notation, which prefixes the geometry string by a SRID indicating the used CRS. For the sample geometries this constitutes as follows (see Listing 3.3).

```
SRID=4326;POINT (6.06799 50.77906)
SRID=25832;LINESTRING (293126.66164 5629309.72698,293125.14552 5629308.42261)
SRID=4326;MULTIPOINT ((6.06799 50.77906), (6.06567 50.7785))
```

Listing 3.3: *EWKT for basic geometries*

Like shown, the EPSG code is added at the beginning of the string to specify the CRS.

GEOGRAPHY MARKUP LANGUAGE (GML)

The OGC issued in 2000 the first version (V1.0) of the Geography Markup Language (GML) standard for modeling and encoding geographic information in an XML fashion. The most recent version 3.2.1 (Portele, 2007) is also adopted as an ISO standard (ISO/TC 211, 2007b).

GML is based on XML technologies, such as XML encoding, XML namespaces or XML schema. It supports encoding of spatial and non-spatial properties of objects. It consists of a set of base schemas, which can be used for the basic three geometries but also additional geometric primitives (0D, 1D, 2D, 3D), geometric composites and aggregates. Furthermore, CRS and topology and other information (e.g. time, coverage) can be modeled.

```xml
<gml:Point srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
  <gml:pos>6.065695599408778 50.778495942043229</gml:pos>
</gml:Point>
```

Listing 3.4: *GML example for a point*

```xml
<gml:LineString srsName="urn:ogc:def:crs:EPSG::25832">
  <gml:posList>293126.661644776759204 5629309.726983548142016
       293125.145516532473266 5629308.422605253756046</gml:posList>
</gml:LineString>
```

Listing 3.5: *GML example for a linestring*

The Listings 3.4 and 3.5 illustrates the GML encoding in version 3.2.1 of the point and the linestring from WKT examples. The attribute *srsName* indicates the CRS, which can be declared in a specific URI format (URL or URN).

GeoJSON

GeoJSON is an open format based on JSON to represent geographic data from the Simple Feature Access specification (Herring, 2011). The RFC standard (Butler et al., 2016) was published in 2016 and defines various JSON objects and their combination to encode geographic features, their properties and spatial extents. However, the CRS is restricted to WGS84 and units of decimal degree.

A *Geometry* object consists of a type and corresponding coordinates. In case of a point, a tuple of longitude and latitude (position) is expected; for a linestring, the coordinates member is a list of two or more positions and, finally for a polygon, coordinates must be an array of linear ring coordinate arrays. Listing 3.6 shows the GeoJSON encoding of a point.

```json
{
  "type": "Point",
  "coordinates": [6.065695599408778, 50.778495942043229]
}
```

Listing 3.6: *Example for a GeoJSON Geometry point geometry*

Besides point, linestring and polygon, other derived geometries can be modeled. This includes e.g. set of points (MultiPoint) or of polygons (MultiPolygon) as well as GeometryCollections. The use of alternative CRSs is not proposed in the RFC. However, in the originally proposed specification (Butler et al., 2008) a CRS can be indicated similarly to the attribute in a GML.

The main object in GeoJSON is a *Feature*, which consists of a member "type", a member "geometry", which is a *Geometry* object, and "properties" for other non-spatial data. A feature encoding for a linestring object is shown in Listing 3.7. Besides the geometry, some properties are added. Additionally, an alternative CRS was specified according to the initially proposed specification in Butler et al. (2008).

```
{
  "type":"Feature",
  "geometry":{
    "type":"LineString",
    "coordinates":[[293126.66164, 5629309.72699] [293125.14552, 5629308.42261]]
  },
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSG::25832"}
    }
  }
  "properties":{
    "name":"Drainage pipe",
    "installed":1981,
  }
}
```

Listing 3.7: *Example for a GeoJSON feature*

The third type of GeoJSON objects is the *FeatureCollection*. It has two members: "type" and "features", whose value is a JSON array composed of GeoJSON features.

GEOBUF

The fourth encoding, we want to portray here, is based on Google Developers' protobuf (Google Developers, 2018) and called Geobuf. It is developed by Mapbox (2018) and provides the capabilities for a compact binary encoding of geographic data. Geobuf can be applied on GeoJSON data to obtain a nearly lossless compression into protobuf. According to Mapbox, Geobuf is 6-8 times smaller than the GeoJSON representation but, simultaneously, faster in encoding and decoding than native JSON parsing/stringifying. Since it is a binary encoding format, Geobuf are non-human readable and can only be created by serializing from other formats or digital objects.

3.4.4 TOPOLOGICAL RELATIONS OF GEOMETRIES

The topological relationship between spatial objects plays an important role in a Geospatial IoT. Chen et al. (2003) e.g. implemented a system for LBSs with the relationships *within* and *distance*. Jin et al. (2013) use the spatial operators *happen-in*, *overlap*, *same-place* and *distance*. In our Geospatial IoT model, capabilities of expressing the relationships between geospatial events or states to each other are essential.

Formalized models for defining the topological relationship between two geometries exist since the early 1990s with e.g. the Four-Intersection Model (4IM) (Egenhofer & Franzosa, 1991) or the Dimensionally Extended Nine-Intersection Model (DE-9IM) model (Egenhofer et al., 1993; Clementini et al., 1993). Each of these models defines a set to describe spatial relations of two geometries in two-dimensional $\mathbb{R}^2$. They are presented briefly in the following, starting with the 4IM.

3.4.4.1 Four-Intersection Model (4IM)

The 4IM describes binary topological relations between two geometries A and B. The relations are defined in terms of the intersections between the boundary (δA) and interior (A°) of A and the boundary (δB) and interior (B°) of B. The model can then be represented by a 2x2-matrix which is called 4-intersection:

$$I_4(A,B) = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \delta B \\ \delta A \cap B^\circ & \delta A \cap \delta B \end{pmatrix} \tag{3.3}$$

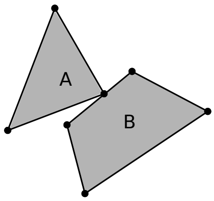
One can distinguish between $2^4 = 16$ binary topological relations if the values of the intersections are considered as empty ($\varnothing$) or non-empty ($\neg\varnothing$).

Let a polygon be defined like before where the boundary is equivalent to the aggregation of all edges and the interior is the body. Given two polygons A, B in $\mathbb{R}^2$, eight of the 16 relations can be realized: disjoint, inside, contains, touches, equals, covers, covered by and overlaps (Egenhofer & Franzosa, 1991). Figure 3.8 (a) shows the manifestation of the 4IM matrix for the A *touches* B depictured in the center.

For two lines in $\mathbb{R}^1$, a set of eight topological relations can be found, which corresponds to Allen's interval relations (see 3.3.4.2). The boundary of a line is represented by the two endpoints, while the interior is the line between the endpoints. However, the 4IM has some major drawbacks: for instance, relations between a line and a polygon cannot be modeled with high granularity. Also, the 4IM does not provide a useful definition of an *equals* relation between two lines in $\mathbb{R}^1$. The following 9IM can compensate these shortcomings (Egenhofer et al., 1993).

$$\begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}$$

(a) *4IM matrix*



$$\begin{bmatrix}0 & 0 & 1\\0 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

(b) *9IM matrix*

Source: Author's illustration

Figure 3.8: *touches-relation and its representation in 4IM and 9IM*

3.4.4.2 Nine-Intersection Model (9IM)

The Nine-Intersection Model (9IM) extends the 4IM by the relation of interior and boundary of an object to the other object's exterior. This extension eliminates the modeling gaps of the 4IM. Let A and B are two objects in $\mathbb{R}^2$ with A's interior (A°), boundary (δA) and exterior (A^-), as well as B's interior (B°), boundary (δB) and exterior (B^-). The corresponding 3x3-matrix represents the nine intersections between the objects' parts, which describe a topological relation:

$$I_9(A,B) = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \delta B & A^\circ \cap B^- \\ \delta A \cap B^\circ & \delta A \cap \delta B & \delta A \cap B^- \\ A^- \cap B^\circ & A^- \cap \delta B & A^- \cap B^- \end{pmatrix} \tag{3.4}$$

Each intersection is characterized by a value empty ($\varnothing$) or non-empty ($\neg\varnothing$). Therefore, the matrix can take $2^9 = 512$ configuration. However in $\mathbb{R}^2$, only a small subset can be realized between two objects.

Between two polygons in $\mathbb{R}^2$ the same set as for the 4IM can be found (Egenhofer & Franzosa, 1991). In Figure 3.8 (b), the matrix for the A *touches* B case is illustrated for the 9IM. However, the 9IM is able to characterize 33 distinct line-line relations in $\mathbb{R}^2$ (Egenhofer & Herring, 1991). Finally, relations between a line and a polygon can be distinguished with finer granularity.

3.4.4.3 Dimensionally Extended Nine-Intersection Model (DE-9IM)

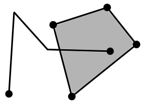
Based on the 9IM the Dimensionally Extended Nine-Intersection Model (DE-9IM) was developed by Clementini & Di Felice (1995). In the so-called dimension extended method, the standard approach is extended by also adding point and line features, which results in 6 major groups of binary relationships: polygon/polygon, line/polygon, point/polygon, line/line, point/line and point/point. The model is further extended by the dimension of the intersection. In the 4IM and 9IM an intersection is either empty ($\varnothing$) or non-empty ($\neg\varnothing$), in the dimensionally extended model an intersection can be either $\varnothing$ (empty), $0D$ (point), $1D$ (line) or $2D$ (area). For 4IM, this results in $4^4 = 256$ different cases, but can be narrowed down to a total of 52 realistic relationship cases (Clementini et al., 1993). Applying the method to 9IM, it gives a total of 81 realistic relationships (Clementini & Di Felice, 1995):

Let A and B are two geometry objects in $\mathbb{R}^2$ with A's interior (A°), boundary (δA) and exterior (A^-) as well as B's interior (B°), boundary (δB) and exterior (B^-). The 3x3 matrix for the DE-9IM is defined as follows:

$$DE - 9IM(A,B) = \begin{pmatrix} dim(A^\circ \cap B^\circ) & dim(A^\circ \cap \delta B) & dim(A^\circ \cap B^-) \\ dim(\delta A \cap B^\circ) & dim(\delta A \cap \delta B) & dim(\delta A \cap B^-) \\ dim(A^- \cap B^\circ) & dim(A^- \cap \delta B) & dim(A^- \cap B^-) \end{pmatrix} \tag{3.5}$$

This adds a large number of different relationships with their own names to the model, which might be unusable and confusing for the user (Clementini et al., 1993). Therefore, the DE-9IM introduces further topological predicates, which are Boolean function for testing the spatial relations between two geometry objects. The model provides eight such spatial relationships between points, lines and polygons (see Table 3.7).

For example, Figure 3.9 (a) shows the relation *crosses* for a line (A) and a polygon (B) depicted on the left-hand side.



$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & \varnothing & 0 \\ 2 & 1 & 2 \end{bmatrix}$$

(a) *Line Crosses Polygon*

$$\begin{bmatrix} \neg\varnothing & * & \neg\varnothing \\ * & * & * \\ * & * & * \end{bmatrix}$$

(b) *Pattern matrix*

Source: Author's illustration

Figure 3.9: *Line crosses polygon relation in the DE-9IM*

The matrices are not distinct for the relations. For the line *crosses* polygon relation in the figure, consider that the line does not end inside the polygon, but crosses the boundary again and ends in the exterior of the polygon. It would still be named a *crosses* relation, although the matrix has a deviating entry ($dim(\delta A \cap B^\circ) = \varnothing$). Thus, pattern matrices can be defined for each primitive with the patterns $\{*, \varnothing, \neg\varnothing\}$, where $*$ denotes a wildcard, $\varnothing$ implies that the intersection is necessarily empty and $\neg\varnothing$ accordingly. The pattern matrix for the crosses relation is given in Figure 3.9 (b). Note that for *crosses*, the given pattern matrix is only valid iff $dim(A) < dim(B)$

Table 3.7: *Topological predicates derived from the DE-9IM*

Predicate Name	Meaning
A *Equals* B	A and B are topologically equal.
A is *Disjoint* to B	A and B have no points in common.
A *Intersects* B	A and B have at least one point in common. (inverse of disjoint)
A *Touches* B	A and B have at least one boundary point but no interior point in common.
A *Crosses* B	A and B have some but not all interior points in common. Further, the dimension of the intersection is less than at least one of A and B.
A is *Within* B	A lies in the interior of B. They do not share boundary points.
A *Contains* B	B lies in the interior of A. They do not share boundary points. (same as B within A)
A *Overlaps* B	A and B have some but not all points in common. The intersection has the same dimension as the geometries themselves.

Source: based on Strobl (2008)

holds. For the other cases $dim(A) = dim(B)$ and $dim(A) > dim(B)$, the pattern matrix must be modified.

The DE-9IM and the derived topological predicates given in Table 3.7 are accepted by the OGC's Simple Feature Access (Herring, 2011). Besides, the eight topological predicates, other can also be derived. Davis (2007) argues that the relations *Covers* and its inverse *CoveredBy* should be added to the predicate list (see Table 3.8).

Table 3.8: *Topological predicates derived from the DE-9IM*

Topological Predicate	Meaning
A *Covers* B	Any point of B lies in the exterior of A.
A is *CoveredBy* B	Any point of A lies in the exterior of B.

Source: based on Strobl (2008)

Covers is related to the *Contains* predicate but also includes the boundary of the first geometry. For instance, a line, which is contained completely in the boundary of a polygon, is not considered to be true for the *Contains* but for the *Covers* relation (Davis, 2007). Thus, *Covers* and *CoveredBy* are implemented additionally in many geospatial software such as Oracle Spatial (Oracle, 2018) or JTS topology suite (LocationTech, 2016).

3.5 ARCHITECTURE OF THE GEOSPATIAL IOT

Based on the characteristics of the Geospatial IoT, the geospatial process ontology (geospatial events and states) derived and defined in the previous sections as well as the fundamentals of IoT systems (see Section 2), we can now conceptualize an architecture for a Geospatial IoT.

The basic building blocks of Geospatial IoT architectures are similar to the components of generalized IoT systems (compare Figure 2.2). IoT devices (1), which have a specific stationary or moving location, measure or drive a geospatial process through sensors or actuators. Whether it is a physical or a virtual thing like in a simulation, a geospatial process is sensed, modeled or driven on events. Thus, things issue and receive information about geospatial processes according to our model of real-world geospatial processes (see Section 3.2.3). As we saw, localizing physical things can be achieved by common localization techniques such as GNSS.

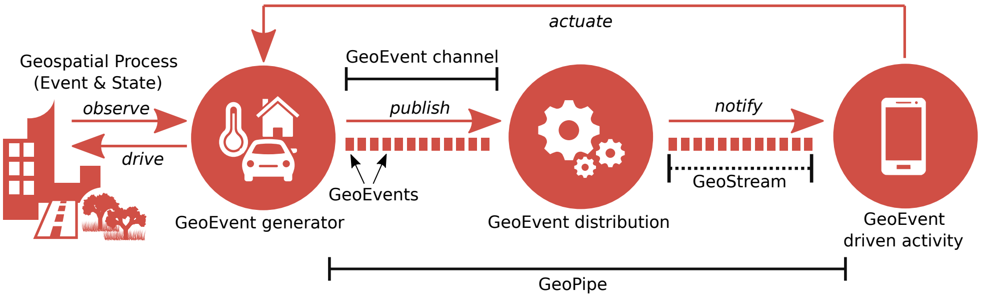
M2M communication technologies (2) find usage to interconnect things with the Internet in a way, that information about geospatial processes can be interchanged bidirectional with the IoT information and services layer. Different message patterns can be used, but for real-time messaging, mechanisms that push the information about geospatial processes to the interested consumers are in favor.

The IoT information and services layer (3) stores the measured information in databases and offers access to the data but also to the underlying layer through web services. In a Geospatial IoT this can be e.g. achieved with existing technologies from the Sensor Web idea (see Section 2.5). The layer is also responsible for the business logic, which can be built traditionally with the SOA pattern. However, as we saw earlier, applying the EDA pattern here is a much more efficient approach. Rieke et al. (2018) calls for an event-driven approach to integrate the Geospatial IoT into traditional SDIs.

The IoT event analysis and visualization layer (4) needs to be able to receive, send and process events about geospatial processes as well as analyze streams of them. Like described previously, dashboards are an important access point to IoT data and devices since they follow principles to display the most important information at a single screen enabling humans to monitor information at a glance (see Section 2.6.2). In web-based GISs, geospatial data is usually displayed in a web mapping application. Combining an IoT dashboard with a web map application supports both, the enhancement of visualizing real-time events about geospatial processes as well as the provision of capabilities to merge with traditional geo data and services. This also holds for the integration of traditional GIS and IoT information and services. Finally, ESP capabilities could be used to find event patterns and correlations through corresponding algorithms detecting notable events in an EDA.

3.5.1 GeoEvent-driven Architecture for a Geospatial IoT

Based on these assumptions and considerations, we develop an architectural design for an event-driven Geospatial IoT in the following. The foundations of the approach are first EDAs from an architectural pattern view (see Section 2.5.3) and, second, the OODA-loop from the control model perspective for smart cities (see Figure 3.2).



Source: Author's illustration

Figure 3.10: *An EDA pattern for the Geospatial IoT*

Figure 3.10 illustrates the assembly of these two perspectives to an architecture for the Geospatial IoT based on eventing. Like in the feedback loop, the control flow is determined by observations of geospatial events and states in the physical world. These observations are performed by IoT devices equipped with sensors and/or actuators. The observed geospatial real-world process (event or state), is fed into the EDA by an event generator, which represents e.g. the sensing device. For convenient purposes, we generalize geospatial states and events into a data type named *GeoEvent* in the following. Although there is a difference in their real-world perceptions, it is useful for multiple reasons.

First, discrete temporal models do not allow for defining instants in time as we saw earlier. Depending on the chronon, a timestamp always defines a period of time from the stated instant to the next possible instant with respect to the used model (see Section 3.3). Secondly and more significantly, in EDA the term "event" describes any information in a message that is published by a producer. So technically, we need to distinguish between events in modeling real-world geospatial processes or phenomena and GeoEvents in a geospatial EDA. Therefore the following definition holds for a GeoEvent in a Geospatial IoT architecture.

Definition 3.5 (GeoEvent)
A GeoEvent is a digital representation of a geospatial process, which can be instantiated as a geospatial event or a geospatial state of a continuant. Thus, a GeoEvent is a 4-tuple:

$$(n, g, t, m)$$

, where n is the name or theme of the underlying geospatial event or state, g describes its spatial component (e.g. a geometry), t represents the temporal component (timestamp or time interval) and m is the message payload itself.

The GeoEvent definition derives from the geospatial event definition (see Definition 3.3) to model real-world geospatial processes, but includes a forth element m, which represents the message payload itself or how Michelson (2006) calls it, the event body. Representing geospatial events and states digitally, the t element may be a timestamp or a time interval. The g element can be any type of spatial reference such as coordinates in direct referencing systems or addresses in indirect systems. Finally, the n parameter describes the theme of the event such as its type or its payload.

The GeoEvent is the basic messaging type in our design. Further, the other components of the EDA pattern (see Section 2.5.3) can be determined easily from Figure 3.10. A GeoEvent generator publishes GeoEvents to a GeoEvent channel. A generator embodies e.g. a sensor node, which observes a geospatial event or a geospatial state, but may be also another system component in the architecture, which issues GeoEvents. The GeoEvent channel describes the medium that is used to transfer the information wrapped in a GeoEvent to filtering and distribution regulators. Obviously, in the Geospatial IoT, the Internet protocol suite is used as the transferring medium. GeoEvent processor performs processing functions such as filtering on the stream of GeoEvents, while downstream GeoEvent-driven activities are invoked by analyzed GeoEvents. These GeoEvent receivers may issue new GeoEvents, for instance to actuate and drive geospatial processes like shown in the figure. Furthermore, we introduce the terms "GeoPipe" and "GeoStream" in the figure, which are defined in the following.

3.5.2 GEOPIPE CONCEPT

The previous considerations point out that a mechanism is needed to interconnect the different building blocks of a Geospatial IoT from physical and virtual devices to processing units sharing GeoEvents in real-time. Hence, our Geospatial IoT architectural design is based on an EDA pattern to determine the flow of events and invoke event handling efficiently according to the feedback loop introduced in

Section 3.1. The fundamental components of EDA (see Section 2.5.3) specify the concept and the different components of the design approach like we described before. At this, the GeoEvent channel is the medium which is used to transfer the events from one place to another. In the IoT domain, it is obvious that this refers to the Internet connectivity, pointedly the use of a TCP/IP stack. However, the actual Message Exchange Pattern (MEP) is not defined by that, since it depends on the used application protocol and its features. So before implementing a prototype for a Geospatial IoT, the exchange pattern for GeoEvents must be defined. Thus, we introduce the concept of GeoPipes and its requirements, which can be considered as a pattern for a Geospatial IoT (Herle & Blankenbach, 2016).

Definition 3.6 (GeoPipe)
A GeoPipe is a specific end-to-end connection between a GeoEvent producer and its distributed consumer(s). It utilizes GeoEvent channels to transfer GeoEvents in a push-based manner. The name of the GeoPipe corresponds to the name of the GeoEvent, which is transferred.

In its simplest form its cardinality is one-to-one, but a GeoEvent distributor might split up a GeoPipe to achieve one-to-many relationships.

Producers can stream their geospatial data to consumers in a push-based way so that the latter one can process the data instantly. Producers and consumers can be of different types and hardware ranging from sensor nodes in WGSNs over high level web services to visualization applications in a web browser and processing units. GeoEvents are pushed through the GeoPipe to consumers in near real-time. A GeoPipe is *unbranched* if producer and consumer of the GeoEvent have a one-to-one relationship. But, a GeoEvent distributor can split up GeoPipes based on rule sets to deliver GeoEvents to multiple consumers, which becomes a *branched* GeoPipe. However, producers and consumers of GeoEvents are decoupled software components and services, so that both have no knowledge about the endpoints of the connection. That includes that neither producers nor consumers knows if the GeoPipe is unbranched or branched. Further, producers and consumers can be connected to multiple GeoPipes at the same time.

3.5.2.1 GeoPipe Requirements

We already addressed some requirements for GeoPipes, some are given by the nature of the EDA pattern. This section provides an overview of specific requirements for implementing GeoPipes, which we extract from the architectural pattern, Geospatial IoT use cases and the modeling of real-world geospatial processes. The compilation of requirements helps to identify useful and mandatory patterns, mechanisms and features to facilitate the selection of a suitable application protocol.

The requirements of GeoPipes are multifaceted since it is designated to connect different kinds of systems and has multiple objectives and issues to solve. The following chosen requirements are crucial for an implementation and are determined by different domains.

MESSAGING PARADIGM

First, the messaging between instances should follow a push-based manner, meaning, if a GeoEvent is published to the GeoPipe it should be pushed to the consumer automatically and instantly. The publisher does not address the consumers of the GeoPipe directly and, thus, does not need to be aware of them. Additionally, GeoPipes should be consumable by multiple consumers. All instances, which are interested in a GeoPipe, receive the corresponding GeoEvents. The publish/subscribe pattern is most suitable for these purposes.

SCALABILITY

In the Geospatial IoT, hundreds of different producers and consumers of GeoEvents might participate in a system. Every instance should be able to create, produce and consume GeoEvents from GeoPipes. Thus, a GeoPipe implementation should be able to deal with an increasing number of instances. The implementation must be scalable.

EFFICIENCY

GeoPipes should facilitate the exchange of geospatial data (GeoEvents) in real-time between instances, possibly between multiple instances at the same time. Since the GeoPipes concept should be used to implement real-time applications, efficiency is a crucial requirement.

INTEROPERABILITY

Instances, which consume and produce GeoEvents using GeoPipes, might be of any type. In Geospatial IoT, the first building block consists of small resource-constrained devices, which observe or drive geospatial processes in the physical world. Although these devices have limited power, storage and restricted processing capabilities, they should have the same rights as any other participants in the event architecture. A small and restricted sensor node in a WGSN should have the same prospects to participate in terms of GeoPipe communication as the industrial machinery in a factory. This especially enforces the next requirement.

LIGHTWEIGHT

An application protocol to implement the GeoPipes concept must be lightweight, meaning the overhead in terms of message size and connection establishment should be as small as possible. Devices with limited resources are not able to handle large protocol headers or data. So, a protocol with a minimal footprint is needed.

SECURITY

Authorization and authentication are also big issues in Geospatial IoT environments. In the GeoPipes concept, these factors concern in particular the consumers and producers of GeoEvents. In critical applications, the questions (1) *who has permission to initiate a GeoPipe and publish GeoEvents?* and (2) *who can consume GeoEvents from a specific GeoPipe?* must be addressed and resolved. Appropriate mechanisms should be available or applicable in the protocol.

RELIABILITY

For similar reasons, message loss should be avoided or avoidable. In critical IoT applications such as in early warning systems, alert messages are only conducive, if they are transmitted reliably. Supporting a reliability mechanism is, therefore, an enormously important feature.

ADAPTABLE PROTOCOL AND OPEN

The final requirement refers to the protocol and software itself, since we do not want to create a totally new protocol or software. For establishing a GeoEvent message type, we need to adapt the protocol and modify the source code of existing software. Thus, the specification of the protocol and the corresponding software should be generally accessible and open source.

A protocol, which meets all these requirements, should support the idea behind the GeoPipes concept. In Section 2.4.2, we already discussed various protocols suggested in the literature and their suitability for IoT applications. Our prototype implementation uses an extension of the lightweight IoT protocol MQTT, which basically meets all the mentioned requirements with some trade-offs. We introduce new message types in the original protocol which are enriched with geospatial information to establish GeoEvents. Thus, we call this extension GeoMQTT, which is described in detail in Chapter 4.

3.5.2.2 Subscribing to GeoPipes

The GeoPipe concept itself describes the connection between a GeoEvent producer and potentially multiple consumers. GeoPipes are initiated by producers by publishing GeoEvents with a certain name, a spatial reference, a timestamp (or time interval) and a message payload. However, consumers must register their interests in specific GeoPipes to receive the corresponding GeoEvents. This mechanism can be referred as "subscriping to GeoEvents", which defines the consumer endpoint of the GeoPipes.

Definition 3.7 (GeoSubscription)
A GeoSubscription can be used by consumers of GeoEvents to connect to a GeoPipe endpoint. It contains of three filters, which are applied to the three parameters

$$(n, g, t)$$

of GeoEvents. Thus, it can be regarded as a 3-tuple:

$$(n_{filter}, g_{filter}, t_{filter})$$

, where n_{filter} is a filter applied to the name and g_{filter} as well as t_{filter} represent filters that are applied to the spatial, respectively the temporal component.

A GeoPipe subscription can be accomplished by applying filters to the metadata of a GeoEvent. The simplest way to subscribe to a GeoPipe is to apply a string comparison between an indicated string and the corresponding name of the GeoEvents. If the strings match, the consumer subscribed to the GeoPipe and its GeoEvents. The mechanism is identical with the topic-based publish/subscribe scheme described in Section 2.3.1.3.

Besides the name, the GeoEvent message has also two other meta information, namely the spatial reference and the timestamp (or time interval). Based on this information, additional filters are also conceivable. As we saw earlier, for both components, spatial and temporal, topological relations can be defined (see Section 3.3.4 and Section 3.4.4). These relationships can be used in combination with spatial and temporal information to apply a spatial and a temporal filter to GeoEvents. The conjunction of all three filters allows consumers to specify their interests in GeoEvents and, therefore, establish GeoPipes more precisely. From these considerations we define a GeoSubscription as in Definition 3.7. How this is implemented in our prototype can be learned in detail in Section 4.3.2.

3.5.3 GEOEVENT PROCESSING & GEOSTREAMS

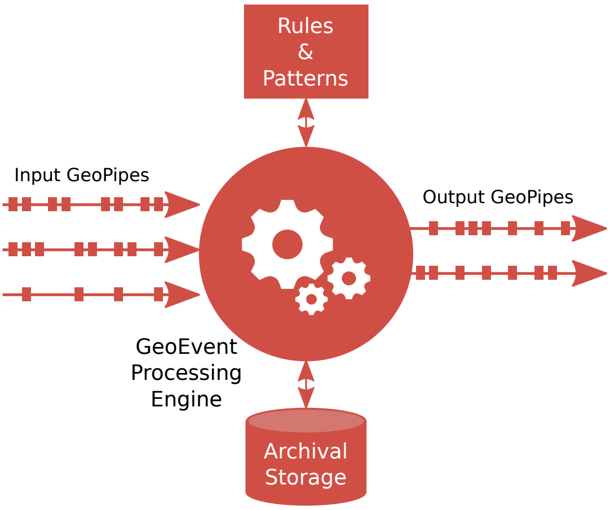
GeoEvents created by GeoEvents generators are sent over the GeoPipe in near-real time occupying specific GeoEvent channels. Like mentioned, GeoPipes can be branched by a GeoEvent distributor based on a specific set of rules. This set of rules can be defined by GeoSubscriptions, which can be registered by consumers to indicate their interests. According to the EDA pattern (see Section 2.5.3), this mechanism can be interpreted as a *simple event processing*, where a GeoEvent filtering is performed based on the three meta information (name, spatial/temporal component) and, subsequently, distributed.

GeoEvent processing can thus be defined as a component of an EDA which takes a GeoPipe as an input and evaluates event processing rules or matches patterns on the GeoEvents flowing in. Based on the analysis, actions are initiated or new GeoEvents in output GeoPipes are generated. Figure 3.11 shows the concept of GeoEvent processing, which is derived from Stonebraker et al. (2005). Events in EDA are processed by engines. They act on different rule sets or patterns and may archive events or computational results in dedicated storage.

The GeoEvent processing engine is connected to GeoPipes, through which GeoEvents are flowing in. Consecutive GeoEvents in a GeoPipe form a stream of GeoEvents. We term in the following a stream of GeoEvents a *GeoStream* and define it in Definition 3.8.

Definition 3.8 (GeoStream)
A GeoStream is a possibly unbounded stream of GeoEvents, which runs through a GeoPipe sequentially. A GeoStream has a name, which is equivalent to the name of the GeoPipe. The input elements $(n, g_1, t_1, m_1), (n, g_2, t_2, m_2), ..., (n, g_t, t_t, m_t)$ *are a sequence of GeoEvents with the name* n*, spatial components* $g_1, ..., g_t$*, timestamps (or time intervals)* $t_1, ..., t_n$ *and message bodies* $m_1, ..., m_t$*.*

An engine, which performs evaluations on or searches patterns in GeoStreams, must be aware of certain requirements and rules. GeoEvent processing engines must process the incoming data in real-time to catch up with new arriving data. Stonebraker



Source: derived from Stonebraker et al. (2005)

Figure 3.11: *GeoEvent (GeoStream) processing*

et al. (2005) lay down eight rules for processing stream data in real-time which also hold for processing GeoStreams:

1. In-stream processing: no storing of event while performing operations.
2. High-level language support.
3. Handle stream imperfections.
4. Predictable and repeatable outcomes.
5. Efficiently store, access, and modify state information, and combine it with live streaming data.
6. Data safety and availability.
7. Distribute processing across multiple processors.
8. Instantaneously processing and responding.

The main requirement claims that event processing engines are efficient, so that performing actions on the GeoStream catches up with the incoming GeoEvents. But simultaneously, it demands the system to stay responsible. Increasing load through increasing temporal resolution of GeoEvents or simply multiple connected GeoPipes must be handled sophisticatedly. Thus, technical solutions should be developed to meet these requirements.

Since GeoEvent processing engines can be of different types, the complexity of the solutions changes according to the desired processing tasks. While the filtering through subscriptions describes a simple processing engine that processes each GeoEvent occurrence in a GeoStream independently, complex engines process incoming GeoEvent in context of prior and future events. Furthermore, ESP engines act on multiple consecutive GeoEvents in a stream. For meeting the efficiency requirements for increasing loads, one solution is to utilize DSP engines which distribute processing across multiple machines. We will see later how increasing load can be met in our implementation.

CHAPTER 4

GEOSPATIAL MQTT (GEOMQTT)

A Geospatial IoT with its derived concepts has several requirements on communication and the applied protocols, we defined before. We implemented the concepts of GeoEvents and GeoPipes as a basis for a GeoEvent-driven architecture by extending the MQTT protocol. This extension, which we call GeoMQTT, introduces novel message types in the base protocol to support the concepts of GeoEvent and GeoSubscriptions.

This chapter explains the extensions we made to MQTT. First, we evaluate the different IoT application protocols (see Section 2.4.2) for the requirements of GeoPipes and explain, why we chose MQTT for the implementation. Subsequently, MQTT is introduced in detail by explaining the messaging model and its features (Section 4.2). Then, the GeoMQTT extension is described. This covers the newly introduced message types as well as the filtering types for GeoPipes (Section 4.3). The implementations for the GeoMQTT broker and clients are subjects of Section 4.4. Finally, we also extended the MQTT-SN protocol with our GeoMQTT extension to support also sensor nodes in WSNs. This is illustrated in the final Section 4.5 of this chapter.

4.1 GEOSPATIAL IOT APPLICATION PROTOCOL EVALUATION

Before implementing the GeoEvent type and the GeoPipe concept, an analysis of the given and established IoT application protocols is mandatory. Since we did not want to implement a complete new protocol for the Geospatial IoT - the requirements of the IoT hold also for the Geospatial IoT - , we decided to extend one of the existing protocols introduced before. Table 4.1 summarizes the messaging mechanisms and core features of the IoT application protocols introduced in Section 2.4.2. Like already discussed, the suitability of the protocols for sending messages in the IoT differs, since their capabilities are partially fundamentally different. This results from the different domains, the protocols were originally developed for. For instance, while HTTP was originally created to transfer data in the WWW such as websites, the initial idea behind the development of XMPP was to design a real-time communication protocol based on XML for instant messaging.

Table 4.1: *Compilation of IoT application protocols*

Protocol	Trans-port	Messaging	Secu-rity	QoS	Header Size
HTTP	TCP	Request/Response	TLS	No	Undefined[1]
CoAP	UDP	Request/Response & Notification	DTLS	Yes	4 Bytes
MQTT(-SN)	TCP & UDP	Publish/Subscribe	TLS & DTLS	Yes	2 Bytes
XMPP	TCP	Request/Response & Publish/Subscribe	TLS	No	Undefined[1]
AMQP	TCP	Publish/Subscribe	TLS	Yes	8 Bytes

[1] variable, but higher order

Source: Author's illustration

The evaluation was performed against the list of requirements for GeoPipes demanded in Section 3.5.2.1. The result of the evaluation gives insight into the suitability of the individual protocols to implement the GeoEvent type and the GeoPipe concept (see Table 4.2). Since we want to implement a GeoEvent-driven architecture for the Geospatial IoT, a push-based messaging pattern is extremely important. This already disqualifies HTTP with its Request/Response mechanism. Depending on the other requirements, it becomes evident that protocols developed and designed for IoT environments meet the requirements best. Features of CoAP and MQTT are both dedicated to resource-constrained devices in high-latency or unreliable networks. Thus, it is hardly surprising that these have their strengths in efficiency, small message size or interoperability in terms of deployed hardware. Simultaneously, they are scalable but have their weaknesses in security issues. For the latter one, the protocols do not provide their own capabilities, but rely on underlying protocols in the communication stack. The other protocols incorporate larger headers, have a larger footprint and, thus, provide no interoperability with resource-constrained devices. They also do not necessarily offer efficiency, e.g. HTTP requires high power and resource consumption (see Section 2.4.2).

So, CoAP and MQTT may be used to implement the architecture. However, we want to employ a fully push-based mechanism in our proposed architecture to initiate real-time data stream. CoAP uses mainly the request/response pattern for communication but also offers a notification mechanism. However, the publish/subscribe mechanism provided by MQTT seems to be the more suitable choice to implement the GeoPipes concept, since it decouples producer and consumer of events in time, space and synchronization. Multiple consumers of events are conceivable without the

Table 4.2: *Evaluation of Requirements Analysis*

Requirement	HTTP	CoAP	MQTT	XMPP	AMQP
Push-based	-	o	++	o	++
Open	+	+	+	+	+
Scalability	-	+	+	+	o
Interoperability	-	+	+	-	-
Efficiency	-	++	+	o	o
Lightweight	--	+	++	-	o
Security	+	-	o	+	+
Reliability (QoS)	-	o	+	-	+
Size	--	+	+	-	+

Source: Author's illustration

knowledge of the producer. Therefore, we choose MQTT as the basis for our intended geospatial extension and the implementation of the GeoEvent-driven architecture for the Geospatial IoT. Nevertheless, it should be mentioned, that MQTT provides a client id and username/password credentials for authentication of devices, but no further security mechanisms. The specification (Banks & Gupta, 2014) advises to use security features from other layers in the stack such as TLS for transport encryption or VPN on the network level.

4.2 MQTT DETAILS

Message Queuing Telemetry Transport (MQTT) is an open and extremely lightweight publish/subscribe protocol for M2M communication with special focus on connections with remote locations and limited network bandwidth. Its design principles cover the minimization of network bandwidth and device resource requirements, but, simultaneously, MQTT attempts to ensure reliability and assures delivery of messages to a specific degree, known as Quality of Service (QoS). These principles favor its use in the IoT world of connected devices as well as in mobile applications.

We already introduced the protocol stack in Section 2.4.2.3 and gave some insights into the standardization process and the version history of MQTT. In this chapter, we focus on the message exchange pattern and the capabilities MQTT provides for implementing the mentioned design principles. These are the foundation for the implemented extension GeoMQTT.

4.2.1 TOPIC-BASED PUBLISH/SUBSCRIBE MODEL

According to Section 2.3.1.3, the publish/subscribe pattern requires three actors: (1) publishers of events, (2) consumers of events and (3) an intermediary distribution layer, which receives the published event and pushes notifications to consumers. In MQTT, this reduces to two components: a client and a server. Clients can be publishers and/or consumers depending on whether the client publishes or is subscribed to messages. According to the specification in Banks & Gupta (2014), clients are responsible for:

1. Establishing the network connection to the server (`connect()`).
2. Publishing messages that other clients might be interested in (`publish()`).
3. Subscribing to request messages that it is interested in receiving (`subscribe()`).
4. Unsubscribing to remove request for messages (`unsubscribe()`).
5. Disconnecting from the server (`disconnect()`).

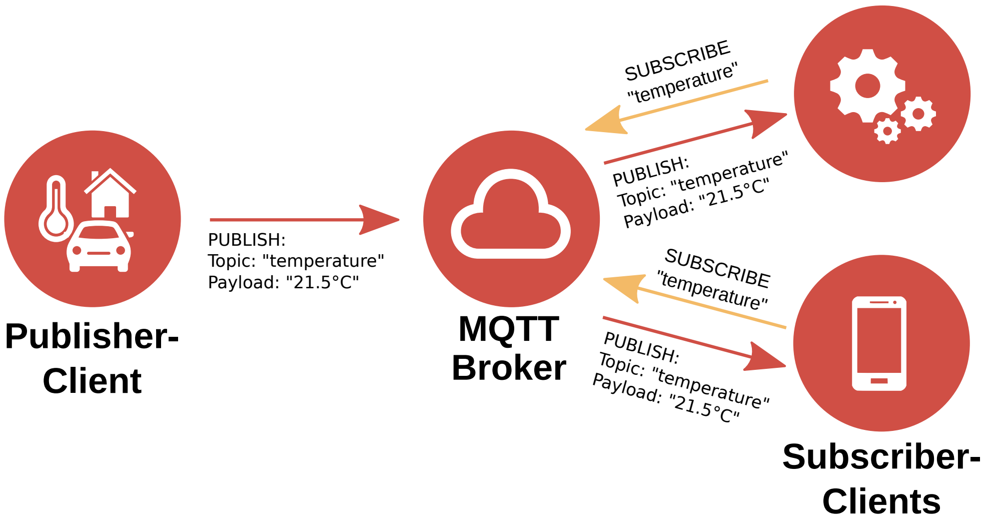
The server is the intermediary layer between clients, which publish messages, and clients, which are subscribed to messages. It provides the logic of the system and may handle thousands concurrently connected clients. It has the following tasks in MQTT:

1. Accepting network connection from clients.
2. Holding the sessions of all persisted clients.
3. Accepting messages published by clients.
4. Processing subscribe and unsubscribe requests from clients.
5. Forwarding of messages that match client subscriptions.

The server is commonly known as the *broker* (or *message broker*). The client/server model in MQTT decouples publisher and subscriber spatially, that means both solely need to be aware of the name or IP as well as the port of the broker. Further, both do not need to run at the same time, which provides time decoupling and, finally, it offers decoupling from synchronization, since subscriber and publisher do not need to interrupt operating during publishing or receiving.

BASIC INTERACTION MODEL

The general interaction between clients and the broker is illustrated in Figure 4.1. In the middle, the message broker is acting as the intermediary between the clients. While the yellow arrow represents the `subscribe()` operation by the subscribing clients on the right, the red arrow indicates publishing of a message by a publisher client (`publish()`), respectively the forwarding of that message to the subscribers by the broker.



Source: Author's illustration

Figure 4.1: *MQTT Broker*

In Section 2.3.1.3, the general idea behind topic-based publish/subscribe mechanisms was explained already. Here an example is depictured. As shown in the figure, the publish message consists of two parts of information. The payload of the message represents the data the publisher client would like to share with other clients. In addition, the client annotates the message with a topic, here "temperature". The broker receives the message, processes the topic of the message and checks if clients have been subscribed to the topic. Both clients on the right performed a `subscribe()` operation with the topic "temperature" in advance, so that the broker forwards the message to both subscribers.

TOPIC NAMES & FILTERS

When a client publishes a message, it is annotated with a so-called *topic name*. This is an UTF-8 encoded string, which is used by the broker to filter messages and, subsequently, forwards them to interested clients like in the example before. A topic can be hierarchically subdivided into several levels, whereby each topic level is separated by the topic level separator, a forward slash. For instance, this may look like the following:

`house/firstfloor/bathroom/temperature` (4.1)

Topics must consist of at least one character and they are case-sensitive. Further, the initialization of topics at the broker is not necessary. A client can publish a message with a new topic directly without creating or registering it before.

Topic filters are like topic names, but are used in subscriptions and provide additional functionalities with so-called wildcards. In the example before (see Figure 4.1), the publisher uses the topic name "temperature" to annotate his message, while the subscribers subscribe with the topic filter "temperature". When the message is received by the broker, it performs a string comparison. If they match, the message is forwarded to the subscribers. However, the topic filter can also be used to subscribe to multiple topic names at the same time by using a single-level or a multi-level wildcard.

