

Report

Data Workflow and Remote-Control Concept for IoT-based Smart Farming

Project Irrimode

by DHI WASY GmbH (associated partner)

Study Project

Maximilian Winderl

06/01/2020

Table of Contents

| | | |
|---------|--|----|
| 1 | Introduction..... | 4 |
| 1 | Project Irrimode | 6 |
| 2 | Theoretical Background | 7 |
| 2.1 | Software..... | 7 |
| 2.1.1 | Modeling Software..... | 7 |
| 2.1.2 | Programming Language and Integrated Development Environments (IDE) | 8 |
| 2.1.3 | Libraries | 9 |
| 2.1.3.1 | NumPy..... | 9 |
| 2.1.3.2 | Pandas..... | 9 |
| 2.1.3.3 | GeoPandas | 9 |
| 2.1.3.4 | Requests..... | 9 |
| 2.1.3.5 | Pickle | 10 |
| 2.1.3.6 | Leaflet Visualization Tool | 10 |
| 2.1.4 | Docker | 10 |
| 2.1.5 | Grafana..... | 11 |
| 2.2 | Standards..... | 12 |
| 2.2.1 | Open Geospatial Consortium (OGC) | 12 |
| 2.2.2 | OGC Sensor Web Enablement (SWE)..... | 12 |
| 2.2.3 | OGC Sensor Things API | 12 |
| 2.2.4 | Quality Modeling Language/ISO 19157 | 15 |
| 2.2.5 | Fraunhofer Open Source SensorThings (FROST) Server | 17 |
| 2.3 | Connectivity..... | 17 |
| 2.3.1 | Global System for Mobile Communication (GSM) | 17 |
| 2.3.1.1 | Messagebird API | 17 |
| 2.3.2 | Low Power Wide Area Network (LPWAN) | 17 |
| 2.3.2.1 | Sigfox..... | 18 |
| 2.3.2.2 | Narrow Band IoT (NB-IoT)..... | 18 |
| 2.3.2.3 | LoRa/LoRaWAN..... | 18 |
| 2.3.2.4 | The Things Network (TTN) | 19 |
| 2.3.2.5 | Cayenne Protocol..... | 21 |

| | | |
|---------|--|----|
| 3 | Implementation..... | 21 |
| 3.1 | Data Workflow..... | 21 |
| 3.1.1 | Access to Third-Party API | 24 |
| 3.1.2 | Data Storage and the SensorThings API..... | 26 |
| 3.1.3 | Data Export FROST | 29 |
| 3.1.4 | Visualization | 30 |
| 3.1.4.1 | Grafana..... | 32 |
| 3.1.4.2 | Leaflet Interactive Map..... | 32 |
| 3.2 | Remote Control Concept | 33 |
| 3.2.1 | Cellular Network (2G)..... | 35 |
| 3.2.1.1 | Hardware..... | 35 |
| 3.2.1.2 | Setup and Configuration | 35 |
| 3.2.2 | LoRa Network | 36 |
| 3.2.2.1 | Hardware..... | 37 |
| 3.2.2.2 | Setup and Configuration | 39 |
| 4 | Discussion..... | 41 |
| 4.1 | Data Workflow..... | 41 |
| 4.1.1 | Data Storage and the SensorThings API..... | 41 |
| 4.1.2 | Alternative Standards..... | 42 |
| 4.2 | Technical Improvements | 43 |
| 4.3 | Visualization..... | 44 |
| 4.4 | Remote Control Concept | 44 |
| 5 | Outlook..... | 46 |
| 6 | Conclusion | 48 |
| 7 | References..... | 49 |

1 Introduction

Water scarcity is a rising issue worldwide. In Germany, drier summers and dwindling water resources are forcing farmers to seek ways to reserve water and irrigate their crops more efficiently. The extreme summer of 2018 had severe consequences for the German farmers and concepts are needed to make sure harvests are not lost due to water stress in these extreme weather conditions. This study project, which is a collaboration between the DHI WASY GmbH and the Chair of Geoinformatics of the Technical University of Munich, aims to advance the automation of irrigation systems. The objective of the project was to solve distinct challenges that occur with the automated irrigation of crops.

Monitoring of parameters, such as climate and soil conditions, water resources, and the state of the crops, is essential to create a fully automated system. A further measure are water balance model, which provide additional data and give farmers and developers the opportunity to simulate different scenarios. Gathering all these information raises the question how to receive, store and use this data effectively to optimize the water usage. Resilient and effective data workflows are crucial to develop optimization algorithms that are user-friendly, transferable and transparent. Different providers for sensor technologies and equipment have distinct ways of delivering their data, thus causing interoperability issues. Moreover, model results are usually delivered in formats that diverge greatly from sensor data, although it is crucial for validation, comparison, and visualization that model and sensor data are available in the same storage format. In the scope of this project, it is examined if data handling problems arising with different data sources can be resolved by transforming all available data to an open web standard.

All parties agreed beforehand that the so-called SensorThings API standard by the Open Geospatial Consortium (OGC) seems suitable. The SensorThings API standard is designed for Internet of Things (IoT) applications. It is the goal of the DHI WASY GmbH to include models into an IoT environment by creating virtual sensor data from model outputs. Therefore, this project examines if this is possible within the framework of the SensorThings API.

Special focus was also laid on the possibilities to store statistical data within the standard. Model data and sensor measurements are afflicted with uncertainty. It is important to store and visualize statistical data, because it impacts the decisions that the operator makes, and any future optimization algorithms must consider statistical parameters to ensure accuracy.

A huge potential for optimization of irrigation and timeseries prediction lies within the scopes of machine learning algorithms [1]. Therefore, another objective was to create a data structure that enables other developers to import the stored data and convert it into a suitable format for further preprocessing and development of algorithms.

An example workflow was built based on data of a pilot project for automated farming. The pilot project is delineated in the following chapter. The sensor data of the pilot fields (obtained from a climate station and soil sensors on-site) is acquired via the application programming interface (API) of the companies providing the technologies. Furthermore, an existing water balance model created with the MIKE-SHE software delivers simulated data on soil moisture content. All data was collected, converted, and implemented within the framework of the OGC SensorThingsAPI standard.

Making an automated system transparent is important to gain trust of farmers. Moreover, hybrid system are desired where the operator can make decisions both based on experience and additional information or algorithms. The visualization of data is essential for the transparency of a system. Realtime applications require real-time visualization and continuous tracking of all parameters must be possible. For this purpose, a simple interactive map-centric web-application was created to visualize all given data. A further visualization tool used is Grafana. Since Grafana supports the SensorThingsAPI standard, dashboards can be employed without further effort. These dashboards were then implemented within the interactive map. Moreover, a graphical user interface has been established to make the data workflow scripts accessible for everyday use.

Another great issue of IoT smart irrigation systems is the remote location that leads to accessibility problems. As already mentioned, the sensors on-site are provided by external companies and have been connected to the cellular network. Therefore, the remote-control concept was designed only to control an on-site pump. Two setups that were chosen to be suitable were applied to achieve this. The first technology is a GSM module controlled by short message service (SMS) coupled to an API that enables the user to send the messages via http request. The second approach is based on a LoRa network. With the help of TheThingsNetwork API, a microcontroller with an implemented LoRa – chip can be controlled via HTTP request. Due to the remote location of the pilot field, it was necessary to setup a gateway. A more detailed description of the hardware is given in chapter 3.2.

In the first chapter, I will present the project Irrimode, which is the foundation of this study project. Following this, a brief overview of related literature will be given. The subsequent chapter delineates the theory, software, and tools that were used within this work, before I outline the implementation of the data workflow and the remote control.

1 Project Irrimode

The core competence of the DHI WASY GmbH lies in the digitization, modelling and visualization of water systems. A collaborated project between the DHI WASY GmbH and various partners has been launched to make irrigation in agriculture more efficient and transparent. Detailed information about this so-called project Irrimode can be found on the corresponding website [2]. Operational partners are Gut Mennewitz GmbH, Ingenieurbüro Irriprojekt Dirk Borsdorff and AGRO-SAT Consulting GmbH and further associated partners are the DLG e.V. Fachzentrum Landwirtschaft, Hochschule Anhalt FB I – Landwirtschaft, Ökotropologie und Landschaftsentwicklung, and Obsthof Am Süßen See GmbH. The project, which is state-aided by the state Sachsen-Anhalt and the European Union fund ELER, aims to digitize and optimize the drop irrigation system of a pilot crop field in Bernburg, Germany [1]. To achieve this, an integrated approach was applied, which comprises different components such as a “MIKE-SHE” model, weather data and real-time sensor data. The system scheme is depicted in Figure 1. A solar pump, which is controlled by a control unit, pumps water from the intermediate storage tank to the fields, a process, which is controllable by valves.

An on-site climate station exists, provided by Pessl Instruments [4]. Sensor readings and estimated properties are available online and can be requested through an API. Available parameters are air temperature, solar radiation, soil temperature, solar panel voltage, precipitation, wind speed, leaf wetness, relative humidity, dew point, reference crop evapotranspiration (ET_0) and vapor-pressure deficit.

Additionally, datalogger are put into place to obtain real-time data on the water content, salinity and soil temperature. Every datalogger consists of various sensors that measure these three parameters in different depths. Three sensors are employed per ten-centimeter interval starting from 5 cm and finishing at 85 cm below the surface. As in the case of the Pessl Instruments climate station, the data is uploaded to a proprietary server of the vendor Sentek Technologies and can be requested through an API.

Furthermore, the MIKE-SHE software offers water balance modelling of the crop fields. A model has been created providing output information on current transpiration, evaporation, and thus water content. The model is meant to be used to optimize the irrigation and to simulate potential scenarios. [1]

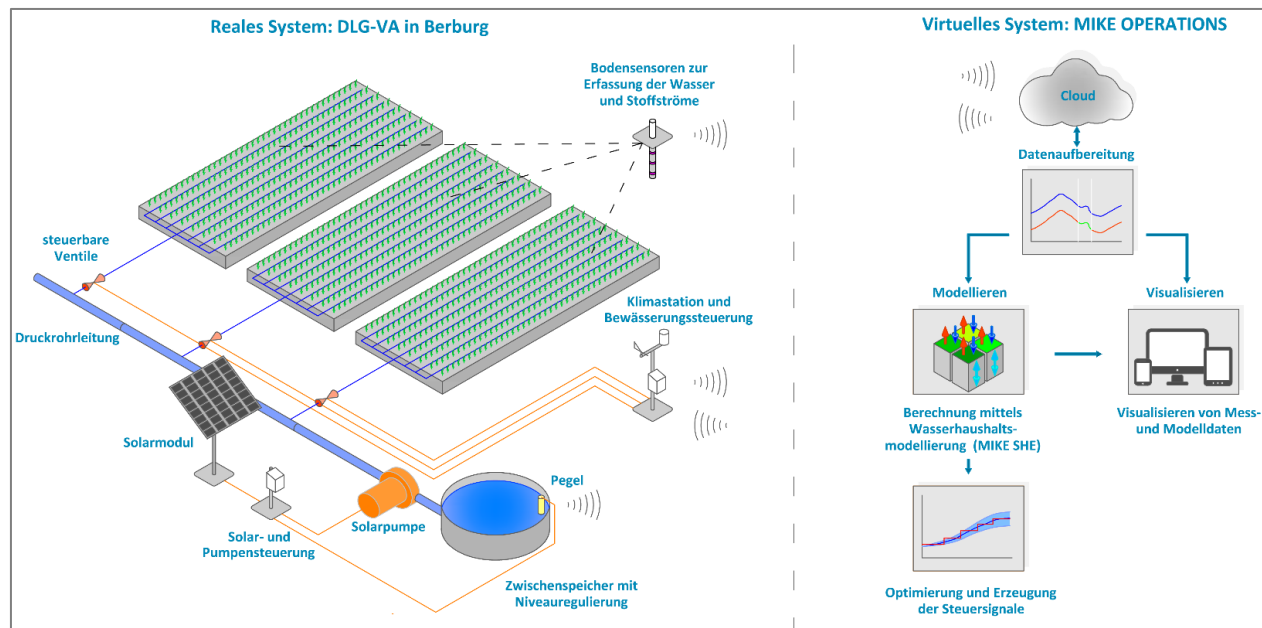


Figure 1: System scheme of the Irrimode project [1].

2 Theoretical Background

2.1 Software

2.1.1 Modeling Software

The MIKE software products are powered by the company DHI. One of the products is the MIKE-SHE software, which is used for water balance modeling and simulation of surface-groundwater interactions. It integrates all significant processes at the catchment scale, such as evapotranspiration, infiltration, overland flow, unsaturated flow, groundwater flow and channel flow [5]. Different approaches both physics-based and stochastic/data-driven are provided for modeling of the individual processes. MIKE-SHE models can hence be considered a holistic approach to implement the whole hydrological cycle with its influences on the crop field.

With the help of this software, it is possible to forecast soil water content and predict various scenarios. The numerical calculations are grid-based, and several layers can be defined to simulate three-dimensional water flow.

All MIKE products are using the proprietary Data File System (DFS) format to handle spatial timeseries data. DFS is a binary format that can be split into three parts: A header section comprises general information such as start time, geographic map projection, etc. The static section saves time-independent static data for certain items. An item could be, for example, a parameter such as soil moisture content. The section that takes the most storage space is the dynamic data section, which contains time-dependent data. The Data File System format can be sub-divided into spatial-dimension-dependent sub-formats. In the case of dfs0 - files, which

describe files with scalar values, one can also talk about model-based virtual sensor timeseries. [6]

The MIKE-SHE model data is an example of how models can be integrated into an IoT environment in form of virtual sensor data. A virtual sensor is defined as a type of software using mathematical models to estimate product properties or process conditions based on physical observations. This is necessary, for example, when a physical sensor is too slow, inaccurate, too expensive, or when the sensor cannot be placed at the location where measurements are needed. In this case, a virtual sensor can be “deployed”, which means the measurements of this location are estimated based on the physical measurements that are delivered by the surrounding sensors. In our case, the virtual sensor data is not based on physical sensor measurements but timeseries are estimated with a vast amount of input data and with the help of the MIKE-SHE software.

2.1.2 Programming Language and Integrated Development Environments (IDE)

Python 3.7 was chosen as the programming language for this project, because it offers built-in data wrangling features and can be used for all kind of sectors, such as web, software, and GUI development, scientific and numeric calculations and for system administration [7]. This makes it suitable not only for the data transformation but also for development of algorithms, as well as for visualization and for the development of a graphical user interface. Some of the libraries that enable this development are described below.

Two IDEs were used for developing the scripts: Jupyter Notebook and PyCharm. Project Jupyter was started in 2014 on basis of the IPython Project. Jupyter products are used “to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages” [8]. One application is the Jupyter Notebook, which is a web-based computational environment that extends the traditional console-based approach by embedding developing, documenting, and execution of code into one. Jupyter Notebook documents are based on JSON and consist of ordered lists of input and output cells that can comprise code, markdown text, mathematics, or plots. Various programming languages are supported by Jupyter. [9]

The second IDE that was used is PyCharm Community Edition. PyCharm is an integrated development environment (IDE) released by JetBrains in 2010. It offers intelligent coding assistance, built-in developer tools and supports Python web development frameworks [10]. Hence in my opinion, PyCharm is more suitable for the implementation of a software library and was used accordingly, whereas the development of the code was done in Jupyter Notebook.

For the programming of a microcontroller the Arduino IDE was used. Arduino IDE is a cross-platform with a code editor, and it offers a simple way to compile and upload programs to an Arduino board. An Arduino IDE program is called a sketch. Features of the program are the sketch editing tools, library manager, and serial monitor. The programming languages C and C++ are both supported by Arduino IDE.

2.1.3 Libraries

2.1.3.1 NumPy

NumPy is a powerful package for scientific computing with Python. The most important features that come with the package are a N-dimensional array object, sophisticated functions, and tools for integration of C/C++ and Fortran code [11]. Further it enables linear algebra, Fourier transformations, and random number calculations [11]. NumPy is also useful for storing multi-dimensional generic data [11]. This is the reason why it was used for handling the model data.

2.1.3.2 Pandas

With the Pandas library, Python developers have a data analysis tool and an integrated high-performances data structure [12]. Some features that are often used within this project are, for example, the reading and writing of data between in-memory data structures and CSV, text files, Microsoft Excel, or SQL databases, merging and joining of data sets, flexible reshaping and pivoting of data sets, and the intelligent data alignment [12]. The library offers a fast and efficient DataFrame object for data wrangling with embedded indexing [12]. Other libraries are built on top of Pandas including statistical, machine learning, and visualization libraries [13]. This makes the Pandas DataFrame object a popular starting point for development of data tools [13].