The single-level wildcard replaces one topic level in the topic name and is indicated with a plus sign "+", while the multi-level wildcard "#" covers multiple topic levels, but can only be the last character in the topic preceded by a forward slash. A topic filter with a multi-level wildcard matches all topic names that begin with the same pattern before the wildcard character "#", regardless of the length or number of following topic levels.

Consider topic name 4.1, subscribers may, for instance, use the following topic filters containing wildcards:

`house/firstfloor/+/temperature` (4.2)

`house/+/+/temperature` (4.3)

`house/firstfloor/#` (4.4)

These topic filters all match topic name 4.1. In addition, topic filter 4.2 replaces the third level, so that topics like `house/firstfloor/office/temperature` are also matched. Topic filter 4.3 uses two single-level wildcards matching all topic names that are also matched by topic filter 4.2 and, in addition, topic names with other floors such as `house/groundfloor/kitchen/temperature`. Last, topic filter 4.4 contains a multi-level wildcard as the last character. Topic names such as `house/firstfloor/heating` or `house/firstfloor/bathroom/tub/waterlevel` are matched.

So, the broker performs a simple string comparison in the trivial case if the topic filter has not wildcards. Otherwise it performs a pattern matching based on the described procedure on the topic names of the incoming messages.

4.2.2 MQTT CONTROL PACKETS

Communication in MQTT works by exchanging a series of MQTT control packages in a defined way between client and server (broker). The type of the control package is determined by four bits in the fixed header of each message. Thus, 16 different control messages are possible, although in MQTT Version 3.1.1 only 14 are occupied. The names of the different types are given in Table 4.3 together with the value, the direction of flow and a description.

Table 4.3: *Control Packet types in MQTT*

Name	Value	Direction of flow	Description
Reserved	0x0	Forbidden	Reserved
`CONNECT`	0x1	C2S[1]	Client request to connect to Server
`CONNACK`	0x2	S2C[2]	Connect acknowledgment
`PUBLISH`	0x3	C2S or S2C	Publish message
`PUBACK`	0x4	C2S or S2C	Publish acknowledgement
`PUBREC`	0x5	C2S or S2C	Publish received (QoS part 1)
`PUBREL`	0x6	C2S or S2C	Publish released (QoS part 2)
`PUBCOMP`	0x7	C2S or S2C	Publish complete (QoS part 3)
`SUBSCRIBE`	0x8	C2S	Client subscribe request
`SUBACK`	0x9	S2C	Subscribe acknowledgement
`UNSUBSCRIBE`	0xA	C2S	Unsubscribe request
`UNSUBACK`	0xB	S2C	Unsubscribe acknowledgement
`PINGREQ`	0xC	C2S	PING request
`PINGRESP`	0xD	S2C	PING response
`DISCONNECT`	0xE	C2S	Client is disconnecting
Reserved	0xF	Forbidden	Reserved

[1] C2S: Client to Server
[2] S2C: Server to Client

Source: based on Banks & Gupta (2014)

The MQTT control packet types share a common fixed header, but have type specific variable header and payload, which depend on the capabilities of the message. In the communication between client and server, the client first establishes a session using the `CONNECT` message. In this message, it states, for instance, a mandatory client identifier. The server answers with a `CONNACK` packet to confirm the successfully established connection. We discussed earlier the `publish()` and `subscribe()` functions of the client, which are invoked by a `PUBLISH`, respectively `SUBSCRIBE` packet type. Basically, `PUBLISH` comprises of the topic name, the payload, as well as flags to control some features (see Section 4.2.3). The `SUBSCRIBE` packet type encapsulates one or more subscriptions composed of the topic filter and an associated QoS level (see Section 4.2.3.1). Subscriptions are acknowledged by a `SUBACK` packet send by the broker to notify the client about the registered subscriptions. The same holds for `UNSUBSCRIBE` and `UNSUBACK` for deregistering subscriptions. These are the most important message types. The others are utilized for specific features, which are described in the next section.

4.2.3 FEATURES

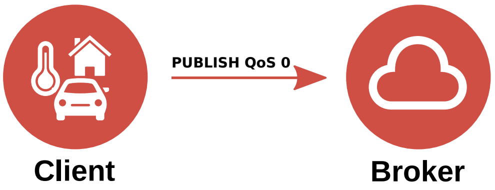
4.2.3.1 QoS

Quality of Service (QoS) denotes a mechanism between the sender and the receiver of a message to achieve a specific guarantee of delivery. This is a key feature in MQTT since it makes communication even in unreliable networks a lot easier. The guarantee of delivery can be expressed by different QoS levels, which should be chosen according to the network reliability and the use case. One of three distinct QoS level can be selected for delivery:

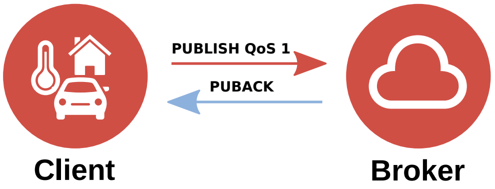
- **QoS 0**: At most once
- **QoS 1**: At least once
- **QoS 2**: Exactly once

In MQTT these levels determine the guarantee of delivery of a `PUBLISH` message in two ways: at first, from a publishing client to the broker and, secondly, from the broker to the subscribing client. In both cases, the specific QoS level is chosen by the client. If a client publishes a message to the broker, it may set a QoS field in the fixed header of the `PUBLISH` packet to indicate the desired level of assurance. Depending on the QoS level, the broker reacts to the `PUBLISH` message (see Figure 4.2).

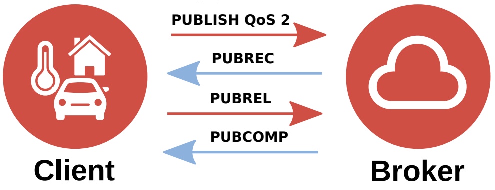
In (a) the QoS level 0 is chosen, meaning that the message is delivered by best-effort. The broker does not acknowledge receipt of the message, so that the sender does



(a) *QoS 0*



(b) *QoS 1*



(c) *QoS 2*

Source: Author's illustration

Figure 4.2: *Different QoS levels and message exchange from publisher to broker*

not know if the message finally received the broker. Therefore, it is also known as "fire and forget". QoS level 1 in (b) assures that a `PUBLISH` message is at least delivered one time to the receiver. The `PUBACK` packet is send by the receiver to signal the receipt of the message. If the message is not acknowledged, the sender resends it after a reasonable amount of time. This might cause multiple deliveries of the same message. Last, the QoS level 2 in (c) provides the guarantee of exactly one time delivery of the `PUBLISH` message. It is quite slow due to the four-part handshake shown in the figure. First, the receiver acknowledges the receipt with a `PUBREC` message. If it is not received by the sender, it resends the `PUBLISH` message with the duplicate flag (DUP) set until it receives an acknowledgement. This acknowledgement is answered by a `PUBREL` packet to signal its receipt. Finally, the broker returns a `PUBCOMP` packet to ensure that the sender and the receiver know of the one-time delivery of the message.

In the other case, if a client sends a `SUBSCRIBE` packet with topic filters, it sets a QoS level field for each topic filter. In the following `SUBACK` message, the broker grants the maximal QoS level for each topic. If the broker receives a message from a publisher, it is forwarded based on matching of topic name and topic filter. Depending on the QoS level chosen in the subscription, the messaging sequence is accordingly to Figure 4.2 but broker and client are interchanged.

4.2.3.2 Persistent Session

When a client connects to the broker, it establishes a session identified by the client id given in the `CONNECT` packet. During this session, the client may publish messages or subscribe to topics, in which it is interested. A session is named *persistent* if its state is stored in the broker even if the client disconnects or is offline. The state of a session covers:

- the existence of a session, even if the rest of the state is empty
- the client's subscriptions
- QoS 1 and QoS 2 messages send by the client that have not been completely acknowledged
- QoS 1 and QoS 2 messages received by the client that have not been completely acknowledged
- QoS 1 and QoS 2 messages that match any subscription of the client during its offline phase

To establish a persistent session, the client requests it during connecting to the broker by setting the clean session flag in the `CONNECT` packet to false. If a session associated with the client id is stored, the broker must resume communications with the client based on the state of this session, otherwise a new session is created. The session must be stored after the client is disconnected. Then after the disconnection of the client, the broker must store further QoS 1 and QoS 2 messages that match any of the client's subscriptions as part of the session state.

If the clean session flag in the `CONNECT` message is set to true, the client and server discard any previous session and start a new one. It lasts as long as the network connection, meaning that the state is not preserved in case of a disconnect.

4.2.3.3 Retained Messages

When a client subscribes to a topic, it normally receives a message annotated with that topic if another client publishes one. Therefore, the subscriber has to wait until the next message is published and has no further knowledge about the status of the topic. In this situation, *retained messages* can be used to avoid this missing knowledge. In a PUBLISH packet the retained flag in the header can be set to instruct the broker to store this message with the corresponding QoS level. Since the broker stores only one retained message per topic, following PUBLISH packets with the same topic and the retained flag set to true overwrites the previous one. Now, if a client subscribes to a topic pattern that matches the topic of the retained message, the broker sends the retained message immediately after the subscription process has finished. The subscriber, however, also knows that the message is retained because the broker sets the retained flag to true.

This way, the subscriber receives a status update of a topic immediately after it subscribed to it. So it can process this information already without waiting for new messages. However, since the retained flag is controlled by the publisher, the received retained message does not necessarily be the last message published with that topic name.

4.2.3.4 Last Will and Testament

The optional Last Will and Testament (LWT) feature of MQTT deals with the problem of ungraceful disconnections of clients, meaning that clients disconnect from the broker without sending a DISCONNECT packet. In IoT systems this may occur e.g. in environments with unreliable networks causing abrupt loss of connection or devices running out of power. The feature can help to notify other clients about the ungraceful disconnection so that they may react in an appropriate way.

For this, a LWT message can be submitted to the broker, which is encapsulated in the CONNECT packet when the client connects. It is basically an ordinary PUBLISH message including a topic, a payload, a QoS and a retained message flag. The broker stores this LWT message until the client disconnects. If it disconnects ungracefully, the broker sends the LWT message to clients that are subscribed to the last will topic. Further, if the retained flag is set to true, the LWT message becomes a retained message for that topic. Otherwise, if it disconnects with a DISCONNECT message, the broker discards the LWT message.

4.2.3.5 Keep Alive

The keep alive feature allows checking if the connection between client and broker is still open and that both are aware of being connected. During connecting to a broker, the client specifies a keep alive time interval in seconds in the header of the CONNECT message. It represents the maximum time interval that is permitted to elapse between two consecutive MQTT packets send by the client. The client is responsible that the interval never exceeds the keep alive value, otherwise the broker disconnects the network connection to the client after one and a half times the keep alive time period.

If the client has no reasonable MQTT packet to send, the client must send a `PINGREQ` packet before the keep alive period exceeds. The broker answers this request by a `PINGRESP` message. This mechanism can also be used irrespective of the keep alive time interval. It allows the client to check if the network and the broker are working as expected. The client should receive a `PINGRESP` packet after a reasonable amount of time, otherwise it cannot be sure that the network is working and should disconnect.

The keep alive time interval may range from one second to more than 18 hours, which is typically adjusted to the application. A keep alive value of zero turns off the keep alive mechanism. In this case, the broker does not necessarily have to disconnect the client due to inactivity.

4.2.4 MQTT OVER WEBSOCKETS

MQTT utilizes a TCP/IP stack to connect clients with the broker. MQTT clients are implemented in several languages providing e.g. Desktop applications or clients running on resource-constrained devices. However, with the basic protocol, the data send by MQTT cannot be received or sent directly in web browsers, although these may become the de facto interface for displaying IoT data (Cope, 2019). MQTT over WebSockets lays the foundation to provide a full-featured MQTT client in web browsers. Clients that implement MQTT over WebSockets are typically provided by a JavaScript library.

The WebSocket protocol provides two-way communication between a web application in a browser and a remote server. With its introduction, it solved the issue that HTTP does not support push notifications and, thus, polling had to be used to retrieve updates from a server (Fette & Melnikov, 2011). So, it offers a full-duplex communication channels over a single TCP/IP connection initiated by a handshake. The initial connection is established by a HTTP request performed by the client to the server. In this handshake the client request for an upgrade of the used protocol from HTTP to WebSockets. The server responds either with a `101 Switching Protocols` message if it agrees, or with any other status code to indicate that the handshake has not been completed. In case of a successful negotiation, the connection switches to WebSockets, so that client and server can transfer binary data over the connection. Otherwise the semantics of HTTP still apply.

In MQTT over WebSockets, the MQTT packets are placed into WebSocket envelopes by clients and broker. Both need to unpack it first from the WebSocket frame before being enabled to process it like an ordinary MQTT message. The MQTT mechanisms such as the connect handshake with `CONNECT` and `CONACK` or the order of packet types for the different QoS are unaffected by this additional WebSocket layer. However, to apply MQTT over WebSockets, the used browser and the broker need to support the WebSockets. Since the protocol was standardized in 2011 (Fette & Melnikov, 2011), modern web browsers already provide built-in support for WebSockets. Also, common MQTT brokers can handle WebSockets natively.

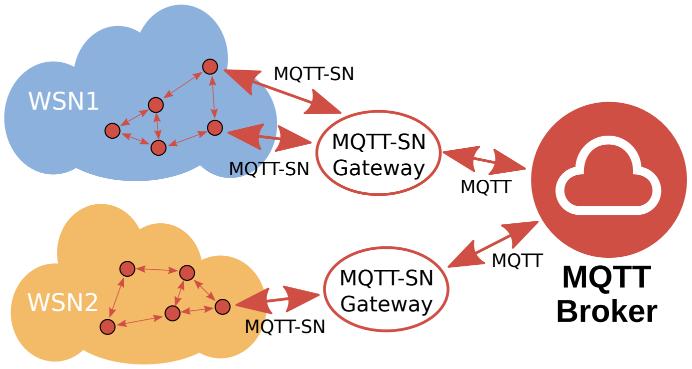
4.2.5 MQTT FOR SENSOR NETWORKS (MQTT-SN)

MQTT is based on a TCP/IP stack, but there is also an extension for connectionless communication protocols like UDP or ZigBee. This extension is especially useful in WSNs. Hence, it is called MQTT for Sensor Networks (MQTT-SN). MQTT-SN was designed in 2008 based on the following design principles (Hunkeler et al., 2008):

- As close as possible to MQTT
- Optimized for tiny battery-operated sensor/actuator devices.
- Considerations of WSN constraints such as high link failure rate, low bandwidth and short message payload
- Network independent

The architecture for MQTT-SN introduces two more components in a MQTT system to bind MQTT-SN clients: the clients themselves and a gateway acting like a translator between the two protocols (see Figure 4.3). The gateway can be implemented using a transparent or an aggregating approach. A transparent gateway establishes a new connection to the broker for each MQTT-SN client, while an aggregating gateway has only one connection to the broker.

MQTT-SN uses more message types than the MQTT protocol to reduce the size of the packets send or to overcome drawbacks of the unreliable transport. For instance, long messages are split into shorter ones. While the `CONNECT` packet in MQTT includes the client identifier as well as the Last Will and Testament topic and payload, the `CONNECT` message in MQTT-SN only requires the client identifier and a flag for the Last Will feature. If the latter one is set to true, the gateway will request the client for a Last Will topic and payload in individual message (`WILLTOPICREQ` and `WILLMSGREQ`). The client answers with a `WILLTOPIC` and a `WILLMSG` packet including the topic, respectively the payload. The gateway gathers this information first, before connecting the client to the broker in a single MQTT `CONNECT` message.



Source: adapted from Hunkeler et al. (2008)

Figure 4.3: *MQTT-SN architecture*

Another way to reduce message size involves reducing redundant information. Consider a device which measures the temperature every five seconds and sends subsequently a `PUBLISH` message with the same topic for each measurement. While in MQTT the entire topic name is send with each message, MQTT-SN clients may only include a two-byte long "topic id" in the `PUBLISH` message. The gateway consults a lookup table to match the corresponding topic name to the topic id and compiles the MQTT `PUBLISH` packet, before sending it to the broker. This requires that the topic name is defined in the lookup table in advance, which the client can achieve by sending a `REGISTER` request with the topic name included first. With the corresponding `REGACK` response by the gateway, the client receives the topic id the gateway assigned to the registered topic.

These strategies help to reduce packet size and solve the problems of unreliable networks, so that a MQTT-SN client is able connected to a MQTT broker through a MQTT-SN gateway. Since the client-gateway combination supports all features of ordinary MQTT clients, the broker cannot distinguish between MQTT and MQTT-SN clients. Additional to the features of MQTT, MQTT-SN provides also other features such as sleeping clients to reduce energy consumption or discovery of gateways. More information about the message types, features and implementation of MQTT-SN clients and gateway can be found in the specification of the protocol (Stanford-Clark & Truong, 2013).

4.3 GEOMQTT EXTENSION

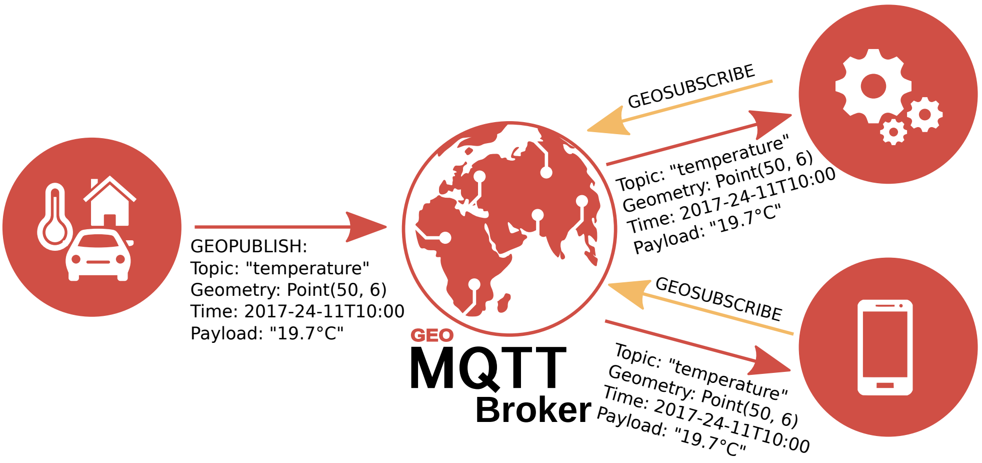
The MQTT protocol, its features and its two extensions, namely MQTT-SN and MQTT over WebSockets, can be utilized to implement a protocol, which supports the idea of sharing GeoEvents through GeoPipes in a GeoEvent-driven architecture for a Geospatial IoT like described in Section 3.5. So, the initial idea behind our extension GeoMQTT lies in modeling GeoEvents with their four attributes: name of the event, spatial component, temporal component and payload. Surely, this could have been accomplished in a simple way, for instance by applying MQTT and placing appropriate encodings in the payload of each `PUBLISH` message. However, this would not meet all requirements for the concept of GeoPipes.

Instead, our favored approach incorporates the parameter triplet (name, spatial component, temporal component) in the metadata of the message packet, since GeoEvents should be the intrinsic data type of our proposed architecture for a Geospatial IoT. Furthermore, simple event processing operations, mainly GeoEvent filtering, should be applicable to the metadata of a GeoEvent. Alongside the methods for clients in the ordinary MQTT protocol (`connect()`, `publish()`, `subscribe()`, etc.), in the GeoMQTT extension we aim at three additional methods enabling clients to deal with GeoEvents:

1. `geopublish()` creates and transmits a GeoEvent to the GeoMQTT broker.
2. `geosubscribe()` registers the client's interest to GeoEvents based on filtering operations on the metadata by using GeoSubscriptions.
3. `geounsubscribe()` deregisters GeoSubscriptions for the client's session.

With these three introduced methods, GeoEvent can be created and published by clients. The GeoMQTT broker receives the GeoEvents and performs filtering operations on the metadata based on registered GeoSubscription. The broker forwards the GeoEvent to subscribers whose GeoSubscriptions match the metadata of the GeoEvent. This process is depicted in Figure 4.4.

The MQTT broker, which is the intermediary distribution layer shown in Figure 4.1, is replace by the GeoMQTT broker, which forwards also GeoEvent besides MQTT `PUBLISH` messages to corresponding subscribers. The client on the left side performs a `geopublish()` with a GeoEvent sending it to the broker. The GeoEvent has four parameters: (1) a MQTT topic name, which corresponds to the name of the GeoEvent (here "temperature"), (2) a geometry, which represents the spatial component of the GeoEvent (here "Point(50,6)"), (3) a timestamp that is the temporal component (here "2017-24-11T10:00") and, finally, (4) the payload of the message (here "19.7 °C"). The GeoMQTT broker analyzes the first three parameters disregarding the



Source: Author's illustration

Figure 4.4: *GeoMQTT Broker*

payload and checks for GeoSubscriptions registered by clients that have performed the `geosubscribe()` operation in advance. In the example, the GeoSubscriptions of the two clients on the right side match the GeoEvent, so that it is forwarded to both. The capabilities of the spatial and the temporal filtering mechanisms are omitted here for simplicity but presented in detail in Section 4.3.2.

Since the filtering operation is not performed exclusively on the topic name of the message by the broker, the extension can be understood as an implementation of an extended topic-based publish/subscribe pattern. Before, we introduced the different types of the publish/subscribe mechanisms in Section 2.3.1.3. From these, the content-based publish/subscribe describes a pattern, in which subscribers choose different filtering criteria along multiple properties. Although, the spatial and the temporal components of the GeoEvent are incorporated rather in the metadata than in the content of the message, this description is the most accurate for the introduced mechanism.

For the three additional methods provided to GeoMQTT clients, we introduce three new control packet types. We aim for interoperability with existing MQTT brokers, therefore changes to the fixed header of control packet are out of question. Hence, these three new control packet types have to be identified by the same four bits in the fixed header. Four bits yield in $2^4 = 16$ possible control types with 14 being already occupied by MQTT control message. We choose the following assignment for the added control types (see Table 4.4).

Table 4.4: *Added Control Packet type in GeoMQTT*

Name	Value	Direction of flow	Description
`SUBACK_GEOSUB`	0x9	S2C[1]or C2S[2]	Subscribe acknowledgement and client's `geosubscribe()` request
`UNSUBACK_GEOUNSUB`	0xB	S2C or C2S	Unsubscribe acknowledgement and client's `geounsubscribe()` request
`GEOPUBLISH`	0xF	C2S or S2C	Performs `geopublish()` to send GeoEvents

1 S2C: Server to Client
2 C2S: Client to Server

Source: based on Herle & Blankenbach (2016)

From Table 4.3, it can be perceived that 0x0 and 0xF are the only free values for control types assigned to the status "reserved". So, we choose the value 0xF for representing the `GEOPUBLISH` control type for sending GeoEvents from clients to broker and vice versa. Left with one free value, the `geosubscribe()` and the `geounsubscribe()` cannot be realized with a completely unused slot. However, since both operations are only performed in one way, specifically send from clients to the broker, we can make use of control types that are send solely from broker to clients. With this double occupancy but regarding the direction of flow, there is no interference in the message exchange based on ordinary MQTT. From these considerations arise that the `SUBACK`, which is only send by the broker to a client to acknowledge or reject a subscription, can be transformed to a `SUBACK_GEOSUB` performing a `geosubscribe()` operation when send from client to server. The same holds for the `UNSUBACK`, which is transformed into a `UNSUBACK_GEOUNSUB` in GeoMQTT.

4.3.1 GEOEVENTS WITH `GEOPUBLISH` PACKET

The `GEOPUBLISH` control packet can be send by clients to a broker to model and publish a GeoEvent. Likewise, a GeoMQTT broker forwards a GeoEvent to a client with the `GEOPUBLISH` control packet, if that client is interested in it expressed by a GeoSubscription. The structure of a `GEOPUBLISH` packet is inspired by the `PUBLISH` packet, because it offers a similar mechanism and provides similar features. Thus, the sole change in the fixed header compared to an ordinary `PUBLISH` message is the allocation of the four bits, which identify the packet type (see Table 4.5).

Since the same feature such as QoS or retained messages are also implemented for the `GEOPUBLISH` message, the flags in the first byte express the same as in a

`PUBLISH` message. The QoS follows the same message exchange pattern like in `PUBLISH` messages, while the retained message feature (RE flag) is only determined by the topic name of the GeoEvent.

Table 4.5: *`GEOPUBLISH` fixed header*

Bit	7	6	5	4	3	2	1	0
Byte 1	`GEOPUBLISH` packet value (0xF)				DUP	QoS level		RE
	1	1	1	1	X	X	X	X
Byte 2	Remaining length							

Source: Author's illustration

Like the fixed header, the variable header of a `GEOPUBLISH` packet is of similar structure then the one in MQTT (see Table 4.6). In the `PUBLISH` packet the variable header contains the topic name followed by a packet identifier. The topic name is an UTF-8 encoded string and is preceded by a length field of two bytes indicating the length of that string. The packet identifier has two bytes and is only present if the QoS level is 1 or 2. The variable header of a `GEOPUBLISH` packet contains four fields in the order given in the figure. Beside the topic name and the packet identifier, a field for the timestamp/time interval and a field for the geometry are placed in front of the packet. Both are also UTF-8 encoded strings and include two bytes for the length field that indicates the number of bytes used by the specific field. For the three metadata fields, the strings may have a maximum length of 2^{16} B $= 65.535$ kB.

The temporal string can either be a timestamp or time interval defined in International Organization for Standardization (ISO) 8601:2004, or a timestamp in Unix time, which represents the seconds that have elapsed since the 1st January 1970 at 00:00:00 UTC. Both notations were already introduced in Section 3.3.3. Additionally, a Unix time interval in the form `<UNIX timestamp>/<seconds>` is also supported. The geometry string of the `GEOPUBLISH` packet can be encoded by major encoding formats for 2D geometries. These include WKT/EWKT, GeoJSON, GML or Geobuf. We already introduced the encodings in detail in Section 3.4.3. They also provide the capability to specify the geometries with a range of different CRSs, which can be parsed and handled by the broker. Although in IoT applications encodings with a smaller size are more likely, the client may freely choose between the encodings and CRSs.

Finally, the payload of the `GEOPUBLISH` packet contains the application message that is being geopublished. The content, its format and type of data is arbitrary and, thus, application specific. While the payload may be empty (zero length), its maximal length can be calculated by subtracting the length of the variable header from the maximum size of a packet, which is 256 MB.

Table 4.6: *`GEOPUBLISH` variable header*

Bit	7	6	5	4	3	2	1	0
Temporal Component								
Byte 1	Length MSB[1](Temporal)							
Byte 2	Length LSB[2](Temporal)							
Byte 3..N	Timestamp or time interval							
Spatial Component								
Byte N+1	Length MSB (Geometry)							
Byte N+2	Length LSB (Geometry)							
Byte N+3..M	Geometry							
Topic Name								
Byte M+1	Length MSB (Topic name)							
Byte M+2	Length LSB (Topic name)							
Byte M+3..O	Topic name							
Requested QoS								
Byte O+1	Packet Identifier MSB							
Byte O+2	Packet Identifier LSB							

[1] MSB: Most Significant Bit
[2] LSB: Least Significant Bit

Source: Author's illustration

4.3.2 GEOSUBSCRIPTION WITH `GEOSUBSCRIBE` PACKET

The `GEOPUBLISH` packet models a GeoEvent with the triplet (timestamp/time interval, geometry, topic name) as metadata of every message send to the broker. The broker needs to decide to which subscribing clients a specific GeoEvent will be forwarded. To establish a GeoPipe between producer and consumer of GeoEvents, we already sketch the mechanism behind "subscribing to GeoPipes" by GeoSubscriptions in Section 3.5.2.2. Basically, this results in GeoMQTT in the `geosubscribe()` operation, which may be invoked by clients.

Based on Definition 3.7 and the three filters of a GeoSubscription $(n_{filter}, g_{filter}, t_{filter})$, we introduce the following filter in GeoMQTT:

1. Topic filter (n_{filter}): The ordinary topic filter of MQTT is inherited.
2. Spatial filter (g_{filter}): This filter consists of a geometry and a spatial relation defined in Section 3.4.4.3.

3. Temporal filter (t_{filter}): Accordingly, this filter consists of a timestamp or interval and a temporal relation (see Section 3.3.4).

The broker applies the three filter of a client's GeoSubscription to the corresponding meta information of the GeoEvents, evaluates the results with a logical conjunction and forwards the GeoEvent to the client if the conjunction yields true.

Clients may perform the `geosubscribe()` operation to register their interest in GeoEvents at the broker by using the introduced `GEOSUBSCRIBE` packet. Its structure orientates on the `SUBSCRIBE` packet, thus, their fixed headers (see Table 4.7) are almost identical. The four bits to identify the control packet type are set to `SUBACK_GEOSUB`, which labels a `GEOSUBSCRIBE` packet if send from client to server like described before. The other flag bits are unused but must be set to `0b0010` respectively. The variable header contains a packet identifier of two byte length.

Table 4.7: *`GEOSUBSCRIBE` fixed header*

Bit	7	6	5	4	3	2	1	0
Byte 1	`SUBACK_GEOSUB` packet value (9)				Reserved			
	1	0	0	1	0	0	1	0
Byte 2	Remaining length							

Source: Author's illustration

In the payload of the `GEOSUBSCRIBE` message, the client may define multiple GeoSubscriptions with a QoS it requests the broker. At least one GeoSubscription/QoS pair must be contained in the message to be accepted by the broker. The payload is similar to the one of `SUBSCRIBE` packets in MQTT, which consist of topic filter/QoS pairs. The UTF-8 encoded topic filter is preceded by two bytes indicating the length of the topic filter. The QoS occupies one byte postfixing the topic filter. These pairs are then packed contiguously in the payload.

A GeoSubscription/QoS pair uses the same notation as a topic filter/QoS pair in `SUBSCRIBE` packets, but adds two fields for the temporal filter and for the spatial filter in front. The structure of a GeoSubscription/QoS pair is shown in Table 4.8.

Like the topic filter, temporal and spatial filter are UTF-8 encoded strings with two preceding bytes for the length. The last two bits of the QoS byte represent the maximum QoS level requested by the client for the specific GeoSubscription. Multiple GeoSubscriptions are simply concatenated in the payload. The broker responds by sending a `SUBACK` control packet, whose type identifier is set to `SUBACK_GEOSUB` because of the double occupancy. Its packet identifier is the same as the `GEOSUBSCRIBE` packet that is acknowledged. The `SUBACK` message contains a return code for each

GeoSubscription/QoS pair, which indicate either that the GeoSubscription failed or shows the maximum QoS that was granted for that GeoSubscription.

Table 4.8: *GeoSubscription/QoS pair in* `GEOSUBSCRIBE` *payload*

Bit	7	6	5	4	3	2	1	0
Temporal Filter								
Byte 1	Length MSB (Temporal filter)							
Byte 2	Length LSB (Temporal filter)							
	Reserved				Relation			
Byte 3	0	0	0	0	X	X	X	X
Byte 4..N	Time stamp/interval							
Spatial Filter								
Byte N+1	Length MSB(Spatial filter)							
Byte N+2	Length LSB(Spatial filter)							
	Reserved				Relation			
Byte N+3	0	0	0	0	X	X	X	X
Byte N+4..M	Geometry							
Topic Filter								
Byte M+1	Length MSB (Topic filter)							
Byte M+2	Length LSB(Topic filter)							
Byte M+3..K	Topic filter							
Requested QoS								
	Reserved						QoS	
Byte K+1	0	0	0	0	0	0	X	X

Source: Author's illustration

In the following sections, a closer look on the temporal and the spatial filter, their capabilities and configurations is taken.

4.3.2.1 Temporal Filter

The first two bytes state the length of the temporal filter, which expects in the subsequent bytes a temporal relation and a timestamp or interval (see Temporal Filter in Table 4.8). The temporal filter (relation & timestamp/interval) can also be an empty field with zero length. In this case, the filter is turned off matching any temporal expression of a `GEOPUBLISH` message.

TEMPORAL RELATION

The temporal relation is set by four bits in the third byte. Overall, 14 different values for the relation between the timestamp/interval in the GeoSubscription and the timestamp/interval in GeoEvents can be applied. These values correspond with the various temporal relations we introduced in Section 3.3.4. The values for the constellations of timestamps and/or time intervals are given in Table 4.9.

Table 4.9: *Coding of the temporal relation*

Value	Point-Point Relation	Interval-Point Relation	Interval-Interval Relation
0x0	p equals q	I contains p	I contains J
0x1	p precedes q	I precedes p	I precedes J
0x2	p equals q	I finished by p	I meets J
0x3	p equals q	I finished by p	I overlaps J
0x4	p equals q	I equivalent to p	I finishes J
0x5	p equals q	I equivalent to p	I starts J
0x6	-	I encloses p	I encloses J
0x7	p equals q	I equivalent to p	I equivalent to J
0x8	-	-	I enclosed by J
0x9	p equals q	I started by p	I started by J
0xA	p equals q	I finished by p	I finished by J
0xB	p equals q	I started by p	I overlapped by J
0xC	p equals q	I started by p	I met by J
0xD	p after q	I preceded by p	I preceded by J

p, q are timestamps
I, J are time intervals

Source: Author's illustration

Based on Allen's interval algebra (interval-interval relation column in the table), we assigned the relation evaluation of point-point relations between two timestamps and interval-point relation between a timestamp and a time interval. This way, a client subscribing e.g. with a time interval and a "meets" relation, also receives GeoEvents with timestamps although "meets" is not defined for an interval-point relation.

TIMESTAMP / TIME INTERVAL

Like mentioned in Section 4.3.1 for the `GEOPUBLISH` packet, the timestamp/interval can be specified by an ISO8601:2004 or UNIX notation. This also holds for both types in the `GEOSUBSCRIBE` packet. Additionally, repeating timestamps/intervals can be indicated by using a cron expression and, in case of an interval, a duration in seconds or in ISO8601:2004 syntax. The following Table 4.10 gives examples for the different possible configuration.

Table 4.10: *Formats of timestamps and time intervals in GeoMQTT*

Time stamp	Example
`<ISOtimestamp>`	2015-08-31T12:00Z
`<UNIXtime>`	1441017813
`<cron>`	(0 0 8 ? * SAT)
Time interval	**Example**
`<start>/<end>`	2015-08-31T12:00Z/2015-09-15T10:00Z
`<start>/<duration>`	2015-08-31T12:00Z/PT2H30M
`<start>/<seconds>`	2015-08-31T12:00Z/10200
`<duration>/<end>`	PT2H30M/2015-08-31T12:00Z
`<seconds>/<end>`	10200/2015-08-31T12:00Z
`<UNIXtime>/<duration>`	1441017813/PT2H30M
`<UNIXtime>/<seconds>`	1441017813/7200
`(<cron>)/<duration>`	(0 0 8 ? * SAT)/PT2H30M
`(<cron>)/<seconds>`	(0 0 8 ? * SAT)/10200

`<start>`, `<end>`, `<duration>` in ISO8601:2004
`<UNIXtime>`, `<seconds>` in seconds
`<cron>` expression according to Terracotta Inc. (2019a)

Source: Author's illustration

The cron expression in the examples represents a timestamp every Saturday at 8:00 am. In combination with a duration either in ISO syntax or seconds, it can be used to define repeating time intervals. For instance, (0 0 8 ? * SAT)/PT2H30M defines an interval, which starts every Saturday at 8:00 am and lasts for 2 hours and 30 minutes. Since not all relations from Table 4.9 are applicable to repeating intervals, currently, only the "contains" relationship is supported for this case.

4.3.2.2 Spatial Filter

Similarly, the spatial filter consists of a spatial relationship and a geometry, so that the broker can match the geometry of incoming GeoEvents against the geometry of the GeoSubscription with respect to the given spatial relationship. The structure of the spatial filter is given in the Spatial Filter in Table 4.8. Basically, it has the same layout as the temporal filter with a preceding two byte length field, one byte for indicating the spatial relationship and, subsequently, the geometry in a common encoding format. By leaving the spatial filter (spatial relation & geometry) empty and setting the length field to zero, the filter is turned off and the broker omits filtering GeoEvents by the spatial component for the corresponding GeoSubscription.

SPATIAL RELATION

In Section 3.4.4 the theoretical background of spatial relationships was described in detail by introducing the concepts of 4IM, 9IM and DE-9IM. In the implementation for the spatial filter, we follow the DE-9IM approach to enable clients to define topological relationships between points, lines and polygons and, thus, between the spatial components of GeoEvents and GeoSubscriptions. In fact, the spatial filter provides the application of one of the eight topological predicates, which are accepted by the OGC's Simple Feature Access (Herring, 2011), and, additionally, the derived predicates *Covers* and *CoveredBy*. The HEX values for the topological predicates are given in Table 4.11.

Table 4.11: *Coding of the spatial relation*

Value	Spatial Relation
0x0	Equals
0x1	Disjoint
0x2	Intersects
0x3	Touches
0x4	Crosses
0x5	Within
0x6	Contains
0x7	Overlaps
0x8	Covers
0x9	CoveredBy

Source: Author's illustration

GEOMETRY

The geometry can be any 2D geometry indicated by common formats for encoding geometries. Like in the `GEOPUBLISH` packet, these formats can be WKT or EWKT, GeoJSON, GML or Geobuf, which were already introduced in Section 3.4.3 in detail.

A geometry can be declared in a specific CRS. This specified CRS and the encoding format are equivalent to the requested CRS, respectively the requested encoding format for GeoEvents that are matched by the GeoSubscription. This includes that if the matching GeoEvent has a geometry, which is specified in a different CRS and/or a different encoding format than the GeoSubscription, the geometry is transformed if possible into the CRS and encoded in the format of the GeoSubscription by the broker, before it is forwarded to the client.

For both geometries (in the `GEOPUBLISH` and the `GEOSUBSCRIBE` messages) also 3D geometries are conceivable but currently not supported. In this case, also topological relationships, which act on 3D geometries, are mandatory. Also other spatial relations or indirect spatial references are imaginable. So, the spatial filter is extensible by other concepts in the future.

4.3.3 UNSUBSCRIBING FROM GEOSUBSCRIPTIONS

The `GEOUNSUBSCRIBE` control packet can be send by clients to perform the `geounsubscribe()` method with the broker. This deregisters GeoSubscriptions for the client's session. The message is marked with the `UNSUBACK_GEOUNSUB` type and answered by a `UNSUBACK` packet, in which the broker acknowledges the deregistering of GeoSubscriptions. Basically, the message has the same structure than the `UNSUBSCRIBE` control packet, which deregisters ordinary subscriptions based on a given topic filter.

This implies that the deregistering process of GeoSubscriptions is solely performed based on the topic filter. The broker deletes each GeoSubscription that has an equivalent topic filter than one included in the `GEOUNSUBSCRIBE` packet disregarding the temporal and spatial filters. This mechanism is chosen, since it simplifies the unsubscribing of GeoSubscriptions massively.

4.4 GEOMQTT IMPLEMENTATIONS

4.4.1 GEOMQTT BROKER

Conceptually, the GeoMQTT broker embodies an event processor in an EDA performing simple event processing tasks on single events (see Section 2.5.3). The task consists of three simple filtering operations, whose rules and patterns are defined by the GeoSubscriptions and, thus, in accordance with the interested subscribers. To achieve this, the introduced control packets for GeoMQTT must be implemented besides the ordinary MQTT messages and features. The implementation details of the GeoMQTT broker are covered in this section.

CONFLICT HANDLING BETWEEN MQTT AND GEOMQTT

Conflicts may occur between the subscriptions of clients (topic subscriptions or Geo-Subscriptions) and MQTT `PUBLISH` messages, respectively GeoMQTT `GEOPUBLISH` messages in the broker. For instance, consider a client that is subscribed to a topic filter with a MQTT subscription and the broker receives a `GEOPUBLISH` message with a topic name, which matches the filter of the MQTT subscription. This situation raises the question whether the message is forwarded to the client or its subscription is ignored. In our broker implementation, we realized the following conflict handling strategies (see Table 4.12).

Table 4.12: *Conflict handling strategies between MQTT and GeoMQTT*

	`GEOPUBLISH`	`PUBLISH`
GeoSubscription	Match by topic, spatial and temporal filter	Not forwarded
Subscription	Temporal and spatial information are ignored, match solely by topic filter and converted to `PUBLISH` message	Match by topic

Source: based on Herle & Blankenbach (2016)

In the trivial cases, the broker verifies the corresponding filters and forwards the messages respectively, meaning, if a client subscribed to a topic with a MQTT subscription and a MQTT publish message is being sent, the broker matches the topic name to the topic filter and forwards the message accordingly. Following this, if the client is subscribed by a topic, temporal and spatial filter (GeoSubscription) and the broker receives a GeoEvent message, all three filters are matched conjunctively to the meta information of the message and potentially forwarded. In case that a client is subscribed to a topic filter with a MQTT subscription and a `GEOPUBLISH` packet is published, the broker ignores the temporal and spatial information of the message and solely matches the topic filter. If it matches, the packet is converted to a MQTT `PUBLISH` messages and sent to the subscriber. Otherwise, if an ordinary `PUBLISH` message is received by the broker and a client is subscribed by a GeoSubscription, the message is not forwarded, even if the topic filter matches the topic name. This way, our conflict handling implementation is also compatible to MQTT clients that do not support the extension.

IMPLEMENTATION

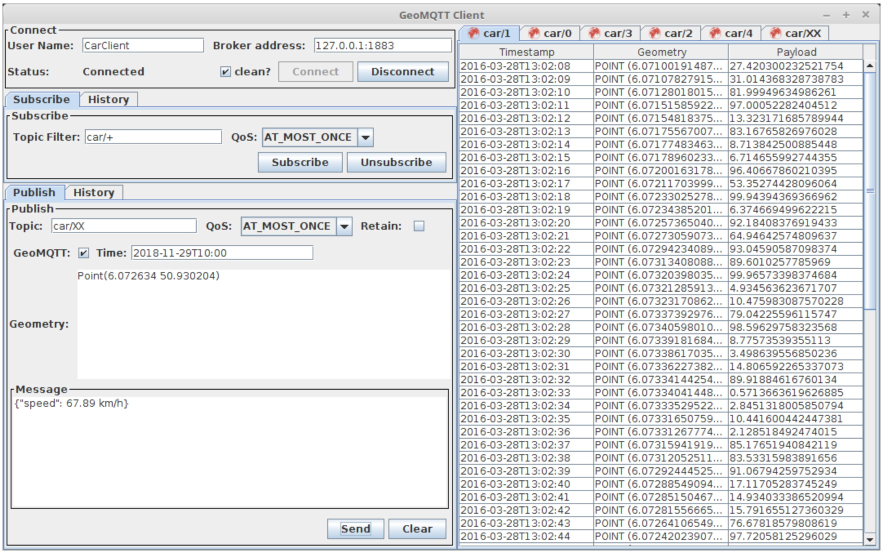
The GeoMQTT Application Programming Interface (API) is implemented in Java utilizing different libraries. For geospatial computations such as encoding, decoding or transformation of geometries the open source GeoTools library (GeoTools, 2017), which is based on the JTS Topology Suite (Eclipse Foundation Inc., 2018a), is applied. Similarly, the open source libraries Quartz (Terracotta Inc., 2019b) and Time4J (Hochschild, 2019) are employed to perform calculations on timestamps and intervals as well as cron expressions.

The developed API is integrated into an existing MQTT broker to extend it to a GeoMQTT broker. The Java-based MQTT Moquette broker (Selva, 2018) was chosen as a basis for the implementation, since it is a fully compliant MQTT broker and provides other interesting features. Additionally, with Hazelcast, Moquette version 0.9 provides a feature, which can be used to make the broker clusterizable. This way load can be distributed among several Moquette instances. We also implemented the appropriate messages for GeoMQTT to also balance the load of GeoEvents. This can especially be interesting when extensive filtering and transformation of coordinates and geometries are required. In the evaluation section (see Section 5.3), we test the performance of the broker especially with these features. In this context, we also describe the Hazelcast mechanism in more detail.

4.4.2 GEOMQTT CLIENTS

Several MQTT clients in different programming languages have been extended by the GeoMQTT features. Thus, the extension can be used by various applications, which use different languages. First, the same Java API described before was integrated in the MQTT client Xenqtt version 1.0 (McClure & Dobs, 2015) to obtain a powerful Java GeoMQTT client. A Graphical User Interface (GUI) was designed to receive and being able to send GeoMQTT messages (see Figure 4.5). This GUI-based client is mainly used to debug a system based on GeoMQTT message exchange, since the messages can be logged in the view on the right side. However, the main benefit of the Java client lies in the facile integration in other Java frameworks.

Similarly, a GeoMQTT client in Python has been developed based on the Paho Python Client (Eclipse Foundation Inc., 2018c). It supports Python 2.7.9+ as well as 3.4+. This client is used for several software plug-ins such as a QGIS plug-in (see Section 6.1) or an extension for a WPS server (see Section 6.4). Finally, a client for WebSockets in JavaScript has been developed. Since the broker supports sending and receiving GeoMQTT messages using WebSockets, a web browser can utilize the GeoMQTT JavaScript client to connect to the broker directly. This client is realized by extending the Paho JavaScript Client (Eclipse Foundation Inc., 2018b). It is also applied in several applications (see Chapter 6).



User Name: CarClient Broker address: 127.0.0.1:1883

Status: Connected ☑ clean? Connect Disconnect

Subscribe

Topic Filter: car/+ QoS: AT_MOST_ONCE

Subscribe Unsubscribe

Publish

Topic: car/XX QoS: AT_MOST_ONCE Retain: ☐

GeoMQTT: ☑ Time: 2018-11-29T10:00

Geometry: Point(6.072634 50.930204)

Message

{"speed": 67.89 km/h}

car/1 | car/0 | car/3 | car/2 | car/4 | car/XX

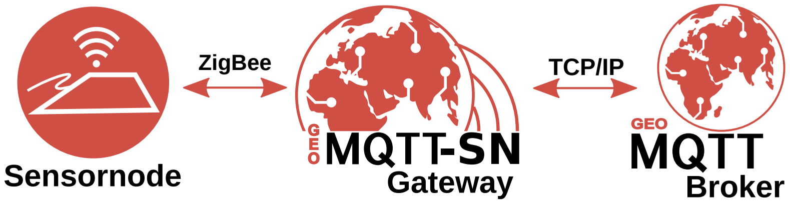
Timestamp	Geometry	Payload
2016-03-28T13:02:08	POINT (6.07100191487...	27.420300232521754
2016-03-28T13:02:09	POINT (6.07107827915...	31.014368328738783
2016-03-28T13:02:10	POINT (6.07128018015...	81.99949634986261
2016-03-28T13:02:11	POINT (6.07151585922...	97.00052282404512
2016-03-28T13:02:12	POINT (6.07154818375...	13.323171685789944
2016-03-28T13:02:13	POINT (6.07175567007...	83.16765826976028
2016-03-28T13:02:14	POINT (6.07177483463...	8.713842500885448
2016-03-28T13:02:15	POINT (6.07178960233...	6.714655992744355
2016-03-28T13:02:16	POINT (6.07200163178...	96.40667860210395
2016-03-28T13:02:17	POINT (6.07211703999...	53.35274428096064
2016-03-28T13:02:18	POINT (6.07233025278...	99.94394369366962
2016-03-28T13:02:19	POINT (6.07234385201...	6.374669499622215
2016-03-28T13:02:20	POINT (6.07257365040...	92.18408376919433
2016-03-28T13:02:21	POINT (6.07273059073...	64.94642574809637
2016-03-28T13:02:22	POINT (6.07294234089...	93.04590587098374
2016-03-28T13:02:23	POINT (6.07313408088...	89.6010257785969
2016-03-28T13:02:24	POINT (6.07320398035...	99.96573398374684
2016-03-28T13:02:25	POINT (6.07321285913...	4.934563623671707
2016-03-28T13:02:26	POINT (6.07323170862...	10.475983087570228
2016-03-28T13:02:27	POINT (6.07337392976...	79.04225596115747
2016-03-28T13:02:28	POINT (6.07340598010...	98.59629758323568
2016-03-28T13:02:29	POINT (6.07339181684...	8.77573539355113
2016-03-28T13:02:30	POINT (6.07338617035...	3.498639556850236
2016-03-28T13:02:31	POINT (6.07336227382...	14.806592265337073
2016-03-28T13:02:32	POINT (6.07334144254...	89.91884616760134
2016-03-28T13:02:33	POINT (6.07334041448...	0.5713663619626885
2016-03-28T13:02:34	POINT (6.07333529522...	2.8451318005850794
2016-03-28T13:02:35	POINT (6.07331650759...	10.441600442447381
2016-03-28T13:02:36	POINT (6.07331267774...	2.128518492474015
2016-03-28T13:02:37	POINT (6.07315941919...	85.17651940842119
2016-03-28T13:02:38	POINT (6.07312052511...	83.53315983891656
2016-03-28T13:02:39	POINT (6.07292444525...	91.06794259752934
2016-03-28T13:02:40	POINT (6.07288549094...	17.11705283745249
2016-03-28T13:02:41	POINT (6.07285150467...	14.934033386520994
2016-03-28T13:02:42	POINT (6.07281556665...	15.791655127360329
2016-03-28T13:02:43	POINT (6.07264106549...	76.67818579808619
2016-03-28T13:02:44	POINT (6.07242023907...	97.72058125296029

Source: Author's illustration

Figure 4.5: *GeoMQTT Java Client - GUI*

4.5 GEOMQTT-SN

For unreliable networks such as WSNs based on ZigBee, the MQTT-SN protocol is also extended with spatiotemporal capabilities. The extension is called GeoMQTT-SN and can be used to connect sensor nodes through a GeoMQTT-SN gateway to a GeoMQTT broker (see Figure 4.6). Like in MQTT-SN, the broker has no knowledge whether a connected client uses GeoMQTT or is connect through a gateway with GeoMQTT-SN. Thus, the latter one is compatible and implements the same mechanisms than GeoMQTT.



Source: based on Herle et al. (2018)

Figure 4.6: *GeoMQTT for Sensor Networks (GeoMQTT-SN)*

To achieve this, new message types are introduced extending MQTT-SN to GeoMQTT-SN. Like in GeoMQTT, the message types `GEOPUBLISH`, `GEOSUBSCRIBE` and `GEOUNSUBSCRIBE` can be used by the client, e.g. a sensor node, to perform the corresponding operations. The control packets are similar to the ones in MQTT-SN. They are translated by the gateway in the appropriate GeoMQTT packet types and vice versa. Additionally, two other packet types (`GEOMREGISTER` & `GEOMREGACK`) are introduced, which can be used to register geometries at the broker to avoid redundancies when publishing a GeoEvent and, thus, reduce message size. This is especially useful for stationary devices whose locations never change.

The general format of a MQTT-SN and also a GeoMQTT-SN message consists of a two or four byte header and a variable payload. The header encodes the length of the entire message (either one or three bytes) and consists of one byte for the message type. While the three byte format of the length field allows for a message size up to 65535 bytes, the one byte format supports message with a maximum length of 256 bytes. The message type field allows theoretically 256 different message types, whilst MQTT-SN already defines 27 message types. So unlike in MQTT, we do not have any issue in choosing a message type value for the five new types. Table 4.13 shows the selected values.

Table 4.13: *Added Control Packet types in GeoMQTT-SN*

Name	Value	Direction of flow	Description
`GEOSUBSCRIBE`	0x1E	C2G[1]	Client's `geosubscribe()` request
`GEOUNSUBSCRIBE`	0x1F	C2G	Client's `geounsubscribe()` request
`GEOPUBLISH`	0x20	C2G or G2C[2]	Performs `geopublish()` to send GeoEvents
`GEOMREGISTER`	0x21	C2G or G2C	Geometry register request
`GEOMREGACK`	0x22	G2C	Acknowledgement to the receipt of `GEOMREGISTER`

[1] C2G: Client to Gateway
[2] G2C: Gateway to Client

Source: Author's illustration

4.5.1 `GEOPUBLISH` IN GEOMQTT-SN

The `GEOPUBLISH` packet orientates on the `PUBLISH` packet in MQTT-SN, but adds flags for spatiotemporal options and a temporal expression as well as a section for a geometry (see Table 4.14).

Table 4.14: *`GEOPUBLISH` packet in GeoMQTT-SN*

	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Message Type							
Byte 3	Flags							
Byte 4:5	TopicID							
Byte 6:7	MessageID							
Byte 8	ST Flags							
	TimeType				GeomType			
Byte 9:m	Temporal Expression							
Byte m+1:n	Geometry							
Byte n+1:o	Data							

Source: Author's illustration

Basically, between the message id and the section for the data, we added the flags and the meta information for our extension. The spatiotemporal flags (ST Flags) determine how the subsequent two section are shaped. This allows for messages of variable message size. For the TimeType the following assignments hold:

- `0b0000`: The message does not provide a timestamp and, thus, the temporal section is empty. The gateway must add a current timestamp when receiving the message and before forwarding it as a GeoMQTT `PUBLISH` message to the broker.
- `0b0001`: The temporal section provides a timeID in the next two bytes, whose mapping is known by the gateway and used in the final GeoMQTT message.
- `0b0010`: A four byte long Unix timestamp is provided in the temporal section.
- `0b0011`: The temporal expression is a time interval which consists of a four byte Unix timestamp and a four byte duration in seconds.

Similarly, the flags for the geometry (GeomType) provide the format of the geometry section. It may have four different manifestations:

- `0b0000`: The geometry is provided by the gateway. The geometry section is of zero length.

- `0b0001`: The geometry field expresses a GeomID (GeometryID) of two byte size. This identifier can either be registered in advanced by a client using a register message (see Section 4.5.3), or the GeomID is hard-coded in the gateway and known by the client.
- `0b0010`: The geometry section represents a point in WGS84 and consists of three bytes for the longitude and three bytes for the latitude. This offers a resolution for the degree values of 4 decimal places, which is sufficient for the typical accuracy of an uncorrected GPS unit (according to van Bentem, 2018).
- `0b0011`: Same as the previous case, but latitude and longitude consist of four bytes each, which gives more than 7 decimal places.
- `0b0100`: The geometry section consists of a length header and an UTF8 encoded geometry string. By default, the length header is one byte. The least significant seven bits encode the data, which provides a length of the geometry string up to $2^7 = 127$ bytes. If the most significant bit is set, the following byte adds to the length header, so that $2^{15} = 32768$ bytes of length are possible.

These options for the temporal and spatial components of a GeoMQTT-SN `GEOPUBLISH` message provide the client ways to optimize the length of the message. For instance, if a sensor node is not equipped with a real-time clock, it may save the bytes for the timestamp by delegating the completion of the message to the gateway. On the other hand, if a sensor node is stationary, its location is not going to change, so that a geometry declaration in each `GEOPUBLISH` message would cause unnecessary redundancies. In this case, the client may use predefined geometries, which are specified in advance.

4.5.2 `GEOSUBSCRIBE` & `GEOUNSUBSCRIBE` IN GEOMQTT-SN

The `GEOSUBSCRIBE` packet in GeoMQTT-SN can be send by the client to the gateway to carry out a GeoSubscription with the broker. Hence, the client must specify a topic filter, a spatial and a temporal filter in the message. Since the message may become quite large, the client has some options to shorten the message. The format of the `GEOSUBSCRIBE` message is an extension of the `SUBSCRIBE` message type but with a different message type value (see Table 4.15).

The topic filter may be indicated by a string encoded topic filter of variable length or with a topic id of two bytes. The four most significant bit of the following byte determine the TimeType, while the other four bytes define the temporal relation like in the base protocol packet (see Table 4.9). Except for the first case, the TimeType and the corresponding temporal expression in the following bytes may take one of the same configurations as in the `GEOPUBLISH` packet. If the TimeType is set to $0b0000$,

Table 4.15: *`GEOSUBSCRIBE` packet in GeoMQTT-SN*

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Message Type							
Byte 3:4	Flags							
Byte 5:6	MessageID							
Byte 7:m	TopicFilter or TopicId							
Byte m+1	Temporal Flags							
	TimeType				Relation			
Byte m+2:n	Temporal Expression							
Byte n+1	Spatial Flags							
	GeomType				Relation			
Byte n+2:o	Geometry							

Source: Author's illustration

the temporal filter is turned off and the length of the temporal expression is zero. For the spatial filter, it holds that the GeomType is identical to the one in the `GEOPUBLISH` packet, but if set to $0b0000$, the filter is turned off. The spatial relation corresponds to the specification given in Table 4.11.

When the gateway receives a `GEOSUBSCRIBE` message by a client, it translates it to a corresponding GeoMQTT message and sends it to the broker. If the gateway accepts the GeoSubscription, it assigns a topic id to the received topic name of the GeoSubscriptions and returns it within a `SUBACK` message to the client. Subsequently, the gateway forwards `GEOPUBLISH` messages with the topic id, a timestamp or interval as well as a geometry to the client.

Since the unsubscribe mechanism for GeoSubsciptions depends solely on the topic filter and is identical to unsubscribing from ordinary subscriptions (see Section 4.3.3), the `GEOUNSUBSCRIBE` message has the same structure than the `UNSUBSCRIBE` packet in MQTT-SN, except for the message type value.

4.5.3 REGISTERING GEOMETRIES BY `GEOMREGISTER`

In MQTT-SN, the `REGISTER` packet can be send by the client to register a topic name at the gateway. The gateway responses with a `REGACK` message to acknowledge the registering process and to transmit a corresponding topic id. The client may use this topic id to annotate `PUBLISH` messages. In GeoMQTT-SN we establish the same mechanism for geometries with the introduced `GEOMREGISTER` packet (see format in Table 4.16). For instance, stationary clients may use this registering process for geometries to reduce redundancies in pending `GEOPUBLISH` packets by specifying a two byte geometry id instead of a full geometry string.

Table 4.16: *`GEOMREGISTER` packet in GeoMQTT-SN*

Bit	7	6	5	4	3	2	1	0
Byte 1	Length							
Byte 2	Message Type							
Byte 3:4	GeomID							
Byte 5:6	MessageID							
Byte 7	GeomType							
Byte 8:n	Geometry							

Source: Author's illustration

If send from client to gateway, the GeomID is empty (0x0000). However if send in the other direction, the gateway may inform a client about a mapping between a GeomID and the corresponding geometry encoded in the geometry section. The message id helps the client to match an acknowledgment message to the geometry register request. Like in the `GEOPUBLISH` packet, the GeomType is set to distinguish between an encoded point in 8 bytes (4 byte latitude and 4 byte longitude) or a geometry string. Last, the geometry field is of variable length and represents the encoded point or the geometry string.

The register request is acknowledged by the gateway with the `GEOMREGACK` packet, which has the same format than the `REGACK` message except for its message type value. Further, the topic id is replaced by the GeomID (see Stanford-Clark & Truong, 2013).

4.5.4 IMPLEMENTATION OF GATEWAY AND CLIENTS

The libraries for GeoMQTT-SN are written in C++. For the gateway implementation, the implementation of Yamaguchi (2017) serves as a basis and is extended to provide the desired message types and functionalities. It is mainly tested with ZigBee Pro as a transport layer protocol running on a Raspberry Pi Model B and Model 2.

The GeoMQTT-SN clients are also implemented in C++. They were implemented from scratch for different platforms. Especially, clients were developed for several versions of the Arduino platform and Libelium's Waspmote (Libelium, 2019). The clients cover the different message types and protocol mechanisms, but also various other functionalities have been implemented. For instance, a cron scheduler is included to enable periodical measurements. Since the mentioned sensing platforms are single-threaded, the clients are implemented as state machines.

CHAPTER 5

GEOMQTT EVALUATION

In this chapter, we evaluate the previously described spatiotemporal extension to MQTT, which we call GeoMQTT, in terms of modeling expressiveness of real-world geospatial events and states and compare the message sizes in the GeoMQTT protocol with common encodings and protocols for spatial events. Further, we test the implemented broker regarding its performance in a distributed testbed with multiple publishers and subscribers of events. Hence, this chapter provides a detailed view on the evaluations and their results. First, the evaluation objectives are defined. The second part deals with the modeling of GeoEvents and GeoSubscriptions and the mentioned comparison to existing formats. Finally, the third part covers the performance tests of the broker. The chapter finishes with a discussion on the results.

5.1 EVALUATION OBJECTIVES

The evaluation of the GeoMQTT protocol aims at investigating the protocols capabilities with respect to the requirement analysis for a Geospatial IoT and the related drawn up concepts. This includes different evaluation questions, which are listed in the following. Trying to answer these evaluation questions helps to assess the suitability of the protocol in an EDA for the Geospatial IoT based on the GeoEvent message type.

1. *Expressiveness* of the introduced message types (GeoEvent & GeoSubscription) for a Geospatial IoT: Are the mechanisms and encodings able to provide a spatial modeling for typical messages in Geospatial IoT applications?

2. *Size* and *lightweight* of the introduced messages: In comparison to other comparable encodings, are the messages in GeoMQTT and GeoMQTT-SN suitable in terms of message size? Can they be used and processed by small resource-constrained devices and, thus, have these devices the same prospects to participate in the architecture?

3. *Efficiency* of the geosubscribe mechanism: Is it possible to connect multiple devices at the same time to the architecture? How many messages can be processed by the message distributer? Do the extended filtering mechanisms demand more processing power? Does this affect the throughput of messages?

How is the performance of a single broker when using advanced features such as geometry transformation?

4. *Scalability* of the messaging middleware: Does horizontal scaling of the broker can be used and does it help to overcome the limitation of a single broker? Can the message throughput be increased by using a GeoMQTT broker cluster?

The first two evaluation objects are investigated in the following Section 5.2. This is performed based on Geospatial IoT scenarios derived from current scientific projects. Subsequently, in Section 5.3 the GeoMQTT broker performance is tested in different testbeds to examine the efficiency and the scalability of the introduced communication mechanisms, especially the advanced filtering capabilities, in GeoMQTT.

5.2 MODELING OF GEOEVENTS & GEOSUBSCRIPTIONS

5.2.1 GEOSPATIAL IOT SCENARIOS

Objects of the real-world connected in a Geospatial IoT can be stationary or mobile. Nevertheless in both cases, they have one or multiple spatial properties. According to the Geospatial IoT framework developed in Section 3.5, objects are exposed to surrounding spatial processes or actuate processes with spatial features. Depending on the type of spatial process - *open* or *closed* - the corresponding messages send by the device take the shape of a GeoEvent with a timestamp or a time interval. In the following, two scenarios of a Geospatial IoT (based on projects) are described distinguished by stationary and mobile devices. Typical spatiotemporal messages and subscriptions are derived for each case.

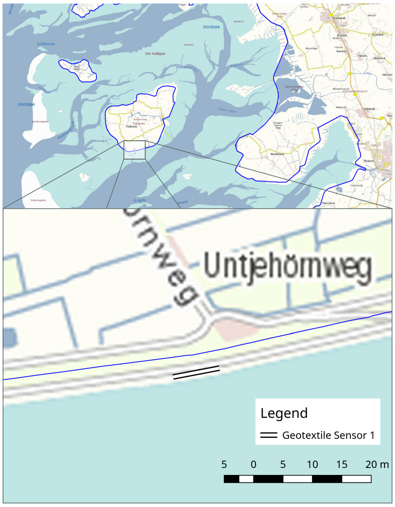
5.2.1.1 Stationary Devices - Structural Monitoring

Stationary devices and sensors are often used in e.g. environmental or structural monitoring, if the sensing campaign aims at observing changes of a specific phenomenon at a fixed location over time. The position and geometry of the sensor or the structure do not change or change only slightly over time. But other property changes of the object can be observed.

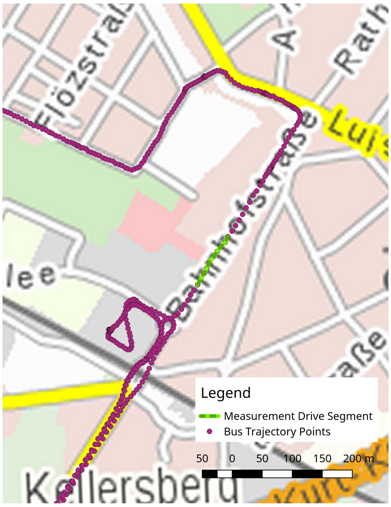
In the EarlyDike[1] project, which ran for three year until in May 2018, a risk-based early warning system for sea dikes was implemented and tested (Herle et al., 2018). Beside other working packages to model and predict external influences for the

[1] https://www.earlydike.de/en/

dike structure such as waves or storm surges, one part of the project dealt with integrating sensors into the dike's body for structural health monitoring of the inner structure. Oblong sensors were developed that was attached to a geotextile and built into the polder side of the dike beneath the surface of the slope. These newly developed sensors measure the moisture penetration, which may occur through infiltration or rise of the seepage line, as well as the deformation. By placing multiple sensor pairs parallel to the dike summit in different heights, the structure can be monitored entirely (see Figure 5.1(a)). Finally, eight sensor pairs were attached to a dike segment, each readout by a sensor node in a WSN. A gateway ensured the connectivity to the Internet and GeoMQTT, respectively GeoMQTT-SN was used as an application protocol. The messages enclosed the measurements, which were stored and analyzed in downstream steps inside the architecture.



(a) *EarlyDike scenario*



(b) *URBMOBI scenario*

Source: Author's illustration

Figure 5.1: *Geospatial IoT scenarios*

The scenario and the implementation in the EarlyDike project show that the monitoring of structures involves sending messages of a GeoEvent type. The object, the device and the sensor itself stay stationary. Nevertheless the geometry and position as well as the sampling time are important pieces of information. So a message that is published by a sensor node must have a geometry (in this case a linestring), a timestamp when the sensor is readout, a topic and a payload indicating the detected moisture penetration. In Section 5.2.2, we explain the spatiotemporal modeling of a GeoEvent in the EarlyDike scenario in detail.

5.2.1.2 Mobile Devices - Environmental Monitoring

Besides stationary sensors, mobile sensors and devices capture spatial measurements at different point in time while moving in space. Mobile positioning technologies such as GNSS ensure that the spatial reference of the devices is updated with a specific accuracy and frequency. Moving sensors gather spatially distributed data and, therefore, create opportunities for extended spatial analysis, but also raise new challenges and problems. The generation and gathering of real-time mobile geo data is additionally driven by new paradigms such as the mobile crowd sourcing/sensing idea (Chatzimilioudis et al., 2012). This development shows that a protocol for the Geospatial IoT must be also prepared for mobile sensing scenarios.

The URBMOBI project (Klok et al., 2014) is chosen to derive such a typical scenario. In the project, sensor units were mounted on the rooftop of buses from the local public transportation service in the region of Aachen, Germany (short: ASEAG). Different environmental parameters such as air temperature or humidity were measured while each bus operated its specific bus line. This was accomplished in different measuring campaigns in the life spans of the project (URBMOBI 1.0 2010-2011, URBMOBI 2.0 2013-2015). Besides the microcontroller board, the developed sensor unit consists of various meteorological sensors, a GPS receiver, a micro-SD card for storing data locally and wireless communication electronics. This setup allows measuring temporally and spatially distributed environmental data at a low-cost level. The project aimed at analyzing urban heat and thermal comfort in cities based on the collected data.

URBMOBI allows for deriving use case scenarios for defining GeoEvents and GeoSubscriptions in a Geospatial IoT application with mobile devices. For instance, a sensor unit mounted on a bus roof measures the air temperature every second while driving from a place A to a place B. This scenario is depictured in Figure 5.1(b) with the drive segment representing the measuring campaign. The measurements for the traveled trajectory are averaged before the system sends a message with the measured value to the server. Thus, the temporal and the spatial components change during the measurement processes, both should be modeled with appropriate dimensionality in a GeoEvent in our proposed architecture.

5.2.2 SPATIOTEMPORAL MODELING

The scenarios described above demonstrate a distinction between stationary and non-stationary, respectively moving or mobile objects equipped with sensors. However, in both scenarios the most important spatial property represents the location of the deployed sensors. In the following the message types for GeoEvents and GeoSubscriptions are constructed for both scenarios.

MODELING GEOEVENTS AND GEOSUBSCRIPTIONS IN EARLYDIKE

In the EarlyDike use case, stationary dikes on the German coastline are equipped with intelligent geotextiles, which are embroidered with carbon fibers in different heights parallel to the dike's summit. Two carbon fibers form an oblong sensor to detect soil moisture and deformations in the dike structure. The measuring process is performed by a sensor node e.g. every two minutes to monitor the inner structure of the dike.

From this scenario, we are able to construct an appropriate **GeoEvent** with the three properties: timestamp, geometry and topic name. The *topic name* can be modeled as e.g.

$$\texttt{dike/untjehoern/geotextile/sensor/1/voltage} \tag{5.1}$$

It consists of several hierarchies identifying the dike's section. `untjehoern` is the name of the dike, while the consecutive hierarchies describe which sensor performed the measurement (`.../geotextile/sensor/1/...`). The last topic part gives the unit of the measured and transferred value (`.../voltage`). The sensor measures the voltage, which drops between the two carbon fibers, to detect the moisture level.

For the *temporal part*, a timestamp identifies the temporal location of the measurement. Since it describes one single point in time, the GeoEvent embodies a Geospatial State (or open geospatial process) of a continuant, more precisely the specific dike section. This point in time can be e.g. given by a timestamp in ISO8601 such as

$$\texttt{2017-08-07T09:32:00Z} \tag{5.2}$$

The *spatial component* of the GeoEvent in EarlyDike describes the location of the oblong sensor. By modeling this property as a linestring with tuples of coordinates, the sensor can be described by its location, shape and extent in the real-world:

```
LINESTRING (8.63726521337415 54.48982784467354, 8.638482143512755
     54.489966406025964, 8.638482143512755 54.489966406025964)
```

Listing 5.1: *Geometry of one moisture sensor in the EarlyDike scenario*

The stated examples of meta information for the GeoEvent in the EarlyDike scenario are used in Section 5.2.3 to evaluate and compare the message size in GeoMQTT and other encoding formats. The payload of the GeoEvent is a voltage value, for instance $3.48V$.

A suitable **GeoSubscription** for clients to receive the published GeoEvents can be modeled in multiple ways. Clients may, for instance, use solely the *topic filter* to specify their interest in the measurements and leave the other filter in wildcard mode. Hence, a client can directly subscribe to a specific sensor or use wildcards to receive multiple GeoEvents. In the scenario, reasonable topic filters might be the following.

$$\texttt{dike/untjehoern/\#} \tag{5.3}$$

$$\texttt{dike/untjehoern/geotextile/\#} \tag{5.4}$$

$$\texttt{dike/untjehoern/geotextile/sensor/+/voltage} \tag{5.5}$$

$$\texttt{dike/untjehoern/geotextile/sensor/1/voltage} \tag{5.6}$$

While in topic filter 5.3 all GeoEvents related to or published by the dike's section of Untjehörn are received by interested clients, topic filter 5.4 is solely applied to GeoEvents specific to the geotextile built into the dike. Topic filters 5.5 and 5.6 look similar, but filter 5.5 applies a wildcard to the sensor id so that GeoEvents emitted by every oblong moisture sensor mounted to the geotextile of the dike section of Untjehörn are received.

The temporal and the spatial filter offer different filtering compositions described in Section 4.3.2 or can be both turned off. Reasonable *temporal filters* can be e.g. the following:

$$\texttt{Equals;2017-08-07T08:00:00Z} \tag{5.7}$$

$$\texttt{Encloses;2017-08-07T08:00:00Z/2017-09-23T11:00:00Z} \tag{5.8}$$

$$\texttt{Encloses;(0/30 0/24 0/12 ? * * *)/28800} \tag{5.9}$$

Temporal filter 5.7 can be used to subscribe to a single point in time with the `Equals` relationship. The filters in 5.8 and 5.9 define a time interval, respectively a repeating time interval with the `Encloses` relation. The repeating time interval in 5.9 utilizes a cron expression, which gives the instants in time every twelve hours, 24 minutes and 30 seconds, and a duration of 28800 seconds (or eight hours). This describes a half tidal cycle, if the subscriber is solely interested in the state of the dike during the flooding phase.

Suitable *spatial filters* might be for instance given in Listing 5.2, here the geometries are encoded in WKT. The first spatial filter specifies a Polygon that describes the dike section of Untjehörn, while the polygon in the second filter (line 2) characterizes the bounding box of the island Pellworm. Both filters use a `Covers` relationship, so that for GeoEvents, whose geometries are inside the specified polygons, the spatial filter function returns "true" as a result.

```
Covers; POLYGON ((8.63837 54.49042, 8.63719 54.49032, 8.63573 54.49021, 8.63578
      54.48964, 8.63722 54.48973, 8.638512 54.48988, 8.63984 54.49008, 8.63969
      54.49057, 8.63969 54.49057, 8.63837 54.49042))
Covers; BBOX(8.587638 54.557351, 8.708814 54.489124)
```

Listing 5.2: *Spatial filters for the EarlyDike scenario*

For the evaluation of the GeoSubscription message size in Section 5.2.4, we use the following composition of filters: topic filter 5.5, temporal filter 5.8 and spatial filter in line 1 of Listing 5.2.

MODELING GEOEVENTS AND GEOSUBSCRIPTIONS IN URBMOBI

In the URBMOBI use case, moving buses are equipped with sensors to monitor the environment. This includes sensors to measure the temperature, the humidity and the solar radiation. Additionally a GPS receiver determines the position of the bus during the monitoring process. Like in the EarlyDike scenario, we can construct a typical **GeoEvent**. We model the *topic name* in the following way:

$$\texttt{bus/line/66/temperature/celsius} \tag{5.10}$$

, where the first three hierarchy levels define the bus line (`bus/line/66/..`), followed by the observed property (here `../temperature/..`) and the used unit as the ultimate level. The *temporal part* is modeled as a time interval since a measurement campaign consists of a trajectory, which the bus travels for ten seconds (see time interval 5.11). Depending on the sensor's sampling rate, the measurements are averaged.

$$\texttt{2016-08-17T16:41:10Z/2016-08-17T16:41:19Z} \tag{5.11}$$

The *spatial part* describes the trajectory, the bus traveled during the measuring campaign. Thus, it is a line, whose vertices represent the measured GPS positions of the bus. An example is given in Listing 5.3.

```
LINESTRING (6.1597 50.87225, 6.159633333 50.87216667, 6.159533333 50.8721, 6.15945
       50.87203333, 6.15935 50.87195, 6.159266667 50.87186667, 6.1592 50.87178333,
      6.159133333 50.8717, 6.15905 50.87161667, 6.158983333 50.87153333)
```

Listing 5.3: *Geometry of a trajectory of a bus*

The stated examples of meta information for the GeoEvent in the URBMOBI scenario are used in Section 5.2.3 to evaluate and compare the message size in GeoMQTT and other encoding formats. The payload of the GeoEvent is an averaged temperature value measured in degree Celsius, for instance $21.48℃$.

Similarly to the EarlyDike scenario, different filters can be constructed to specify a **GeoSubscription**. We can imagine the following *topic filter* depending on the subscriber's interests. Using topic filter 5.12, a client may subscribe to every GeoEvent published by buses. Further, while topic filter 5.13 describes all temperature measurements in degree Celsius published by every bus line, topic filter 5.14 omits wildcards and can be used to receive the specific GeoEvents annotated by the same topic name. The latter one is chosen for the message size evaluation of GeoSubscriptions in Section 5.2.4.

$$\texttt{bus/\#} \tag{5.12}$$

$$\texttt{bus/line/+/temperature/celsius} \tag{5.13}$$

$$\texttt{bus/line/66/temperature/celsius} \tag{5.14}$$

Also different *temporal filters* can be conceived in the scenario. In the evaluation of the message sizes later, we use the temporal filter in 5.15. It evaluates to true if the measurement is contained in the given time interval. At this, it does not matter if the GeoEvent is a geospatial event with a time interval like in the URBMOBI case, or a geospatial state with a timestamp. For more information, see the temporal relationships between intervals and points in time in Section 3.3.4.

$$\texttt{Contains}; \texttt{2015-09-01T11:00:05+02:00/2016-09-01T11:00:06+02:00} \tag{5.15}$$

The *spatial filter* can be multifaceted depending on the use case and interests of the subscribers. For instance line 1 in Listing 5.4 uses a `Contains` relationship and a polygon that describes a road segment. For every GeoEvent, which is contained by the polygon, the filter is evaluated to true. This first example is used as well in the comparative analysis of the formats in the next section. The second example (line 2 in Listing 5.4) exemplifies a spatial filter with an `Intersects` relation. The geometry describes a polygon created by a 1 km buffer around the city center of Aachen (historic market square). Here a EWKT format was used since the coordinates are given in a CRS with metric units. The idea behind this filter in the URBMOBI scenario is to monitor and receive GeoEvents published by buses, which are in close proximity to the center of the city.

```
Contains; POLYGON ((6.158429038058124 50.871240749670065,6.159239664227631
     50.872029079445475, ... ,6.158429038058124 50.871240749670065))
Intersects; SRID=31466;BUFFER(POINT (2505976.633777643088251
     5626672.894559904932976), 1000)
```

Listing 5.4: *Spatial filters for the URBMOBI scenario*

For the constructed scenarios and their use cases, the expressiveness of GeoEvents and GeoSubscriptions can be assessed. Both, the GeoEvent for expressing geospatial events and states and the GeoSubscription messages are sufficient for spatiotemporal modeling of the real-world phenomena and specification of interest in them. Thus, it can be used to implement an event-driven Geospatial IoT like proposed in Chapter 3 from a modeling point of view.

5.2.3 GEOEVENT ENCODING COMPARISON

The spatiotemporal modeling evaluation of the introduced messages in GeoMQTT assessed the requirement for expressiveness of real-world geospatial events and states as well as the capabilities of defining GeoSubscriptions. Following this, the second evaluation objective call for reasonable size of the encoded messages. Relatively small encodings empower and encourage small resource-constrained devices to participate in a Geospatial IoT. Thus, in this section the sizes of the encoded GeoEvents in GeoMQTT are analyzed and compared to encoded GeoEvents in similar formats. Considering this, comparable formats must be identified first. The following list covers candidates to encode GeoEvents and assess their applicability.

1. *Event Pattern Markup Language (EML)* can be used to describe event patterns for event processing and analysis based on an XML format. This includes a data type for EML event objects, which can be used to encode spatiotemporal events with different properties (Everding & Echterhoff, 2008).

2. *Observations and Measurements (O&M) 2.0* is a conceptual schema providing models for the exchange of information describing observation acts. Observations may have data types including primitive types but also complex ones such as time, location and geometry (Cox, 2013). There are implementations in XML and JSON. The encodings are associated with the SWE standards. Especially, the JSON encoding is used in the SensorThings API.

3. *Common Alerting Protocol (CAP)* is a digital message format based on XML for all types of alerts and notifications. Temporal information can be defined with timestamps, a time interval with an effective timestamp and a point of expiry. The encoding provides an area element, which can be set to a textual and

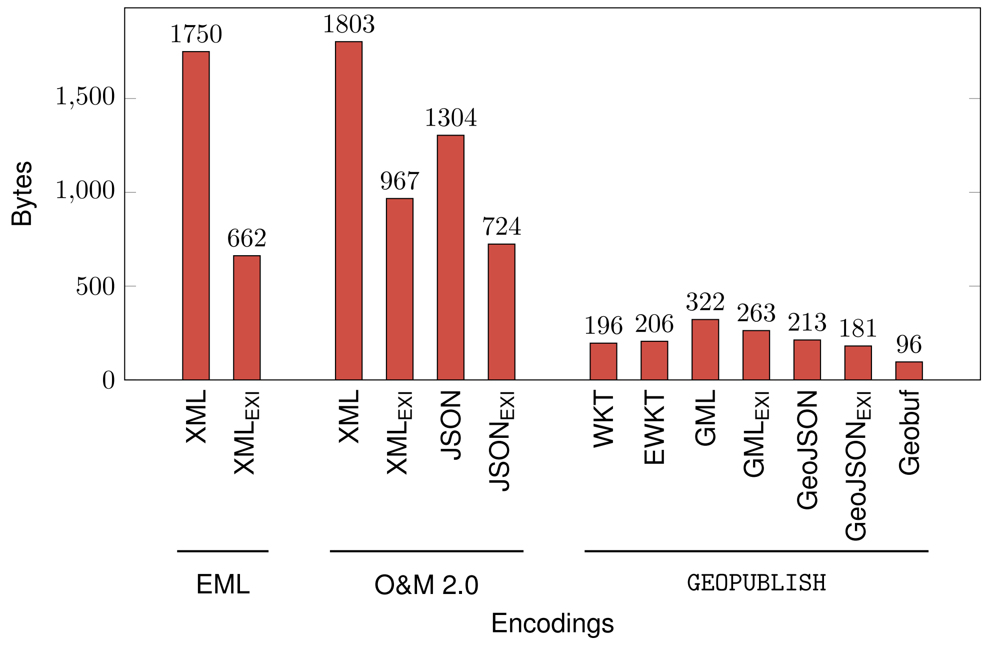
coded description such as postal codes or, preferably, described by a geospatial shape (Westfall, 2010). However, the latter one only offers polygons and circles and, thus, is not suitable for the GeoEvents modeled in our scenarios.

4. *Moving Feature Access (MF)* is an OGC standard that specifies encoding representations of movement of geographic features with rigid bodies. The features are described by using space-time coordinates. Several encodings of the model exist: Comma Separated Values (CSV) (Asahara et al., 2015a), XML (Asahara et al., 2015b) and JSON (Kim & Ogawa, 2017). The MF JSON encoding applies to representations and formats of GeoJSON, but adds new terms to specify dynamic attributes of moving features. The encoding standard can be used to encode the measurement campaigns in the URBMOBI scenario.

Therefore, for our scenarios we encode the GeoEvents in EML, O&M 2.0 and, additionally, in MF in the URBMOBI case. While EML is an XML derivative, O&M 2.0 can be implemented in XML and in JSON. The MF encoding is solely performed in JSON. In addition, both - XML and JSON encodings - can be compressed by the EXI mechanism. Efficient XML Interchange (EXI) and EXI4JSON are binary encodings of XML respectively JSON developed by the W3C's Efficient Extensible Interchange Working Group (Kamiya, 2018). They reduce the verbosity of the documents and the cost of parsing significantly. For the encoding in GeoMQTT's `GEOPUBLISH` message, we use the supported geometry encoding formats also including EXI if applicable (e.g. in GML). ISO8601 timestamps or time intervals are used to encode the temporal component in each `GEOPUBLISH` message.

MESSAGE SIZES IN THE EARLYDIKE SCENARIO

Figure 5.2 shows the resulting sizes of the messages in the EarlyDike scenario. It becomes clear that the `GEOPUBLISH` encodings of the GeoEvent are smaller in comparison to the other chosen formats. Encoded in an EML document, the GeoEvent constructed in the last section has a size of 1750 B, it can be reduced by the EXI mechanism to 662 B without information loss. Similarly, the XML document of the O&M 2.0 version has the largest size with 1803 B, which can be reduced to 967 B in the EXI case, respectively 724 B in the JSON_{EXI} format. When using GeoMQTT's `GEOPUBLISH`, the largest message is obtained encoding the geometry in GML (322 B). While the smallest `GEOPUBLISH` message using a lossless geometry encoding is the WKT format with 196 bytes, it can be even downsized by applying Geobuf encoding. However, due to the encoding with 6 digits after decimal point, this leads to precision loss. In comparing these encoding formats, the reader should bear in mind, that EML and O&M 2.0 are encoding standards for documents. For exchanging them between entities, a suitable transfer protocol such as HTTP is still needed, while in GeoMQTT's `GEOPUBLISH` the transfer protocol is already included.

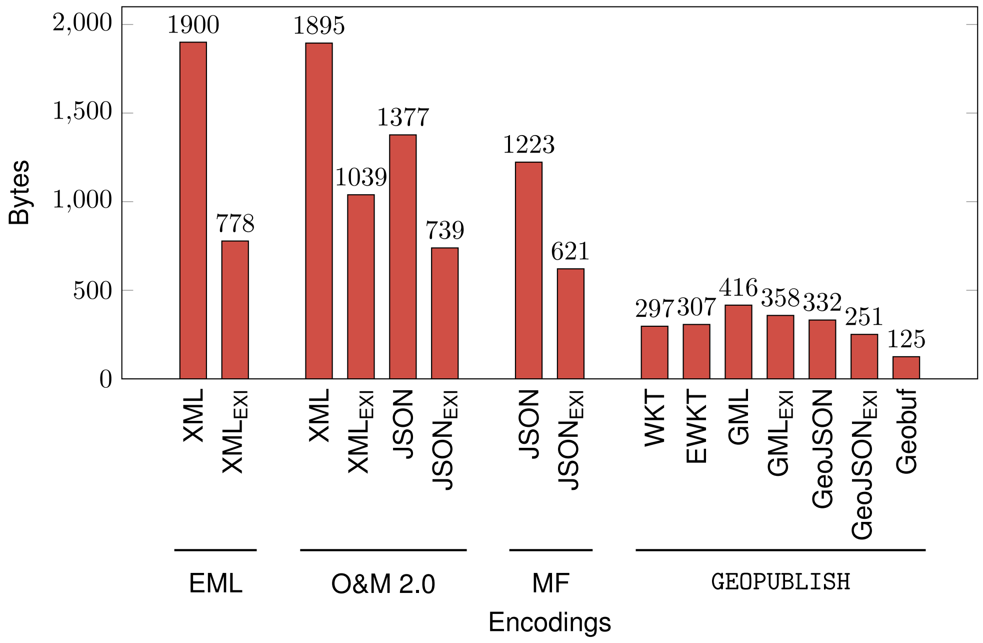


Source: Author's illustration

Figure 5.2: *Encoding sizes of a GeoEvent in the EarlyDike scenario*

MESSAGE SIZES IN THE URBMOBI SCENARIO

In the URBMOBI scenario, similar messages can be constructed. Additionally to EML and O&M 2.0, the message size of MF with a JSON encoding is investigated. The resulting message sizes are depictured in Figure 5.3. Here again, the encodings of GeoMQTT's `GEOPUBLISH` outperform the documents of EML, O&M 2.0 and MF in terms of small message size. With 1900 B the plain EML document has the largest size, with EXI it can be reduced to 778 B. O&M 2.0 has also quite large sizes with the smallest at 621 B when using JSON$_{EXI}$. The MF versions have comparable sizes to the different versions of EML and O&M 2.0. Again the smallest lossless `GEOPUBLISH` message is obtained by WKT encoding with 297 B. Geobuf, however, decreases this size to 125 B by reducing the resolution to 6 digits after decimal point.