2.1.3.3 GeoPandas

In the scope of this project, geospatial data plays a major role. Hence, the GeoPandas library was used to make the handling of geospatial data easier. GeoPandas is an extension of the Pandas library. It allows spatial operations on geometric types by extending the datatypes used by Pandas. The geometric operations are done by another library called shapely. The Pandas DataFrame object is replaced by the GeoDataFrame object of GeoPandas. [14]

2.1.3.4 Requests

Requests is a library for using the Hypertext Transfer Protocol (HTTP) 1.1. This package embeds the urllib3 library and replaces the urllib2 default module of python. It offers an easy way to communicate with HTTP and enables integration of web services into Python. [15]

2.1.3.5 *Pickle*

Pickling is the process of converting a python object hierarchy into a byte stream. The pickle module therefore establishes a binary protocol for serializing and de-serializing a Python object structure. A pickled object can be saved as local file, which can then be reloaded back into Python. This makes the Pickle module a tool for storing all common Python datatypes as local Pickle files. [16]

2.1.3.6 *Leaflet Visualization Tool*

Leaflet is an open-source source library for creating interactive maps. It is based on JavaScript and has most functions inherited that are needed by developers to build the map [17]. Since in the scope of this project, not JavaScript but the Python language will be used, the folium library is of importance. This library implements the leaflet.js library into a python environment plus adds some data wrangling tools [18].

2.1.4 *Docker*

Docker is the leading company in the containerization market [19]. Containers are used to package everything necessary to run an application, such as code, runtime, system tools, system libraries and settings [20]. In a way, containers act like virtual machines but there are distinctions: containers run on the hosts operating system, whereas a virtual machine (VM) includes a full copy of an operating system. Thus, virtual machines take up a lot more storage space [8, 9]. The technology Docker put forward was designed to facilitate creation, shipment, deployment and running of applications by packaging software applications into standardized units [20]. A standardized unit of software is called a docker container. With this, Docker established an industry standard for containers that are lightweight and secure [20].

The Docker Engine is the interface between containers and resources. The engine was originally created for Linux operating systems but thanks to virtualization technologies it is now also possible to use it on Windows or Mac OS [22]. Since containers are fairly light-weight and start up is fast, docker containers are becoming popular in cloud applications. The automated scaling of applications is made easier [22].

For the scope of this study project, it is important to understand some distinct components of docker. An overview of the structure is given in Figure 2. First, one must differentiate between images and containers. Images pose read-only, immutable files that contain all necessary files to run a container [23]. The relation between an image and a container can be compared to the relation of a class to an object [23]. Once run, the docker image becomes the docker container, if the Docker Engine is used, which means the application defined in the image is running within the container [20]. As shown in Figure 2, the required information to create a specific image are stored in a local Dockerfile. Docker containers and images on the other side are stored within a

local Docker instance. However, Docker images can be saved as archive file with tar, which is a software for collecting many files into one [24]. This enables the user to store local backup files that can be loaded back into an image. Another instance of the data structure of Docker are volumes. They are necessary to manage application data and are useful for backups and exchange of data between docker containers [25]. The storage of application data is described in detail on the documentation page of docker [25].

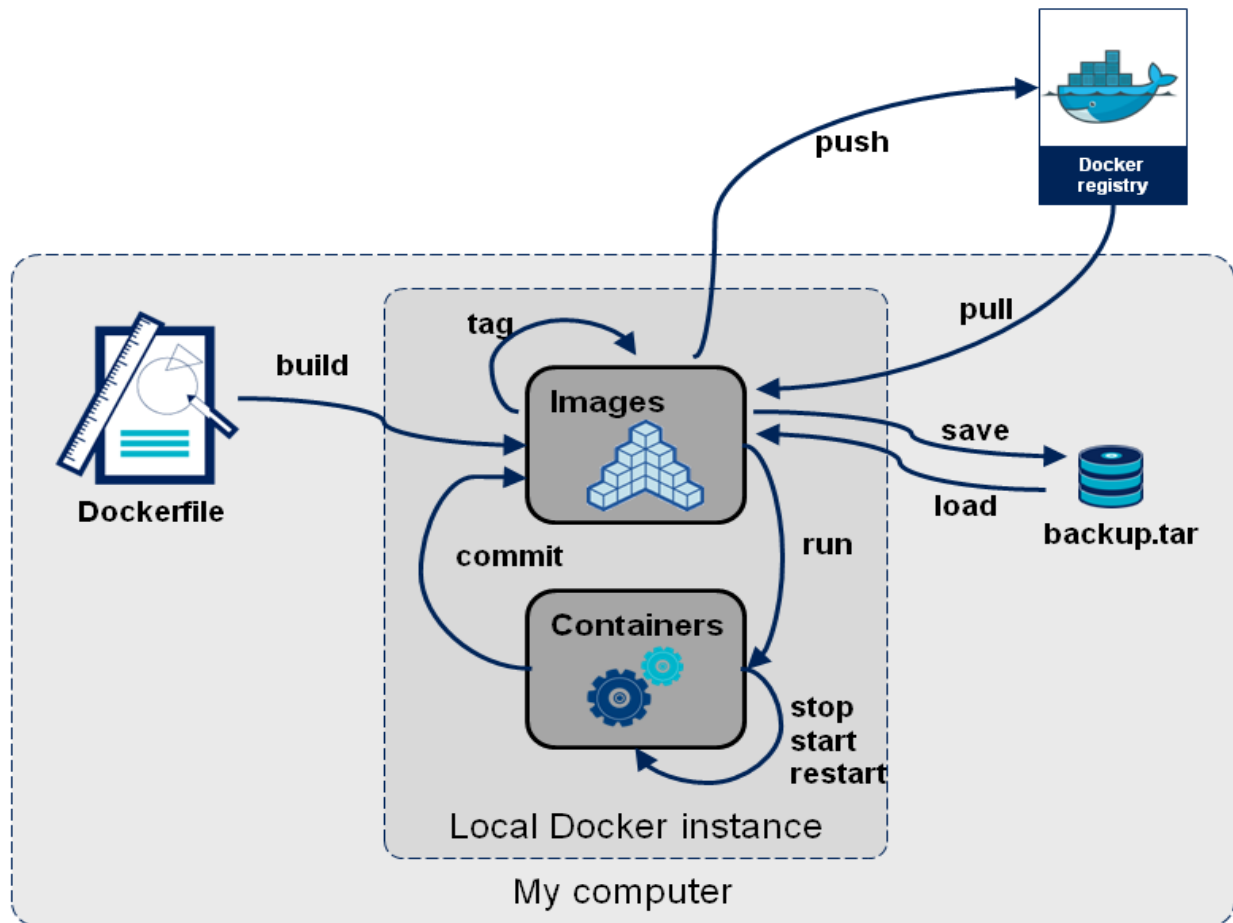


Figure 2: Overview of Docker structure, from [26].

2.1.5 Grafana

Grafana is an open-source observability platform that supports over 30 data sources. Data is queried from the source and visualized in form of dynamic dashboards. Grafana also comprises alerting and ad-hoc filtering features. Many plugins for distinct ways of visualization are available for Grafana, such as the "DarkSky" weather forecast plugin, "Heatmaps", "Worldmap Panel", "GeoLoop", "Plotly", "Trend Box", and many more. Furthermore, Grafana supports various data sources. Most importantly for this project, Grafana provides a plugin for the SensorThings API as data source. This makes Grafana an example of a service that can be used as plug-and-play application when standards are put into place. [27]

2.2 Standards

2.2.1 Open Geospatial Consortium (OGC)

The Open Geospatial Consortium is an organization that comprises more than 530 businesses, government agencies, research organizations and universities [14]. It is the objective of the consortium to make geospatial information and services fair, accessible, interoperable, and reusable (FAIR). To make this happen, the OGC released various royalty-free, publicly available, open standards, such as, for example, Web Map Service (WMS), Geography Markup Language (GML), KML, and Open Modelling interface (OpenMI) [28]. Existing standards for the water industry are the Water Modeling Language (WaterML) and the Groundwater Modeling Language (GroundwaterML) [14, 15].

2.2.2 OGC Sensor Web Enablement (SWE)

The OGC Sensor Web Enablement describes a suite of standards that were developed to enable discovery, access, tasking, as well as eventing and alerting of sensor resources in a standardized way [31]. It includes the following standards: Sensor Modeling Language (SensorML), Observation and Measurements (O&M), Sensor Observation Service (SOS), and the Sensor Things API [32]. The SensorML describes sensors and measurement processes, the O&M handles real-time sensor observations, whereas the SOS and Sensor Things API are standards for the retrieval of sensor descriptions and observations, hence, represent web service interface standards. The standard used in this project for accessing and storing sensor and model data is the Sensor Things API, which is explained in the following chapter.

2.2.3 OGC Sensor Things API

The Sensor Things API is the latest standard of the SWE suite and updates the older xml encoded standards, which were complex in use [33]. The standard offers a web-based approach to connect IoT devices, data and applications in an open, geospatial, and unified way [34]. The Sensor Things API is based on the REST principles, JSON encoding, and can make use of MQTT and OASIS Open Data protocol and URL conventions [34]. MQTT is a machine-to-machine protocol specialized for IoT applications, characterized by its small size, low power consumption, and small data packets [35]. The OASIS Open Data protocol builds on HTTP and JSON, follows the REST principle, and uses Uniform Resource Identifiers (URI) to address and access resources [36]. Various queries are supported such as sorting, pagination, filtering, selecting, and expanding. The filtering functions implemented in the standard enable users to make specific requests. The functions embed logical, mathematical, and comparison operators, plus string, mathematical, geospatial, and date and time functions [33].