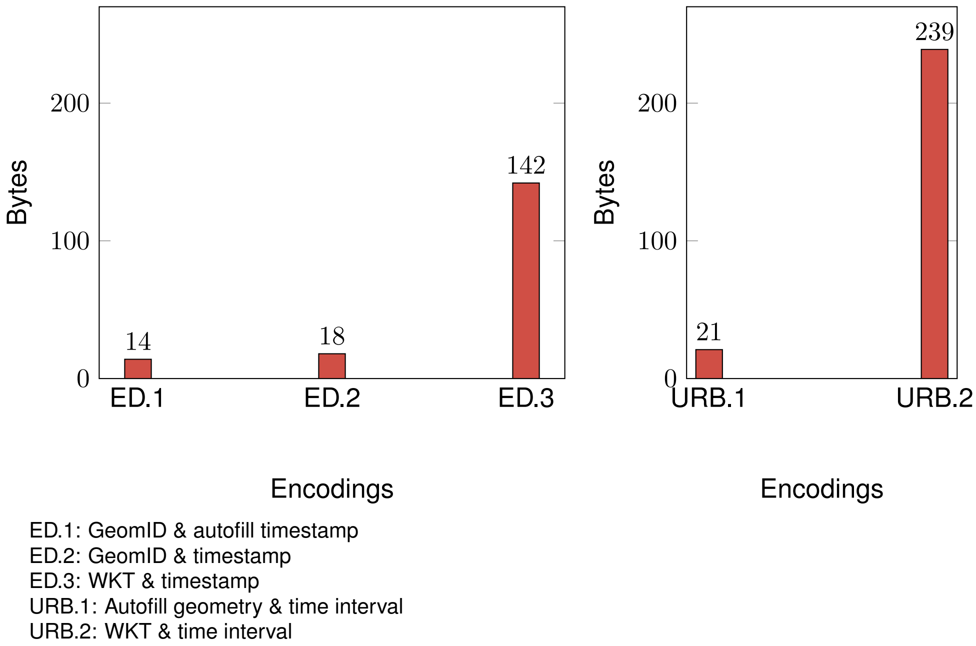


Source: Author's illustration

Figure 5.3: *Encoding sizes of a GeoEvent in the URBMOBI scenario*

Message sizes in GeoMQTT-SN for both scenarios

Finally, we analyze several versions and configurations of GeoMQTT-SN's GEOPUBLISH regarding the size of the sample message in the EarlyDike and URBMOBI scenario. As described in Section 4.5, the GEOPUBLISH message of GeoMQTT-SN, which is sent from sensor nodes in a WSN to the gateway, uses mechanisms to reduce the message size. Subsequently, the gateway translates these reduced-size packets to GEOPUBLISH messages in GeoMQTT. This is achieved by e.g. letting the gateway append the specification of the geometry or the timestamp. In the EarlyDike use case the geometry can easily be added by the gateway since the position of the sensor is stationary. Either the sensor node registers the geometry in advance with a GEOMREGISTER message or it is already hard coded into the gateway's configuration. In its GEOPUBLISH message, the sensor node indicates solely the corresponding GeomID with two bytes. The resulting message sizes are plotted in the left chart of Figure 5.4. By entrusting the completion of the message to the gateway, the packet size can be reduced to 14 B, if both - geometry and timestamp - are omitted (see ED.1). If the timestamp is added to the message (ED.2), the size increases by four bytes. Finally, if the GeoMQTT-SN client is also in charge of defining the geometry, the size adds up to 142 B when the WKT encoding is used. In the URBMOBI scenario, the bus is a moving object and, thus, the geometry (here trajectory) changes for each GeoEvent. Therefore, the geometry as well as the time interval are most likely included in the packet. In the right chart of Figure 5.4, this case is shown in URB.2. For the previously defined sample GeoEvent, this leads to a `GEOPUBLISH` message in GeoMQTT-SN of 239 B, which is not significantly smaller than the same message in GeoMQTT. However, we can also think about a scenario, in which a bus is equipped with a Personal Area Network (PAN) of multiple sensor nodes and a gateway to connect to the Internet. The sensor node is responsible for the temperature measurement and the forwarding to the gateway, but the gateway determines the positions and the trajectory. Then, the packet size can be decreased to 21 B because the gateway adds the geometry of the message.



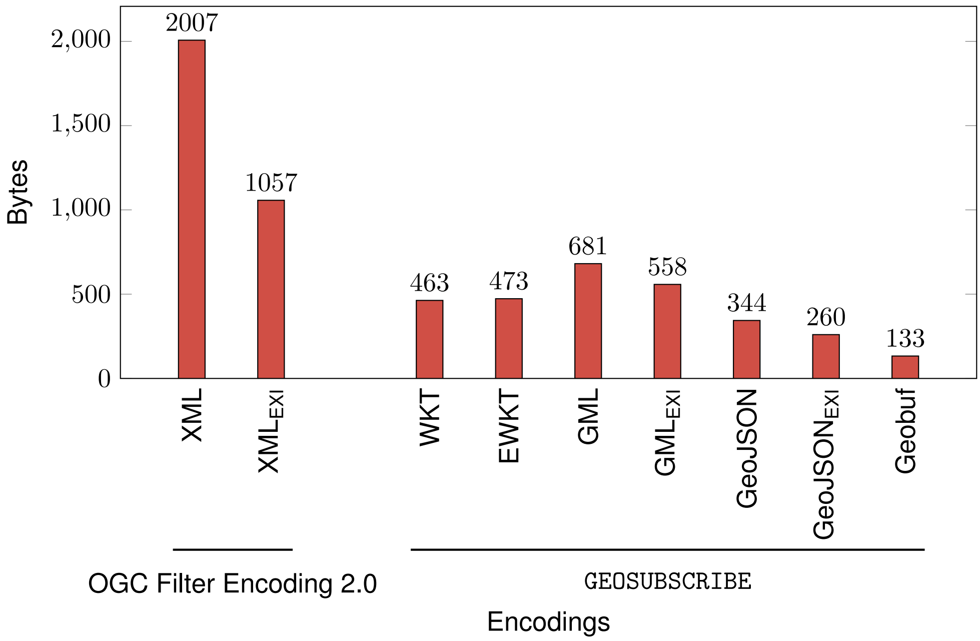
Source: Author's illustration

Figure 5.4: *GeoMQTT-SN encoding sizes of a GeoEvent in the EarlyDike & URBMOBI scenarios*

This evaluation shows that also small resource-constrained devices in a sensor network can participate in an architecture based-on GeoMQTT. The message sizes can be reduced to a very small size without losing the expressiveness of a GeoEvent. However, it should be mentioned that the dominating factor in terms of packet size is the geometry encoding. If it is not known by the gateway or changes frequently, it must possibly be included in the message, which leads to a significantly larger packet, or it must be registered by the GeoMQTT-SN client in advance resulting in a larger number of messages in the network.

5.2.4 GeoSubscription Encoding Comparison

In addition to the comparison of GeoEvents in different formats, the sizes for encoding GeoSubscriptions are evaluated. The GeoSubscriptions that are constructed in Section 5.2.2 for the two scenarios are the basis for the evaluation. For GeoMQTT, we include the scenario specific GeoSubscription in a `GEOSUBSCRIBE` packet type and compare different geometry encodings. Timestamps or time intervals are specified by their ISO8601 versions and are not changed for the different packets.



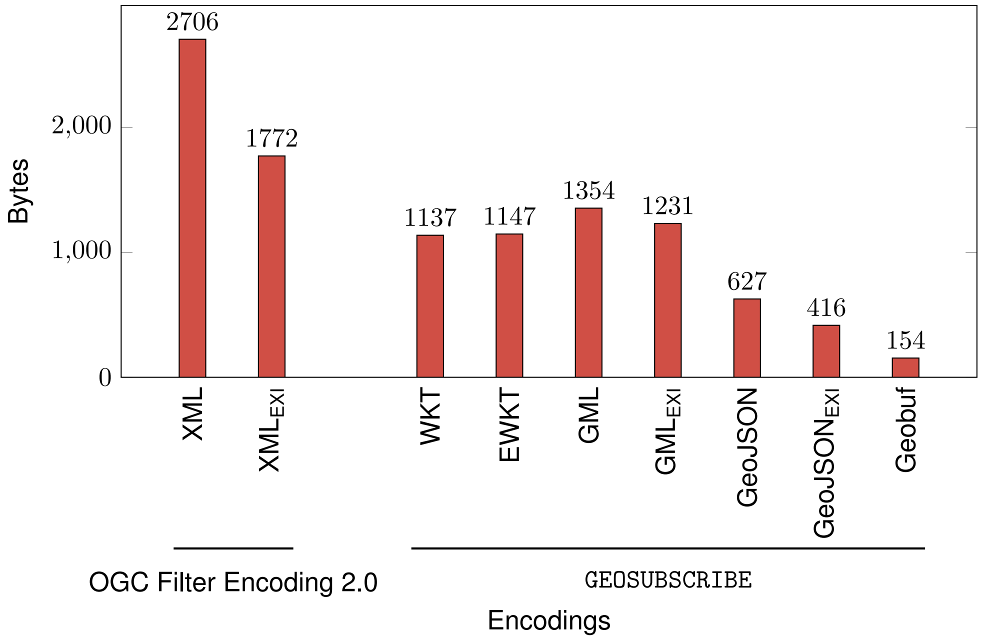
Source: Author's illustration

Figure 5.5: *Encoding sizes of a GeoSubscription in the EarlyDike scenario*

Furthermore, we identify the OGC Filter Encoding Standard (FES) 2.0 as an alternative candidate to encode GeoSubscriptions. The OGC standard describes an XML and KVP encoding of a system neutral syntax for expressing projections, selection and sorting clauses called a query expression (Vretanos, 2014). Several filters can be specified such as comparison filters on data types like integers or strings, but also temporal and spatial filters. The expressiveness of the standard is applicable to define GeoSubscriptions. Thus, we use the XML version to model the stated GeoSubscriptions and compare them to the `GEOSUBSCRIBE` packet type of GeoMQTT. Like in the comparison for GeoEvents, the EXI versions of the encodings are also included.

GeoSubscription sizes in the EarlyDike scenario

The encoding sizes of the GeoSubscription in the EarlyDike use case are depictured in Figure 5.5. Clearly, the sizes of the `GEOSUBSCRIBE` packet outperform the FES 2.0 documents. Encoding the GeoSubscription in the XML version of the FES 2.0 results in a quite large document of 2007 B, while the EXI version has still a size of 1057 B. For the `GEOSUBSCRIBE` message, we obtain similar results compared to the GeoEvent encoding. The smallest lossless size is provided by the WKT geometry encoding with 463 B. The Geobuf version reduces the precision of the geometry's coordinates but also eventuates in half of the size (133 B).



Source: Author's illustration

Figure 5.6: *Encoding sizes of a GeoSubscription in the URBMOBI scenario*

GeoSubscription sizes in the URBMOBI scenario

Similar results are obtained by the URBMOBI use case (see Figure 5.6). Here, a polygon with a high precision representing an area around a road segments is used in the spatial filter. Unsurprisingly, the FES 2.0 document has a large size with 2706 B as well as the EXI version with 1772 B. The compression of the Geobuf encoding including its precision loss reduces the size of a `GEOSUBSCRIBE` message to 154 B. Still, WKT provides the smallest lossless encoding of a `GEOSUBSCRIBE` packet with 1137 B.

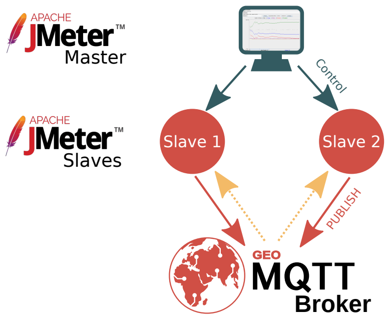
5.3 Broker Performance Testing & Scaling

After evaluating the expressiveness and the sizes of the introduced message types in GeoMQTT, we proceed with the other two objectives of this evaluation raised in Section 5.1. It covers the efficiency and lightweight of the implemented broker and its mechanisms as well as the scalability of the system. So, the focus of this evaluation step is on the performance of the broker during message distribution. This involves the investigation of the broker during heavy message load using multiple test scenarios. For instance, we like to assess the functional stability of the broker while varying the throughput of messages per second, or the behavior when changing the number of subscribers and the number of brokers.

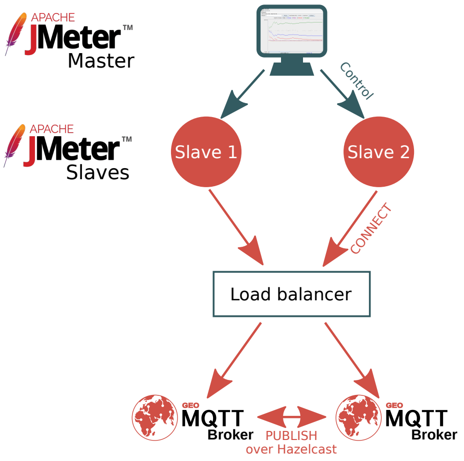
5.3.1 Testbed Specifications

Before describing the details of the performance test and the different parameter variation, this section introduces the specification of the testbeds which are used for the tests. Basically, two testbeds are set up, to meet the objectives and evaluate the efficiency and scalability of the implemented GeoMQTT broker. Both are depictured in Figure 5.7.

The main difference between the two testbeds is the number of brokers involved. While in testbed 1 (see Figure 5.7 (a)) a single GeoMQTT instance is deployed, in testbed 2 (see Figure 5.7 (b)) a GeoMQTT broker cluster consisting of two brokers is installed to distribute the load of the messages utilizing the hazelcast framework. Thus, the first testbed aims at investigating the efficiency of the broker implementation and testbed 2 is employed to assess the scalability. However in both testbeds, GeoMQTT clients are scalable deployed on two machines using the JMeter framework. Clients and server as well as the applied technologies are defined in the following.



(a) *Testbed 1 with one broker*



(b) *Testbed 2 with two brokers*

Source: Author's illustration

Figure 5.7: *Geospatial IoT scenarios*

Setup of the Clients

The clients in our performance tests are distributed deployed on two machines and use the GeoMQTT Java implementation mentioned before. To plan, control and execute tests on the testbeds, the Apache JMeter tool is used.

Apache JMeter[2] is a Java application originally designed to load test functional behavior and measure performance of Web applications but has been expanded to other test functions as well. Heavy load on a server can be simulated to analyze overall performance under different load types. A full featured testing GUI facilitates the design, recording and debugging of test plans. However, a command-line mode is also included to perform load test from any Java compatible Operating System (OS) without the overhead of the GUI. Additionally, JMeter allows for distributed stress testing, in which a master system running the JMeter GUI is configured that controls the execution of test plans on registered slave machines. These slaves run a JMeter server, which takes commands from the GUI and sends requests to the target system(s). In both of our testbeds (see Figure 5.7) we make use of this distributed testing feature. The advantage of this feature is the capability to replicate a test across many computers and simulate a larger load on the server.

JMeter supports neither MQTT nor GeoMQTT innately. But with its plug-in system, a mechanism to integrate extensions and, ultimately, support also other protocols is

[2] https://jmeter.apache.org/

provided by the software. In that way, MQTT can be supported by multiple available plug-ins. Based on the MQTT plug-in written by the GitHub user "winglet" [3], we developed a plug-in for GeoMQTT. This involved replacing the used MQTT client with our GeoMQTT Java client and implementing appropriate advanced functionalities in the GUI. The latter one enables the user to specify spatiotemporal components of messages in the publisher sampler, respectively spatiotemporal filters in the subscriber sampler of a test plan. This way, the plug-in is prepared to publish and subscribe to GeoEvents expressed by `GEOPUBLISH` messages in GeoMQTT, so that we can perform the desired load test on the broker.

Table 5.1: *Hardware specifications in testbeds*

	Master	Slave	GeoMQTT broker
Processor	Intel i7-4770	Intel Xeon E5-2603 v3	Intel Xeon E5-2603 v3
CPUs	8	1	1
Clock Speed (GHz)	3.4	1.6	1.6
RAM (GB)	16	2	2
Ethernet (Mb/s)	1000	1000	1000

Source: Author's illustration

The JMeter master system runs with Linux Mint 18.2 ("Sonya") and has the hardware specifications given in Table 5.1. The two slave systems have equal hardware specifications. The OS Debian 8.10 ("jessie") is installed. Both, master and slave systems, use Apache JMeter Version 5.0 (r1840935). The virtual machines for the JMeter slaves, the GeoMQTT brokers as well as the HAProxy in testbed 2 run on the same physical server machine, so the network speed is negligible.

SETUP OF THE BROKER

The GeoMQTT broker is compiled as a runnable jar and deployed as a Docker[4] container on the machines. In testbed 1, the Docker container is deployed on a single machine running Debian 8 ("jessie") with the hardware specifications given in

[3] https://github.com/zerogvt/mqttws-jmeter

[4] https://www.docker.com/get-started

Table 5.1. The same specifications hold for the broker machines in testbed 2. Here the load is additionally balanced by a **HAProxy**[5]. HAProxy is a TCP load balancer and proxy server, which spreads incoming requests across multiple endpoints. It is often used when too many concurrent connections over-saturate the capability of a single server. In our testbed, it is deployed on a dedicated machine and configured with the two given GeoMQTT broker instances as endpoints. Instead of connecting directly to a broker machine, GeoMQTT clients connect to the proxy which distributes the incoming `CONNECT` requests to the GeoMQTT brokers in a round-robin fashion. Subsequently, the TCP connection is established between client and assigned broker. The HAProxy machine has the same hardware specifications and OS like the GeoMQTT broker machines.

This setup allows distributing the load on several instances of the GeoMQTT broker. However, without a communication mechanism between the brokers in the cluster, they might not receive all events and, thus, cannot notify every subscriber. Like in testbed 2 with two brokers (see Figure 5.7)(b), if a subscriber is connected to the left broker, incoming events at the right broker, which match the subscriptions of the subscriber, are not known and cannot be forwarded to the subscribers. Therefore, the events need to be exchanged between all brokers in the broker cluster. For this purpose, **Hazelcast**[6] is applied in testbed 2. Hazelcast is an open-source in-memory data grid written in Java. Data is distributed in the main memory across multiple machines. The architecture allows for horizontal scaling by adding new instances of the GeoMQTT broker. Also, it prevents the system to fail in case of a server error since the data is distributed.

The assumption holds that by scaling the broker horizontally and with the help of hazelcast and HAProxy, the load, the throughput of messages and the number of subscribers can be increased. Therefore, the GeoMQTT broker implements also the Hazelcast interface to be able to establish a broker cluster.

5.3.2 PERFORMANCE TEST PLANS

We aim for different performance tests to assess the efficiency and the scalability of the broker. These tests are mainly performed on GeoMQTT messages, but the broker's capabilities of processing ordinary MQTT subscriptions and events are also evaluated and serve as a reference for the advanced filtering mechanisms. We expect that both introduced filters in GeoMQTT, temporal as well as spatial filter, significantly increase the processing load in the broker and, thus, decrease the possible message throughput. Also additional features like coordinate transformation might affect throughput and latency. However, we also like to assess if the horizontal

[5]http://www.haproxy.org/

[6]https://hazelcast.org/

scaling of the broker in testbed 2 can counterbalance this decrease in message throughput. For these purposes, we construct the following test plans:

1. PubQoS1: MQTT `PUBLISH` message with QoS 1, no subscribers (reference)
2. GeoPubQoS1: GeoMQTT `GEOPUBLISH` message with QoS 1, no geo subscribers
3. PubSub: MQTT `PUBLISH` message with QoS 1, single subscriber (reference)
4. GeoPubGeoSub: GeoMQTT `GEOPUBLISH` message with QoS 1, single geo subscriber (no transformation of coordinates)
5. $\text{GeoPubGeoSub}_{\text{multisub}}$: GeoMQTT `GEOPUBLISH` message with QoS 1, multiple geo subscribers (no transformation of coordinates)
6. $\text{GeoPubGeoSub}_{\text{transform}}$: GeoMQTT `GEOPUBLISH` message with QoS 1, single geo subscriber (transformation of coordinates)
7. $\text{GeoPubGeoSub}_{\text{multisub,transform}}$: GeoMQTT `GEOPUBLISH` message with QoS 1, multiple geo subscriber (transformation of coordinates)
8. $\text{GeoPubGeoSub}_{\text{multisub,transform,scale}}$: GeoMQTT `GEOPUBLISH` message with QoS 1, multiple geo subscriber (transformation of coordinates); horizontal scaling of broker

The first seven test plans are run and evaluated on testbed 1 to assess the capabilities of a single GeoMQTT broker instance. To determine the deviations to MQTT, test plan 1 and 3 are conducted. The last test plan assesses the scalability of the broker and is therefore driven on testbed 2.

Metrics and Parameters

For all test plans, the **message throughput** is varied, firstly by modifying the throttle parameter in the clients, which indicates the time lag between two sent messages. Empirically tested, this parameter should not (and is not) decreased below 50 ms. So, a single client can increase the message throughput to 20 msg/s. Secondly, to raise the message throughput further, the number of clients can be increased using distributed clients with the JMeter framework. Every `PUBLISH` message is send with QoS level 1 and answered by the broker with a `PUBACK` to ensure the receipt of the message (see Section 4.2.3.1).

In test plans involving subscribers it holds that every subscriber is subscribed to all messages send by every publisher. In the best case, this means that each subscriber receives every `PUBLISH` message ever send during the test. The subscribers are also running on the JMeter slave machines in parallel to the clients, which send the events. The subscriptions itself are annotated with QoS level 0.

Every test plan is executed for roughly two minutes. This way we can measure the performance under steady load. For measuring the efficiency of the broker, we will observe the following metrics during each test plan if applicable.

- *Publisher latency*: Difference in time between sending `PUBLISH` message and receiving the `ACK` for the message.
- *Subscriber latency*: Difference in time between sending `PUBLISH` message by a client and reception of the message at the subscriber (aka. round-trip latency).
- *Error rate*: Portion of messages which are not received by the subscriber but received by the broker. (downstream message loss)

Timestamps are integrated into the messages, so that the latencies can be computed at arrival time. The latencies are then averages for each test. Since the publishers and subscribers might run on different machines, the latencies are only considered, if the message is sent and received by the same machine. Otherwise deviations in the clocks of the machines would result in incorrect latencies due to issues in time synchronization in millisecond. Further during some tests, the CPU usage is measured. This way, we can determine why and when downstream message loss occurs.

TEST MESSAGES

The GeoEvent that are encoded in `GEOPUBLISH` messages in GeoMQTT include a topic name, a time stamp, a geometry and a payload. The topic name consists of three hierarchy levels which are partly machine specific (see Topic name 5.16). The `<machineName>` and `<threadNumber>` are replaced by the clients with the machine and thread specific name. Including this, we can compute the latencies correctly like described before. The topic name in test plans using MQTT messages are likewise.

`<machineName>/<threadNumber>/temperature` (5.16)

The temporal component is set to the current machine's time (time of sending) with millisecond resolution. The geometry is set to one of the points specified in Listing 5.5. Usually the point in line 1 is used, however for test involving coordinate transformation, we use the geometry in line 2.

```
POINT (0.0 0.0)
POINT (6.067672129371917 50.779453271121561)
```

Listing 5.5: *Geometries in the performance tests*

The payload of each GeoMQTT `GEOPUBLISH` message is set to the string "7℃". In MQTT messages, the payload is instead filled with the current time of the machine with millisecond resolution, since otherwise it would not be included in the message and determining the different latencies would not be possible.

TEST SUBSCRIPTIONS

For the subscriptions and the GeoSubscriptions we use in the test plans involving subscribers, topic filter 5.17 is applied. With the first two hierarchy levels set to the wildcard "+", we ensured that every message can be received by every subscriber.

$$\texttt{+/+/temperature} \tag{5.17}$$

While the temporal filter is turned off by leaving it blank and, thus, is evaluated to "true" for every message, the spatial filter is set to one of the following given in Listing 5.6. The filter in line 1 is normally used, but in test plans involving coordinate transformation the filter in line 2 is applied.

```
1 EQUALS; POINT(0.0 0.0)
2 COVERS; SRID=5652;POLYGON ((32293268.2479151 5629435.74512573,32293286.7656027
      5629439.03272728, ... ,32293268.2479151 5629435.74512573))
```

Listing 5.6: *Spatial filters for the test plans*

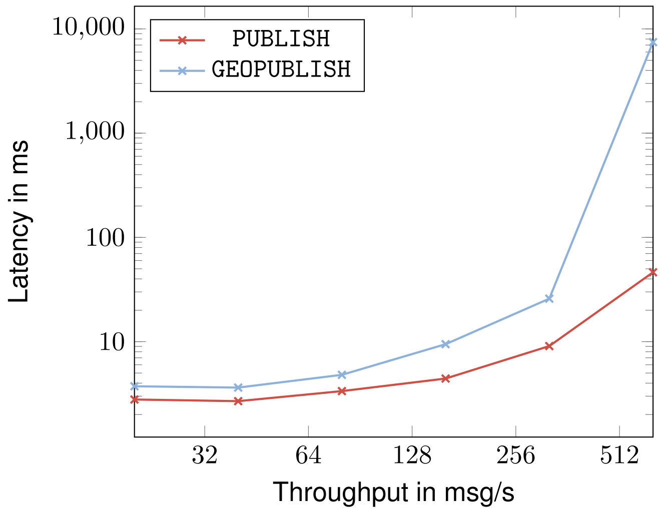
The geometry of the second spatial filter describes the building of the civil engineering faculty at RWTH Aachen University in CRS "ETRS89 / UTM zone 32N". It is constructed in a way that the geometries of the GeoEvents used in the corresponding test plans evaluates to true for each subscriber. The geometries of the GeoEvents are then first transformed to the GeoSubscription's CRS before sending them to the subscriber.

5.3.3 PERFORMANCE TEST RESULTS

The results of the test plans constructed in the last section are shown and analyzed in the following parts. For simplified comparison, the test plans are grouped if they match thematically.

5.3.3.1 PubQoS1 & GeoPubQoS1

For the PubQoS1 and the GeoPubQoS1 test plans, testbed 1 is used. The aim of the test plans is to measure the publisher latency if the throughput in messages per second (msg/s) is increased on a single broker. The messages are assigned a QoS level 1, so that the broker must acknowledge their receipt. In general, we expect a higher latency at a higher throughput. First, the test plan is performed by using MQTT `PUBLISH` messages and, afterwards, GeoMQTT `GEOPUBLISH` packets are sent. Since the broker does not have to check for and validate against subscriptions, the test plan solely focuses on measuring load while parsing and acknowledging messages. Figure 5.8 shows the result for both test plans.



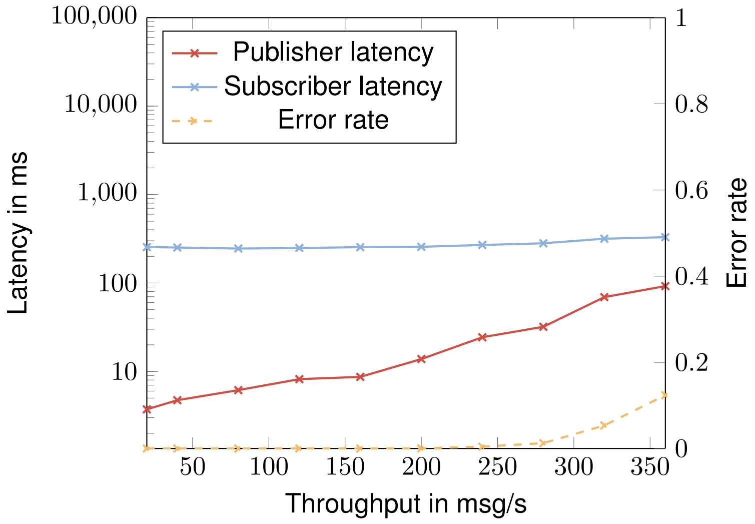
Source: Author's illustration

Figure 5.8: *Latency of `(GEO)PUBLISH` message with QoS 1*

The graph illustrates the publisher latencies for `PUBLISH` and `GEOPUBLISH` messages on a single broker in testbed 1, while the throughput is modified. Clearly, the base MQTT packet performs better than the `GEOPUBLISH` message in GeoMQTT. For a lower throughput (e.g. 20-40 msg/s) the latencies range around 2.7 ms, respectively 3.7 ms for GeoMQTT and the broker is nowhere near an over-saturation. So basically, from the difference between the two message types, we can conclude that the parsing of the additional temporal and spatial information takes roughly an additional millisecond for the specific message throughput. Subsequently, the results show that the higher the message throughput, the more the latencies increase. With a throughput of around 640 msg/s, the latency for GeoMQTT events rises to around 7.5 s, which is a non-acceptable state for the system. By contrast with 46.26 ms, the latency using MQTT `PUBLISH` is still acceptable.

5.3.3.2 PubSub & GeoPubGeoSub

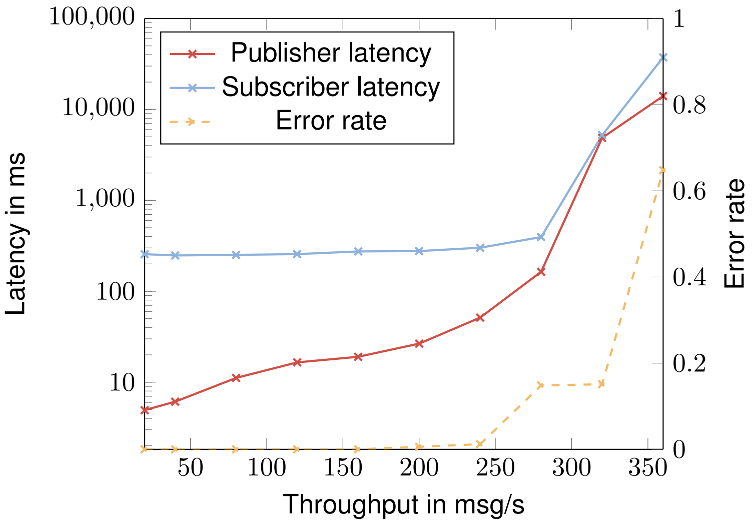
Whereas the first two test plans exclude the subscriptions of message, we want to assess the latency of a single subscriber in the following. Additionally, the publisher latencies are also recorded. Figure 5.9 shows the results for test plan PubSub, in which a MQTT client subscribes to each message with the topic filter `+/+/temperature`. The message throughput is stepwise increased from 20 msg/s to 360 msg/s. The publisher latencies rise with increasing throughput to roughly 92 ms at 360 msg/s. The subscriber latency (or round-trip latency) increases only slightly and is with 330 ms at 360 msg/s still acceptable. However, downstream message loss (0.04%) occurs already at a 200 msg/s throughput. At 360 msg/s, 12.375% of the messages are not received by the subscriber.



Source: Author's illustration

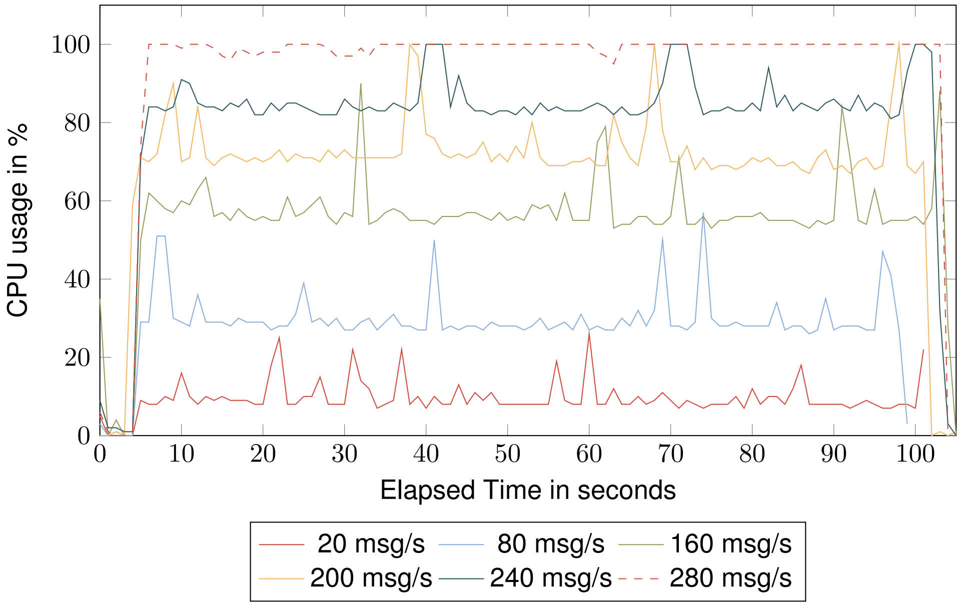
Figure 5.9: *Round-trip latency using one subscriber in MQTT*

Comparing these results to the version with `GEOPUBLISH` messages, the results differ significantly. The graphs in Figure 5.10 illustrate the test plan results for the GeoMQTT version. The latencies for publisher and subscriber are still acceptable at a throughput of 280 msg/s with 164.7 respectively 394.92 ms, but skyrocket with higher throughput to several seconds. Message loss also occurs already at 200 msg/s (0.585%) and significantly increases to 64.79% at 360 msg/s. When analyzing the CPU usage during these load tests (see Figure 5.11), we can clearly perceive that the higher throughput and the accompanied larger amount of filter evaluations lead to a heavier utilization of the CPU of the GeoMQTT broker machine. With 20 msg/s this usage is considered decent with around 10% and peaks at 22% during the test. If the throughput increases tenfold (200 msg/s), the CPU usage averages around 73%, but peak load can be at 100%. Even higher throughputs lead to higher incidences of full CPU usage. Hence, a rise in the error rate is the logical consequence.



Source: Author's illustration

Figure 5.10: *Round-trip latencies using one subscriber in GeoMQTT*



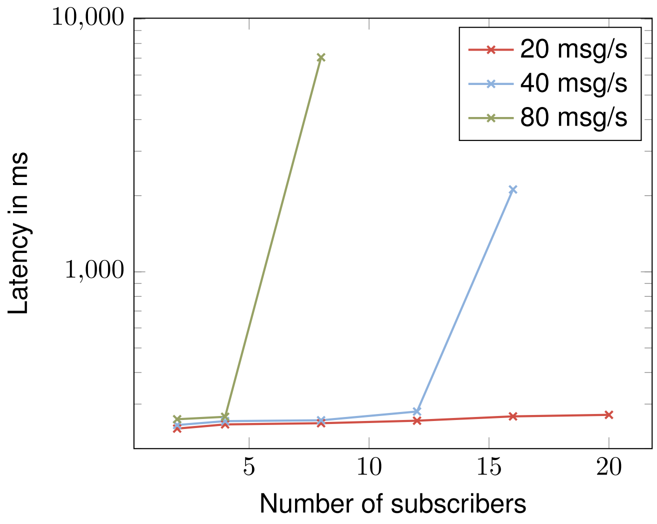
Source: Author's illustration

Figure 5.11: *CPU usage of the broker at different throughput levels during the GeoPubGeoSub test plan*

5.3.3.3 GeoPubGeoSub$_{\text{multisub}}$

Based on the results of the previous test plans, we focus on the number of subscribers in the next performance test. We modify the throughput of messages as well as the number of subscribers. Each subscriber receives every published `GEOPUBLISH` message. The filter evaluation is performed based on topic and spatial filter leaving the temporal filter turned off. However, the spatial filtering mechanism still does not involve coordinate transformation. For several parameter variations the results in Figure 5.12 are shown.

For a throughput of 20 msg/s, the broker can handle up to 20 subscribers easily at the same time. The subscriber latencies range from 240 to 272.87 ms and no downstream message loss occurs. For 12 subscribers, similar results can be obtained with a 40 msg/s throughput. However, using 16 subscribers and 40 msg/s, the subscriber latency increases to 2.12 s and message loss occurs (7.87%). Setting the throughput further to 80 msg/s, the broker reaches its limits already at 8 concurrent subscribers leading to a subscriber latency of 7.05 s and a downstream message loss of already 70.58%.



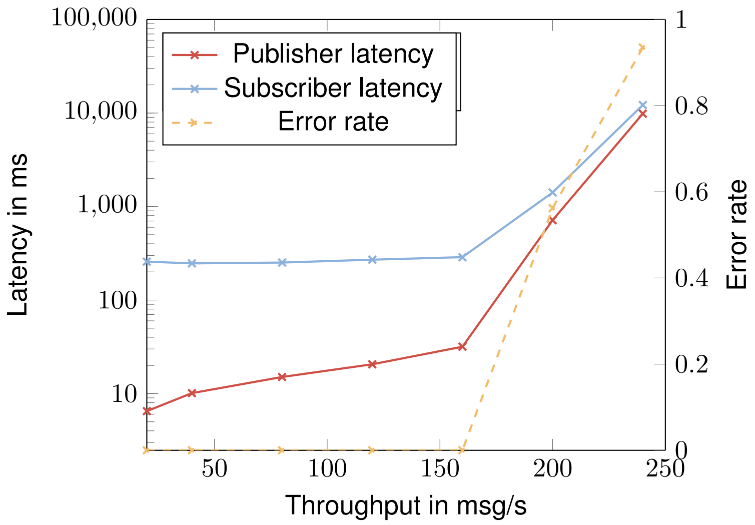
Source: Author's illustration

Figure 5.12: *Round-trip latencies for multiple subscribers and throughputs*

5.3.3.4 GeoPubGeoSub$_{\text{transform}}$ & GeoPubGeoSub$_{\text{multisub,transform}}$

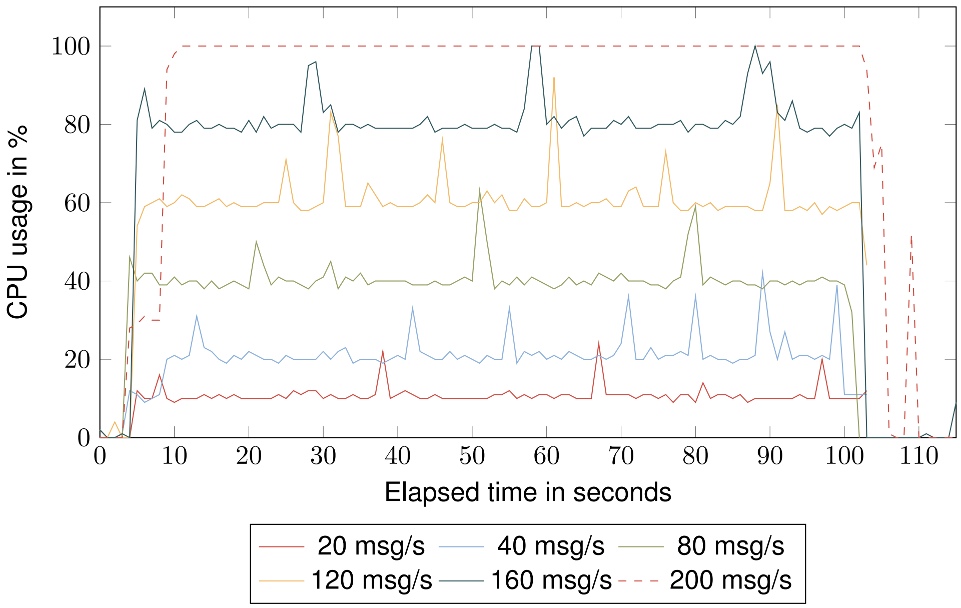
In the following, we repeat the previously conducted test plans that include GeoSubscriptions, but also add the processing step of coordinate transformation for every message the broker forwards it to the subscribers. First, the GeoPubGeoSub test plan is rerun, which involve a single GeoSubscription made by a client. Again the throughput is varied and the latencies as well as the CPU usage are recorded. The results for the latencies are plotted in Figure 5.13.

The figure shows that a throughput of 160 msg/s is unproblematic: the publisher latency is still low with 31.70 ms, the subscriber latency with 287.59 ms is as high as in the test without coordinate transformation and message loss occurs scarcely - the error rate is 0.03125%. However, if the throughput is increased to 200 msg/s or even further, the measured latencies and error rates soar. The system is not capable of processing and distributing the events any more. Compared to the performance test without coordinate transformation, the capabilities in terms of processing at a high throughput are reduced significantly. Also if we take a closer look at CPU load during the test plans (see Figure 5.14), this becomes evident. A throughput level of 160 msg/s leads to a CPU usage of around 80% with peaks to 100%, but a load of 200 msg/s is not processable by the single GeoMQTT broker in testbed 1.



Source: Author's illustration

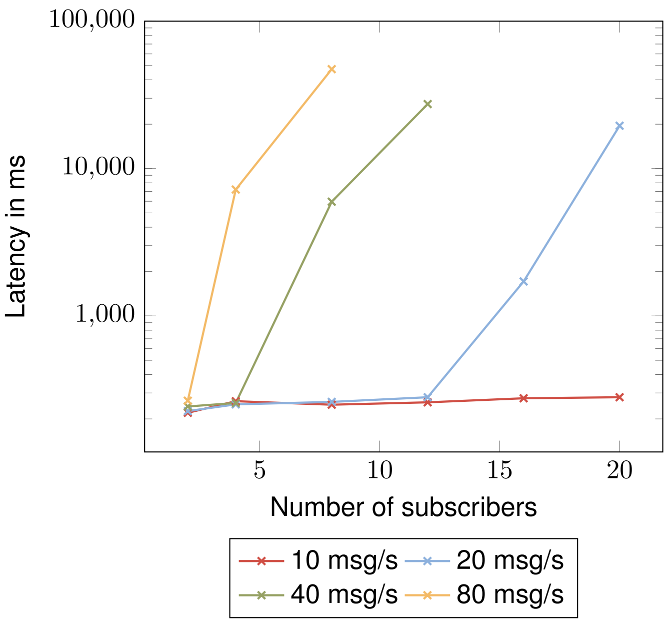
Figure 5.13: *Round-trip latency using one subscriber at different throughput levels with coordinate transformation*



Source: Author's illustration

Figure 5.14: *CPU usage of the broker at different throughput levels with single subscriber and coordinate transformation*

When rerunning the GeoPubGeoSub$_{\text{multisub}}$ test plans with coordinate transformation and modifying the number of subscribers, the number of transformations can be increased further. We measure the subscriber latencies illustrated in Figure 5.15 for several parameter variations. In comparison to the first test with multiple subscribers, the subscriber latencies for 20 msg/s and 20 subscribers are clearly above 15 seconds and message loss occurs (error rate: 62.85%). Already 16 concurrent subscribers lead to a subscriber latency of 1.714 s. However, if we reduce the throughput to 10 msg/s, the test plan is handled smoothly. For throughputs of 40 or even 80 msg/s, only a small number of subscribers can be served in a reasonable amount of time. Furthermore, the error rates are already quite high having small amounts of subscribers. For instance, with a throughput of 80 msg/s and 4 subscribers, the downstream message loss is already at an error rate of 77.98%.



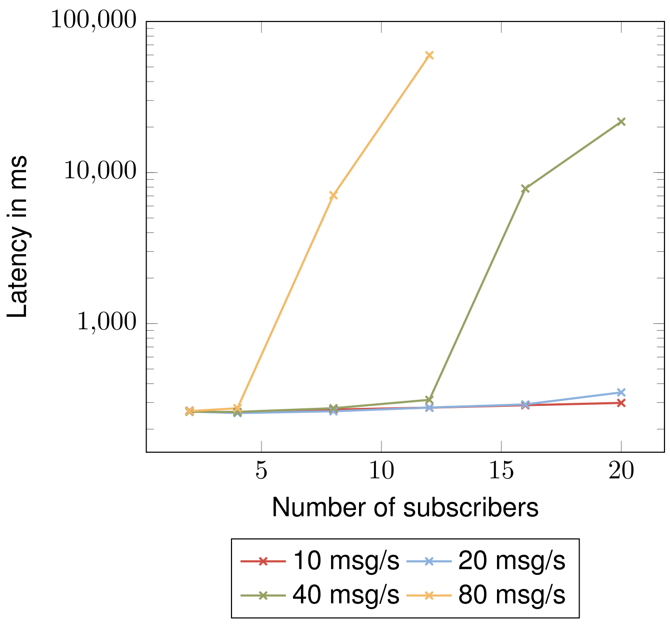
Source: Author's illustration

Figure 5.15: *Round-trip latencies for multiple subscribers and coordinate transformation*

5.3.3.5 GeoPubGeoSub$_{\text{multisub,transform,scale}}$

To evaluate the scalability of the GeoMQTT broker, the test plan GeoPubGeoSub$_{\text{multisub,transform}}$ is also driven on testbed 2. In this testbed, two

GeoMQTT brokers form a broker cluster, in which `PUBLISH` and `GEOPUBLISH` messages are exchanged between each other using hazelcast. The connections of the GeoMQTT clients are divided in a round-robin fashion by a HAProxy instance (see Section 5.3.1). This setup might divide the processing load induced by the filtering mechanisms to two brokers, if the subscribers are equally distributed among them. In the following test plans, we can control this distribution process by first connecting the subscribers to the cluster. Then after a time delay, the publishers also connect to the broker cluster. With equally distributed subscribers, we expect a balanced distribution of processing load and the best performance results we can obtain from the testbed and for the tests. The previous results show that the bottle neck of the broker is the processing of the GeoSubscriptions. We expect that the broker cluster can improve the performance and, thus, provide a suitable scalability mechanism.



Source: Author's illustration

Figure 5.16: *Round-trip latencies for multiple subscribers and coordinate transformation in testbed 2*

Similarly to the latter tests using a single broker instance, the throughput of messages and the number of subscribers are varied in the $\text{GeoPubGeoSub}_{\text{multisub,transform,scale}}$ test plan. Again, each subscriber is subscribed to every message, so that in the best case, it receives all messages. Also the GeoSubscriptions use a different CRS than the `GEOPUBLISH` messages entailing a transformation of coordinates for every message and every GeoSubscription. For the test plan and parameter variations we collected the results plotted in Figure 5.16.

Compared to the same tests on a single GeoMQTT broker (see Figure 5.15), the results of the broker cluster clearly prove the superiority in terms of performance. While on a single broker the tests with 20 msg/s and 16, respectively 20 subscribers, lead to high subscriber latencies and error rates, the same tests on the GeoMQTT broker clusters operate smoothly with almost no impact in comparison to less subscribers or a lower throughput (e.g. 10 msg/s). For a throughput of 40 msg/s, 12 subscribers are still trouble-free, but with 16 subscribers the latency increases and downstream message loss occurs. The error rate is already at 19.736%. Similar results are obtained for 80 msg/s and 8 subscribers, so that the capacity limit of the broker cluster is reached.

Overall however like shown, the performance in terms of message throughput can be improved by utilizing a GeoMQTT broker cluster. Thus, scalability of the protocol can be ensured by appropriate mechanisms, here in form of a broker cluster and load distribution.

5.4 Discussion on Evaluation Results

The evaluation objectives in Section 5.1 were raised to assess the properties of GeoMQTT and its suitability regarding the requirements for a Geospatial IoT (see Chapter 3). Especially, the focus has been on questions about *expressiveness*, *message size*, *efficiency* and *scalability*. Here, we like to discuss the findings and assess the capabilities of the GeoMQTT protocol and the implementation.

GeoEvents and GeoSubscriptions like defined in Chapter 3 should be representable in the Geospatial IoT architecture. With the evaluation based on the scenarios for stationary and mobile devices, we showed that at least in these use cases, the **expressiveness** of the messages in GeoMQTT is granted. However, we can think about other scenarios, in which for instance the expressiveness of the spatial component in GeoMQTT in its current version is insufficient. E.g. in Geospatial IoT applications, which involve the orientation of GeoEvents, the currently used geometry representation is unable to cover these aspects. The same holds for the spatial filter in GeoSubscriptions. A filter for things facing in a certain direction cannot be modeled with the current version of the protocol.

The message size comparisons illustrate, that the **lightweight** of the introduced messages in GeoMQTT can be positively assessed. In comparison to other encoding standards, the sizes for the modeled GeoEvents and GeoSubscriptions with its corresponding packet types `GEOPUBLISH` and `GEOSUBSCRIBE` outperform their competitors. Thereby, the encoding of the geometries in the spatial component determines mainly the deviation in size. It should be mentioned that the comparison evaluates two different types of encodings: in GeoMQTT the messages are packet encodings, while the other standards such as EML feature documents encodings. For the latter ones, embedding in an appropriate transfer protocol is still needed. Nevertheless, the modeling in GeoMQTT is still smaller, so that we can argue for the lightweight of the protocol.

Based on the performance tests we run on the GeoMQTT broker, the **efficiency** of the proposed implementation could be evaluated. In comparison to plain MQTT and its subscribing mechanism, the broker demands for more processing power when the extended filtering mechanisms of GeoMQTT are applied. The latencies are in general increasing with the GeoMQTT extension. Additionally depending on the amount of GeoSubscriptions and messages, the feasible throughput of messages is restrained. When applying coordinate transformations due to alternative CRSs in the GeoSubscriptions, the performance of the GeoMQTT broker in terms of message throughput is further restricted. Coordinate transformations to another CRS need additional processing resources. The overall assessment of the efficiency is challenging, since it depends heavily on the application, particularly the number of publishers, of subscribers and the frequency of GeoEvents send through the system. In the test plans involving a single broker instance, a lower message throughput of 20 msg/s and up to 12 subscribers with coordinate transformation yields in an acceptable latency. But increasing message throughput and/or number of subscribers result in higher latencies and message loss. Contrary, while decreasing the number of subscribers, the message throughput can be incremented. With a single subscriber and coordinate transformation, a message throughput of 160 msg/s can be achieved without problems. Like said, the message throughput depends on the specific application (e.g. sampling rate of sensors). For the scenarios introduced in Section 5.2, the achieved efficiency with one broker instance is sufficient.

Equally important to the efficiency of the GeoMQTT broker is its **scalability**. If a scalable solution can be applied, the performance capabilities of the broker can be expanded. Therefore, we deployed a cluster with two GeoMQTT brokers in another testbed applying load distribution. The results of the test plans we reran on the cluster show that the performance in terms of feasible throughput can be increased by replicating the broker instances. The limitation of the single broker instance in testbed 1 can be overcome by scaling it horizontally. So our implemented GeoMQTT extension meets the scalability requirement of a GeoEvent-based architecture for a Geospatial IoT. However, further questions arise from this observation such as the question about the scaling factor. Thus, further investigations and evaluations involving a cluster with multiple brokers should be performed in future studies.

CHAPTER 6

GEOMQTT INFORMATION & SERVICES

The third building block of IoT architectures consists of an information and a service layer (see Section 2.5). In this chapter, we present the integration of the GeoMQTT protocol in this layer for different types of software and services. With the integration in a desktop GIS, we bridge the mechanisms of our Geospatial IoT architecture to contemporary GIS technology (Section 6.1). Further, we describe a RESTful access point to the architecture in Section 6.2, as well as bridge the GeoMQTT protocol into the service-oriented Sensor Web architecture implemented by the OGC SWE standards (Section 6.3). Finally, we introduce an extension of the WPS interface that enables processes to receive and run stream processing algorithms on GeoEvents published by e.g. GeoMQTT (see Section 6.4). Further, this extension is also used to invoke processes on DSP systems.

6.1 GEOMQTT PLUG-IN FOR QGIS

Traditionally, desktop GISs are expert software systems to support capturing, manipulating, analyzing, managing and visualizing spatial annotated or geographic data. The evolution of GIS coincides with the development of computers and the Internet technology as we have seen in the introduction of this thesis. The traditional architecture of desktop GIS software manage and analyze spatial data in a standalone environment like a desktop computer, while service-oriented GIS relies on distributed web services (Yue et al., 2014). Consequently, GIS is supposed to play also an important role in the Geospatial IoT.

A literature review reveals conceivable reasons to interconnect IoT with GIS technologies. For instance, Gubbi et al. (2013) state that data collected within IoT is often geo-related and sparsely distributed. They favor a framework based on Internet GIS to cope with challenges of visualizing the data. According to Kamilaris & Ostermann (2018), who conducted a review of IoT research projects and its relation to geospatial analysis, the tools of desktop GIS offer large potential for understanding, modeling and visualizing natural or artificial ecosystems while using IoT as a sensing infrastructure. Hence, we like to integrate the Geospatial IoT and desktop GISs to investigate the interaction of the two technologies. As a proof-of-concept, we used the open source QGIS software and its plug-in system to implement a GeoMQTT plug-in. With its help, real-time data from corresponding clients can be received, visualized and analyzed by the software.

6.1.1 QGIS AND ITS PLUG-IN SYSTEM

QGIS is a free and open source GIS software. It is composed of two programs: QGIS Desktop and QGIS Browser. While the browser is used to manage and preview data, QGIS Desktop is a traditional desktop GIS with the corresponding basic functionalities to manage, display, analyze and style geo data. The volunteer-led development of QGIS was started by Gary Sherman in 2002. In 2007 it was incubated with the Open Source Geospatial Foundation (OSGeo). Version 1.0 was finally released in 2009 (Menke et al., 2015). At the time of writing, version 3.6 (Noosa) has been released recently and can be downloaded from the project's website[1].

The QGIS Desktop GUI consists of a menu bar, toolbars, panels and a map display (Menke et al., 2015) and, thus, is comparable to similar GIS software such as ESRI's ArcGIS or Intergraph's Geomedia. Figure 6.1 shows a screenshot of the utilized LTR version 2.18.x (Las Palmas). The main view is the map display, which shows the styled data. Most capabilities of QGIS Desktop can be accessed by the menu bar and the floating or docked toolbars. Likewise, floating or docked panels provide a variety of functionalities. For instance, in the Layers panel, the user may access or rearrange the different layers.

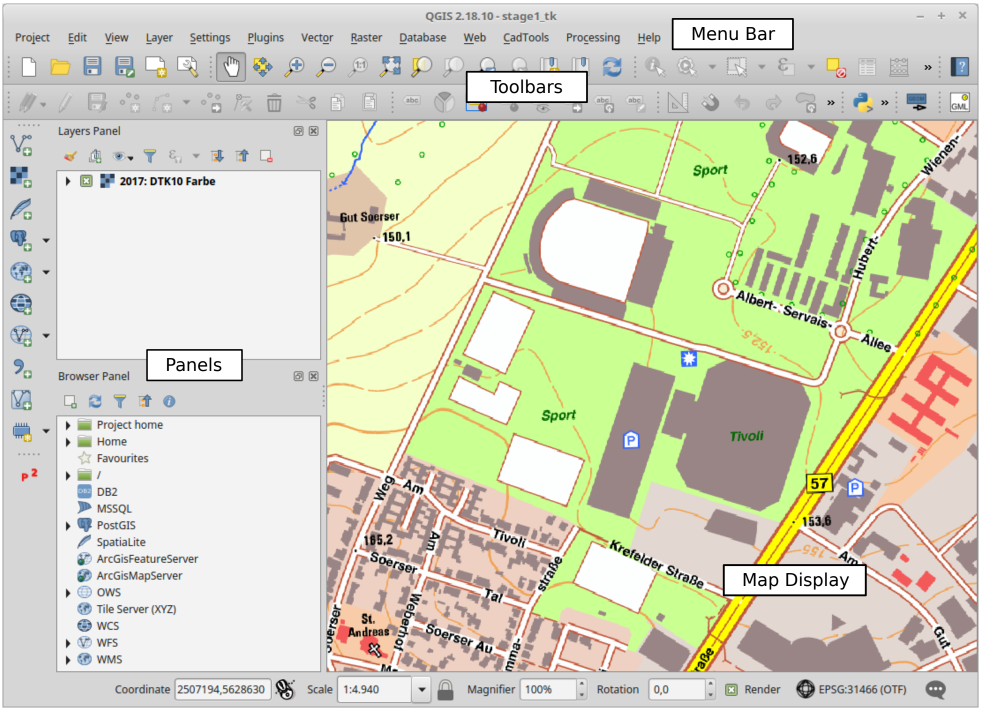
The set of basic functionalities can be enhanced using plug-ins. Because QGIS is written in C++, this was originally the only language to develop plug-ins. However, since version 0.9.x, QGIS supports scripting using Python. This includes a Python console to run scripts based on objects and methods in the QGIS API, but also the possibility to write applications and plug-ins in Python. Since the QGIS code depends on Qt libraries, PyQt4[2] is used in the Python bindings. Employing Python, it became much easier to develop and to distribute plug-ins. Additional information about the plug-in system and a guide for developing plug-ins can be e.g. found in QGIS Project (2018).

Since QGIS introduced Python support, many plug-ins with different functionalities have been implemented or are still developed (e.g. Braden, 2015; Duarte et al., 2018). A QGIS plug-in[3] which integrates QGIS with MQTT is also available but is not under active development and does not run under the current version.

[1]https://qgis.org

[2]https://pypi.org/project/PyQt4/

[3]https://github.com/nzfarmer1/telemetrylayer



Source: Author's illustration

Figure 6.1: *QGIS Desktop Version 2.18.x LTR (Las Palmas)*

6.1.2 GeoMQTT in QGIS Desktop

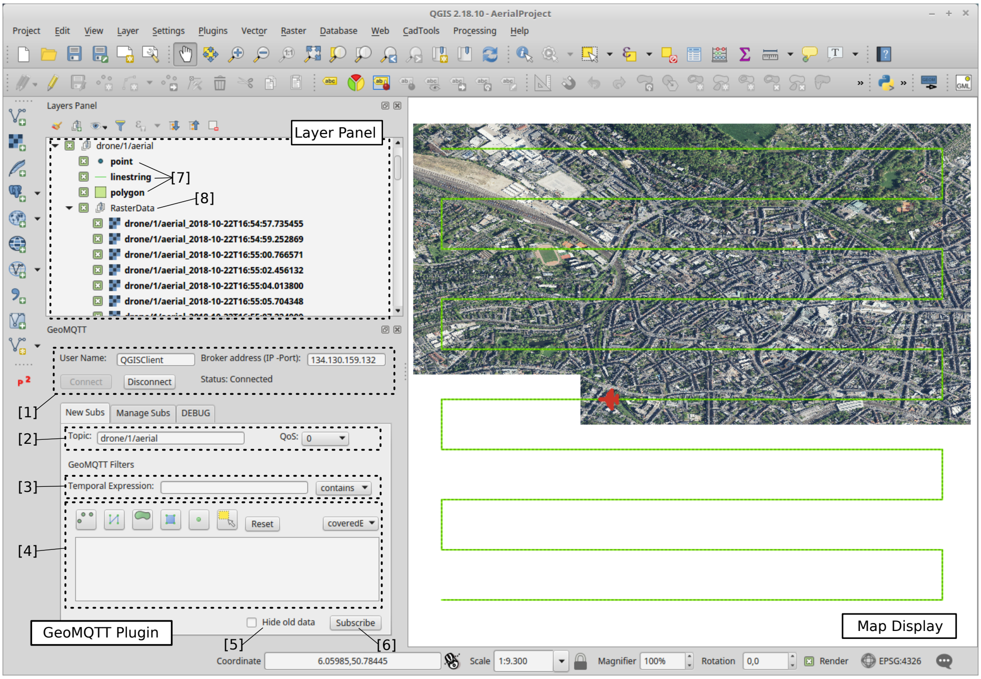
Integrating Geospatial IoT data sources with a desktop GIS such as QGIS by using the designated protocols is required to receive, manage, analyze and visualize IoT data in real-time. Thus, the project to integrate GeoMQTT in QGIS can be easily realized by utilizing the QGIS plug-in system and the already developed GeoMQTT Python client (see Section 4.4.2). The requirements for the plug-in are the following:

1. The user can connect to one or multiple arbitrary GeoMQTT brokers with an arbitrary user name.
2. The user may subscribe to GeoEvents by using the GeoSubscription mechanism of GeoMQTT. This includes
   (a) specifying a topic filter according to the MQTT syntax,
   (b) specifying a temporal expression and a temporal relation (temporal filter),

   (c) specifying a geometry and a spatial relation according (spatial filter),
   (d) specifying a QoS level.

3. The user may unsubscribe from a GeoSubscription by using the `UNGEOSUBSCRIBE` message and the corresponding topic filter.
4. Incoming GeoEvents are added to the layer tree and presented to the user in the map display of QGIS.
5. All events are captured in corresponding data types and the user has access to all received data.

The plug-in was designed as a floating or dockable panel for QGIS in the Qt 4 Designer software[4]. Figure 6.2 shows the GUI of the GeoMQTT plug-in attached to the bottom left corner of the QGIS application.


QGIS 2.18.10 - AerialProject
Project Edit View Layer Settings Plugins Vector Raster Database Web CadTools Processing Help
Layers Panel
Layer Panel
[7]
[8]
GeoMQTT
[1]
[2]
[3]
[4]
[5]
[6]
New Subs Manage Subs DEBUG
GeoMQTT Filters
Connect Disconnect
Reset
Hide old data Subscribe
GeoMQTT Plugin
Map Display

- drone/1/aerial
  - point
  - linestring
  - polygon
  - RasterData
    - drone/1/aerial_2018-10-22T16:54:57.735455
    - drone/1/aerial_2018-10-22T16:54:59.252869
    - drone/1/aerial_2018-10-22T16:55:00.766571
    - drone/1/aerial_2018-10-22T16:55:02.456132
    - drone/1/aerial_2018-10-22T16:55:04.013800
    - drone/1/aerial_2018-10-22T16:55:05.704348

User Name: QGISClient Broker address (IP -Port): 134.130.159.132
Status: Connected
Topic: drone/1/aerial QoS: 0
Temporal Expression: contains
coveredE
Coordinate 6.05985,50.78445 Scale 1:9.300 Magnifier 100% Rotation 0,0 Render EPSG:4326

Source: Author's illustration

Figure 6.2: *GeoMQTT plug-in attached to the QGIS Desktop window*

[4]`http://doc.qt.io/archives/qt-4.8/designer-manual.html`

In the GeoMQTT Plug-in panel, the user can connect to a broker by specifying the address and his user name [1]. After the connection is successfully established, GeoSubscriptions can be registered by the QGIS client: the user may specify a topic filter [2], a temporal filter [3] and a spatial filter [4]. The geometry for the spatial filter can be given by a string or with the help of the map display. The geometry of existing features can also be selected. With the subscribe button [6], the user fulfills the subscription. A successful GeoSubscription leads to a newly added layer group in the layer panel. The name of this layer group corresponds to the chosen topic filter. The group consists of three layers for vector data (point, linestring and polygon) [7] and a layer group for raster data [8]. If a `GEOPUBLISH` for a GeoSubscription arrives, the data is added to that layer group according to the geometry type of the message or the payload of the message in case of raster data. The layers consist of historical and new arriving data. With the hide-old-data option [5], the user can additionally indicate that only new arriving data should be visible in the map display.

6.1.3 Use Cases and Future Use

Various use cases can be implemented with the GeoMQTT extension. Figure 6.2 illustrates a use case, in which raster data are streamed into QGIS: In a first step, an Unmanned Aerial Vehicle (UAV) takes aerial photographs of the surface of the earth. The photographs are streamed to a ground station, orthorectified by a computing unit and the resulting orthophotos are published with GeoMQTT. The QGIS user can subscribe directly to the orthophotos, which are visualized on arrival in the layer panel.

Also vector data can be updated in real-time by using the GeoMQTT plug-in. For instance, in a vehicle tracking scenario such as fleet tracking, the most recent positions are published by the vehicles by using GeoMQTT. The positions can be streamed in real-time into QGIS using the plug-in. Subsequently, the user might run traffic analyses or trajectory mining tasks directly on this data. This vehicle tracking scenario shows that real-time data combined with the analysis capabilities of QGIS would potentially add tremendous value to GIS systems. In fact with the WPS extension presented in Section 6.4, a mechanism is created based on a service to call stream processing capabilities on a server. Integrating this service into the GeoMQTT plug-in for QGIS would add further distributed stream processing functions to the desktop GIS.

6.2 A RESTful Access Point to GeoMQTT

GeoMQTT like MQTT is a M2M protocol and, therefore, is tailored to requirements of machines. Users, on the other hand, need simpler methods to interact with machines. The standards of the Sensor Web (see Section 2.5.1) are one way to achieve this, but it only holds for sensor data. Having a more general access point to the GeoMQTT broker would be a huge benefit, especially for developers. Therefore, we conceptualized and implemented an architecture build upon the Representational State Transfer (REST) principle, which bridges GeoMQTT and REST. The developed architecture is presented in this section.

6.2.1 REST and the Geospatial IoT

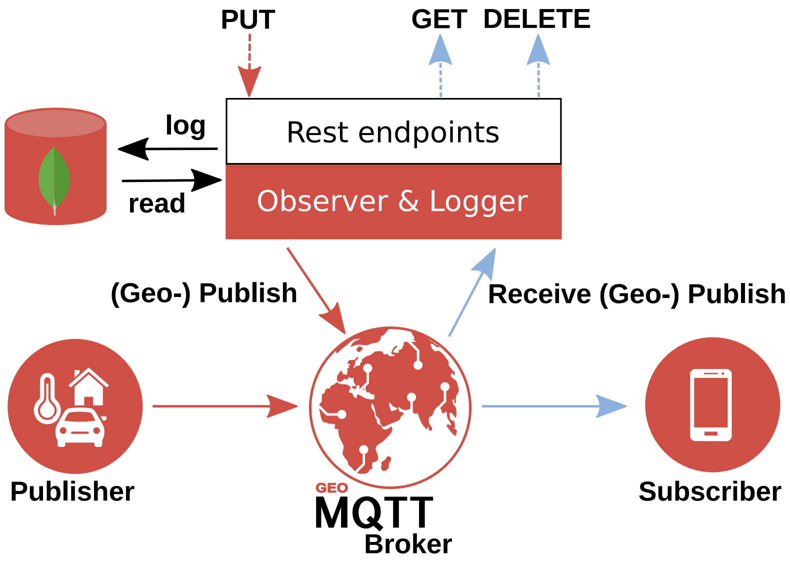
REST and RESTful web services were first introduced by Roy Fielding in his doctoral thesis (Fielding, 2000). It describes an architectural style for large-scale distributed hypermedia systems by providing a set of fundamental architectural constraints. The main concept in RESTful systems is the notion of resource which describes a logical object. This might be a physical object, abstract concepts such as collections of objects or dynamic and transient concepts such as server-side states or transactions (Guinard et al., 2010b). The architectural style described by REST is detached from technologies and implementations. However, RESTful web services use the HTTP methods GET, POST, PUT and DELETE for retrieving, creating, updating and, respectively, removing resources provided by a web server.

In some IoT projects RESTful architectures are used to access or update resources. For instance, in the PortoLivingLab historical sensor data can be requested via a RESTful API (Santos et al., 2018). In Isikdag & Pilouk (2016) the RESTful interface of a graph database is requested by IoT nodes to update a sensor resource. Guinard et al. (2010b) analyze the technologies behind REST architectures and propose how these can be applied to the WoT. They argue that RESTful architectures are the most effective solution for the WoT. Naik (2017) concludes in an IoT protocol comparison that HTTP-based RESTful clients and servers represent the most interoperable, since they solely need to support an HTTP stack for message exchange. Therefore, REST components such as RESTful web services have its rightful place in IoT architectures, although with its request/response mechanism, push-based real-time communication is not possible. To benefit from the extensive interoperability through a simple HTTP stack, establishing a bridge between GeoMQTT and REST based on HTTP seems useful.

6.2.2 Bridging GeoMQTT and REST

When adding a RESTful interface to GeoMQTT two different semantic models of communication are bridged, the publish/subscribe interaction scheme of MQTT and the request/response pattern of HTTP. According to Collina et al. (2012) or Koster (2013), it is useful to couple the REST-oriented web architecture and the real-time properties of MQTT to close the gap between machines and developers in the IoT. Collina et al. (2012) implement a so-called QEST broker to expose MQTT topics as REST resources and vice versa. For instance, the REST resource */topics/room/237/temperature* corresponds to the topic *room/237/temperature*. By requesting the resource with an HTTP GET, the response consists of the latest published value issued with the corresponding topic. Similarly, a HTTP PUT request at */topics/room/237/temperature* publishes the value of the request body with the corresponding topic.

We follow a similar approach in the implementation of our REST bridge. Unlike Collina et al. (2012), we do not integrate the REST interface directly in our GeoMQTT broker but use an observer client, which subscribes to all messages/events. Additionally, we use the bridge as a message logger, which does not solely store the latest value on a specific topic but all messages that are received. The REST-GeoMQTT bridge is shown in Figure 6.3.



Source: based on Herle et al. (2016a)

Figure 6.3: *REST-GeoMQTT Bridge*

The observer and logger client subscribes to all MQTT and GeoMQTT messages at the broker by using the multi-level wildcard # for topics and, respectively, a wildcard for the temporal and spatial filters. It logs the received `PUBLISH` and `GEOPUBLISH` messages in separate collections in a MongoDB[5] database. The bridge itself is

[5] `https://www.mongodb.com`

implemented in the Spring framework[6] and has two different REST endpoints: one for MQTT and one for GeoMQTT.

For MQTT, the REST resources are mapped to topics according to the approach used by Collina et al. (2012). For instance, the HTTP resource */publish/room/237/temperature* is mapped to the topic *room/237/temperature*. In a HTTP GET request, it is also possible to use the single-level wildcard *+* or URL-encoded multi-level wildcard *#* in topic filters. The bridge queries the MongoDB database for logged `PUBLISH` messages and responses with a list of messages in JSON format. Setting the optional URL parameter *size* to 1 allows the user to retrieve only the most recently published message that matches the topic filter. Accordingly, with the HTTP PUT request of a resource, the request body is published to the GeoMQTT broker by using the corresponding topic name. Wildcards, however, are not allowed in the resource since they are also prohibited in topic names in MQTT `PUBLISH` messages as well.

In GeoMQTT, the topic is handled similarly to the MQTT case - e.g. the resource */geopublish/temperature* is exposed as the topic *temperature*. The HTTP GET request has four optional parameters: *from*, *to*, *geometry* and *size* like before. *from* and *to* are used to define a time interval whilst geometry expects a geometry in WKT format. Like in the GeoMQTT protocol, all OGC Simple Feature Access geometries are supported extended by a `BBOX` and `BUFFER` type. The time interval and geometry are both evaluated with a `Covers` relation. If not specified the temporal filter and spatial filter are set to wildcards. The bridge requests the MongoDB with temporal, spatial and topic filter and returns a GeoJSON FeatureCollection of the logged `GEOPUBLISH` messages. Hereby, the spatial filter is evaluated with a `Covers` relation in respect to the geometries of the `GEOPUBLISH` messages stored in the database. The HTTP PUT request for the GeoMQTT endpoint expects two required parameters besides the corresponding wildcard-free topic name as the resource: the time parameter as an ISO8601 timestamp and the geometry parameter in WKT syntax. Like the MQTT case, the request body and the parameters form a `GEOPUBLISH` message that is sent to the GeoMQTT broker.

In addition, we implemented HTTP DELETE for the two endpoints to manage the database. It deletes the matching entities in the database and returns them as a JSON, or a GeoJSON document respectively. The request parameters are identical to the parameters in the HTTP GET requests except for the *size* parameter.

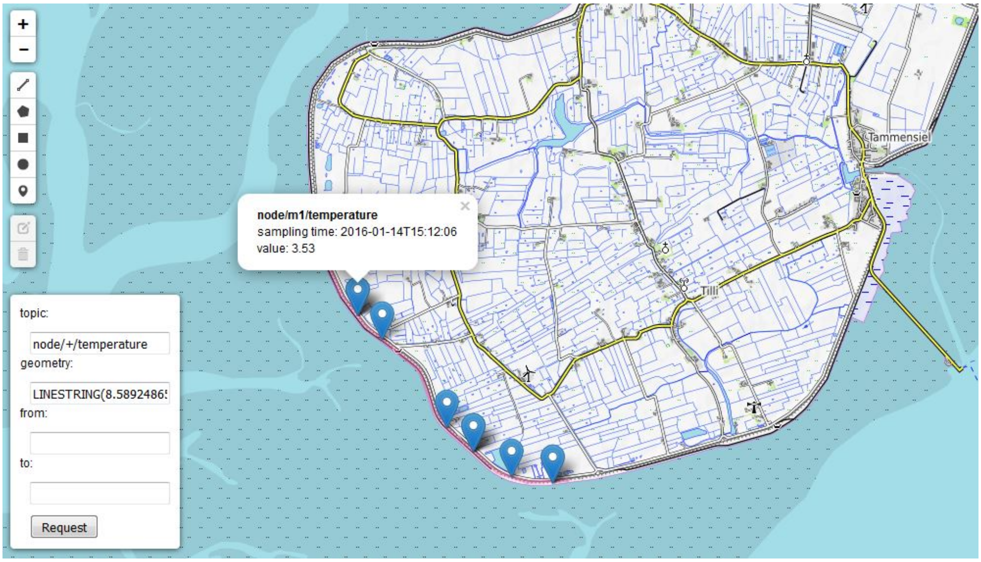
As mentioned, since HTTP uses the request/response mechanism, it cannot fully support a publish/subscribe interaction scheme. WebSockets could be one possible solution to solve this issue (Fette & Melnikov, 2011). In fact, we already implemented a GeoMQTT client with WebSockets. But Collina et al. (2012) argue that WebSockets

[6] `https://spring.io`

do not implement the concept of URI after opening the communication and, therefore, do not support the pure REST approach which involves exposing resources as topics in our solution. They enhance the implementation by a long polling approach for retrieving real-time updates in the browser without using WebSockets. However, this is not implemented in our approach so far.

6.2.3 REST-GEOMQTT BRIDGE WEB APPLICATION

The REST bridge is used in the EarlyDike (see Section 5.2.1.1) project to log and easily obtain events published by sensor nodes, which are deployed at dike lines to monitor the structure of sea dikes. We set up a web map application to request the GeoEvents and plot them in a map. Since the response type of the service is a FeatureCollection of events in GeoJSON format, it is quite effortless to plot the events, here in a leaflet[7] map container (see Figure 6.4). The corresponding REST request for the requested data in the application in Figure 6.4 is depicted in Listing 6.1.


node/m1/temperature
sampling time: 2016-01-14T15:12:06
value: 3.53

topic: node/+/temperature
geometry: LINESTRING(8.5892486
from:
to:

Source: based on Herle et al. (2016a)

Figure 6.4: *Web map application to request GeoMQTT events using the REST bridge*

[7] `https://leafletjs.com`

```
http://earlydike.de:8080/rest/geopublish/node/+/temperature?
geometry=LINESTRING(8.589248657226562 54.517893120052946,8.590707778930664, ...)
```

Listing 6.1: *HTTP GET request to retrieve events from REST-GeoMQTT bridge*

The REST endpoint here is */rest/geopublish*, the requested resource corresponds to all stored messages which matches the topic filter *node/+/temperature*, where the *+* wildcard is used to replace the id of the sensor node and, therefore, retrieve all measured temperatures of every available sensor node. Additionally, the stored messages are filtered by the specified geometry, a linestring, which represents the southwestern first order dike line of the German North Sea island Pellworm (purplish polyline in the map in Figure 6.4). The temporal filter, however, is not specified and, thus not applied in the request.

6.3 Bridging the SWE Standards

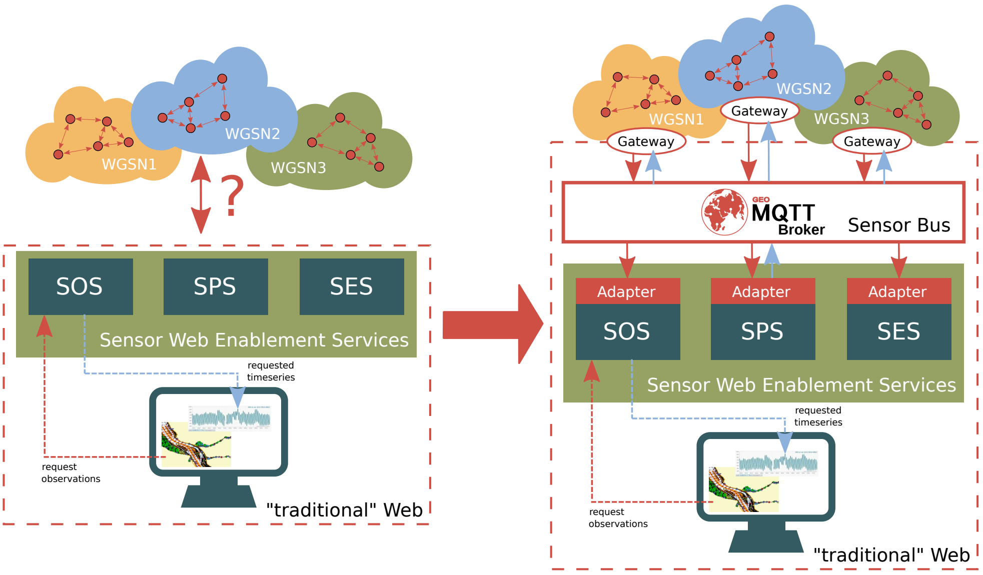
Besides the RESTful access point to GeoMQTT, integration into a SOA such as the service-oriented Sensor Web is desirable. By bridging these systems, observations, measurements and access to sensors can be offered to users or software agents as web services in a standardized way. Furthermore, it closes the interoperability gap between sensor networks and SWE services. Our solution is described in the following.

6.3.1 Interoperability Gap

The vision of the Sensor Web, which was introduced in Section 2.5.1, includes the idea of establishing a standard foundation for plug and play web-based sensor networks (Botts et al., 2007). Walter & Nash (2009) argue that it is not an easy task to connect sensors as data providers to SWE-service. Both, knowledge of the sensor network's format and of the SWE standards such as the SOS must be gathered to solve this task. Further, the high-level design of the SWE standards and the low-level protocols and formats of WGSNs lead to an interoperability gap. Even though the transactional profile of the SOS allows for storing live sensor data in its data base, it is hardly directly usable by sensor nodes. First, off-the-shelf WGSNs do not support the standards and, second, using the standards requires high amount of processing power. In environments with limited resources such as small sensor nodes encoding data in XML structures and using HTTP as a protocol is a huge disadvantage in terms of processing and transmitting.

6.3.2 Closing the Gap with the Sensor Bus

This interoperability gap needs to be closed in a sophisticated way. Therefore, Bröring et al. (2010) suggest an intermediary layer to bypass the gap between the low-level protocols used in WGSNs and the high-level protocols of the SWE standards. They introduce a so-called Sensor Bus, which follows a message bus pattern to facilitate the integration of sensors and sensor data feeds into as Sensor and Spatial Data Infrastructure (SSDI) based on SWE standards. Adapter applications connect to the Sensor Bus waiting for new sensor data to arrive and forward the data into the SWE services with the service's compliant methods. For instance, arriving data is persistently stored in the database, which is accessible by the *InsertObservation* request of the SOS service. The Sensor Bus concept is illustrated in Figure 6.5.



Source: based on Herle et al. (2016b)

Figure 6.5: *Closing the interoperability gap with a Sensor Bus*

The figure on the left shows the interoperability gap between the SWE services and the WGSNs. The Sensor Bus as an intermediary layer is introduced in the figure on the right. It can be implemented using a bunch of different protocols. For instance, Bröring et al. (2010) show the implementation in a proof-of-concept with four different technologies: Internet Relay Chat (IRC), XMPP, Java Message Service (JMS) and Twitter. Bröring (2012) concludes in his thesis that the Sensor Bus with its publish/subscribe mechanism is useful for automated registration of sensors and their data with the Sensors Web services. Other researchers seize on this concept. Geipel et al. (2015), for instance, also utilize XMPP to notify client using the Sensor Bus concept.

6.3.3 Sensor Bus in the EarlyDike Project

In the EarlyDike project, we adapt the idea of the Sensor Bus. However, we use GeoMQTT for the implementation of the bus since the other protocols mentioned in the previous section do not meet our requirements. First, the sensor nodes in the WGSN should be independent clients in the system and, therefore, should also be able to receive messages. Protocols that rely exclusively on a TCP/IP stack (e.g. IRC or XMPP) are not suitable, since most WGSNs use connectionless transmission protocols such as ZigBee. With the GeoMQTT-SN extension (see Section 4.5), these capabilities are already implemented in the GeoMQTT protocol. Secondly, the resource constraints restrict the abilities of sensor platforms to process, create and transmit large data formats such as XML and, thus, rule out the use of protocols such as XMPP.

The realization of the Sensor Bus involves the implementation of adapter applications, which are basically GeoMQTT clients that connect to the bus and translates between incoming `GEOPUBLISH` messages and the SWE services (see Figure 6.5). The topic names, the timestamp/time interval and the geometry of the messages are mapped to the metadata of the corresponding SWE services. As a proof-of-concept, this is implemented in the EarlyDike project with the SOS service to store sensor data.

The SOS service uses O&M 2.0 to represent and transfer (spatiotemporal) measurements. O&M defines a conceptual model as an XML based GML application schema. An observation in the basic observation model is related to a procedure, which represents the process that created the observation, for instance, a physical sensor or a simulation. Further, it consists of observed properties, which describe the properties that are observed, e.g., "temperature" or "humidity". A Feature of Interest (FOI) is linked to each observation. It is a computational representation of a real-world feature (e.g. "dike of Untjehörn"). The FOI has a shape property for the geometry of the observation. The observation's result can be of any type, from single values to n-dimensional coverage of values. Finally, an observation has three temporal properties: a phenomenon time to represent the time when the result applies to the observed property, a result time which represents the production time stamp of the observation and an optional valid time that defines the time period for which the result is usable (Bröring et al., 2011). Additionally, the SOS data model needs an observation offering URI which groups observations.

In the implemented adapter, the shape property of the FOI is associated with the geometry of the `GEOPUBLISH` message, while the temporal properties are derived from the timestamp or time interval. The payload of the GEOPUBLISH message represents the result of the observation. The other properties are generated from the different levels of the topic name. For instance, the topic name *SOS/insertObservation/labdike/geotextile/1/voltage/V* is used in the EarlyDike project. The first two levels determine which operation must be executed, here an *InsertObservation*, which is sent to the deployed SOS server. The derived properties are given for the example topic in Table 6.1:

Table 6.1: *Mapping of the topic "SOS/insertObservation/labdike/geotextile/1/voltage/V" to mandatory O&M attributes*

O&M attribute	Derived value from topic name
procedure	SOS/procedure/labdike/geotextile
featureOfInterest	SOS/featureOfInterest/labdike/geotextile/1
offering	SOS/offering/labdike/geotextile
observedProperty	voltage
Unit of Measurement (UOM)	V

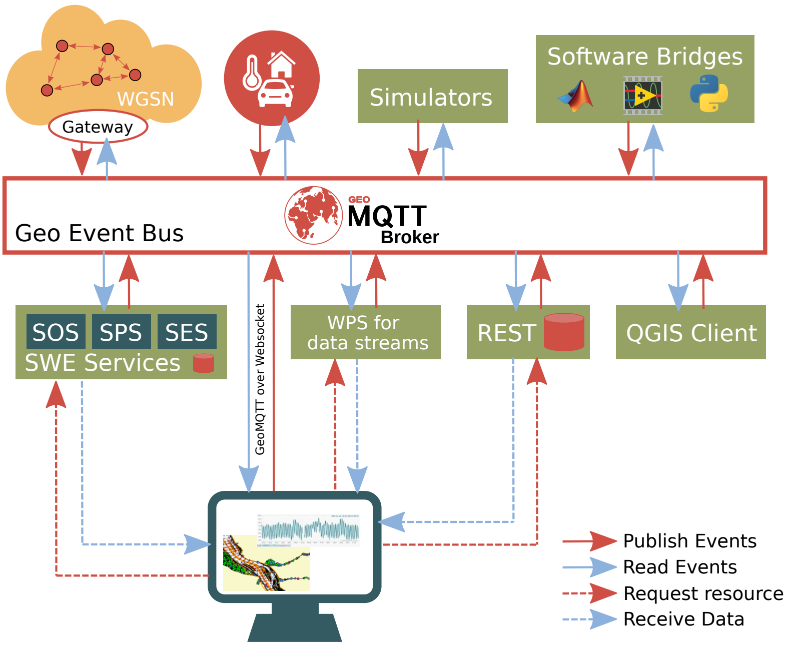
Source: Author's illustration

For each GEOPUBLISH message the adapter receives, it derives the URI and generates the *InsertObservation* XML. Before posting the XML to the SOS server, the adapter checks if the sensor already exists in the server. If it exists, the adapter continues with the request, otherwise an *InsertSensor* request is performed in advance. This request needs basically the same properties (procedure, offering and FOI), which are encoded in SensorML sub-elements. Following this message handling, the adapter supports plug-and-play behavior for new sensors if the GEOPUBLISH topic name complies with the described structure.