There are two functionalities provided in the standard: sensing and tasking [7]. In this context, sensing describes how to manage and retrieve observation and metadata IoT sensor systems,

whereas the tasking part is about parameterizing sensors and actuators [7]. While the tasking part is not relevant for this work, the sensing part is the base of the data storage within the workflow concept.

Sensor Things API comprises various entities: “Things, Locations, Sensors, ObservedProperties, Historical Locations, Datastream, Observation and FeatureOfInterest”. Their individual attributes and the relations between the entities are depicted in Figure 3. A short description of the entities is necessary to grasp the data structure:

- Thing: Following the ITU Telecommunication Standardization Sectors definition, a Thing is “an object of the physical world or the information world (virtual things), which is capable of being identified and integrated into communication networks” [37].
- Location/FeatureOfInterest: The Location entity, as defined in the OGC SensorThings API standard, is the last known location of one or multiple Things. The entity FeatureOfInterest is often identical to the location particularly in in-situ applications. It has a direct one-to-one relation to the observation entity. The observation result is allocated to a phenomenon, which is a property of the FeatureOfInterest. However, in remote sensing applications, the ultimate location of the Thing may not be identical to the FeatureOfInterest but rather describe a remotely sensed geographical area or volume. [34]
- Historical Location: This entity describes location of last known and previous locations of the Thing. [34]
- Datastream: Data produced by the same Sensor and the same ObservedProperties are grouped by the Datastream entity. [34]
- Sensor: A Sensor is defined as an instrument that observes either a property or a phenomenon that aims to produce an estimate of the value of the property. [14, 16]
- ObservedProperty: An ObservedProperty describes an Observation phenomenon. [34]
- Observation: An Observation is the measurement or otherwise determination of a property's value [14, 16].

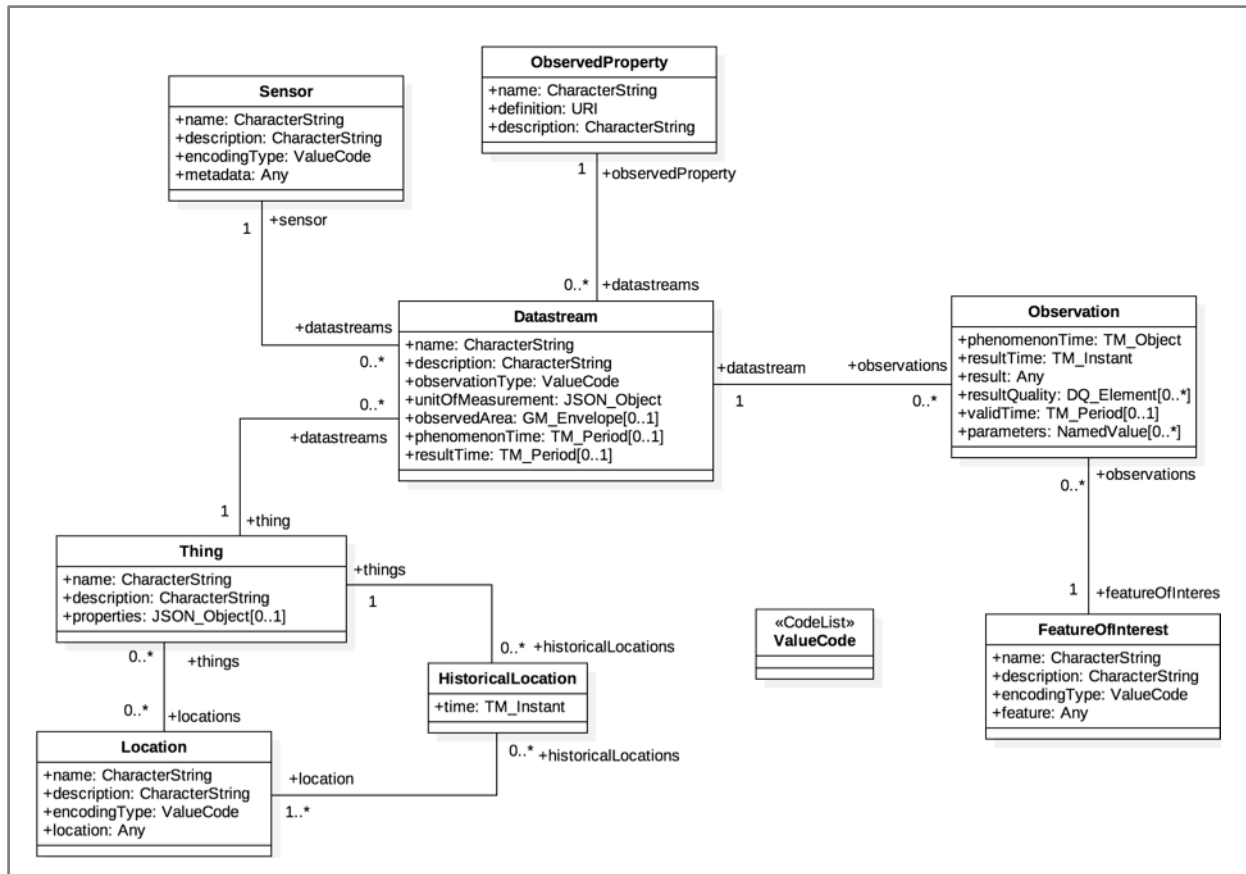


Figure 3: Sensing entities of the OGC SensorThingsAPI standard [34].

The Sensor Things API standard additionally offers a MultiDatastream and a data array extension that supports complex result types in form of array data [34]. The UML data model for the extensions is shown in Figure 4. Its structure is like the original, but the Datastream entity is replaced by the MultiDatastream. In contrary to the original Datastream entity, the MultiDatastream is based on arrays, which means that multiple ObservedProperties are linked to one MultiDatastream. Furthermore, the Observation result is now of the type JSON-Array. This offers the opportunity to store multiple values in one result array, which is particularly interesting for the storage of multi-dimensional raster data. With this extension, different approaches are possible to store the model simulation results.

There are several free and open source implementations of the Sensor Things API, such as Eclipse Whiskers, GOST, FROST, and Mozilla STA [22 - 25].

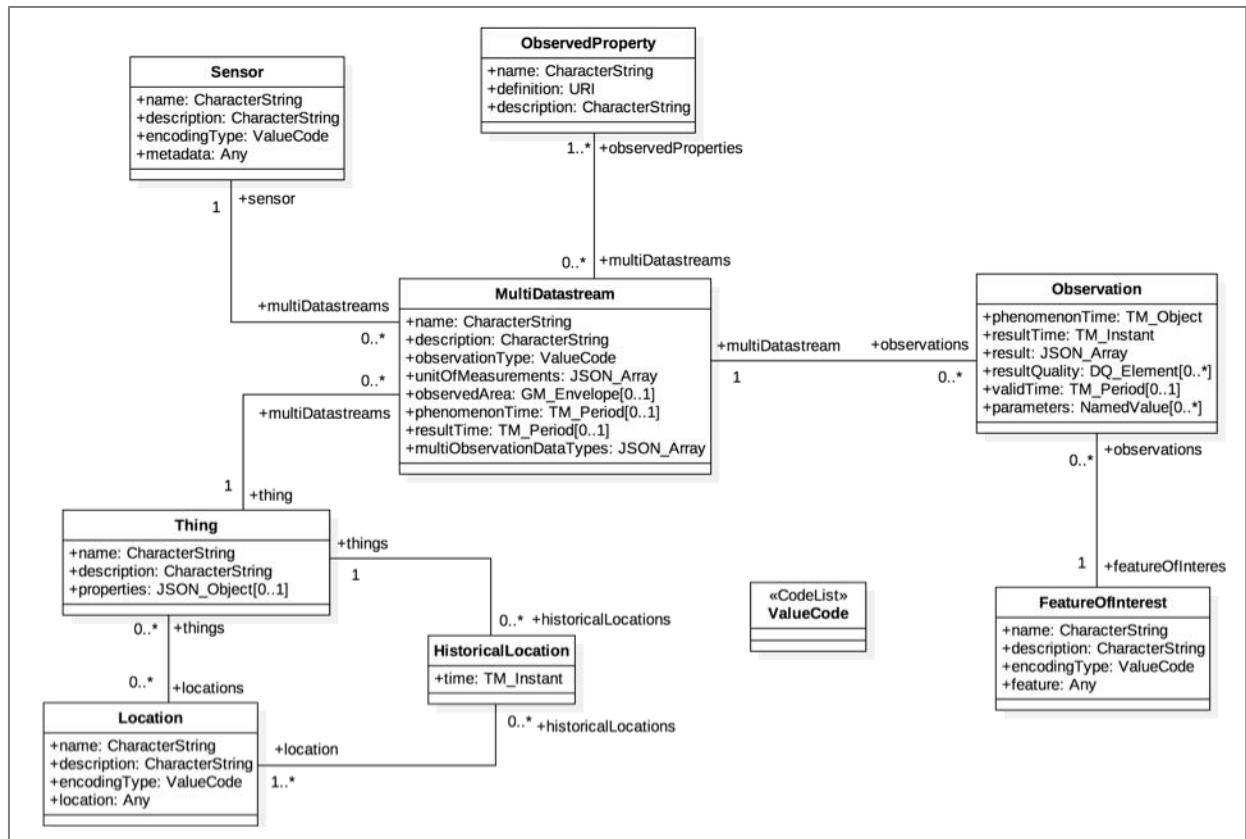


Figure 4: MultiDatastream Extension Entities. [34]