6.3.4 GEOEVENT BUS EXTENSION

A procedure in the observation model represents not only a physical sensor but also simulations or processes. However, sometimes these simulations or processes rely on the data issued by the physical sensor. For example, before persistently stored in a database, raw measured sensor data must be processed further which sometimes cannot be accomplished by the sensor node itself. So, a post processor must be notified of the new measured data and process the data before writing it to the database. For these cases, Ghobakhlou et al. (2014) suggest a service, the so-called Sensor Data Processing Service (SDPS). The service subscribes to a MQTT topic on which the raw sensor data is published and post processes each message if it matches a specific pattern. Subsequently, the measured data is stored in a SOS database.

Since these situations occur in the EarlyDike project quite frequently, we transformed the Sensor Bus to an event bus and used the concept of GeoPipes using GeoMQTT to connect producer and consumer of GeoEvents in a push-based way. We call this bus the GeoEvent bus. (Post-) Processing services log in the GeoEvent Bus and subscribe to the GeoMQTT topics on which the sensor data are published. The results of the processing service are republished to the event bus on a different topic. Data storage services such as the SOS subscribe to the post processed messages and only store this data persistently omitting the raw values. That way, e.g. outliers or erroneous data can be filtered easily or data aggregation functions can be applied to reduce data for storage. For instance, services for filtering or aggregation tasks can be set up by the implemented WPS extension for data streams, which is presented in Section 6.4. Figure 6.6 shows the GeoEvent Bus with the different producers and consumers of GeoEvents.



Source: based on Herle et al. (2016b)

Figure 6.6: *GeoEvent Bus with different producers and consumers of GeoEvents*

Basically, the figure on the left-hand side shows the Sensor Bus concept while other GeoMQTT information and services such as the REST bridge (Section 6.2) or the QGIS client (Section 6.1) can also easily connect to the GeoEvent Bus to receive data from it. However, also producers of data and processes acting on GeoStreams can be linked together to form new event-driven process chains. In the EarlyDike project, different simulators such as a storm surge simulator or a wave simulator were implemented, which are based on data issued by physical sensors or the results of other simulations. In this context, a set of bridges to other 3rd party software were implemented as well. These include a Matlab[8], a LabVIEW[9] and a SWAN[10] bridge since other project partners relied on these software products. The bridges, written in Python, manage a local socket to provide an Interprocess Communication (IPC) interface to the 3rd party applications. The applications as well as the corresponding bridge listen to and can publish messages to the socket. This way, 3rd party software can be connected to the GeoEvent Bus and receive or generate GeoEvents.

6.4 Enhancing the WPS Interface with GeoPipes Support

In our conceptual architecture (see Section 3.5), we introduced the notions of GeoPipes, GeoEvent and GeoStreams. These concepts can be implemented using the GeoMQTT protocol. Now, we like to look at processing of GeoEvents and GeoStreams by connecting geoprocessing mechanisms with GeoPipes. We aim for a geoprocessing interface, which supports GeoStreams as an input, processes the incoming GeoEvents in real-time and can issue processed GeoStreams. To achieve this, the WPS interface of the OGC is enhanced by novel input and output types to support the concept of GeoPipes (Herle & Blankenbach, 2017). The concept and the technical implementation as a proof-of-concept are presented in this section.

6.4.1 Introduction in Real-Time Geoprocessing

The diffusion of location-aware IoT devices increases the demand to process data streams in real-time. Especially in time-critical applications such as monitoring infrastructures, the need for real-time analysis and response becomes mandatory. In real-time geoprocessing both dimensions - spatial and temporal - are used to retrieve knowledge and support decisions. But, combining the two dimensions in a real-time system brings numerous computational challenges and opportunities for collection, storage and especially processing. According to Nittel (2015), future real-time geoprocessing systems will most likely focus on spatiotemporal analysis instead of spatial analysis over snapshots of spatial data. Therefore, existing methods might require new algorithms and implementations to compute and deliver real-time

[8] `https://www.mathworks.com/products/matlab.html`

[9] `https://www.ni.com/en-us/shop/labview.html`

[10] `http://swanmodel.sourceforge.net/`

results from much larger data sets. Typical tasks of these systems include real-time geospatial queries over large amounts of data streams while keeping up with incoming data, finding patterns in data streams or computing cross-correlations with other streams respectively historical data. For these systems a formal foundation of time and spatiotemporal concepts is required. McCullough et al. (2011) define, for instance, a typology of real-time geoprocessing, which they draw from Worboys & Hornsy (2004) model about representing the dynamic nature of real-world phenomena within GIS (see Section 3.2.2). They simplify this four-stage model into snapshot geoprocessing and stream geoprocessing. While snapshot geoprocessing still refers to static processing of spatiotemporal data, in which input data is specified once, read in once and the operation has a finite lifetime, stream geoprocessing is a radically different approach. Since geospatial data streams are an unbounded sequence of tuples with a time dimension but also a space dimension, querying and processing these open-ended data streams meet different requirements than processing static finite data. The amount of spatiotemporal data increases rapidly over time, and, thus, applying data stream models to geospatial data becomes relevant when querying or analyzing them in real-time (Appice et al., 2014). Different data stream models are applied in data stream systems to be able to query continuously arriving data tuples (Babcock et al., 2002). Common techniques are the so-called window approaches. A count-based window model, for instance, decomposes the data stream into non-overlapping windows of fixed size. These windows can be queried once they are completed. After processing, the windows are discarded. The sliding window model, on the other hand, considers the most recent data of the stream. It has a fixed window size and is similar to a first-in, first-out queue ordered by time, which is updated if a new data tuple arrives. Queries are compiled and executed on that queue. A data stream model forms the basis for further knowledge discovery in the (geo) data stream. Subsequent analyses could be summarization tasks of the data streams, such as sampling or histograms, or more advanced and complex processing in the field of data stream mining to predict values, cluster or find anomalies (Appice et al., 2014).

6.4.2 OGC WPS Interface and Real-Time Processing

In modern SSDIs standardized geo web services are used to guarantee interoperability. The commonly used interface standard for geospatial processing is the OGC WPS. It standardizes rules for the inputs and outputs of deployed services, as well as the request methods of a service. The interface in version 1.0.0 is standardized by the OGC since 2007 (Schut, 2007). Since version 1.0.0 has some drawbacks, a Standard Working Group (SWG) was formed in 2009 to work on a new interface standard, WPS 2.0, to evaluate and process change requests. Finally, in 2014 the new standard WPS 2.0 has been released (Müller & Pross, 2018). However, since there was no server-side implementation of the WPS 2.0 standard by the time of implementing the extension, we focus here on the version 1.0.0, keeping in mind to be able to transfer the concept also to the new version. Several open source WPS 1.0 server implementations exist, for instance the 52North WPS server written in Java or the PyWPS server implemented in Python version 3. An overview of different implementations and their performances can be found in Poorazizi & Hunter (2015).

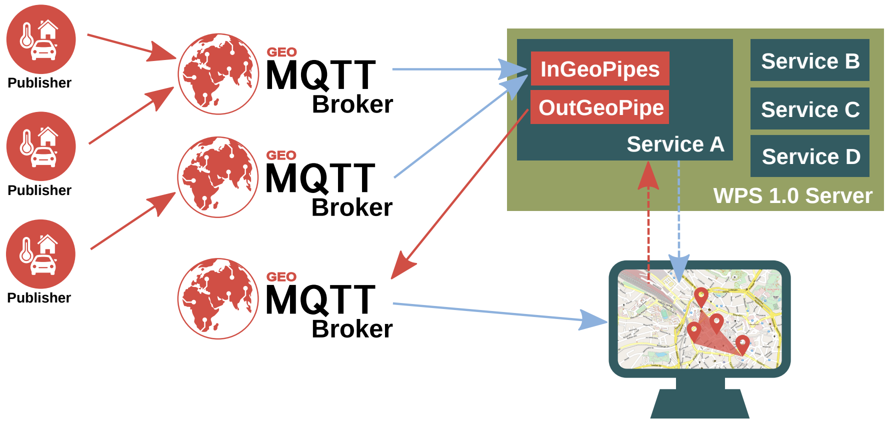
The WPS is a request/response interface with three core operations, which can be used by clients to interact with the server. The *GetCapabilities* operation is used to inform the requesting client about service meta data in form of an XML-document that describes the capabilities of a specific server implementation. Clients can request detailed information about each process with the *DescribeProcess* operation by specifying the process identifier in the request. Finally, to invoke a process, clients may use the *Execute* operation including the input parameters in the body of the request. The server parses the request, executes the process with respect to the inputs and returns the result to the client. The WPS interface was developed to process geospatial data, vector and/or raster data, but can also be used to implement non-spatial processes. Therefore, input and output data can be of three different types. *LiteralValue* parameters basically represent string data, which is sent directly to the server. Server and client can specify units used and the atomic data type for these parameters. Various data types, such as integer, string or date, can be chosen. The *ComplexValue* type represents large datasets, which can also be binary. This type is usually used for geospatial raster or vector datasets. WPS servers specify acceptable input rules with an XML schema and MIME types, which the client should follow. For instance, raster data are sent using base64 encoding, while vector data are usually encoded in GML or other formats such as GeoJSON. However, the standard (Schut, 2007) specifies the content of the *ComplexValue* data structure as "any". Accordingly, custom types can be defined as well. Last, the *BoundingBoxValue* can be used to define an area of interest with a left-bottom and a right-top corner using some CRS.

The WPS interface is based on synchronous HTTP as described and, thus, has some restrictions in asynchronous real-time processing. For long-lasting complex computations, which exceed the HTTP time-out duration, a polling approach is defined for asynchronous operations in the standard. A requesting client can poll the server to check the state of the processes, e.g. if it has finished or not. According to Resch et al. (2010), the significant overhead in exchanging messages is a major disadvantage of this approach, since the client must continuously poll the server. They conclude that a notification mechanism seems to be a more suitable and optimized approach. In Westerholt & Resch (2015) a WPS server is extended by such a push mechanism to inform the client about the state of process execution. To notify the client, the extension uses the WebSocket Protocol, so that messages can be received in a browser. Although in this prototype, notifications are only used to inform the client, the architectural approach could also be utilized to integrate complex geospatial analysis into event-driven, real-time workflows. This includes the input of live geospatial information into the process itself. Across the literature, some exemplary real-time applications are described, which process sensor data with the WPS interface. For instance, in Schaeffer et al. (2012) or Kmoch et al. (2016) a simulation process is started by a WPS service, first querying a SOS for the most recent available sensor data collected in a sensor network. The WPS service is regularly invoked in a cycle after the previous process is finished. So in these solutions, there are still time and methodology gaps between the data streams emitted by sensor networks and the service processing the near real-time sensor data. To be able to process live geospatial information in a WPS service without polling a database actively (e.g. from a geo data stream published by sensors), a notification mechanism needs to be integrated in the process itself. In Foerster et al. (2012), a full streaming WPS is described utilizing the HTTP Live Streaming protocol to integrate data streams into the process. In this approach, the WPS can receive input data streams, process them and send intermediate results back to the client as an output stream. The streams are represented by the playlist data format, which helps to transport multimedia data chunks.

6.4.3 INTEGRATING GEOPIPES IN THE WPS INTERFACE

The GeoPipes concept can be coupled with the WPS 1.0 to provide real-time geoprocessing functionalities applied to spatiotemporal data streams. Since the WPS interface is not suitable or tailored for providing real-time geospatial processes, some authors came up with solutions described before to overcome these drawbacks. Our approach to perform geoprocessing tasks on an unbounded sequence of data tuples involves connecting the service to GeoPipes. Therefore, we define the in- and output data types *InGeoPipes*, *OutGeoPipes*, *InPipes* and *OutPipes*, which are in our architecture currently realized with GeoMQTT, respectively MQTT clients. The architecture concept is illustrated in Figure 6.7.

A service “A” deployed on a WPS 1.0 server is invoked with an *Execute* request. In the request *InGeoPipes* and *OutGeoPipes* are defined, which encode GeoSubscriptions connection details. For each GeoPipe, the service connects to a GeoMQTT broker with a client in a dedicated thread. If it is an *InGeoPipe*, the client subscribes to the specified GeoSubscription. The service computes a custom function on the incoming GeoStream and eventually publishes a message to an *OutGeoPipe*. At this, a process may connect to different *InGeoPipes* and *OutGeoPipes*. Both types of pipe are defined by the user of the service. Unlike Westerholt & Resch (2015) in which the server chooses the event server of the outputs and provides the connection details in the response of the *Execute* query, the connection details for the *OutGeoPipes* are also specified by the client and, therefore, an input to the service. To receive



Source: Author's illustration

Figure 6.7: *Proposed architecture for connecting the WPS interface with GeoPipes*

results published to the *OutGeoPipe*, the requesting client must connect to the pipe itself and independently from the WPS service. The benefits of this design decision include that the WPS standard and the XML structures for the input do not require non-compliant customizations. It also facilitates fusing different streams and chaining stream processes since the client has control about the endpoints of the streams. However, this implementation implies that the client has profound knowledge about possibly multiple remote servers, which can be unfavorable in certain applications.

GEOPIPES INPUT TYPES

For the input and output types definition the rule set of data types of the WPS 1.0 standard are used. The WPS interface defines three different input data types: *LiteralData*, *ComplexData* and *BoundingBoxData*. For the definition of a (Geo-)Pipe the *BoundingBoxData* input type is not suitable, because it strictly expects a predefined XML data structure with geographic coordinates for a rectangular area. However, the *LiteralData* and *ComplexData* input types can both be used to model and submit the connection details for a (Geo-)Pipe.

The *LiteralData* input consists of a simple literal value. A "dataType" attribute can be included in the parameter. Typical data types are strings or integers but it is also possible to choose an "anyURI" data type. So, we can define subscriptions for *InPipes/InGeoPipes* and topic names to publish to for *OutPipes/OutGeoPipes* in an URI. Table 6.2 shows the URI syntax for the different versions of the pipes.

Table 6.2: *URI syntax for different types of pipes*

Data type	Uniform Resource Identifier (URI)
InGeoPipe	`geomqtt://[clientid:password@]<address>:<port>/<topicFilter>?temporal=<relation>,<temporalExpression>&spatial=<relation>,<geometry>`
InPipe	`mqtt://[clientid:password@]<address>:<port>/<topicFilter>`
OutGeoPipe	`geomqtt://[clientid:password@]<address>:<port>/<topicName>`
OutPipe	`mqtt://[clientid:password@]<address>:<port>/<topicName>`

Source: based on Herle & Blankenbach (2017)

The scheme of the URI determines the used protocol. Further, for instance an *InGeoPipe* requires the address and port of the broker and potentially the credentials (client id & password) to log in. The GeoSubscription details are specified in the path and query parts of the URI. Whilst the path represents the topic filter, the temporal filter and the spatial filter are defined in the query part. With this URI syntax method, it is only possible to define a single GeoSubscription at a time. In an *OutGeoPipe* or *OutPipe* only a topic name, which is the path of the URI, is defined.

Table 6.3: *MIME types for different types of pipes*

Data type	Multipurpose Internet Mail Extensions (MIME)
InGeoPipe	application/x-ogc-ingeopipe; subtype=geomqtt
InPipe	application/x-ogc-inpipe; subtype=mqtt
OutGeoPipe	application/x-ogc-outgeopipe; subtype=geomqtt
OutPipe	application/x-ogc-outpipe; subtype=mqtt

Source: based on Herle & Blankenbach (2017)

For a more sophisticated solution, the details of the pipes can also be specified within a custom XML encoding in a *ComplexData* input type. The WPS standard states that the content of the *ComplexData* data structure can be of any type. Thus, this approach is suitable and valid here. The *ComplexData* version has some advantages in comparison to the *LiteralData* version. For instance, it is more flexible in defining multiple GeoSubscriptions in one *InputGeoPipe*. Additionally, with the created XML schema files, the input can be validated automatically when the user sends the request. Based on the WPS Best Practices Discussion Paper (Schaeffer et al., 2012), Table 6.3 shows the introduced MIME types for the different pipes.

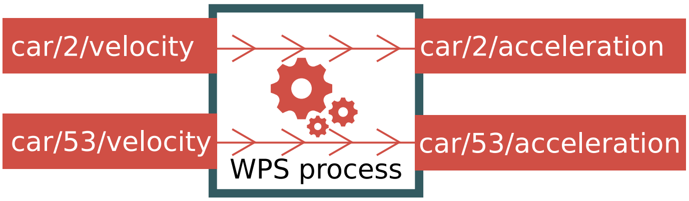
Like the encoding in *LiteralData*, the address and port of the broker are also defined in the *ComplexData* representation (see Listing 6.2). A login tag can be used to submit credentials (omitted here). The *Geosubscribe* tag indicates a single GeoSubscription with topic, temporal and spatial filters. It can be used multiple times. The XML is parsed by the server, validated against the XML schema file and then used to connect to the broker and register the GeoSubscriptions. The other pipes are defined similarly. A schema is created for each of the types and handled in the same way. Like mentioned, output pipes are used to publish processing results and, thus, expect solely a topic name to publish to.

```xml
<pipe:GeoMQTTInput xmlns:pipe="http://www.gia.rwth-aachen.de/geopipes">
  <pipe:Address>localhost</pipe:Address>
  <pipe:Port>1883</pipe:Port>
  <pipe:Geosubscribe>
    <pipe:TopicFilter>temperature</pipe:TopicFilter>
    <pipe:TemporalFilter relation="CONTAINS"></pipe:TemporalFilter>
    <pipe:SpatialFilter relation="EQUALS">POINT(6 51)</pipe:SpatialFilter>
  </pipe:Geosubscribe>
</pipe:GeoMQTTInput>
```

Listing 6.2: *XML-encoded InGeoPipe*

Named wildcards mechanism

A special case in MQTT and GeoMQTT are the single-level and multi-level wildcards in topic filters. Imagine a service subscribes to a topic filter *car/+/velocity* where the wildcard "+" represents the car id for each car individually. The service computes the acceleration for each car individually and publishes the result to an output pipe with a topic name, which is customized to each car with respect to its id (see Figure 6.8).



Source: based on Herle & Blankenbach (2017)

Figure 6.8: *GeoPipes and named wildcards example*

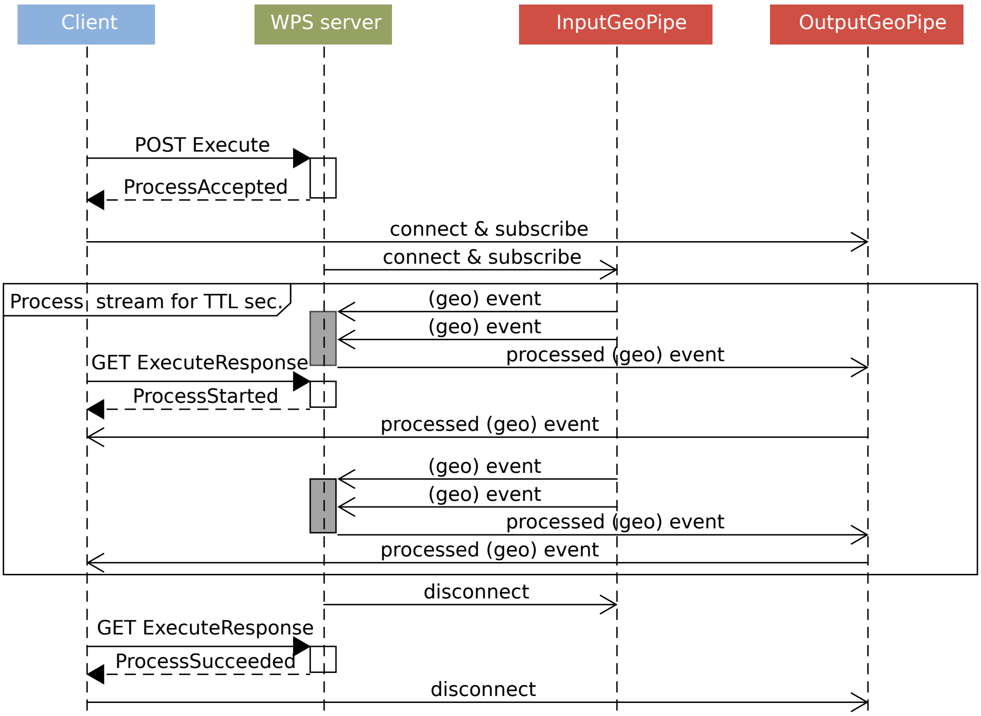
For these situations, the named wildcards mechanism in topic filters of a subscription is introduced. In the *Execute* method of the WPS service, the user can assign a name in curly brackets to wildcards used in topic filters. In the previous example the topic filter would be *car/+{car_id}/velocity*. This is especially useful if we establish an output pipe which uses the *{car_id}* variable to distinguish between entities. In the topic name of the output pipe, the user then just needs to set the specific name of the variable. In this example, the topic name would be *car/{car_id}/acceleration*. In a similar way, this mechanism can also be used for the multi-level wildcard. In this case, the variable would potentially consist of a string representing multiple hierarchy levels of the topic.

HANDLING WPS DRAWBACKS

As explained, the WPS interface standard is designed to support operations with a finite lifetime by using HTTP requests. On the contrary, (geospatial) data streams are an open-ended unbounded sequence of tuples. Processing these streams implies to start an operation with (possibly) an infinite lifetime. In addition, the set of methods of the WPS interface does not include a method to stop a process. A process, which is executed with the *Execute* method, runs until it succeeds or fails. The requesting client is not capable of interfering with the process execution. Thus, in McCullough et al. (2011) the WPS server implementation was modified by adding a *StopExecuting* method to facilitate the management of continuous computing jobs. However, this is obviously not compliant to the standard and, therefore, not implemented in our solution.

To avoid an infinite lifetime of the process or zombie processes, our simple approach involves an extra parameter in each *Execute* request, which limits the lifetime of the process. The Time-To-Live (TTL) parameter is specified by the requesting client in seconds. The operation is applied to the streams for this amount of time. Subsequently, the service disconnects from (possibly) multiple brokers and exposes the resulting XML response of the process to the client.

The preferred way of invoking a WPS process in our approach is the asynchronous mode. This has some advantages in comparison to the synchronous mode. For instance, if the *Execute* request is accepted by the server, the immediate *ProcessAccepted* response allows the client to establish a connection to the output pipes to receive the results of the stream process. In synchronous mode on the other hand, the client can only connect directly to the broker without knowing the state of the request. Furthermore in asynchronous mode, an *Execute* request for a long-running stream process (the TTL parameter is set to a high value) does not exceed the HTTP time-out duration. Figure 6.9 visualizes the sequence of messages exchanged in our streaming WPS mechanism. The asynchronous mode of the WPS server is utilized here. The data streams are processed for TTL seconds. After the process time is exceeded, the client as well as the process disconnects from the pipes.



Source: based on Herle & Blankenbach (2017)

Figure 6.9: *GeoPipes integration in WPS - sequence of messages*

6.4.4 Implementation and Sample Processes

The GeoPipes extension for the WPS interface described in the previous paragraphs is implemented and tested in PyWPS-4, a server side implementation of the OGC WPS standard written in Python version 3 (PyWPS Development Team, 2009). It was chosen since it is easy to customize for our needs and new processes can be set up with low effort. Furthermore, a GeoMQTT client in Python is already implemented. Thus, processes using GeoMQTT as input and/or output GeoPipes can be implemented without great effort.

PyWPS-4 also offers some features we are using in our processes. For instance, input data can be validated in different modes up to a very strict validation using a given XML schema file. Additionally, PyWPS-4 is under active development and WPS 2.0 features will be implemented in the future (Čepický & De Sousa, 2016). With this server implementation, we set up some real-time geo processes, two of them are presented in the following sections. The processes use stream data models, such as the sliding window technique explained before. All described processes are compliant to the WPS standard, which means that the operation set is not modified and the processes use the TTL parameter to specify a finite runtime. Other additional processes are described in Herle & Blankenbach (2017) but omitted here.

EXAMPLE PROCESS - DYNAMIC CONVEX HULL

The generic implementation of the GeoPipes extension in our adjusted PyWPS instance allows us to run dynamic geometry algorithms. We set up a real-time processing service which calculates the dynamic convex hull of a changing set of points utilizing Overmars & van Leeuwen's (1981) algorithm. The algorithm is known to be efficient when inserting or deleting points from the set. For our implementation, we used the version of Cisneros (2007). The in- and outputs of the service are specified in Table 6.4.

Table 6.4: *Inputs and Outputs for dynamic convex hull service*

	Name	WPS Input	Data type/Format
Inputs	points	ComplexData	application/x-ogc-ingeopipe; subtype=geomqtt
	convexhull	ComplexData	application/x-ogc-outgeopipe; subtype=geomqtt
	ttl_points	LiteralData	integer
	ttl	LiteralData	integer
Outputs	response	LiteralData	boolean

Source: based on Herle & Blankenbach (2017)

The *points-InGeoPipe* represents the input stream for the geometries. Only the geometry and timestamp information of the GeoMQTT `GEOPUBLISH` messages are used. If a geometry is inserted into the set of points, the updated convex hull geometry is published with a GeoMQTT message in the *OutGeoPipe* defined in *convexhull*. The *ttl_points* parameter allows the user to specify the time the incoming points are part of the set. After the defined seconds the messages (and their geometries) are deleted from the set of points. The updated convex hull geometry is published once again to the *OutGeoPipe*. The stream process stops after *ttl* seconds. A demo of the process can be found on the website[11] .

[11] `http://wpsdemo.gia.rwth-aachen.de/convexhull.html`

EXAMPLE APPLICATION - ONLINE MAP MATCHING

The dynamic convex hull algorithm is a good example for how we can utilize and process spatiotemporal messages received by the GeoPipes. However, it is not a very realistic use case. In the research field of trajectory data mining, more realistic examples can be found. Trajectory data mining describes the process of knowledge discovery from trajectory data. Ultimately, methods of trajectory data mining can be used to e.g. analyze, query or classify mobility patterns or traffic (Zheng, 2015). In the processing chain of trajectory data mining one essential step is to match the raw and noisy locations from GNSS to a graph, for instance a road network. This preprocessing step is called map matching, respectively online map matching if it is applied to a data stream. We set up an online geometric map matching algorithm which matches the received GNSS points to the nearest road and issues the projected map matched location. The input parameters for the WPS service are defined in Table 6.5.

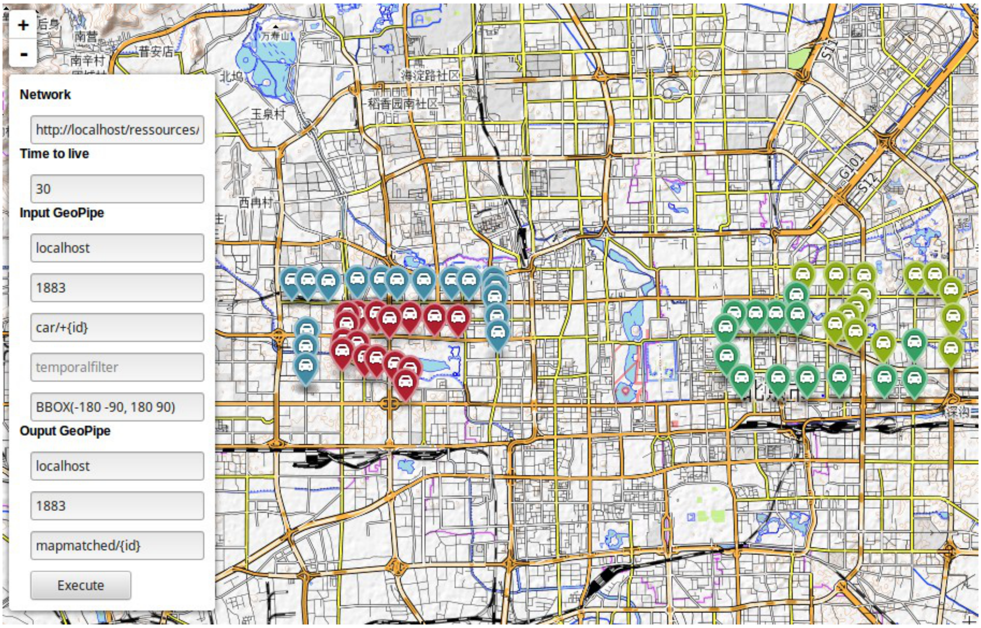
Table 6.5: *Inputs and Outputs for map matching service*

	Name	WPS Input	Data type/Format
Inputs	gpslocation	ComplexData	application/x-ogc-ingeopipe; subtype=geomqtt
	mmposition	ComplexData	application/x-ogc-outgeopipe; subtype=geomqtt
	road_network	LiteralData	application/gml+xml
	ttl	LiteralData	integer
Outputs	response	LiteralData	boolean

Source: based on Herle & Blankenbach (2017)

The *gpslocation-InGeoPipe* represents the input stream of the data, the *mmposition-OutGeoPipe* is used to publish the map matched position and the *road_network* is a user-defined road network, which can be specified as a reference pointing to a GML file or a WFS request. With the named wildcard mechanism, the implementation allows to distinguish between different entities, if, for instance, their identifiers are encoded in the topic name. That way, the topic names in the output GeoEvents can be adjusted to the specific map matched entity. As a proof-of-concept, we implemented a web application for the simple online map matching algorithm of cars. A screenshot of the client is illustrated in Figure 6.10.

Since the implemented simple map matching algorithm only projects the received location onto the nearest street in the road network, it is not a stream but a snapshot geoprocessing application. This means, the process could also be started with each


Network
http://localhost/ressources/
Time to live
30
Input GeoPipe
localhost
1883
car/+{id}
temporalfilter
BBOX(-180 -90, 180 90)
Ouput GeoPipe
localhost
1883
mapmatched/{id}

Source: Herle & Blankenbach (2017)

Figure 6.10: *Web application for the online map matching WPS process*

location individually. However, if we consider more sophisticated online map matching algorithms, which also include the history of GNSS locations and topological features, then it becomes a stream geoprocessing algorithm. For instance, Mattheis et al. (2014) introduced a Hidden Markov Model online map matching algorithm, which also considers the previous states (locations) of the car.

Map matching is a necessary preprocessing step in trajectory data mining. We only implemented this simple algorithm but with the OutGeoPipe defined, various subsequent other services could be set up subscribing to the GeoPipe to e.g. cluster cars or analyze traffic conditions. So, a chaining of different stages in a mining chain can be realized by coupling the GeoPipes together. This also supports the chaining idea of the original WPS interface.

Control DSP system with the WPS Interface

In an extended study, we implement efficient online map matching algorithms in a scalable Distributed Stream Processing (DSP) architecture and invoke the offered services using the WPS interface with GeoPipes (Laska et al., 2018). To achieve this, we setup an Apache Storm stream processing cluster and realized the input piping with GeoMQTT and Apache Kafka. The map matched positions are published by the system utilizing GeoMQTT, so that clients may directly subscribe to the stream processing results. As a proof-of-concept the proposed architecture was implemented and evaluated utilizing distributed map matching algorithms. However, it can also be used to implement applications for several other use cases of deploying and evaluating distributed stream processing algorithms that operate on spatiotemporal data streams originating from IoT devices.

In this proposed architecture GeoMQTT is applied to setup real-time stream processing pipelines on GeoEvents in an easy fashion. It serves as the glue between IoT devices, the real-time stream processing architecture and the end user of the results. For the use case, the described WPS extension and the implemented process to control the distributed stream processing offer the map matching algorithm as a standardized real-time geoprocessing service to end users. Basically in- and outputs are identical to the simple map matching algorithm described before; however, the underlying stream processing is outsourced to the DSP architecture, which is dedicated to stream processes and provides more processing power than a single server.

CHAPTER 7

CONCLUSION & FUTURE WORK

In this thesis, an event-driven architecture for the emerging Geospatial IoT is proposed, implemented and evaluated. Based on the fundamental concepts and technologies of the IoT, a protocol for exchanging spatiotemporal data in real-time is conceived and created in a prototypical system. This chapter summarizes in Section 7.1 first the developments and findings we achieved in this thesis. This covers the results of the different research areas we address to implement a Geospatial IoT platform:

1. Conceptual view on the Geospatial IoT: modeling, data types, mechanisms
2. Implementation of the concept: requirements, functionalities, evaluation
3. Integration of GeoMQTT into IoT concepts and contemporary GIS technologies

After reflecting the achievements, in Section 7.2 we look at future work in these areas, especially for the concepts and implementations realized in this thesis.

7.1 ACHIEVEMENTS AND REFLECTION ON RESEARCH OBJECTIVES

ARCHITECTURAL CONCEPT FOR A GEOSPATIAL IOT

Regarding the conceptual view on the Geospatial IoT, we raised several research objectives in the beginning of the thesis. These covered especially the form and notion of the Geospatial IoT, its involving things, their spatial nature and data models that are used in a Geospatial IoT. Starting from that, we approached the term *Geospatial IoT* by investigating the spatial nature of smart things and the essential components of IoT systems. Physical things in the real world have five spatial properties: location, shape, size, orientation and sphere of influence. These can be measured by different means, e.g. the location with GNSS. Further, integrating spatial concepts into IoT systems, especially in the OODA feedback loop of CPSs, it becomes evident that it is driven by spatial events. Subsequently, we analyzed the term event and the spatiotemporal modeling in GIS to find an appropriate ontology. Based on the conjunction of Galton's ontology *processes as patterns of occurrence* and Peuquet (1994)'s *Triad Framework*, we derived the definitions and modeling of

Geospatial Events and *Geospatial States*. Both describe a *Geospatial Process* with a spatial component in the real world. But while the Geospatial Event models an occurrent in the real world over a time interval, the Geospatial State represents a state of a continuant at a specific point in time. With these established, the essential modeling concepts for a Geospatial IoT were found and we could proceed with designing an appropriate architecture.

Based on these observations, further research questions include the types of data and their related concepts and mechanisms, which can be subsequently derived for implementing an architecture for the Geospatial IoT. We found that the abstract data types of geospatial processes form the basis of events and states in the real world issued e.g. by real-world objects. Taking this, we came up with a data type for our Geospatial IoT architecture. The *GeoEvent* combines both abstract data types in a general message type. It describes a 4-tuple consisting of a theme, a temporal component (timestamp or time interval), a spatial component as well as the body (or payload) of the message. This message type is the basic mechanism to exchange data in our proposed Geospatial IoT architecture. With GeoEvents and the EDA pattern, we developed and defined accompanying components such as the GeoPipe, the GeoStream or the GeoSubscription. The latter one represents the primary way for participants in the architecture to express their interest in GeoEvents. It consists of three filters applied to the first three parameters of GeoEvents. With its help consumers may connect to a GeoPipe. When connected, a consumer receives a stream of GeoEvents (GeoStream) associated with the GeoPipe. With these components and mechanisms, we could design a GeoEvent-driven architecture for a Geospatial IoT.

IMPLEMENTATION OF A GEOEVENT-DRIVEN ARCHITECTURE FOR A GEOSPATIAL IOT

With the architectural pattern for a Geospatial IoT established, various research questions occurred such as the proper type of communication or the qualification of various IoT protocols for the architecture. The concept of the GeoEvent-driven architecture for a Geospatial IoT has different requirements for an implementation: on the one hand mechanism, which are determined by the nature of EDA and, on the other hand, technical demands from the fundamentals of IoT platforms and systems. We raised requirements such as the messaging pattern, the lightweight or the adaptability and analyzed messaging candidates with respect to their suitability. We found MQTT to be the best matching candidate regarding our requirement engineering. The main reasons for this choice was, first, the publish/subscribe mechanism, which provides the messaging in a push-based fashion, so that data streams in real-time can be initiated. And secondly, the already given adaptability to resource-constrained environments such as sensor networks in IoT infrastructures.

Based on the technical specifications of the MQTT protocol in version 3.1.1, we introduced the extension GeoMQTT including new message types to publish GeoEvents, subscribe to them via a GeoSubscription, or to perform an unsubscribe process. To issue a proper GeoEvent, the `GEOPUBLISH` packet consists of a timestamp or time interval, a geometry, which can be specified in multiple encoding formats and different CRSs, a topic name and a payload. For specifying a client's interest in GeoEvents, the `GEOSUBSCRIBE` type was introduced. It is used to register one or multiple GeoSubscriptions with their three filter types. These are then evaluated by the broker to distribute the GeoEvents accordingly. At this, the topic filter has the common functionality like in MQTT, while the temporal and spatial filter consists of a temporal relation and a timestamp or time interval, respectively a spatial relation from the DE-9IM set of predicates and a reference geometry. Further, clients may unsubscribe from GeoSubscriptions using the topic filter and the `GEOUNSUBSCRIBE` packet type. Conceptually, these advanced functionalities transferred the topic-based publish/subscribe mechanism of MQTT to a content-based publish/subscribe based on three types of information in GeoMQTT.

We implemented a GeoMQTT API in Java, which is used to extend an existing MQTT broker and client with the introduced message types and the spatiotemporal capabilities. Furthermore, several other clients were created to support other programming languages and environments. These include a Python client and a JavaScript client. Especially the latter one can be used with the technology of WebSockets to connect a web application to a GeoMQTT broker and, thus, receive GeoEvents in real-time in a browser. With these clients established, we were also able to connect contemporary GIS technologies quite effortless to the GeoEvent-driven architecture like we will summarize later in this conclusion. On top of that, we advanced the extension MQTT-SN for connectionless transmission, such as ZigBee in sensor networks, with our spatiotemporal functionalities naming it GeoMQTT-SN. The focus of this advancements laid on the number of bytes transmitted for a GeoEvent or a GeoSubscription. Different strategies were used and implemented to reduce the message sizes. This included e.g. splitting up messages into multiple packets or outsourcing information completion from small sensor nodes to Internet gateways. GeoMQTT-SN client and gateway were implemented in C++ runnable on single-board computers such as Arduino or Raspberry Pi.

The implemented protocol and its features must meet the constraints and requirements of a Geospatial IoT. Questions on how the prototypical implementation can be evaluated with respect to the nature of a highly-connected system emerged. These involved on the one hand the compliance with the intended design and, on the other hand, the testing of real-world scenarios within the Geospatial IoT and its applications. Therefore, GeoMQTT was evaluated against the remaining requirements for a GeoEvent-driven architecture for a Geospatial IoT. The evaluation focused first on the expressiveness of the introduced messages regarding the conceptual data type of GeoEvents and the mechanism of GeoSubscriptions using Geospatial IoT scenarios. The results show that for the specific use cases the expressiveness is sufficient, but other applications may need other modeling capabilities such as the orientation of things, which cannot be modeled currently. Second, the message sizes were assessed in comparison to other encodings. We found that for the scenario modeling, the packet sizes are very small in comparison to other formats and, thus, beneficial in IoT environments. To test the efficiency and scalability, the GeoMQTT broker and its performance were tested in two testbeds and with several test plans simulating different message loads. We observed that in comparison to plain MQTT, the performance in terms of message throughput is clearly decreased, since the broker must evaluate two additional filters. However, we showed that by using a broker cluster, this performance reduction can be compensated. We found that the broker is horizontally scalable and, thus, the architecture may react to changed conditions such as an increasing number of messages.

INTEGRATION OF GEOMQTT INTO IOT CONCEPTS AND CONTEMPORARY GIS TECHNOLOGIES

GeoMQTT can be used to deploy a GeoEvent-driven architecture for the Geospatial IoT. With this architecture established, we can create Geospatial IoT applications and integrate GeoEvents and related concepts into contemporary GIS technologies. In this research field, several questions occurred, which were investigated by the following developments. Especially, the contributed effort and effectiveness of integrating the Geospatial IoT in existing GIS technologies were focused on. Further, we wanted to investigate if existing GIS technology can cope with protocols and data in a Geospatial IoT.

Approaching these questions, we implemented first a GeoMQTT plug-in for QGIS, an open source desktop GIS. The interaction between real-time GeoEvents and geospatial analysis functionalities offered by these expert system promises large potentials for understanding, modeling and visualizing natural or artificial ecosystems. The plug-in utilizes the created Python GeoMQTT client, so that users can specify and register GeoSubscriptions with the broker. Whether vector or raster data, the plug-in parses the incoming GeoEvents and loads them into the map display of QGIS. Subsequently, geospatial analyses and algorithms can be applied. This proof-of-concept illustrates that conventional desktop GISs can integrate data from the Geospatial IoT, especially GeoEvents in our architecture. Since QGIS does not follow an event model, the incoming spatiotemporal GeoEvents must be mapped in suitable in-build data types. Thus, it takes some efforts to adjust the different data types. Although the developed plug-in proofs that QGIS can receive and visualize GeoEvents in real time, further research about its capabilities of handling massive amounts of concurrent GeoEvents should be conducted.

In distributed GISs such as SDIs, several architectural patterns find its usage. GeoMQTT and the proposed architecture are based on the EDA pattern, but sometimes other architectural styles might be beneficial. Following the WoT idea, we implemented a RESTful bridge for providing an access point to GeoMQTT using HTTP methods. The software also serves as a logger system, so that historical GeoEvents are stored in a database and can be retrieved by HTTP GET request. Further, with HTTP PUT new GeoEvents can be published into the system. Although the pull-based approach of HTTP seems to be contrary to the idea of data exchange in real-time, the bridge has its rightful place in the architecture: HTTP is probably the most interoperable message exchange mechanism in the Internet, only a browser is needed. Thus, it can be used in an easy way e.g. by developers to insert GeoEvents into the architecture. We found that requesting and publishing of GeoEvents in a GeoJSON format using the RESTful bridge facilitates integration into existing desktop and Internet GIS solutions such as QGIS or web mapping applications based on OpenLayers or Leaflet.

Like the ROA style, the architectural pattern of SOA is favorable in some applications. The idea of the Sensor Web implements a SOA with the help of the SWE standards. It proposes a concept in which sensor data and sensor networks are accessible by web services using HTTP methods. With GeoMQTT we implemented a Sensor Bus to interconnect the services of the Sensor Web with the event-driven style of our Geospatial IoT architecture. Since low level resource-constrained sensor nodes are not able to use HTTP and XML encodings directly, the implementation closes also the interoperability gap between sensors and services. We proved the concept by an implementation of a bridge between GeoMQTT and the SOS, which offers historical sensor data as a service. Further, we extended the Sensor Bus concept into a GeoEvent Bus, which connects not only sensor nodes and service, but also other data producers such as analysis software or visualization entities such as web browsers.