2.2.4 Quality Modeling Language/ISO 19157

The resultQuality attribute of the entity Observation in the Sensor Things API standard requires a Data Quality Element as type. A Data Quality Element is defined in the ISO 19157 [43]. In this standard, it is clarified how to describe the quality of geographic data. The ISO 19157 defines the components needed for the description of data quality, the procedures for evaluating the quality, the specifying of components and content structures of a register for data quality measures, and establishment of principles for reporting data quality [43].

The Data Quality Element is an abstract object that will be represented by one of the different types, depicted in Figure 5. In this work, the resultQuality attribute is used to store statistical data. For the uncertainty of model output data, for example, the DQ_QuantitativeAttributeAccuracy would be of interest.

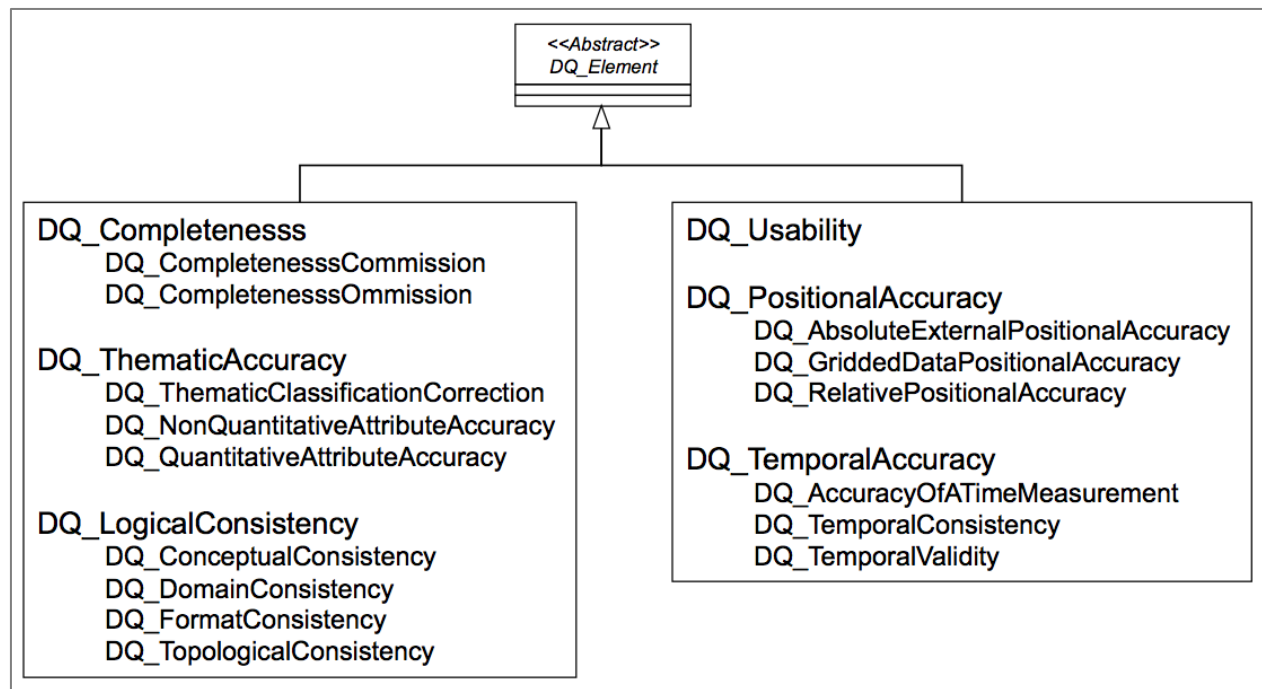


Figure 5: Data Quality Element Types. [44]

The Quality Modeling Language (QualityML) is a profile of the ISO 19157 but extends it by defining semantics and vocabularies for the quality concept [45]. It widely uses expressions from the Uncertainty Modeling Language (UncertML) [46]. The UncertML is based on a discussion paper of the OGC from 2009 [47]. It is a conceptual model that encapsulates probabilistic uncertainties [46]. The UncertML never became an official OGC standard and the UncertML website was shut down in 2016 [45]. However, the QualityML was instituted, which extends the uncertainty concept by alternative metrics [45].

2.2.5 Fraunhofer Open Source SensorThings (FROST) Server

As mentioned, there are a multiple open-source server that implement the Sensor Things API standard. Since our department already had experiences with the Fraunhofer Open Source SensorThings (FROST) server and it is the first implementation that includes all the extensions of the OGC Sensor Things API standard, this server was chosen for the storage of the sensor and model data. FROST is an open-source server application based on JavaEE, PostgreSQL, and PostGIS. The server is designed to interconnect IOT devices, data, and applications over the Internet, thus, enabling IOT systems [48]. The FROST server is characterized by its scalability, enabling systems both on small scale (e.g., Raspberry PI) and local servers, as well as for cloud application [33]. Furthermore, when using the FROST server, the user can be sure that the data is stored truly type-conserving. Moreover, type-specific ordering plus type-safe filtering are possible [33]. For attributes whose types are not specifically defined within the Sensor Things API standard (of type “any”) the FROST server can store any type that is valid in JSON [33]. There are many ways to set up the FROST server. It can either be installed either separately for HTTP or MQTT, or all-in-one. The deployment can be done with the help of Docker.

2.3 Connectivity

2.3.1 Global System for Mobile Communication (GSM)

In the late 1980s caused by the switch from national regulation of networks to privatization, new ways of telecommunication were developed. The Global System for Mobile Communication (GSM) is one of the standards that resulted from this development. GSM describes the protocols used in the second generation (2G) cellular networks. Most 2G networks operate in the bands of 900 MHz or 1800 MHz. [39, 40]

2.3.1.1 Messagebird API

The service company Messagebird provides an API to send, receive, and control Short Message Service (SMS), Voice and Whatsapp messages. This makes it possible to integrate Short Message Services messages into the remote-control system and automatize the communication between user and a receiver. [51]

2.3.2 Low Power Wide Area Network (LPWAN)

In times of rapid increase of IoT devices, long range networks have become more popular. Different technologies are available for the long-range radio communication including the three leading ones Sigfox, LoRa and NB-IoT. A recent comparative study on these technologies regarding their advantages, quality of service, network coverage, and cost has been conducted by Kais Mekki et. al [52]. LPWANs are characterized by their low power-consumption, low costs,

low data rate and long range. Low power wide area networks are therefore ideal for IoT applications, which do not require high data rates.

2.3.2.1 Sigfox

Sigfox is a commercial network operator using unlicensed ISM bands. Their technology provides bidirectional data transmission, where the downlink only occurs after an uplink signal [52]. The range of coverage is very high whereas, on the downside, the maximum messages per day are restricted to 140 [52]. Sigfox implemented their own base stations equipped with cognitive software-defined radios. The user must pay per device and amount of messages to make use of the base stations [53]. The maximum data rate is 100 bps with a payload length of maximum 12 bytes (UL)/ 8 bytes (DL) [52].

2.3.2.2 Narrow Band IoT (NB-IoT)

The Narrow Band IoT is a special LPWAN technology because it can coexist with GSM and LTE under licensed frequency bands. Three operation modes are possible: stand-alone operation, guard-band operation (utilization of unused resource blocks within the LTE carriers guard band), and in-band operation (utilization of resource blocks within LTE carrier).[52]

The protocol is built on the LTE protocol but has reduced functionalities to decrease the data rate. In summary, the NB-IoT is based on the LTE network infrastructure but has been adopted to the characteristics required for IoT applications. The use of the network is not free, coverage is lower compared to LoRa and Sigfox and energy consumption is generally higher due to QoS handling and synchronous communication for uplink and downlink [52].

2.3.2.3 LoRa/LoRaWAN

LoRa is a physical layer technology, which also uses unlicensed ISM bands to implement the network. It provides different classes (A, B, C) which are based on different uplink/downlink schemes [54]. A standardized protocol called LoRaWAN is applied for the data transfer using LoRa modulation. LoRa modulation is built on Chirp spread-spectrum (CSS) technology, which uses chirp signals for communication [55]. The LoRaWAN protocol makes use of different spreading factors. This factor defines the total number of symbols (coding information bits), which is calculated by using the spreading factor as exponent with base 2 [55]. Hence, the spreading factor is used to accustom the data rate to range. The data rate further depends on the bandwidth. A higher data rate comes at the price of lower range and vice versa [52].

Depending on the region, LoRaWAN can use bandwidths that vary between 125 kHz, 250 kHz and 500 kHz. The general range is lower than what the Sigfox base stations provide. The maximum payload length for each message is 243 bytes. [56]

The European LoRaWAN network operates in a frequency band between 863 MHz and 870 MHz [57]. The frequency regulations of Europe specify so-called duty-cycles, which is a fraction of one period when a system or signal is active [58]. A period is defined as the time it takes for a signal to switch on and off again [58]. Section 7.2.3 of the ETSI EN300:220 standard defines specific sub-bands and their duty cycles for Europe: [59]

- g (863.0 – 868.0 MHz): 1%
- g1 (868.0 – 868.6 MHz): 1%
- g2 (868.7 – 869.2 MHz): 0.1%
- g3 (869.4 – 869.65 MHz): 10%
- g4 (869.7 – 870.0 MHz): 1%

These specifications apply to every device, which transmits on one of these frequencies. Most of the LoRaWAN channels show a duty-cycle of 1% or lower. The network needs to be smart when scheduling messages on gateways. However, developers also must consider these limitations when designing a LoRaWAN based system by reducing their payload size, transmission intervals and avoiding downlink messages. [57]

LoRaWAN uses identifiers to know which specific device, application, or gateway is communicating: there are the DevEUI (unique; 64 bit end-device identifier), the DevAddr (non-unique; 32 bit device address), the AppEUI (unique; 64 bit application identifier), and the GatewayEUI (unique; 64 bit gateway identifier). [60]

Some microcontroller boards support the LoRa network as they have a respective communication interface implemented. Some boards have LoRa transceiver embedded (e.g., Adafruit Feather M0 RFM95 Lora) but the protocol must be implemented manually. Certain others have a LoRaWAN chip, which implements the LoRaWAN protocol (e.g., Seeeduno LoRaWAN) so that the developer can use the implemented protocol via a library.

2.3.2.4 The Things Network (TTN)

The Things Network (TTN) is a community network that operates LoRa networks world-wide, free of charge, as part of the LoRa Alliance [61]. They do not only maintain a LoRa network, but also provide a network server for public use. Because it is a community-based initiative, anyone can set up and run a gateway using TTN. At the moment, there are about 10 000 gateways available in 147 countries [61]. The local TTN communities work on the establishment and maintenance of the network.

To make use of the LoRa gateways, each device or node must be registered with TTN. A free user account must be created, and a new application must be set up to add devices under the respective application. For every registered device the individual history of sent and received data packages can be observed online. [61]

The preferred option to connect with TTN is the Over-the-Air Activation (OTAA). Devices perform a network connection process during which a dynamic DevAddr is allocated and security keys are negotiated with the system. This is the most secure way to establish a connection [60]

Since the TTN provides an open network, it follows a fair access policy that limits the data every node can send and receive per day. The uplink airtime is restricted to 30 seconds, whereas the downlink messages are limited to 10 messages per node per 24 hours. [62]

Crucial aspects for fulfilling the duty-cycle regulations and the TTN policy are airtime and range, both of which depend on transmission interval, payload, bandwidth, spreading factor, and confirmation messages.

As elaborates before, the data rate depends on the bandwidth and the spreading factor. The higher the data rate the shorter the airtime and, hence, the lower the energy consumption due to reduced active time. Thus, if one increases the bandwidth or reduces the spreading factor, the payload will be transmitted in lower air time. However, the range will be diminished consequently because the gateway would be more sensitive to noise. The TTN website provides an airtime calculator for users to test their data transmissions [63]. A good starting point is a spreading factor of 7 with a bandwidth of 125 (SF7BW125), which has the least airtime and energy consumption, and if the range is not enough, adapt it respectively. [64]

The LoRaWAN protocol has a feature for the adaptation of the data rate. When the adaptive data rate (ADR) is enabled, the network will automatically optimize the data rate. The regional parameters define where it makes sense to activate this feature [65].

The payload generally must be as low as possible to keep airtime and energy consumption low. Therefore, it makes sense to use binary encoding. A suitable payload format is the Cayenne Protocol, which is discussed in the following section. [64]

One further factor for energy efficiency is the transmission power. With a reduced transmission power, the range will be reduced but the device consumes less energy. In general, the time interval between messages should at least a few minutes to ensure a low energy consumption and fulfillment of regulations and the TTN policy. [64]

All these rules apply to the uplink. However, downlinks are different, because during the time when the gateway sends a downlink, all channels are blocked for uplink signals. Thus, it is generally recommended not to use downlinks, or minimize its use and payloads. Furthermore, confirmation uplinks are to be avoided, if they are not necessary. The data rate of the downlink is based on the uplink. Hence, if the uplink airtime is low the downlink airtime will be low, too. [64]

2.3.2.5 Cayenne Protocol

The Cayenne Low Power Payload (LPP) is used by “myDevices.com” to implement LoRaWAN nodes into their Cayenne platform. It is a format which offers a simple and convenient way for LPWAN networks such as LoRaWAN to send information. The Cayenne LPP complies with the limitation of the payload size by structuring the payload format into channels, types, and values. This allows the device to send various sensor information at once. The data types are based on the IPSO Alliance Smart Objects Guidelines [66]. The data types are specified properties, such as temperature, humidity, etc. According to the data type, the data size will vary. By using the Cayenne Protocol, the payload can be as low as 11 bytes. The payload that is send is encoded in hexadecimal. [67]

3 Implementation

The complete software library, all scripts and Jupyter notebooks are available as GitHub repository [68]. The structure and scripts are examined in the following chapter.

3.1 Data Workflow

The main goal of this study project was to develop a data workflow that enables storage of data from different data sources (including model data) in a standardized way. Furthermore, it should facilitate visualization, and provide an interface for further preprocessing and optimization algorithms. The result data workflow is illustrated in Figure 6. First, an overview of the whole workflow is given before the individual parts are elaborated in more detail.

The sensor data is requested from the corresponding API of the providers. The data is converted and split up into metadata and timeseries for further transfer. Before the data is deployed in the Sensor Things API standard, it is saved in excel (metadata) and csv-files (timeseries). These files represent an interface for users of the data workflow. One can manually adapt parameters and validate the data before it is forwarded to the Sensor Things API standard. The request and conversion of the timeseries has been done for the two providers Pessl Instruments and Sentek Technologies. However, other sensor providers store their data in other formats. Hence, this pre-interface, consisting of excel and csv-files, will be the target format for other sensor data that may be requested in the future.

The model data (dfs3 format) of a MIKE-SHE model, which simulated moisture content for several months in 2018, is imported, converted to a NumPy array and split up for different storage options. This is one example for how model data can be put into a programming environment.

As discussed, the Sensor Things API Web Interface Service standard was chosen to store and interact with the sensor and model data. As examined in chapter 2.2.5, the FROST Server implements the desired standard. Hence, this server implementation was chosen as the base for

the storage of the sensor and model data. The interaction with the server is managed by the metadata excel sheets.

Once the timeseries data is deployed on the server, it can be requested and is automatically converted to a Pandas DataFrame object. This format is ideal, since Pandas is an essential tool for further preprocessing and development of machine learning algorithms.

Visualization was achieved by using Grafana and an interactive Leaflet map. The Leaflet map is based on the csv-files, shapefiles, model data, and weather forecast data of the German Meteorological Service (DWD). In addition to this, Grafana was set up to create interactive dashboards on base of the FROST Servers standardized data. These dashboards were then embedded as iframes into the Leaflet map.