Further, we also integrated our proposed Geospatial IoT architecture as well as its related concepts and protocols into GeoEvent processing capabilities. We enhanced the WPS interface of the OGC in a way that not only static data can be given as input and output. We introduced new data types, namely the GeoPipes concept, which can be used to specify input and output (geo) data streams. As a proof-of-concept, MQTT and GeoMQTT as push-based technologies were utilized to implement stream processing services. The extension with its new data types influences the modeling side of the service, so that the interface itself stays compliant to the standard. With push-based protocols as input and output types, processes can be created, which facilitate sensor fusion or CSP. By chaining different GeoPipes issued or taken as an input by several processes, processing chains acting on these (geo) data streams can be established. Further, we coupled the enhanced WPS interface with DSP capabilities, so that we are also able to provide stream processing clusters based on frameworks such as Apache Storm as services. This might be of interest in the future when the number of IoT devices and of (geo) data streams will increase tremendously.

With these applications and extensions, we showed that an architecture for the Geospatial IoT can be implemented with a spatiotemporal model as a basis message exchange type. We created a protocol, which is adjusted to the specific requirements of the IoT and, simultaneously, can be used to publish spatiotemporal-enriched data as a stream. GeoMQTT provides the capabilities to deploy a GeoEvent-driven architecture and, thus, enables message exchange and initialization of control loops in real-time. Further, we showed that the integration of existing contemporary GIS technology into the architecture is possible and adds value in terms of visualization, exploring and processing Geospatial IoT data. Hence, we conclude that the raised research questions can be answered sufficiently, although additional questions arose while writing this thesis. There is still room for further research in the field of Geospatial IoT. Some ideas to enhance our proposed solution are outlined in the next section.

7.2 Future Work

For the different fields of our research and the implemented approaches, we can think about future enhancements. This covers the modeling of data types and mechanisms in our conceptual architecture for the Geospatial IoT as well as the implementation and mechanisms in GeoMQTT, the evaluation methodology and the Geospatial IoT related concepts and services. Some of the ideas and future work are presented in this section.

Concept of GeoEvents and GeoEvent-based architecture for a Geospatial IoT

The spatial modeling of GeoEvents is currently only accomplished by the geometry of the corresponding real-world geospatial event or state. Other spatial properties are omitted or cannot be integrated into the spatial component. Like already mentioned, additional spatial properties might be useful in Geospatial IoT application such as the orientation or the pose in 3D space. Talking about 3D, currently solely 2D objects and their relationships given by DE-9IM are supported. In the future however, 3D objects as well as their relationships for specifying GeoSubscription in the third dimension are suitable and useful additions. With these integrated, additional spatial properties such as shape or size can be easily derived.

From an architectural point of view, the proposed GeoEvent-based architecture solely runs on the flow of GeoEvents. It is the single data type which is used for controlling the flow or transferring data regardless of the origin. We can think of introducing advanced or derived sub data types based on GeoEvents to distinguish e.g. different payloads or origins. This would also allow to modify the GeoSubscription mechanism to a type-based publish/subscribe system.

GEOMQTT MECHANISMS & IMPLEMENTATION

However, also the implementation of the current concepts in GeoMQTT can be improved further. Besides the improvements in modeling GeoEvents, the spatial component of GeoEvents or GeoSubscriptions can be extended by e.g. indirect georeferencing systems. Currently, the specification of geometries solely depends on coordinates or lists of coordinates. If we would introduce indirect georeferencing systems such as place names or postal codes, the subscribing process might become easier for clients, especially human clients. Humans memorize words much easier than coordinates and, thus, might be able to express their interests in GeoEvents without much foreknowledge. For instance, a human might be interested in all GeoEvents issued in the city center of Aachen and would be able to express this need by addresses or postal codes. Furthermore, other types of spatial relations such as directional or distance relations might be useful. A "nearby" relation for example can be reasonable in certain applications. For the temporal filtering mechanism we can think of providing support for more colloquial expressions such as "every Monday", too. In fact, this can be already expressed by the supported cron expressions but it is also only applicable with expert knowledge about the syntax and semantics. Also for temporal relation, it holds that e.g. distance relations can be fruitful.

A general future feature of the GeoMQTT mechanism could be *updating* GeoSubscriptions. Usually, a GeoSubscription is made once by a client. The subscriber can unsubscribe to it, but cannot update it by e.g. an alternative geometry in the spatial filter. But especially the updating of a geometry for filtering is beneficial and mandatory in some cases. For example, a moving entity such as a car subscribes spatially with a GeoSubscription to every GeoEvents that occurs in its surrounding. If the entity moves, the geometry of the spatial filter representing its surrounding must be updated frequently. Currently, subscribers have to unsubscribe first and, subsequently, renew their GeoSubscriptions. An efficient mechanism and/or message type for updating GeoSubscriptions could avoid this additional effort.

Also the current implementation of the GeoMQTT protocol can be improved. This starts with the efficient storage and evaluation of GeoSubscription, to advanced security issues such as topic specific authorization and continues with GeoMQTT clients written in programming languages to connect also other tools or other things to the Geospatial IoT. Furthermore, we implemented the GeoMQTT extension based on MQTT Version 3.1.1. In the meantime in March 2019, MQTT Version 5.0 was released by its standardization organization as an OASIS standard. It introduces advanced functionalities and message types in the protocol. A migration of the GeoMQTT extension to MQTT Version 5.0 in the near future seems reasonable.

GEOMQTT EVALUATION

Our methodology involved the conceptualization of an architecture, the implementation with GeoMQTT protocol and, the evaluation of the protocol against the raised requirements. This evaluation was realized in two phases, first the modeling expressiveness and size of the GeoMQTT messages based on constructed IoT scenarios were investigated and, secondly, the efficiency and scalability of the GeoMQTT broker were assessed. For both phases, further evaluations in the future are needed. In case of modeling real-world geospatial events and states we saw that other geospatial scenarios than the assessed ones can be found, in which the expressiveness of the message types is brought to its limits. This includes use cases, in which additional spatial properties are of main interest. Identifying these real-world use cases through advanced evaluation steps should improve also the modeling of the data types in the architecture. This also carries weight for the sizes of the `GEOPUBLISH` and `GEOSUBSCRIBE` packets, which should then be investigated further.

The stress test we conducted to assess the efficiency and the scalability in the second phase of the evaluation show already interesting results. However, in the test plans we evaluated the efficiency by setting fixed test parameters. The throughput of messages per second and the number of subscribers were not modified during the execution of the test plan itself. These are non-realistic scenarios since during operating time the combination of clients connecting, disconnecting, subscribing or publishing is highly variable. In our test plans we mainly tested for the peak performance of the GeoMQTT broker. In real applications, messages are distributed in a more random fashion. In the future, we think that it would be more reasonable to test the system with realistic loads. Scientific approaches such as the queuing theory could be applied to simulate incoming and outgoing messages in a realistic way. It also would make sense to construct several scenarios and requirements for different IoT applications. For instance, in an early-warning system the message throughput and requirements probably totally differ from these in time-uncritical applications. Prospective evaluations should take these differentiations into account.

Also the scalability test can be extended in the future. We evaluated if the efficiency in terms of message throughput can be increased by deploying a GeoMQTT broker cluster. The tests with two brokers were promising and we could clearly prove that the performance in contrast to a single broker can be increased by employing a cluster. Further tests should investigate to which factor this scaling is employable. The performance improvement depending on the replication factor should subsequently be determined.

GEOMQTT INFORMATION AND SERVICES

In addition, several related GeoMQTT information and services applications should be developed to provide assess points or adapters to software and standards for out Geospatial IoT architecture. This includes the integration of the protocol into newer standards of the IoT and the geospatial world. For instance, we introduced in the fundamentals chapter the SensorThings API and the Publish/Subscribe interface standard of the OGC. Both standards use already publish/subscribe mechanisms. Hence, integrating GeoMQTT as an extension to the used protocols would be straightforward.

A more general improvement poses the introduction of a name service for topics. In EDAs every consumer participating in the system should understand the messages that are exchanged. A business lexicon or an ontology is therefore a mandatory concept for these systems. In our proposed architecture with GeoEvents as a basis concept and GeoMQTT as an implementation of it, this cannot be ensured currently. Especially the topic name and the payload of the `GEOPUBLISH` and `PUBLISH` packets are arbitrary strings, respectively arbitrary data. To understand the topic name and the content of the payload, a client must know the semantic meaning of both. In the newer MQTT Version 5.0, this issue is partly tackled by including an optional MIME content type in each message to describe the content of the payload. However, for applications a semantic service for topic names and the data would be beneficial. This would facilitate the understanding of the structure of the topic names, the underlying ontology and the corresponding messages.

Finally, advanced applications involving processing and visualization capabilities in the GeoEvent-based architecture for the Geospatial IoT should be implemented. For instance, an IoT dashboard framework with an integrated WebGIS for visualizing GeoEvents or GeoStreams in real-time could give a sophisticated access points to the architecture for human clients. The technology for receiving and publishing GeoEvents from a browser is already implemented with the JavaScript client using WebSockets. Currently however, it is only used for demos or smaller web applications. An adaptable thick client with rich functionalities in form of a dashboard could be used to prototype applications resting on the proposed architecture quickly. Also we can think about implementing other applications with connecting factors to the geospatial domain. These include applications in related domains such as the industry 4.0, Building Information Modeling (BIM) or UAVs.

BIBLIOGRAPHY

Abel, D. J., Taylor, K., Ackland, R., & Hungerford, S. (1998), "An exploration of GIS architectures for internet environments," *Computers, Environment and Urban Systems*, 22, 7–23.

Aceves, E. & Larios, V. M. (2012), "Data Visualization for Georeferenced IoT Open Data Flows for a GDL Smart City Pilot," *IEEE-GDL CCD smart cities white paper*, 1–5.

Akyildiz, I., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002a), "A Survey on Sensor Networks," *IEEE Communications Magazine*, 40, 102–114.

Akyildiz, I., Su, W., Sankarasubramaniam, Y., & Cayirci, E. (2002b), "Wireless sensor networks: a survey," *Computer Networks*, 38, 393–422.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015), "Internet of things: A Survey on Enabling Technologies, Protocols, and Applications," *IEEE Communications Surveys and Tutorials*, 17, 2347–2376.

Alkhatib, H., Faraboschi, P., Frachtenberg, E., Kasahara, H., Lange, D., Laplante, P., Merchant, A., Milojicic, D., & Schwan, K. (2014), "IEEE CS 2022 Report," IEEE Computer Society.

Allen, J. (1983), "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, 26, 832–843.

Allen, J. F. (1984), "Towards a General Theory of Action and Time," *Artificial Intelligence*, 23, 123–154.

Alonso, L., Barbarán, J., Chen, J., Díaz, M., Llopis, L., & Rubio, B. (2018), "Middleware and communication technologies for structural health monitoring of critical infrastructures: A survey," *Computer Standards and Interfaces*, 56, 83–100.

Alspaugh, T. A. (2019), "Allen's Interval Algebra," *available at* `https://www.ics.uci.edu/{~}alspaugh/cls/shr/allen.html` (Accessed: 2019-05-15).

Appice, A., Ciampi, A., Fumarola, F., & Malerba, D. (2014), *Data Mining Techniques in Sensor Networks : Summarization, Interpolation and Surveillance*, London: Springer.

Asahara, A., Shibasaki, R., Ishimaru, N., & Burggraf, D. (2015a), "OGC Moving Features Encoding Part II: Simple CSV 1.0," *OGC® Implementation Standard*, OGC 14-084, 1–26.

Asahara, A., Shibasaki, R., Ishimaru, N., & Burggraf, D. (2015b), "OGC® Moving Features Encoding Part I: XML Core," *OGC® Implementation Standard*, OGC 14-083.

Ashton, K. (2009), "That 'Internet of Things' Thing," *available at* `https://www.rfidjournal.com/articles/view?4986` (Accessed: 2019-05-15).

Augustin, A., Yi, J., Clausen, T., & Townsley, W. M. (2016), "A Study of LoRa: Long Range & Low Power Networks for the Internet of Things," *Sensors*, 16, 1–18.

Babcock, B., Babu, S., Datar, M., Motwani, R., & Widom, J. (2002), "Models and Issues in Data Stream Systems," in: *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pp. 1–30.

Balshe, M., Peon, R., & Thomson, M. (2015), "Hypertext Transfer Protocol Version 2 (HTTP/2)," *IETF Standards Track*, RFC 7540.

Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R., & Sturman, D. (1999), "An Efficient Multicast Protocol for Content-Based Publish-Subscribe Systems," *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, 262–272.

Banks, A., Briggs, E., Borgendale, K., & Gupta, R. (2019), "MQTT Version 5.0," *OASIS Standard*.

Banks, A. & Gupta, R. (2014), "MQTT Version 3.1.1," *OASIS Standard*.

Beard, K. (2006), "Modelling Change in Space and Time: An Event-Based Approach," in: Drummond, J., Billen, R., Joao, E., & Forrest, D. (eds.): *Dynamic and Mobile GIS: Investigating Changes in Space and Time*, Boca Raton: CRC Press, 1st ed., pp. 55–76.

Beinat, E., Steenbruggen, J., & Wagtendonk, A. (2007), "Location Awareness 2020: A foresight study on location and sensor services," Vrije Universiteit, Spatial Information Laboratory (SPINlab), Amsterdam.

Bendel, S., Springer, T., Schuster, D., Schill, A., Ackermann, R., & Ameling, M. (2013), "A Service Infrastructure for the Internet of Things based on XMPP," in: *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pp. 385–388.

Benoit, L., Briole, P., Martin, O., Thom, C., Malet, J. P., & Ulrich, P. (2015), "Monitoring landslide displacements with the Geocube wireless network of low-cost GPS," *Engineering Geology*, 195, 111–121.

Bensky, A. (2016), *Wireless Positioning Technologies and Applications*, London: Artech House Publishers, 2nd ed.

Berners-Lee, T., Fielding, R., & Frystyk, H. (1996), "Hypertext Transfer Protocol – HTTP/1.0," *Network Working Group Memo*, RFC 1945.

Bifet, A. & Kirkby, R. (2009), "Data Stream Mining. A Practical Approach," University of Waikato, Hamilton.

Bigagli, L. & Rieke, M. (2017), "The new OGC Publish/Subscribe Standard - applications in the Sensor Web and the Aviation domain," *Open Geospatial Data, Software and Standards*, 2, 1–10.

Bill, R. (2016), *Grundlagen der Geo-Informationssysteme*, Berlin: Wichmann Verlag, 6th ed.

Blankenbach, J. (2007), *Handbuch der mobilen Geoinformation: Architektur und Umsetzung mobiler standortbezogener Anwendungen und Dienste unter Berücksichtigung von Interoperabilität*, Heidelberg: Wichmann Verlag.

Bonomi, F., Milito, R., Natarajan, P., & Zhu, J. (2014), "Fog Computing: A Platform for Internet of Things and Analytics," in: Bessis, N. & Dobre, C. (eds.): *Big Data and Internet of Things: A Roadmap for Smart Environments. Studies in Computational Intelligence*, Springer, vol. 546, heidelberg ed., pp. 169–186.

Botts, M., Percivall, G., Reed, C., & Davidson, J. (2007), "OGC Sensor Web Enablement: Overview and High Level Architecture. - Version 3," *OpenGIS® White Paper*, OGC 07-165.

Botts, M., Percivall, G., Reed, C., & Davidson, J. (2008), "OGC® Sensor Web Enablement: Overview and High Level Architecture," in: Nittel, S., Labrinidis, A., & Stefanidis, A. (eds.): *GeoSensor Networks: Second International Conference, GSN 2006, Boston, MA, USA, October 1-3, 2006, Revised Selected and Invited Papers*, pp. 175–190.

Botts, M. & Robin, A. (2014), "OGC SensorML: Model and XML Encoding Standard - Version 2.0," *OGC® Encoding Standard*, OGC 12-000.

Bouguera, T., Diouris, J. F., Chaillout, J. J., Jaouadi, R., & Andrieux, G. (2018), "Energy Consumption Model for Sensor Nodes Based on LoRa and LoRaWAN," *Sensors*, 18, 1–23.

Boyd, J. R. (1987), *A Discourse on Winning and Losing*, Maxwell: Air University Press.

Braden, S. E. (2015), "An Open Source Alternative for Crater Counting Using QGIS and the CircleCraters Plugin," in: *46th Lunar and Planetary Science Conference*, pp. 1–2.

Braeckel, A. & Bigagli, L. (2016), “OGC Publish/Subscribe Interface Standard 1.0, SOAP Protocol Binding Extension,” *OGC® Implementation Standard*, OGC 13-133.

Braeckel, A., Bigagli, L., & Echterhoff, J. (2016), “OGC® Publish/Subscribe Interface Standard 1.0 - Core,” *OGC® Implementation Standard*, OGC 13-131.

Brand, K., Blankenbach, J., & Kolbe, T. (2017), *Leitfaden – Mobile GIS . Von der GNSS-basierten Datenerfassung bis zu Mobile Mapping. Version 3.0*, München: Selbstverlag, Runder Tisch GIS e.V, 6th ed.

Bröring, A. (2012), “Automated On-the-fly Integration of Geosensors with the Sensor Web,” Ph.d. thesis, University of Twente.

Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., & Lemmens, R. (2011), “New generation Sensor Web Enablement.” *Sensors*, 11, 2652–2699.

Bröring, A., Foerster, T., Jirka, S., & Priess, C. (2010), “Sensor Bus: An Intermediary Layer for Linking Geosensors and the Sensor Web,” in: *COM.Geo ’10 Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research and Application*, pp. 1–8.

Bröring, A., Stasch, C., & Echterhoff, J. (2012), “OGC Sensor Observation Service Interface Standard - Version 2.0,” *OpenGIS® Implementation Standard*, OGC 12-006.

Bukhsh, Z. A., van Sinderen, M., & Singh, P. M. (2015), “SOA and EDA: A Comparative Study. Similarities, Differences and Conceptual Guidelines on their usage,” in: *12th International Joint Conference on e-Business and Telecommunications (ICETE)*, pp. 213–220.

Burke, J., Estrin, D., Hansen, M., Parker, A., Ramanathan, N., Reddy, S., & Srivastava, M. B. (2006), “Participatory Sensing,” *available at* `https://escholarship.org/uc/item/19h777qd` (Accessed: 2019-02-20).

Butler, H., Daly, M., Doyle, A., Gillies, S., Hagen, S., & Schaub, T. (2016), “The GeoJSON Format,” *IETF Standards Track*, RFC 7946.

Butler, H., Daly, M., Doyle, A., Gillies, S., Schaub, T., & Schmidt, C. (2008), “The GeoJSON Format Specification,” *available at* `http://geojson.org/geojson-spec` (Accessed: 2018-11-20).

Campelo, C. E. & Bennett, B. (2013), “Representing and Reasoning about Changing Spatial Extensions of Geographic Features,” in: Tenbrink, T., Stell, J., Galton, A., & Z, W. (eds.): *Spatial Information Theory. COSIT 2013. Lecture Notes in Computer Science*, Springer, Cham, vol. 8116, pp. 33–52.

Cannata, M., Antonovic, M., Molinari, M., & Pozzoni, M. (2014), “istSOS, Sensor Observation Management System: a Real Case Application of Hydro-Meteorological Data for Flood Protection,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XL-5/W3, 111–117.

Casals, L., Mir, B., Vidal, R., & Gomez, C. (2017), “Modeling the Energy Performance of LoRaWAN,” *Sensors*, 17, 1–30.

Čepický, J. & De Sousa, L. M. (2016), “New implementation of OGC Web Processing Service in Python programming language. PyWPS-4 and issues we are facing with processing of large raster data using OGC WPS,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B7, 927–930.

Chatzimilioudis, G., Konstantinidis, A., Laoudias, C., & Zeinalipour-yazti, D. (2012), “Crowdsourcing with Smartphones,” *IEEE Internet Computing*, 16, 36–44.

Chen, L. & Shang, S. (2018), “Approximate spatio-temporal top-k publish/subscribe,” *World Wide Web - Special Issue on Big Data Management and Intelligent Analytics*, 1–23.

Chen, X., Chen, Y., & Rao, F. (2003), “An Efficient Spatial Publish/Subscribe System for Intelligent Location-Based Services,” *Proceedings of the 2nd international workshop on Distributed event-based systems*, 1–6.

Chen, Y. & Kunz, T. (2016), “Performance Evaluation of IoT Protocols under a Constrained Wireless Access Network,” in: *2016 International Conference on Selected Topics in Mobile Wireless Networking (MoWNeT)*, pp. 1–7.

Cheng, B., Zhu, D., Zhao, S., & Chen, J. (2016), “Situation-Aware IoT Service Coordination Using the Event-Driven SOA Paradigm,” *IEEE Transactions on Network and Service Management*, 13, 349–361.

Cisneros, J. A. (2007), “Maintenance of the Convex Hull of a Dynamic Set,” Master thesis, University of Edinburgh.

Claramunt, C. & Theriault, M. (1995), “Managing Time in GIS An Event-Oriented Approach,” in: Clifford, J. & Tuzhilin, A. (eds.): *Recent Advances in Temporal Databases*, pp. 23–42.

Clementini, E. & Di Felice, P. (1995), “A Comparison of Methods for Representing Topological Relationships,” *Information Sciences*, 3, 149–178.

Clementini, E., Felice, P., & Oosterom, P. (1993), “A Small Set of Formal Topological Relationships Suitable for End-User Interaction,” in: Abel, D. & Chin Ooi, B. (eds.): *Advances in Spatial Databases - Proceedings of Third International Symposium, SSD ’93*, Berlin, Heidelberg: Springer, pp. 277–295.

Collina, M., Corazza, G. E., & Vanelli-Coralli, A. (2012), “Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST,” in: *2012 IEEE 23rd International Symposium on Personal, Indoor and Mobile Radio Communications - (PIMRC)*, pp. 36–41.

Cope, S. (2019), “Using MQTT Over WebSockets with Mosquitto,” *available at* `http://www.steves-internet-guide.com/mqtt-websockets/` (Accessed: 2019-02-06).

Cox, S. (2011), “Open Geospatial Consortium - Observations and Measurements - XML Implement - Version 2.0,” *OGC® Implementation*, OGC 10-025.

Cox, S. (2013), “OGC Abstract Specification Geographic information — Observations and measurements - Version 2.0,” *OGC® Standard: Abstract Specification*, OGC 10-004.

Crocker, D. (1982), “Standard for the Format of ARPA Internet Text Messages,” RFC 822.

Curry, E. (2004), “Message-Oriented Middleware,” in: Mahmoud, Q. (ed.): *Middleware for Communications*, New Jersey: John Wiley & Sons Inc, pp. 1–28.

Curry, E., Derguech, W., Hasan, S., Kouroupetroglou, C., & ul Hassan, U. (2019), “A Real-time Linked Dataspace for the Internet of Things: Enabling “Pay-As-You-Go” Data Management in Smart Environments,” *Future Generation Computer Systems*, 90, 405–422.

D’Ambrosia, J. (2018), “IEEE 802 LAN/MAN Standards Committee,” *available at* `http://www.ieee802.org/` (Accessed: 2019-01-08).

Dangermond, J. (2017), “Five GIS Trends Changing the World according to Jack Dangermond, President of Esri,” *Geoawesomeness*, online.

Davis, M. (2007), “Quirks of the "Contains" Spatial Predicate,” *available at* `http://lin-ear-th-inking.blogspot.com/2007/06/subtleties-of-ogc-covers-spatial.html` (Accessed: 2018-11-20).

Dawson, F. & Stenerson, D. (1998), “Internet Calendaring and Scheduling Core Object Specification (iCalendar),” *Network Working Group Standards Track*, RFC 2445.

Doyle, A. (2000), “OpenGIS® Web Map Server Interface,” *OGC® Implementation Specification*, OGC 00-028.

Duarte, L., Silva, P., & Teodoro, A. C. (2018), “Development of a QGIS Plugin to Obtain Parameters and Elements of Plantation Trees and Vineyards with Aerial Photographs,” *ISPRS International Journal of Geo-Information*, 7, 1–20.

Duquennoy, S., Grimaud, G., & Vandewalle, J.-J. (2009), "The Web of Things : interconnecting devices with high usability and performance," in: *ICESS 2009, May 2009, HangZhou, Chile.*, pp. 1–8.

Echterhoff, J. (2010), "OWS-7 Event Architecture Engineering Report Copyright," *OGC Public Engineering Report*, 10-060r1.

Echterhoff, J. & Everding, T. (2008), "OpenGIS Sensor Event Service Interface Specification (proposed) - Version 0.3.0," *OpenGIS® Discussion Paper*, OGC 08-133.

Echterhoff, J. & Everding, T. (2011), "OGC® Event Service - Review and Current State," *OGC® Public Discussion Paper*, 11-088r1.

Eclipse Foundation Inc. (2018a), "LocationTech JTS Topology Suite," *available at* `https://projects.eclipse.org/projects/locationtech.jts` (Accessed: 2019-02-08).

Eclipse Foundation Inc. (2018b), "Paho JavaScript Client," *available at* `https://github.com/eclipse/paho.mqtt.javascript` (Accessed: 2019-02-18).

Eclipse Foundation Inc. (2018c), "Paho Python Client," *available at* `https://github.com/eclipse/paho.mqtt.python{#}more-information`.

Egenhofer, M. J., Clarke, K. C., Gao, S., Quesnot, T., Franklin, W. R., Yuan, M., & Coleman, D. (2016), "Contributions of GIScience over the Past Twenty Years," in: Onsrud, H. & Kuhn, W. (eds.): *Advancing Geographic Information Science: The Past and NExt Twenty Years*, Needham: GSDI Association Press, pp. 9–34.

Egenhofer, M. J. & Franzosa, R. D. (1991), "Point-set topological spatial relations," *International Journal of Geographical Information Systems*, 5, 161–174.

Egenhofer, M. J. & Herring, J. R. (1991), "A Mathematical Framework for the Definition of Topological Relationships," *4th International Symposium on Spatial Data Handling*, 2, 803–813.

Egenhofer, M. J., Sharma, J., & Mark, D. M. (1993), "A Critical Comparison of the 4-Intersection and 9-Intersection Models for Spatial Relations: Formal Analysis," in: McMaster, R. & Armstrong, M. (eds.): *Proceedings of AutoCarto 11*, pp. 1–12.

Ericsson (2011), "More than 50 Billion Connected Devices," *Ericsson White Paper*.

Eugster, P. (2007), "Type-Based Publish/Subscribe: Concepts and Experiences," *ACM Transactions on Programming Languages and Systems*, 29, 1–50.

Eugster, P. T., Felber, P. A., Guerraoui, R., & Kermarrec, A.-M. (2003), "The many faces of publish/subscribe," *ACM Computing Surveys*, 35, 114–131.

Eugster, P. T., Guerraoui, R., & Damm, C. H. (2001), “On Objects and Events,” in: *Proceedings of the 16th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pp. 254–269.

Everding, T. & Echterhoff, J. (2008), “Event Pattern Markup Language (EML) - Version 0.3.0,” *OGC© Discussion Paper*, OGC 08-132.

Fazio, M. & Puliafito, A. (2015), “Cloud4sens: a cloud-based architecture for sensor controlling and monitoring,” *IEEE Communications Magazine*, 53, 41–47.

Fette, I. & Melnikov, A. (2011), “The WebSocket Protocol,” *IETF Standards Track*, RFC 6455.

Few, S. (2006), *Information Dashboard Design: The Effective Visual Communication of Data*, Sebastopol: O’Reilly Media, Inc., 3rd ed.

Fielding, R. T. (2000), “Architectural Styles and the Design of Network-based Software Architectures,” Ph.d. thesis, University of California.

Filipponi, L., Vitaletti, A., Landi, G., Memeo, V., Laura, G., & Pucci, P. (2010), “Smart City: An Event Driven Architecture for Monitoring Public Spaces with Heterogeneous Sensors,” in: *Proceedings of the 4th International Conference on Sensor Technologies and Applications (SENSORCOMM)*, IEEE, pp. 281–286.

Foerster, T., Baranski, B., & Borsutzky, H. (2012), “Live Geoinformation with Standardized Geoprocessing Services,” in: Gensel, J., Josselin, D., & Vandenbroucke, D. (eds.): *Bridging the Geographic Information Sciences, International AGILE’2012 Conference, Avignon (France), April, 24-27, 2012*, Berlin, Heidelberg: Springer, pp. 99–118.

Galton, A. (2004), “Fields and Objects in Space, Time, and Space-time,” *Spatial Cognition and Computation*, 4, 39–68.

Galton, A. (2009), “Spatial and temporal knowledge representation,” *Earth Science Informatics*, 2, 169–187.

Galton, A. (2015), “Outline of a Formal Theory of Processes and Events, and Why GIScience Needs One,” in: *Proceedings of the 12th International Conference on Spatial Information Theory (COSIT)*, New York: Springer, pp. 3–22.

Galton, A. (2016), “The Ontology of Time and Process,” in: *3rd Interdisciplinary School on Applied Ontology*.

Galton, A. (2018), “Processes as Patterns of Occurrence,” in: Stout, R. (ed.): *Process, Action, and Experience*, Oxford: Oxford University Press, pp. 41–57.

Gartner (2017), “Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016,” *Press Release*.

Geipel, J., Jackenkroll, M., Weis, M., & Claupein, W. (2015), “A Sensor Web-Enabled Infrastructure for Precision Farming,” *ISPRS International Journal of Geo-Information*, 4, 385–399.

GeoTools (2017), “GeoTools Documentation,” *available at* `http://geotools.org/` (Accessed: 2019-02-08).

Gessler, R. & Krause, T. (2015), *Wireless-Netzwerke für den Nahbereich: Eingebettete Funksysteme: Vergleich von standardisierten und prorietären Verfahren*, Wiesbaden: Springer Vieweg, 2nd ed.

Ghobakhlou, A., Sallis, P., & Wang, X. (2014), “A Service Oriented Wireless Sensor Node Management System,” in: *Proceedings of the IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pp. 1475–1479.

Gibbons, P. B., Karp, B., Ke, Y., & Nath, S. (2003), “IrisNet : An Architecture for a World-Wide Sensor Web,” *IEEE Pervasive Computing*, 2, 22–33.

Glabsch, J., Hesse, C., Heunecke, O., Keller, F., & Schuhbäck, S. (2011), “Kosteneffizientes Geo-Monitoring von Hochwasserschutzanlagen und Bauwerken mit GPS/GNSS-Sensornetzen,” in: *Exhibition and International Conference on Climate Impact. Acqua alta, Hamburg*.

Godfrey, R., Ingham, D., & Schloming, R. (2012), “OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0,” *OASIS Standard*.

Goodchild, M. F. (2007), “Citizens as sensors: the world of volunteered geography,” *GeoJournal*, 69, 211–221.

Goodchild, M. F. (2010), “Twenty years of progress: GIScience in 2010,” *Journal of Spatial Information Science*, 1, 3–20.

Google Developers (2018), “Protocol Buffers,” *available at* `https://developers.google.com/protocol-buffers/` (Accessed: 2018-11-20).

Gorlatova, M., Sarik, J., Grebla, G., Cong, M., Kymissis, I., & Zussman, G. (2015), “Movers and Shakers: Kinetic Energy Harvesting for the Internet of Things,” *IEEE Journal on Selected Areas in Communications*, 33, 1624–1639.

Grenon, P. & Smith, B. (2004), “SNAP and SPAN: Towards Dynamic Spatial Ontology,” *Spatial cognition and computation*, 1, 69–103.

Gross, N. (1999), “The Earth Will Don An Electronic Skin,” *available at* `https://www.bloomberg.com/news/articles/1999-08-29/14-the-earth-will-don-an-electronic-skin` (Accessed: 2019-05-15).

Grothe, C. (2010), “An Aeronautical Publish / Subscribe System Employing Imperfect Spatiotemporal Filters,” Ph.d. thesis, Technische Universität Darmstadt.

Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013), “Internet of Things (IoT): A vision, architectural elements, and future directions,” *Future Generation Computer Systems*, 29, 1645–1660.

Guinard, D., Trifa, V., Karnouskos, S., Spiess, P., & Savio, D. (2010a), “Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services,” *IEEE Transactions on Services Computing*, 3, 223–235.

Guinard, D., Trifa, V., Mattern, F., & Wilde, E. (2011), “From the Internet of Things to the Web of Things : Resource Oriented Architecture and Best Practices,” in: Uckelmann, D., Harrison, M., & Michahelles, F. (eds.): *Architecting the Internet of Things*, Berlin Heidelberg: Springer, pp. 97–129.

Guinard, D., Trifa, V., & Wilde, E. (2010b), “A Resource Oriented Architecture for the Web of Things,” in: *Proceedings of 2010 Internet of Things (IOT), IoT for a green Planet*, pp. 1–8.

Gutierrez, J., Villa-Medina, J. F., Nieto-Garibay, A., & Porta-Gandara, M. A. (2014), “Automated Irrigation System Using a Wireless Sensor Network and GPRS Module,” *IEEE Transactions on Instrumentation and Measurement*, 63, 166–176.

Güting, R. H. & Schneider, M. (2005), *Moving Objects Databases*, Morgan Kaufmann Publishers.

Hakiri, A., Berthou, P., Gokhale, A., & Abdellatif, S. (2015), “Publish/Subscribe-Enabled Software Defined Networking for Efficient and Scalable IoT Communications,” *IEEE Communications Magazine*, 53, 48–54.

Haklay, M., Mazumdar, S., & Wardlaw, J. (2018), “Citizen Science for Observing and Understanding the Earth,” in: Mathieu, P.-P. & Aubrecht, C. (eds.): *Earth Observation Open Science and Innovation*, Cham: Springer International Publishing, pp. 69–88.

Haller, S. (2010), “The Things in the Internet of Things,” in: *Proceedings of Internet of Things Conference 2010*.

Hartke, K. (2015), “Observing Resources in the Constrained Application Protocol (CoAP),” *IETF Standards Track*, RFC 7641.

He, S. & Chan, S. H. (2016), “Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons,” *IEEE Communications Surveys and Tutorials*, 18, 466–490.

Herle, S., Becker, R., & Blankenbach, J. (2016a), “Bridging GeoMQTT and REST,” in: *Proceedings of the Geospatial Sensor Webs Conference 2016, August 29 - 31, 2016, Münster*.

Herle, S., Becker, R., & Blankenbach, J. (2016b), “Smart sensor-based geospatial architecture for Dike Monitoring,” *IOP Conference Series: Earth and Environmental Science*, 34.

Herle, S., Becker, R., Blankenbach, J., Quadflieg, T., & Schüttrumpf, H. (2018), “Dateninfrastruktur für ein kontinuierliches echtzeitfähiges Geomonitoring,” in: Busch, W. (ed.): *Tagungsband GeoMonitoring 2018*, Clausthal-Zellerfeld, pp. 1–14.

Herle, S. & Blankenbach, J. (2016), “GeoPipes using GeoMQTT,” in: Sarjakoski, T., Santos, M. Y., & Sarjakoski, T. (eds.): *Geospatial Data in a Changing World: Selected papers of the 19th AGILE Conference on Geographic Information Science*, Cham: Springer International Publishing, pp. 383–398.

Herle, S. & Blankenbach, J. (2017), “Enhancing the OGC WPS interface with GeoPipes support for real-time geoprocessing,” *International Journal of Digital Earth*, 11, 48–63.

Herring, J. R. (2011), “OpenGIS Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture - Version 1.2.1,” *OpenGIS® Implementation Standard*, 06-103r4.

Hill, L. L. (2006), *Georeferencing: The Geographic Associations of Information*, Cambridge: MIT Press.

Hochschild, M. (2019), “Time4J,” *available at* `https://github.com/MenoData/Time4J` (Accessed: 2019-02-08).

Hohpe, G. & Woolf, B. (2003), *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Hornsby, A. & Bail, E. (2009), “μXMPP: Lightweight Implementation for Low Power Operating System Contiki,” in: *Proceedings of the International Conference on Ultra Modern Telecommunications and Workshops*, IEEE, pp. 1–5.

Hornsby, K. & Egenhofer, M. J. (2000), “Identity-based change: A foundation for spatio-temporal knowledge representation,” *International Journal of Geographical Information Science*, 14, 207–224.

Hornsby, K. S. & Cole, S. (2007), “Modeling moving geospatial objects from an event-based perspective,” *Transactions in GIS*, 11, 555–573.

Horton, M. & Adams, R. (1987), “Standard for Interchange of USENET Messages,” *Network Working Group Memo*, RFC 1036.

Huang, C. (2014), “GeoPubSubHub: A Geospatial Publish/Subscribe Architecture for the World-Wide Sensor Web,” Ph.d. thesis, University of Calgary.

Hui, J. & Thubert, P. (2011), “Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks,” *IETF Standards Track*, RFC 6282, 1–24.

Huisman, O. & de By, R. A. (eds.) (2009), *Principles of Geographic Information Systems: an introductory textbook*, Enschede: ITC, 4th ed.

Hunkeler, U., Truong, H. L., & Stanford-Clark, A. (2008), “MQTT-S — A Publish/Subscribe Protocol for Wireless Sensor Networks,” *Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE ’08)*, 791–798.

Ibarra-Esquer, J. E., González-Navarro, F. F., Flores-Rios, B. L., Burtseva, L., & Astorga-Vargas, M. A. (2017), “Tracking the Evolution of the Internet of Things Concept Across Different Application Domains,” *Sensors*, 17, 1–24.

IBM & Eurotech (2010), “MQTT V3.1 Protocol Specification,” Eurotech, IBM.

Isikdag, U. & Pilouk, M. (2016), “Integration of Geo-Sensor Feeds and Event Consumer Services for Real-Time Representation of IoT Nodes,” *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLI-B4, 267–274.

ISO/IEC JTC (2013), “Information technology - Telecommunications and information exchange between systems - Near Field Communication - Interface and Protocol (NFCIP-1),” *ISO/IEC Standard*, 18092.

ISO/IEC JTC 1 (1994), “Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model,” *ISO Standard*, 7498-1.

ISO/IEC JTC 1 (2014), “Information technology – Advanced Message Queuing Protocol (AMQP) v1.0 specification,” *ISO/IEC Standard*, 19464.

ISO/IEC JTC 1 (2016), “Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1,” *ISO/IEC Standard*, 20922.

ISO/TC 154 (2004), “Data elements and interchange formats – Information interchange – Representation of dates and times,” *ISO Standard*, 8601.

ISO/TC 211 (2007a), “Geographic information – Spatial referencing by coordinates,” *ISO Standard*, 19111.

ISO/TC 211 (2007b), “Geoinformation – Geography Markup Language (GML),” *ISO Standard*, 19136.

ISO/TC 211 (2015), “Geographic information – Well-known text representation of coordinate reference systems,” *ISO Standard*, 19162.

ISO/TC 268 (2018), “Sustainable cities and communities - Guidance on establishing smart city operating models for sustainable communities,” *ISO Standard*, 37106.

ITU (2005), “The Internet of Things,” *ITU Internet Reports*.

ITU-T (2012), “Overview of the Internet of Things,” *Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities*, Y.2060.

Jazayeri, M. A., Liang, S. H., & Huang, C. Y. (2015), “Implementation and Evaluation of Four Interoperable Open Standards for the Internet of Things,” *Sensors*, 15, 24343–24373.

Jin, B. & Chen, H. (2010), “Spatio-Temporal Events in the Internet of Things,” in: *Proceedings of the 8th International Conference on Embedded and Ubiquitous Computing (EUC)*, pp. 353–358.

Jin, B., Zhuo, W., Hu, J., Chen, H., & Yang, Y. (2013), “Specifying and detecting spatio-temporal events in the internet of things,” *Decision Support Systems*, 55, 256–269.

Jirka, S., Bröring, A., & Stasch, C. (2009), “Discovery mechanisms for the sensor web.” *Sensors*, 9, 2661–2681.

Johnston, S. & Cox, S. (2017), “The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams,” *Electronics*, 6, 1–7.

Kamilaris, A. & Ostermann, F. (2018), “Geospatial Analysis and the Internet of Things,” *ISPRS International Journal of Geo-Information*, 7, 1–22.

Kamiya, T. (2018), “Efficient Extensible Interchange Working Group,” *available at* `https://www.w3.org/XML/EXI/` (Accessed: 2019-05-15).

Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., & Alonso-Zarate, J. (2015), “A Survey on Application Layer Protocols for the Internet of Things,” *Transaction on IoT and Cloud Computing*, 3, 11–17.

Kim, K.-S. & Ogawa, H. (2017), “OGC Moving Features Encoding Extension - JSON - Version 1.0,” *OGC® Best Practice*, 16-140r1.

Kim, M.-S. (2018), “Research issues and challenges related to Geo-IoT platform,” *Spatial Information Research*, 26, 113–126.

Kitchin, R. & McArdle, G. (2017), “Urban data and city dashboards: Six key issues,” in: Kitchin, R., Lauriault, T., & McArdle, G. (eds.): *Data and the City*, London: Routledge.

Klok, L., van der Mark, P., & Nieuwkoop, E. (2014), “Urbmobi - A Mobile Measurement Device for Urban Environmental Monitoring,” in: *Proceedings of the Third International Conference on Countermeasures to Urban Heat Island (IC^{2}UHI)*, pp. 1221–1230.

Klopfer, M. & Simonis, I. (eds.) (2009), *Sany: an open architecture for sensor networks*, SANY Consortium.

Klyne, G. & Newman, C. (2002), “Date and Time on the Internet: Timestamps,” *Network Working Group Memo*, RFC 3339.

Kmoch, A., Klug, H., White, P., & Reichel, S. (2016), “SensorWeb Semantics on MQTT for responsive Rainfall Recharge Modelling,” in: *Proceedings of the 19th AGILE International Conference on Geographic Information Science*, pp. 1–4.

Koster, M. (2013), “M2M Protocol Interoperability Using the Smart Object API,” *available at* `http://iot-datamodels.blogspot.com/2013/10/m2m-protocol-interoperability-using.html` (Accessed: 2019-05-15).

Kotsev, A., Pantisano, F., Schade, S., & Jirka, S. (2015), “Architecture of a Service-Enabled Sensing Platform for the Environment,” *Sensors*, 15, 4470–4495.

Kreps, J. (2014), “Questioning the Lambda Architecture,” *available at* `https://www.oreilly.com/ideas/questioning-the-lambda-architecture` (Accessed: 2019-05-15).

Küpper, A. (2005), *Location-Based Services: Fundamentals and Operation*, Hoboken: John Wiley & Sons.

Lan, L., Wang, B., Zhang, L., Shi, R., & Li, F. (2015), “An Event-driven Service-oriented Architecture for the Internet of Things,” *Proceedings of the Asia-Pacific Services Computing Conference 2014 (APSCC 2014)*, 68–73.

Langran, G. (1990), “Temporal GIS design tradeoffs,” in: *Journal of the Urban and Regional Information Systems Association*, pp. 16–25.

Laska, M., Herle, S., Klamma, R., & Blankenbach, J. (2018), “A Scalable Architecture for Real-Time Stream Processing of Spatiotemporal IoT Stream Data—Performance Analysis on the Example of Map Matching,” *ISPRS International Journal of Geo-Information*, 7, 1–15.

Latvakoski, J., Iivari, A., Vitic, P., Jubeh, B., Alaya, M., Monteil, T., Lopez, Y., Talavera, G., Gonzalez, J., Granqvist, N., Kellil, M., Ganem, H., & Väisänen, T. (2014), “A Survey on M2M Service Networks,” *Computers*, 3, 130–173.

Lee, E. A. (2008), “Cyber Physical Systems: Design Challenges,” *Proceedings of the 11th IEEE Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, 363–369.

Lee, G. M., Crespi, N., Choi, J. K., & Boussard, M. (2013), "Internet of Things," in: Bertin, E., Crespi, N., & Magedanz, T. (eds.): *Evolution of Telecommunication Services: The Convergence of Telecom and Internet: Technologies and Ecosystems*, Berlin, Heidelberg: Springer, vol. 7768, pp. 257–282.

Liang, Q., Nittel, S., Hahmann, T., Informatics, S., & Science, I. (2016a), "From Data Streams to Fields: Extending Stream Data Models with Field Data Types," in: Miller, J. A., O'Sullivan, D., & Wiegand, N. (eds.): *Geographic Information Science*, Cham: Springer International Publishing, vol. 9927, pp. 178–192.

Liang, S., Huang, C., & Khalafbeigi, T. (2016b), "OGC SensorThings API Part 1: Sensing - Version 1.0," *OGC® Implementation Standard*, 15-078r6.

Liang, S. & Khalafbeigi, T. (2019), "OGC SensorThings API Part 2 – Tasking Core - Version 1.0," *OGC® Implementation Standard*, 17-079r1.

Libelium (2018), "Waspmote Plug & Sense! Sensor Guide v7.5," *available at* `http://www.libelium.com/development/plug-sense/documentation/waspmote-plug-sense-sensors-guide/` (Accessed: 2019-05-15).

Libelium (2019), "Waspmote," *available at* `http://www.libelium.com/products/waspmote/` (Accessed: 2019-02-15).

Liebig, T. & Morik, K. (2013), "Insight: Report on end-user requirements, test data, and on prototype definitions," TU Dortmund and Insight Consortium Members.

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., & Zhao, W. (2017), "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal*, 4, 1125–1142.

LocationTech (2016), "JTS Topology Suite - Features," *available at* `https://locationtech.github.io/jts/jts-features.html` (Accessed: 2019-05-15).

Logre, I., Mosser, S., Collet, P., & Riveill, M. (2014), "Sensor Data Visualisation: A Composition-Based Approach to Support Domain Variability," in: Cabot, J. & Rubin, J. (eds.): *Modelling Foundations and Applications. Proceedings of the 10th European Conference (ECMFA 2014)*, Cham: Springer, pp. 101–116.

Lopez, M. A., Lobato, A., & Duarte, O. C. M. B. (2016), "A Performance Comparison of Open-Source Stream Processing Platforms," in: *Proceedings of the 2016 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6.

Luzuriaga, J. E., Perez, M., Boronat, P., Cano, J. C., Calafate, C., & Manzoni, P. (2015), "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks," *Proceedings of the 12th Annual IEEE Consumer Communications and Networking Conference (CCNC 2015)*, 931–936.

Maheshwari, P. & Pang, M. (2005), "Benchmarking message-oriented middleware: TIB/RV versus SonicMQ," *Concurrency Computation Practice and Experience*, 17, 1507–1526.

Manley, J. H. (1974), "Embedded computers: software cost considerations," in: *Proceedings of the National Computer Conference 1974*, pp. 343—-347.

Mapbox (2018), "Geobuf," *available at* `https://github.com/mapbox/geobuf` (Accessed: 2018-11-20).

Maréchaux, J.-L. (2006), "Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus," *IBM Developer Works*, 1269—-1275.

Marjani, M., Nasaruddin, F., Gani, A., Karim, A., Hashem, I. A. T., Siddiqa, A., & Yaqoob, I. (2017), "Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges," *IEEE Access*, 5, 5247–5261.

Marz, N. & Warren, J. (2015), *Big Data: Principles and Best Practices of Scalable Realtime Data Systems*, Greenwich, USA: Manning Publications Co.

Masolo, C., Borgo, S., Gangemi, A., Guarino, N., & Oltramari, A. (2003), "WonderWeb Deliveralbe D18 - Ontology Library," Laboratory for Applied Ontology, ISTC-CNR, Trento, Italy.

Matheus, R., Janssen, M., & Maheshwari, D. (2018), "Data science empowering the public: Data-driven dashboards for transparent and accountable decision-making in smart cities," *Government Information Quarterly*, 1–9.

Mattern, F. & Flörkemeier, C. (2010), "Vom Internet der Computer zum Internet der Dinge," *Informatik-Spektrum*, 33, 107–121.

Mattheis, S., Khaled Al-Zahid, K., Engelmann, B., Hildisch, A., Holder, S., Lazarevych, O., Mohr, D., Sedlmeier, F., & Zinck, R. (2014), "Putting the car on the map: A scalable map matching system for the Open Source Community," *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, P-232, 2109–2119.

Mayer, S. & Guinard, D. (2011), "An Extensible Discovery Service for Smart Things," in: *Proceedings of the 2nd International Workshop on Web of Things (WoT'11)*, pp. 1–6.

McClure, J. & Dobs, J. (2015), "XenQTT," *available at* `https://github.com/TwoGuysFromKabul/xenqtt` (Accessed: 2019-05-15).

McCullough, A., Barr, S., & James, P. (2011), "A Typology of Real-Time Parallel Geoprocessing for the Sensor Web Era," in: Foerster, T., Broering, A., Baranski, B., Pross, B., Stasch, C., Everding, T., & Maes, S. (eds.): *Proceedings of the*

Workshop on Integrating Sensor Web and Web-based Geoprocessing at AGILE 2011 conference, pp. 1–5.

Medina, C. A., Perez, M. R., & Trujillo, L. C. (2017), “IoT Paradigm into the Smart City Vision: A Survey,” in: *Proceedings of the 2017 IEEE International Conference on Internet of Things (iThings), IEEE Green Computing and Communications (GreenCom), IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 695–704.

Menke, K., Pirelli, L., Smith, R., & van Hoesen, J. (2015), *Mastering QGIS*, Birmingham: Packt Publishing Ltd.

Michelson, B. M. (2006), “Event-Driven Architecture Overview: Event-Driven SOA Is Just Part of the EDA Story,” Patricia Seybold Group and Elemental Links Inc.

Morales, J. & Garcia, M. (2015), “GeoSmart Cities: Event-driven geoprocessing as enabler of smart cities,” in: *Processing of the 2015 IEEE First International Smart Cities Conference (ISC2)*, pp. 1–6.

Mourelatos, A. P. D. (1978), “Events, processes, and states,” *Linguistics and Philosophy*, 2, 415–434.

Müller, M. & Pross, B. (2018), “OGC WPS 2.0.2 Interface Standard: Corrigendum 2,” *OGC® Implementation Standard*, 14-065r2.

Naghavi, M. (2012), “Cloud Computing as an Innovation in GIS & SDI: Methodologies, Services, Issues and Deployment Techniques,” *Journal of Geographic Information System*, 4, 597–607.

Naik, N. (2017), “Choice of Effective Messaging Protocols for IoT Systems: MQTT, CoAP, AMQP and HTTP,” in: *Proceedings of the 2017 IEEE International Symposium on Systems Engineering (ISSE 2017)*, pp. 1–7.

Nastase, L. (2017), “Security in the Internet of Things: A Survey on Application Layer Protocols,” in: *Proceedings of the 21st International Conference on Control Systems and Computer (CSCS 2017)*, pp. 659–666.

Nittel, S. (2009), “A Survey of Geosensor Networks: Advances in Dynamic Environmental Monitoring,” *Sensors*, 9, 5664–5678.

Nittel, S. (2015), “Real-time Sensor Data Streams,” *SIGSPATIAL Special*, 7, 22–28.

Nittel, S., Stefanidis, A., Cruz, I., Egenhofer, M., Goldin, D., Howard, A., Labrinidis, A., Madden, S., Voisard, A., & Worboys, M. (2004), “Report from the First Workshop on Geo Sensor Networks,” *ACM SIGMOD Record*, 33, 141–144.

O’Hara, J. (2007), “Toward a commodity enterprise middleware,” *ACM Queue*, 5, 48–55.

Oki, B. M., Pfluegl, M., Siegel, A., & Skeen, D. (1993), “The Information Bus - An Architecture for Extensible Distributed Systems,” in: *Proceedings of the 14th ACM symposium on Operating systems principles (SOSP’93)*, pp. 58–68.

O’Leary, N. & Piper, A. (2019), “MQTT,” *available at* `http://mqtt.org/` (Accessed: 2019-05-15).

Olsson, J. (2014), “6LoWPAN demystified,” Texas Instruments Inc, Dallas.

Oracle (2018), “SDO-GEOM Package (Geometry),” *Oracle Spatial Developer’s Guide*.

O’Sullivan, D. & Igoe, T. (2004), *Physical Computing: Sensing and Controlling the Physical World with Computers*, Boston: Thomson Course Technology.

Overmars, M. H. & van Leeuwen, J. (1981), “Maintenance of Configurations in the Plane,” *Journal of Computer and System Sciences*, 23, 166–204.

Özgövde, A. & Grüninger, M. (2010), “Foundational Process Relations in Bio-Ontologies,” in: *Proceedings of the 6th International Conference on Formal Ontology in Information Systems (FOIS 2010)*, pp. 243–256.

Özsoyoglu, G. & Snodgrass, R. T. (1995), “Temporal and Real-Time Databases: A Survey,” *IEEE Transactions on Knowledge and Data Engineering*, 7, 513–532.

Pakkala, D. & Latvakoski, J. (2007), “Distributed Service Platform for Adaptive Mobile Services,” *International Journal of Pervasive Computing and Communications*, 2, 135–148.

Palattella, M. R., Dohler, M., Grieco, A., Rizzo, G., Torsner, J., Engel, T., & Ladid, L. (2016), “Internet of Things in the 5G Era: Enablers, Architecture, and Business Models,” *IEEE Journal on Selected Areas in Communications*, 34, 510–527.

Peuquet, D. & Qian, L. (1996), “An Integrated Datatbase Design for Temporal GIS,” in: *Proceedings of the 7th International Symposium on Spatial Data Handling*, pp. 21–31.

Peuquet, D. J. (1994), “It’s about Time: A Conceptual Framework for the Representation of Temporal Dynamics in Geographic Information Systems.” *Annals of the Association of American Geographers*, 84, 441–461.

Peuquet, D. J. (1999), “Time in GIS and geographical databases,” in: Longley, P., Goodchild, M., Maguire, D., & Rhind, D. (eds.): *Geographical Information Systems: Principles and Technical Issues*, New York: John Wiley & Sons, vol. 1, 2nd ed., pp. 91–103.

Peuquet, D. J. (2005), “Theme section on advances in spatio-temporal analysis and representation,” *ISPRS Journal of Photogrammetry and Remote Sensing*, 60, 1–2.

Peuquet, D. J. & Duan, N. (1995), “An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographical data,” *International Journal of Geographical Information Systems*, 9, 7–24.

Polous, K. (2016), “Event Cartography: A New Perspective in Mapping,” Ph.d. thesis, Technische Universität München.

Poorazizi, M. E. & Hunter, A. (2015), “Evaluation of Web Processing Service Frameworks,” *OSGeo Journal*, 14, 1–24.

Portele, C. (2007), “OpenGIS® Geography Markup Language (GML) Encoding Standard - Version 3.2.1,” *OpenGIS® Standard*, OGC 07-036.

PostGIS Development Group (2018), “PostGIS EWKB, EWKT and Canonical Forms,” *available at* `http://postgis.net/docs/manual-2.5/using{_}postgis{_}dbmanagement.html{#}EWKB{_}EWKT` (Accessed: 2019-05-15).

Przybylla, M. & Romeike, R. (2014), “Physical Computing in Computer Science Education,” in: *Proceedings of the 9th Workshop in Primary and Secondary Computing Education (WiPSCE’14)*, pp. 136–137.

Pustejovsky, J. (1991), “The syntax of event structure,” *Cognition*, 41, 47–81.

PyWPS Development Team (2009), “Python Web Processing Service (PyWPS), Software, Version 4.0.0,” *available at* `http://pywps.org` (Accessed: 2019-05-15).

QGIS Project (2018), “PyQGIS developer cookbook, Release 2.18,” QGIS Project.

Ray, P. P. (2018), “A survey on Internet of Things architectures,” *Journal of King Saud University - Computer and Information Sciences*, 30, 291–319.

Razzaque, M. A., Milojevic-Jevric, M., Palade, A., & Cla, S. (2016), “Middleware for Internet of Things: A Survey,” *IEEE Internet of Things Journal*, 3, 70–95.

Resch, B., Blaschke, T., & Mittlboeck, M. (2010), “Live Geography : Interoperable Geo-Sensor Webs Facilitating the Vision of Digital Earth,” *International Journal On Advances in Networks and Services*, 3, 323–332.

Resnick, P. (2001), “Internet Message Format,” *Network Working Group Standards Track*, RFC 2822.

Ribeiro, A., Vieira, J., Sousa, V., & Cardoso, A. (2015), “Demonstration of GIS web-based platform for experimentation supported by geosensors in a WSN,” in: *Proceedings of the 3rd Experiment International Conference (exp.at 2015)*, pp. 137–138.

Rieke, M., Bigagli, L., Herle, S., Jirka, S., Kotsev, A., Liebig, T., Malewski, C., Paschke, T., & Stasch, C. (2018), "Geospatial IoT—The Need for Event-Driven Architectures in Contemporary Spatial Data Infrastructures," *ISPRS International Journal of Geo-Information*, 7, 1–29.

Rios, L. G. & Diguez, J. A. I. (2014), "Big Data Infrastructure for analyzing data generated by Wireless Sensor Networks," in: *Proceedings of the 2014 IEEE International Congress on Big Data*, pp. 816–823.

Sachs, K., Kounev, S., Appel, S., & Buchmann, A. (2009), "Benchmarking of Message-Oriented Middleware," in: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, pp. 1–2.

Saint-Andre, P. (2004), "Extensible Messaging and Presence Protocol (XMPP): Core," *Network Working Group Standards Track*, RFC 3920.

Saint-Andre, P. (2011), "Extensible Messaging and Presence Protocol (XMPP): Core," *IETF Standards Track*, RFC 6120.

Saint-Andre, P. (2018), "XEP-0045: Multi-User Chat," *XEP Standards Track*, XEP-0045.

Saint-Andre, P. & Cridland, D. (2016), "XEP-0001: XMPP Extension Protocols," *XEP Standards Track*, XEP-0001.

Saint-Andre, P., Smith, K., & Tronçon, R. (2009), *XMPP: The Definitive Guide Building Real-Time Applications with Jabber Technologies*, Sebastopol: O'Reilly Media Inc.

Santos, P., Rodrigues, J., Cruz, S., Lourenco, T., D'Orey, P., Luis, Y., Rocha, C., Sousa, S., Crisóstomo, S., Queirós, C., Sargento, S., Aguiar, A., & Barros, J. (2018), "PortoLivingLab: An IoT-based Sensing Platform for Smart Cities," *IEEE Internet of Things Journal*, 5, 523–532.

Sauter, M. (2015), *Grundkurs Mobile Kommunikationssysteme*, Wiesbaden: Springer Vieweg, 6th ed.

Schaeffer, B., Baranski, B., Foerster, T., & Brauner, J. (2012), "A Service-Oriented Framework for Real-Time and Distributed Geoprocessing," in: Bocher, E. & Neteler, M. (eds.): *Geospatial Free and Open Source Software in the 21st Century. Proceedings of the first Open Source Geospatial Research Symposium (OGRS 2009)*, Berlin, Heidelberg: Springer, pp. 3–20.

Schulte, R. & Natis, Y. (2003), "Event-Driven Architecture Complements SOA," *Decision Framework*, DF-20-1154, 1–7.

Schut, P. (2007), "OpenGIS Web Processing Service - Version 1.0.0," *OGC® Standard*, 05-007r7.

Seeger, H. (1999), “Spatial referencing and coordinate systems,” in: Longley, P., Goodchild, M., Maguire, D., & Rhind, D. (eds.): *Geographical Information Systems: Principles and Technical Issues*, New York: John Wiley & Sons, 2nd ed., pp. 427–436.

Selva, A. (2018), “Moquette,” https://github.com/andsel/moquette*available at* `https://github.com/andsel/moquette` (Accessed: 2019-02-08).

Semtech Corporation (2013), “SX1272/3/6/7/8 LoRa Modem,” *Designer’s Guide*, AN1200.13, 1–9.

Serbanati, A., Medaglia, C. M., & Ceipidor, U. B. (2011), “Building blocks of the internet of things: State of the art and beyond,” in: Turcu, C. (ed.): *Deploying RFID - Challenges, Solutions, and Open Issues*, IntechOpen, pp. 351–366.

Shekhar, S., Jiang, Z., Ali, R. Y., Eftelioglu, E., Tang, X., Gunturi, V. M. V., & Zhou, X. (2015), “Spatiotemporal Data Mining: A Computational Perspective,” *ISPRS International Journal of Geo-Information*, 4, 2306–2338.

Shelby, Z., Hartke, K., & Bormann, C. (2014), “The Constrained Application Protocol (CoAP),” *IETF Standards Track*, RFC 7252.

Shi, D., Xu, H., Su, R., & You, Z. (2010), “A GEO-related IOT Applications Platform Based On Google Map,” in: *Proceedings of the 7th IEEE International Conference on E-Business Engineering*, IEEE, pp. 380–384.

Shukla, A. & Simmhan, Y. (2017), “Benchmarking Distributed Stream Processing Platforms for IoT Applications,” in: Nambiar, R. & Poess, M. (eds.): *Performance Evaluation and Benchmarking. Traditional - Big Data - Internet of Things. Revised Selected Papers of the 8th TPC Technology Conference (TPCTC 2016)*, Cham: Springer International Publishing, pp. 90–106.

Simonis, I. (2006), “OGC® Sensor Alert Service Candidate Implementation Specification,” *OpenGIS® Best Practices*, 06-028r3.

Simonis, I. (2008), “OGC® Sensor Web Enablement Architecture,” *OGC® Best Practice*, 06-021r4.

Simonis, I. & Echterhoff, J. (2006), “Draft OpenGIS® Web Notification Service Implementation Specification - Version 0.0.9,” *OpenGIS® Best Practices Paper*, OGC 06-095.

Simonis, I. & Echterhoff, J. (2011), “OGC® Sensor Planning Service Implementation Standard - Version 2.0,” *OpenGIS® Implementation Standard*, OGC 09-000.

Smith, B., Almeida, M., Bona, J., Brochhausen, M., Ceusters, W., Courtot, M., Dipert, R., Goldfain, A., Grenon, P., Hastings, J., Hogan, W., Jacuzzo, L., Johansson, I., Mungall, C., Natale, D., Neuhaus, F., Overton, J., Petosa, A., Rovetto, R., Ruttenberg, A., Ressler, M., Rudniki, R., & Schulz, S. (2015), “Basic Formal Ontology 2.0 - Specification and User’s Guide,” .

Sowa, J. F. (2000), *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Pacific Grove: Brooks Cole Publishing Co.

Stanford-Clark & Truong, H. L. (2013), “MQTT For Sensor Networks (MQTT-SN) Protocol Specification - Version 1.2,” IBM Corporation.

Stonebraker, M., Çetintemel, U., & Zdonik, S. (2005), “The 8 Requirements of Real-Time Stream Processing,” *ACM SIGMOD Record*, 34, 42–47.

Strobl, C. (2008), “Dimensionally Extended Nine Intersection Model (DE-9IM),” in: Shekhar, S. & Xiong, H. (eds.): *Encyclopedia of GIS*, Boston: Springer, pp. 240–245.

Tan, Y., Vuran, M. C., & Goddard, S. (2009), “Spatio-Temporal Event Model for Cyber-Physical Systems,” in: *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops*, pp. 44–50.

Tanenbaum, A. S. & Wetherall, D. J. (2013), *Computer Networks*, London: Pearson Education, 5th ed.

Teklemariam, G. K. (2018), “CoAP-Based Enablers for Designing Efficient and Reliable Distributed IoT Applications,” Doctoral thesis, Ghent University.

Terhorst, A., Moodley, D., Simonis, I., Frost, P., McFerren, G., Roos, S., & van den Bergh, F. (2008), “Using the Sensor Web to Detect and Monitor the Spread of Vegetation Fires in Southern Africa,” in: Nittel, S., Labrinidis, A., & Stefanidis, A. (eds.): *GeoSensor Networks. Revised Selected and Invited Papers of the Second International Conference, (GSN 2006)*, Berlin, Heidelberg: Springer, pp. 239–251.

Terracotta Inc. (2019a), “Cron Trigger Tutorial,” *available at* `http://www.quartz-scheduler.org/documentation/quartz-2.x/tutorials/crontrigger.html` (Accessed: 2019-02-07).

Terracotta Inc. (2019b), “Quartz Job Schedular,” *available at* `http://www.quartz-scheduler.org/` (Accessed: 2019-02-08).

Thakur, G. S., Bhaduri, B. L., Piburn, J. O., Sims, K. M., Stewart, R. N., & Urban, M. L. (2015), “PlanetSense: A Real-time Streaming and Spatio-temporal Analytics Platform for Gathering Geo-spatial Intelligence from Open Source Data,” in: *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL’15)*, pp. 1–4.

Theorin, A., Bengtsson, K., Provost, J., Lieder, M., Johnsson, C., Lundholm, T., & Lennartson, B. (2017), “An Event-Driven Manufacturing Information System Architecture for Industry 4.0,” *International Journal of Production Research*, 55, 1297–1311.

Tönjes, R., Ali, M., Bargnahi, P., Ganea, S., Ganz, F., Haushwirth, F., Kjaergaard, B., Kümper, D., Mileo, A., Nechifor, S., Sheth, A., & Tsiatsis, V. (2014), “Real-Time IoT Stream Processing and Large-scale Data Analytics for Smart City Applications,” in: *Proceedings of the European Conference on Networks and Communications 2014*, pp. 1–5.

Traversat, B., Abdelaziz, M., Doolin, D., Duigou, M., Hugly, J., & Pouyoul, E. (2003), “Project JXTA-C: Enabling a Web of Things,” in: *Proceedings of the 36th Annual Hawaii International Conference on System Sciences.*, p. 282.

Trieloff, C., Hara, J. O., Atwell, M., & Contributers (2008), *AMQP - Advanced Message Queuing Protocol - Protocol Specification*.

van Bentem, A. (2018), “Best practices when sending GPS location data,” *available at* `https://www.thethingsnetwork.org/forum/t/best-practices-when-sending-gps-location-data/1242` (Accessed: 2019-02-15).

van der Zee, E. & Scholten, H. (2013), “Application of geographical concepts and spatial technology to the Internet of Things. The role of location in real-time smart environments.” *FEWEB Research Memorandum*, 2013-33.

van der Zee, E. & Scholten, H. (2014), “Spatial Dimensions of Big Data: Application of Geographical Concepts and Spatial Technology to the Internet of Things,” in: Bessis, N. & Dobre, C. (eds.): *Big Data and Internet of Things: A Roadmap for Smart Environments*, Cham: Springer International Publishing, pp. 137–168.

Venkateswara Rao, K., Govardhan, A., & Chalapati Rao, K. (2012), “Spatiotemporal Data Mining: Issues, Tasks And Applications,” *International Journal of Computer Science & Engineering Survey (IJCSES)*, 3, 39–52.

Vilain, M. B. (1982), “A System for Reasoning about Time,” *Proceedings of Second National Conference on Artificial Intelligence (AAAI-82)*, 197–201.

Vitolo, C., Elkhatib, Y., Reusser, D., Macleod, C. J., & Buytaert, W. (2015), “Web technologies for environmental Big Data,” *Environmental Modelling & Software*, 63, 185–198.

Vretanos, P. (2014), “OGC Filter Encoding 2.0 Encoding Standard – With Corrigendum - Version 2.0.3,” *OGC® Implementation Standard*, 09-026r2.

Waher, P. (2017), "XEP-0323: Internet of Things - Sensor Data," *XEP Standards Track*, XEP-0323.

Walter, K. & Nash, E. (2009), "Coupling Wireless Sensor Networks and the Sensor Observation Service – Bridging the Interoperability Gap," in: *Proceedings of the 12th AGILE International Conference on Geographic Information Science*, pp. 1–9.

Wan, J., Li, D., Zou, C., & Zhou, K. (2012), "M2M Communications for Smart City: An Event-Based Architecture," in: *Proceedings of the IEEE 12th International Conference on Computer and Information Technology (CIT'12)*, pp. 895–900.

Wang, F., Hu, L., Zhou, J., & Zhao, K. (2015), "A Survey from the Perspective of Evolutionary Process in the Internet of Things," *International Journal of Distributed Sensor Networks*, 11, 1–9.

Wang, H., Xiong, D., Wang, P., & Liu, Y. (2017), "A Lightweight XMPP Publish/Subscribe Scheme for Resource-Constrained IoT Devices," *IEEE Access*, 5, 16393–16405.

Wang, S. (2010), "A CyberGIS Framework for the Synthesis of Cyberinfrastructure, GIS, and Spatial Analysis," *Annals of the Association of American Geographers*, 100, 535–557.

Wang, S., Wan, J., Li, D., & Zhang, C. (2016), "Implementing Smart Factory of Industrie 4.0: An Outlook," *International Journal of Distributed Sensor Networks*, 12, 1–10.

Weiser, M. (1991), "The Computer for the 21st Century," *Scientific American*, 265, 94–104.

Werner-Allen, G., Lorincz, K., Welsh, M., Marcillo, O., Johnson, J., Ruiz, M., & Lees, J. (2006), "Deploying a Wireless Sensor Network on an Active Volcano," *IEEE Internet Computing*, 10, 18–25.

Westerholt, R. & Resch, B. (2015), "Asynchronous Geospatial Processing: An Event-Driven Push-Based Architecture for the OGC Web Processing Service," *Transactions in GIS*, 19, 455–479.

Westfall, J. (2010), "Common Alerting Protocol Version 1.2," *OASIS Standard*.

Wiener, P., Stein, M., Seebacher, D., Bruns, J., Frank, M., Simko, V., Zander, S., & Nimis, J. (2016), "BigGIS: A Continuous Refinement Approach to Master Heterogeneity and Uncertainty in Spatio-Temporal Big Data," in: *Proceedings of the 24th International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL'16)*, pp. 1–4.

Winters, R. M., Tsuchiya, T., Lerner, L. W., & Freeman, J. (2016), “Multi-Modal Web-Based Dashboards for Geo-Located Real-Time Monitoring,” in: *Proceedings of the 2nd Web Audio Conference (WAC-2016)*, pp. 1–6.

Worboys, M. (2005), “Event-oriented approaches to geographic phenomena,” *International Journal of Geographical Information Science*, 19, 1–28.

Worboys, M. & Hornsy, K. (2004), “From Objects to Events: GEM, the Geospatial Event Model,” in: Egenhofer, M. J., Freska, C., & Miller, H. (eds.): *Geographic Information Science. Proceedings of the 3rd International Conference GIScience 2004*, Berlin, Heidelberg: Springer, pp. 327–343.

XMPP Software Foundation (2018), “XMPP Extension Protocols,” *available at* `https://xmpp.org/extensions/` (Accessed: 2018-12-10).

Yamaguchi, T. (2017), “MQTT-SN over UDP and XBee,” *available at* `https://github.com/ty4tw/MQTT-SN` (Accessed: 2019-02-15).

Yavari, A., Jayaraman, P. P., & Georgakopoulos, D. (2016), “Contextualised Service Delivery in the Internet of Things,” in: *Proceedings of the IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 454–459.

Yick, J., Mukherjee, B., & Ghosal, D. (2008), “Wireless sensor network survey,” *Computer Networks*, 52, 2292–2330.

Yokotani, T. & Sasaki, Y. (2017), “Comparison with HTTP and MQTT on required network resources for IoT,” in: *Proceedings of the International Conference on Control, Electronics, Renewable Energy, and Communications (ICCEREC 2016)*, pp. 1–6.

Yue, P. & Jiang, L. (2014), “BigGIS: How Big Data Can Shape Next-Generation GIS,” in: *Proceedings of the 3rd International Conference on Agro-Geoinformatics*, pp. 1–6.

Yue, P., Zhang, C., Zhang, M., & Jiang, L. (2014), “Sensor Web Event Detection and Geoprocessing over Big Data,” in: *Proceedings of the 2014 IEEE Geoscience and Remote Sensing Symposium*, pp. 1401–1404.

Zaborovsky, V., Lukashin, A., & Muliukha, V. (2016), “Robotic Operations Network: Cyber-Physics Framework and Cloud Centric Software Architecture,” in: Rawat, D., Rodrigues, J., & Stojmenovic, I. (eds.): *Cyber-Physical Systems: From Theory to Practice*, Boca Raton: CRC Press, pp. 259–282.

Zafari, F., Gkelias, A., & Leung, K. (2017), “A Survey of Indoor Localization Systems and Technologies,” *IEEE Communications Surveys & Tutorials*, 1–32.

Zakaria, S., Rey, G., Mohamed, E., Lavirotte, S., Fazziki Abdelaziz, E., & Tigli, J.-Y. (2015), “Smart Geographic object: Toward a new understanding of GIS Technology in Ubiquitous Computing,” *IJCSI International Journal of Computer Science Issues*, 12, 52–61.

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014), “Internet of Things for Smart Cities,” *IEEE Internet of Things Journal*, 1, 22–32.

Zdraveski, V., Mishev, K., Trajanov, D., & Kocarev, L. (2017), “ISO-Standardized Smart City Platform Architecture and Dashboard,” *IEEE Pervasive Computing*, 16, 35–43.

Zhang, Y., Duan, L., & Chen, J.-L. (2014), “Event-driven SOA for IoT services,” in: *Proceedings of the 2014 IEEE International Conference on Services Computing (SCC 2014)*, pp. 629–636.

Zheng, Y. U. (2015), “Trajectory Data Mining : An Overview,” *ACM Transaction on Intelligent Systems and Technology*, 6, 29.

List of Abbreviations

4IM	Four-Intersection Model
9IM	Nine-Intersection Model
6LoWPAN	IPv6 over Low-power WPAN
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
BLE	Bluetooth Low Energy
CAP	Common Alerting Protocol
CEP	Complex Event Processing
CoAP	Constrained Application Protocol
CPS	Cyber-Physical System
CRS	Coordinate Reference System
CSP	Complex Stream Processing
DE-9IM	Dimensionally Extended Nine-Intersection Model
DSP	Distributed Stream Processing
DTLS	Datagram Transport Layer Security
EDA	Event-driven Architecture
EML	Event Pattern Markup Language
EPSG	European Petroleum Survey Group Geodesy
ESP	Event Stream Processing
ETRS89	European Terrestrial Reference System 1989
EWKT	Extended Well-Known Text
EXI	Efficient XML Interchange

FES	OGC Filter Encoding Standard
FTP	File Transfer Protocol
GeoMQTT	Geospatial MQTT
GeoMQTT-SN	GeoMQTT for Sensor Networks
GIS	Geographic Information System
GIScience	Geographic Information Science
GML	Geography Markup Language
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GSN	Geo Sensor Network
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
IM	Instant Messaging
INSPIRE	Infrastructure for Spatial Information in Europe
IIoT	Industrial IoT
IoT	Internet-of-Things
IP	Internet Protocol
IPv6	Internet Protocol Version 6
IPC	Interprocess Communication
IRC	Internet Relay Chat
ISM	Industrial, Scientific and Medical Band
ISO	International Organization for Standardization
JMS	Java Message Service
JSON	JavaScript Object Notation

KVP	Key-Value Pair
LBS	Location-Based Service
LoRa	Long Range
LoRaWAN	Long Range Wide Area Network
LPWAN	Low Power Wide Area Network
M2M	Machine-to-Machine
MEP	Message Exchange Pattern
MF	Moving Feature Access
MIME	Multipurpose Internet Mail Extensions
MOM	Message-Oriented Middleware
MQTT	Message Queuing Telemetry Transport
MQTT-SN	MQTT for Sensor Networks
NFC	Near Field Communication
O&M	Observations and Measurements
OGC	Open Geospatial Consortium
OODA	Observe-Orient-Decide-Act
OS	Operating System
OWS	OGC Web Service
QoS	Quality of Service
REST	Representational State Transfer
RFID	Radio-Frequency Identification
ROA	Resource-oriented Architecture
RPC	Remote Procedure Call
SAS	Sensor Alert Service
SASL	Simple Authentication and Security Layer
SDI	Spatial Data Infrastructure

SensorML	Sensor Model Language
SES	Sensor Event Service
SMTP	Simple Mail Transfer Protocol
SN	Sensor Network
SOA	Service-oriented Architecture
SOAP	Simple Object Access Protocol
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SRID	Spatial Reference System Identifier
SSDI	Sensor and Spatial Data Infrastructure
SWE	Sensor Web Enablement
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
UOM	Unit of Measurement
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UWB	Ultra Wide Band
WFS	Web Feature Service
WGS84	World Geodetic System 1984
WGSN	Wireless Geo Sensor Network
WKB	Well-Known Binary
WKT	Well-Known Text
WLAN	Wireless Local Area Network
WMAN	Wireless Metropolitan Area Network

WMS	Web Map Service
WNAN	Wireless Neighborhood Area Network
WNS	Web Notification Service
WoT	Web of Things
WPAN	Wireless Personal Area Network
WPS	Web Processing Service
WS-N	OASIS Web Services Notification
WSN	Wireless Sensor Network
WWAN	Wireless Wide Area Network
WWW	World Wide Web
XEP	XMPP Extension Protocol
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

List of Figures

LIST OF TABLES

Veröffentlichungen des Geodätischen Instituts der Rheinisch-Westfälischen Technischen Hochschule Aachen

Die 'Veröffentlichungen des Geodätischen Instituts ... ' wurden bis 2009 im Rahmen des wissenschaftlichen Schriftenaustausches verteilt und nicht über den Buchhandel vertrieben. Einzelne Exemplare können - soweit nicht vergriffen - gegen Unkostenerstattung beim Geodätischen Institut der RWTH Aachen (*Mies-van-der-Rohe-Str. 1, 52074 Aachen*) bezogen werden. Die Hefte Nr. 1 -29 können auf Anfrage ausgeliefert werden. Die Hefte ab Nr. 64 stehen nur noch Online zur Verfügung.

Nr. 30	PHILIPS, J.:	Ein photogrammetrisches Aufnahmesystem zur Untersuchung dynamischer Vorgänge im Nahbereich (Diss. 1981)[2]
Nr. 31	BUSCH, W.:	Eigenschaften stationärer hydrostatischer Präzisions-Höhenmeßsysteme für kontinuierliche Langzeitbeobachtungen, untersucht an einem neuentwickelten Schlauch-wagensystem (Diss. 1980)[3]
Nr. 32	FORAMITTI, H.:	Bezugsebenen für Projektionen von Kulturgutbeständen (1981)[2]
Nr. 33	REUTER, R.:	Integralformeln der Einheitssphäre und harmonische Splinefunktionen (Diss. 1982)[3]
Nr. 34	URBAN, G.; JANSEN, M. (Hrsg.):	Dokumentation in der Archäologie- Techniken, Methoden, Analysen. Veröffentlichung zum Seminar am 5-6. Dezember 1981 in Aachen (1982)[3]
Nr. 35	WOLF, H.; RINNER, K.:	Festkolloquium aus Anlaß des 70. Geburtstages von o.Prof.Dr.techn. Fritz Löschner am 27. Mai 1982[1]
Nr. 36	KAHLER, D.:	Untersuchungen zur Varianzschätzung von Punktkoordinaten im terrestrisch photogrammetrischen Stereomodell (Habil. -schrift 1984)[3]
Nr. 37	GOTTWALD, R.:	Zur Genauigkeitssteigerung und Erstellung eines automatisierten Datenflusses beim trigonometrischen Nivellement mit kurzen Zielweiten (Diss. 1985)[3]

Nr. 38	SCHWARZ, W.:	Zur Ermittlung der integralen Temperatur der Atmosphäre mit Ultraschall für Refraktionsbestimmungen im Nahbereich (Diss. 1985)[2]
Nr. 39	BRAUER, H.:	Clusteranalytische Methoden zur Strukturierung einer städtischen Bodenpreissammlung (Diss. 1986)[2]
Nr. 40	WITTE, B.; u.a.:	Festschrift zur Vollendung des 65. Lebensjahres und zur Emeritierung von o.Prof.Dr.-Ing. E. Hektor (1986)[2]
Nr. 41	WÜLLER, H.:	Entwicklung und Untersuchung eines Rotationsnivellierinstrumentes und einer photoelektrischen Nivellierlatte zur Automatisierung des geometrischen Nivellements (Diss. 1987)[2]
Nr. 42	SCHAFFELD, H.G.:	Eine Finite-Elemente-Methode und ihre Anwendung zur Erstellung von Digitalen Geländemodellen (Diss. 1987)[1]
Nr. 43	KAMPMANN, G.:	Zur kombinativen Norm-Schätzung mit Hilfe von L1-, L2-, und der Boskovic-Laplace- Methode mit den Mitteln der Linearen Programmierung (Diss. 1988)[3]
Nr. 44	BENNING, W.:	Programmsystem KAFKA - Komplexe Analyse flächenhafter Kataster-Aufnahmen, Modell und Anwendung der Ausgleichung hybrider Lagemessungen (2. Auflage 1989)[3]
Nr. 45	BENNING, W.:	3D-Adjustment of Hybrid Geodetic Measurements (1990)[3]
Nr. 46	JAKOBS, M.:	Entwicklung eines elektro-mechanischen Aligniersystems zur meßtechnischen Erfassung räumlicher Verformungszustände in der Bauwerksüberwachung (Diss. 1990)[2]
Nr. 47	SCHOLZ, T.:	Zur Kartenhomogenisierung mit Hilfe strenger Ausgleichungsmethoden (Diss. 1992)[2]

Nr. 48	YIN, L.:	Untersuchungen zur Arbeitsweise und Genauigkeit von elektrooptischen Distanzmessern nach dem Impulslaufzeitverfahren (Diss. 1992)[2]
Nr. 49	LEHMKÜHLER, H.:	Die geodätische Deformationsanalyse als Mustererkennungsaufgabe (Diss. 1992)[2]
Nr. 50	SPARLA, P.:	Experimentelle Untersuchungen zur Ermittlung des Brechungsindex der Atmosphäre mit Hilfe von elektronischen Sensoren (Diss. 1993)[2]
Nr. 51	SCHMIDT, H.:	Meßunsicherheit und Vermessungstoleranz bei Ingenieurvermessungen (Diss. 1994)[2]
Nr. 52	SCHWERMANN, R.:	Geradengestützte Bildorientierung in der Nahbereichsphotogrammetrie (Diss. 1995)[2]
Nr. 53	BENNING, W. (Hrsg.):	125 Jahre Geodäsie an der RWTH Aachen, Festschrift (1995)[3]
Nr. 54	BRAESS, M.:	Strukturbasierte Merkmalszuordnung in kurzen stereoskopischen Videosequenzen (Diss. 1997)[2]
Nr. 55	AUSSEMS, T.:	Positionsschätzung von Landfahrzeugen mittels Kalman-Filterung aus Satelliten- und Koppelnavigationsbeobachtungen (Diss. 1999)[2]
Nr. 56	MÜLLER, J.:	Homogenisierung dreidimensionaler Szenarien nach der Methode der kleinsten Quadrate (Diss. 1999)[2]
Nr. 57	LIU, Z.:	Knowledge-based Text Recognition for the Automatic Interpretation of Reduced Scale Drawings (Diss. 1999)[2]
Nr. 58	BERZEN, N.:	Entwurf und Implementierung eines portablen persistenten Objektspeichers für C++ in heterogenen Rechnerumgebungen (Diss. 2000)[2]
Nr. 59	BREZING, A.:	Entwicklung eines Expertensystems zur wissensbasierten Deformationsanalyse (Diss. 2000)[2]

Nr. 60	HETTWER, J.:	Numerische Methoden zur Homogenisierung großer Geodatenbestände (Diss. 2003)^{2}
Nr. 61	SCHINDLER, R.:	Wissensbasierte Extraktion semantischer Informationen aus Vermessungsrissen (Diss. 2004)^{2}
Nr. 62	KAMPSHOFF, S.:	Integration heterogener raumbezogener Objekte aus fragmentierten Geodatenbeständen (Diss. 2005)^{3}
Nr. 63	BECKER, R.:	Differentialgeometrische Extraktion von 3D-Objektprimitiven aus terrestrischen Laserscannerdaten (Diss. 2005)^{2}
Nr. 64	LANGE, J.:	Mess- und Auswertungstechnik zur Riss- und Faserdetektion bei Betonbauteilen (Diss. 2009)
Nr. 65	OSTERMEYER, L.:	Zur effizienten Verarbeitung XML- repräsentierter Massendaten in der Normbasierten Austauschschnittstelle (Diss. 2010)
Nr. 66	LICHTENSTEIN, M.:	Strukturbasierte Registrierung von Punktwolken unter Verwendung von Bild- und Laserscannerdaten (Diss. 2011)
Nr. 67	BLANKENBACH J. (Hrsg.):	Festschrift zur Emeritierung von Univ. Prof. Dr.-Ing. W.Benning (2012)
Nr. 68	FOCKE, I.:	Geometrische Strukturanalyse von Glasfasern in Textilbeton (Diss. 2013)
Nr. 69	REAL EHRLICH, C.:	Echtzeit-Positionierung für Fußgänger innerhalb von Gebäuden auf Basis von Smartphone-Sensoren (Diss. 2018)
Nr. 70	BLUT, C.:	Mobile Augmented Reality for Semantic 3D Models. A Smartphone-based Approach with CityGML (Diss. 2019)

PREISE : 1)= EUR 10.00, 2)= EUR 12.00, 3)= EUR 15.00 zuzüglich gesetzlicher MwSt.
(STAND: 12/2018)