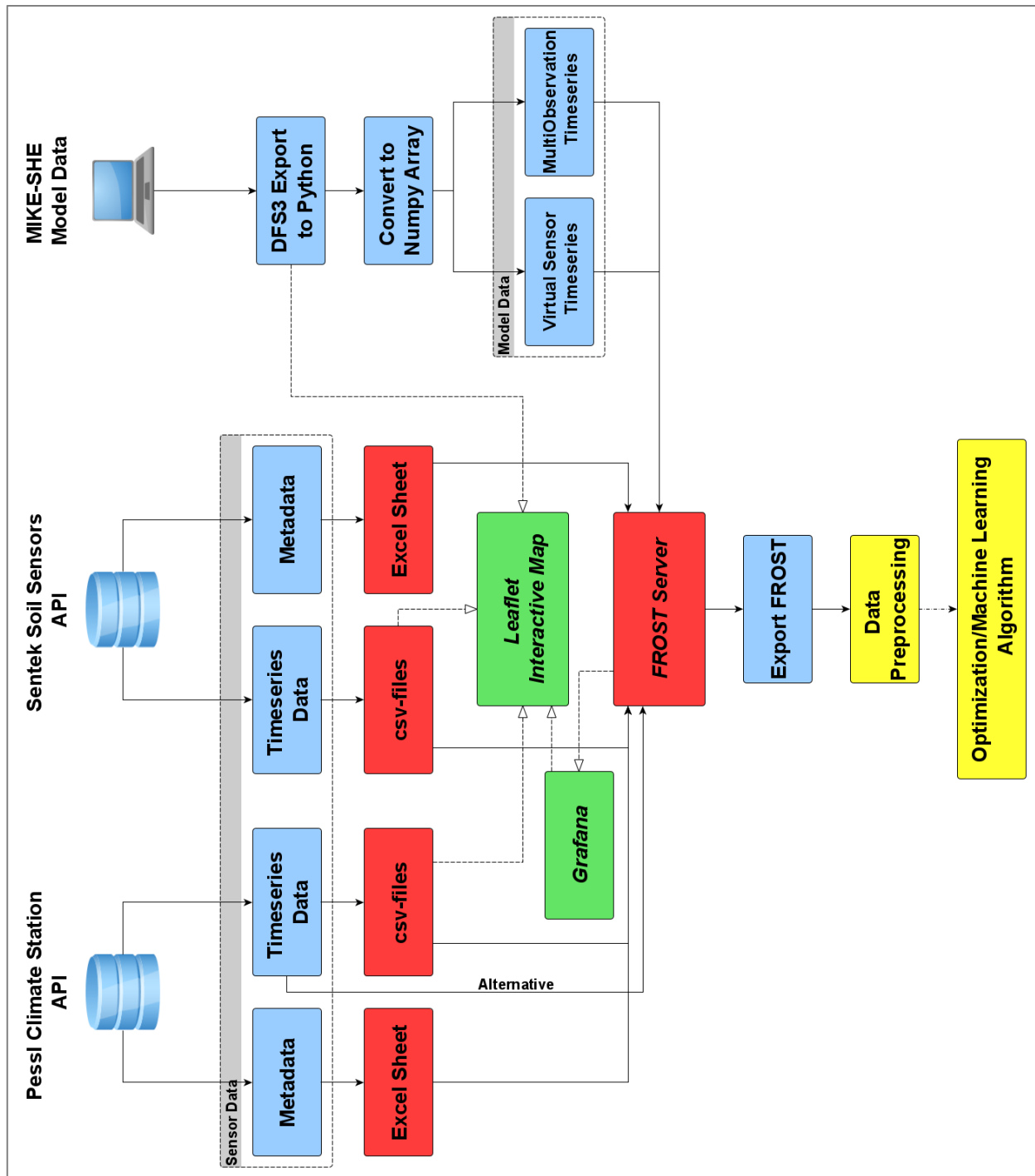


Figure 6: Data workflow as flowchart. BLUE: data transformation; RED: data storage; GREEN: visualization, YELLOW: further data use.

3.1.1 Access to Third-Party API

The data transformation from the provider APIs to the OGC Sensor Things API was conducted as pilot example on how to standardize sensor data from different data sources. Both Pessl Instruments and Sentek Technologies do not provide standardized data yet. The API of Sentek Technologies provides xml and json formatted data, whereas the Pessl Instrument API supplies data in json format. Thus, with these data sources two common formats are covered. Therefore, the transformation scripts can be adjusted in the future for other data sources with the same format.

The file containing all functions required for the transformation of sensor, is the `data_import.py` file. The import codes are based on the excel sheets created for each sensor provider ("`spreadsheet_sentek.xlsx`" and "`spreadsheet_pessl.xlsx`"), which define the local directory paths of all sensor csv- and svg-files. For future use, these excel sheets need to be adjusted to the new sensor names.

At first, when there are no files existing, the "`create_csv_sentek`" function will create new csv-files for all Sentek Technologies sensors and will update them to the last known timestamp.

A corresponding function for Pessl Instruments sensors is "`create_csv_pessl`". However, for the Pessl Instruments sensors this will only save one dataset that is received by making one request. To update the timeseries within the csv-file to the latest known timestamp, the function "`update_csv_pessl`" must be called subsequent to creating the csv-files. If the "`create_csv_pessl`" function is run, even though there are already files existing with the same names defined in the "`spreadsheet_pessl.xlsx`", they will be overwritten. Both the update function of both Pessl and the update function of Sentek Sensors read out the last values from the existing csv-files to find the last timestep they must make a request from.

The metadata, such as sensor names, are exported to another excel sheet named "`parameters.xlsx`" for the subsequent POST request to FROST that is elaborated in the following chapter about storage.

The model data was exported to Python with the help of a tool, developed internally by the DHI Group. The tool is called "`pydhi`" and enables imports of Data File System - formatted result data from MIKE-SHE models. After import, the data is converted to a NumPy array for further usage and to embed it into the leaflet map.

Since the model data is a local resource and incorporates complex raster data in Data File System format, different storage technique have been examined to find a suitable solution. The ideal case for further use within and between models would be the storage of the model output as raster data. This possibility was examined, but it was discovered that the Sensor Things API is not suitable for this purpose, which will be further discussed in chapter 4.1.1. For visualization and comparison to sensor data, this approach would not be an ideal fit in any case. Therefore, different approaches have been developed.

Storing the data as virtual sensor timeseries in form of grid points with coordinates and one value, has been proven to be a good storage strategy. This way, it can be visualized as graph and compared to the measured sensor data. Thus, this approach was chosen as first option. However, this only allows for storage of two spatial dimensions. A way to solve this is to store each node individually with information on its depth. Arrays that contain additional data on depth or further parameters are desired for multidimensional plots.

Fortunately, the FROST server offers the MultiDatastream extension defined in the scope of the Sensor Things API standard (see chapter 2.2.3). This extension enables an array-based storage with further information. Within the scope of the project, this was tested by storing arrays with additional data on depth. The MultiDatastream extension is an essential instance of the Sensor Things API that makes it possible to store distributed model data. These two approaches enable an integration of model data into an IoT environment.

As depicted in Figure 1, there is an alternative for the data handling of the Pessl Instrument data, which sends POST requests directly to the FROST servers Observation entity without saving it as csv-file. This is done by the function “direct_post_pessl”, which requests data as do the functions described before, but converts it to JSON instead, which is required for the FROST server. Even though, in this way, the computational time for import, conversion, and export of timeseries to FROST is reduced, it is generally important to have an interface to manage the data exchange between the FROST Server and data sources. Therefore, it is recommended to always update the csv-files with the identical data, when a POST request is made. If this is not the case, information may be lost, or duplicates may occur.

In the next chapter, it will be further elaborated how the Sensor Things API data model is implemented.

3.1.2 Data Storage and the SensorThings API

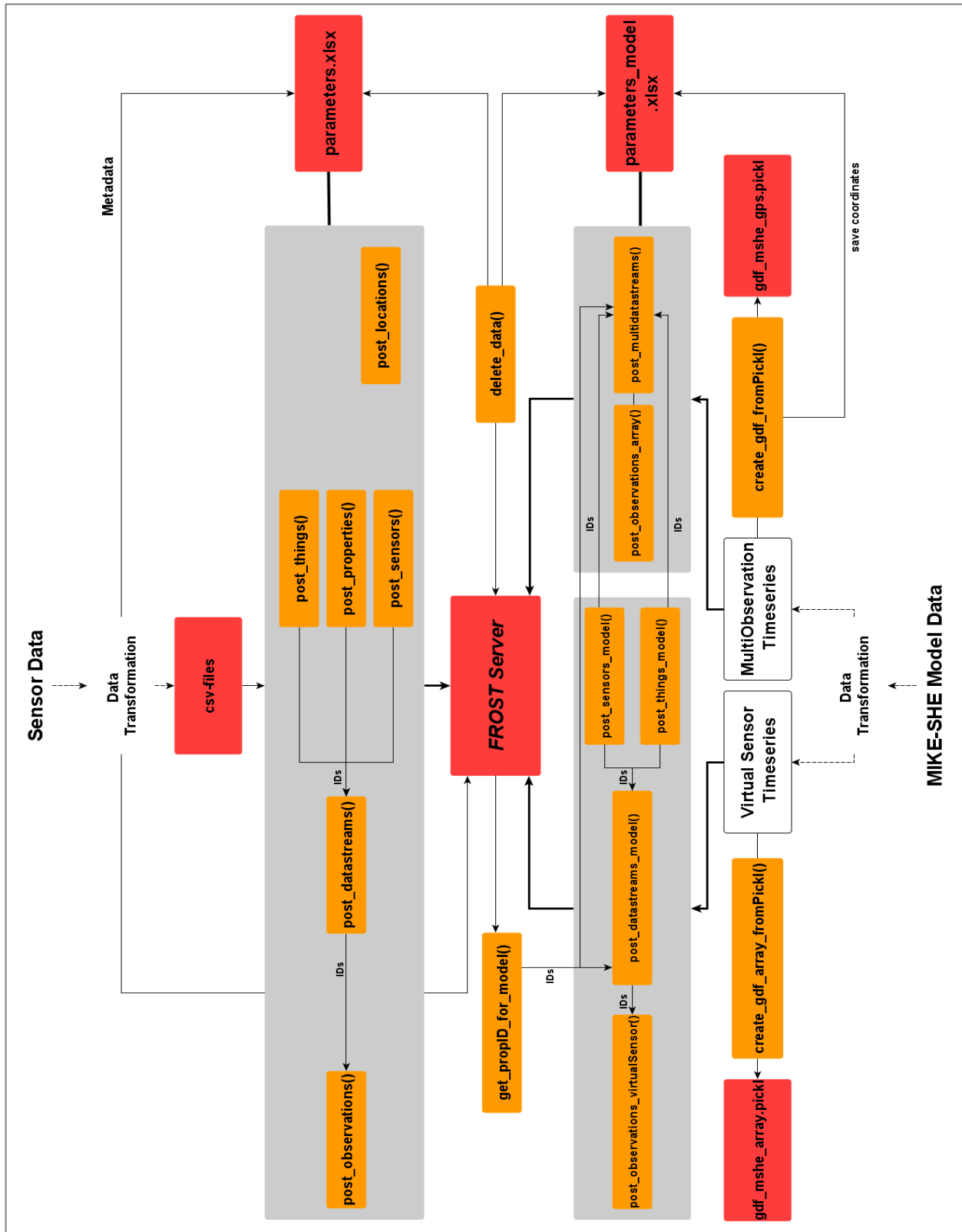


Figure 7: Data workflow FROST Server Communication. ORANGE: functions; RED: data storage; GREY: POST request functions of different entities.

The aim of the project was to standardize model and sensor data and store it efficiently. As already mentioned, the Fraunhofer Open Source Sensor Things Server was used to achieve this goal. This server implements the OGC Sensor Things API standard. Communication scripts have been implemented that are based on HTTP requests and JSON, as defined in the Sensor Things API standard (see chapter 2.2.5). The file containing the functions is the “FROST.py”. The workflow with all functions is depicted in Figure 7.

The upper part of the figure deals with the POST requests of the sensor timeseries that are saved in the csv-files and their corresponding metadata. Every entity of the Sensor Things API standard has its own function to upload the data to the server. The metadata and the links to the csv-files are managed by the excel sheet parameters.xlsx. The crucial factor is the IDs of the entities, which need to be accurate in the excel sheet for the POST request to work:

As shown in Figure 3, the Datastream has exactly one Thing, one Sensor, and one ObservedProperties. The IDs of these related entities must be defined in the JSON object. Hence, the entities Things, Sensors, and ObservedProperties must exist before the respective Datastream entity can be created. Consequently, for the Observation entity POST request to be successful, the corresponding Datastream entity must exist when the POST request is sent.

After each post, the IDs are automatically filled into the respective column of the corresponding sheet in the parameter.xlsx file. However, the IDs of the entity’s Sensors, Things, and ObservedProperties must be assigned manually to each other in the sheet “Manual Assignment”, because the code cannot know to which thing a sensor belongs to and which sensor records what property.

The excel file is color-coded to make it easier for the user to decide which column is updated automatically and which must be adjusted manually. Furthermore, it can be checked in the excel file if the POST request was successful. This way, a user-friendly pre-interface is created that can be adapted and validated manually before it is send to the FROST server for long-term storage.

The entities of the Sensor Things API had to be defined for the available sensor and model data. Every data logger has been defined as one individual Thing while the sensors of every datalogger are all added as individual Sensor entities. In other words, every datalogger was defined as one thing with 30 sensors connected to it over the Datastream entity.

Putting the model data into the Sensor Things API framework was more complicated, because of the separation of Things and Sensors. Intuitively, the created virtual sensors would fit into the scope of the Sensor entity, however, the location entity is merely related to the Things entity (Figure 3). Therefore, it is necessary to save the virtual sensors as Things to save the location information. Furthermore, when creating a Datastream entity, it is mandatory to connect a Sensor. Hence, one must also define the virtual sensors as Sensors. Consequently, every virtual sensor was saved as Sensor and Thing entity. Further options to solve this problem are discussed in section 4.1.

The lower part of Figure 7 shows how the MIKE-SHE model data is added to the FROST Server. It is split up into the virtual sensor timeseries and the multi-observation approach. The virtual sensor timeseries functions are not fundamentally different from the sensor data scripts. Thus, the communication to the FROST Server is also based on an excel file (parameters_model.xlsx). It has almost identical characteristics as the parameters.xlsx file but contains an additional sheet with the MultiDatastream parameters.

Since in the model output data only gives values for the property soil moisture, this exact property ID is received by a GET request with a filter in the `get_propID_for_Model` function. This is an example of a specific filtered request, which is defined in the Sensor Things API, that shows the advantages of adapting a Web Service Interface standard.

Because the UML data model of the MultiDatastream extension is similar in many aspects, the functions for the entities Things, Sensors, ObservedProperties, and Locations are complementary to the POST request functions of the sensor data. However, there are some differences to the JSON object when the MultiDatastream and Data Array extension are applied. MultiDatastreams group a selection of complex Observation results [34]. This means that for every index of the attribute `multiObservationDataTypes` the respective observed property must be defined, thus, each array index must relate to an ID of the ObservedProperties entity. The `multiObservationDataTypes` attribute defines the dimension of the data array result in the Observation entity. Therefore, one must make a POST request with a result array that has the same dimension as the `multiObservationDataTypes` attribute of the corresponding MultiDatastream.

Another ambition of this work was to find a way to store statistical data in a standardized way. It has been discovered that there is a way to achieve this within the Sensor Things API. The `resultQuality` attribute of the Observation entity lets the user store data according to the ISO 19157. Unfortunately, there was no model dataset with statistical information available to implement this in the FROST server. However, it was tested successfully, if a POST request implementing statistical data based on the ISO 19157 is possible. It has also been discovered that the `resultQuality` attribute of the entity Observations in the Sensor Things API standard is extensible, hence, further approaches are possible. For instance, the QualityML can be applied to describe statistical sensor data in more detail.

3.1.3 Data Export FROST

For further use of the data stored in the FROST Server, a function was needed that exports the observation data. As elaborated in section 2.1.3.2, the Pandas DataFrame object is a popular starting point when it comes to preprocessing and development of machine learning algorithms. Hence, the objective was to export the standardized data and convert it to this format.

The function that was written to do exactly this is called “request_observations_FROST”. It is also based on the parameter.xlsx excel file for specifying what observations to request. The resulting DataFrame object includes all requested Datastreams and has two columns for each property, which contain the timestamp in one and the observation value in the other.

The full dataframe is saved as pickle-file because it takes a large amount of computational time to send enormous amounts of GET requests to the FROST Server for observation data. This pickle file can be imported with the function “get_pickle_observations”.

One example how the data could be further preprocessed and used was implemented. The so-called ARIMA algorithm was implemented, which forecasts timeseries. The ARIMA function requires the input variable “property”, which defines what value to look for in the pickle imported dataset. According to this variable, the ARIMA algorithm is applied to the corresponding data.

The results of the ARIMA algorithm are inaccurate. The script only represents an example for the data workflow.

3.1.4 Visualization

The sensor and model data were visualized in the form of an interactive leaflet map and as interactive plot in Grafana. Both are powerful tools to increase transparency of irrigation systems. Therefore, the interactive Grafana dashboards were integrated into the leaflet map to bring the features of both tools together. The functions and relations used to generate this integrated approach are depicted in Figure 8.

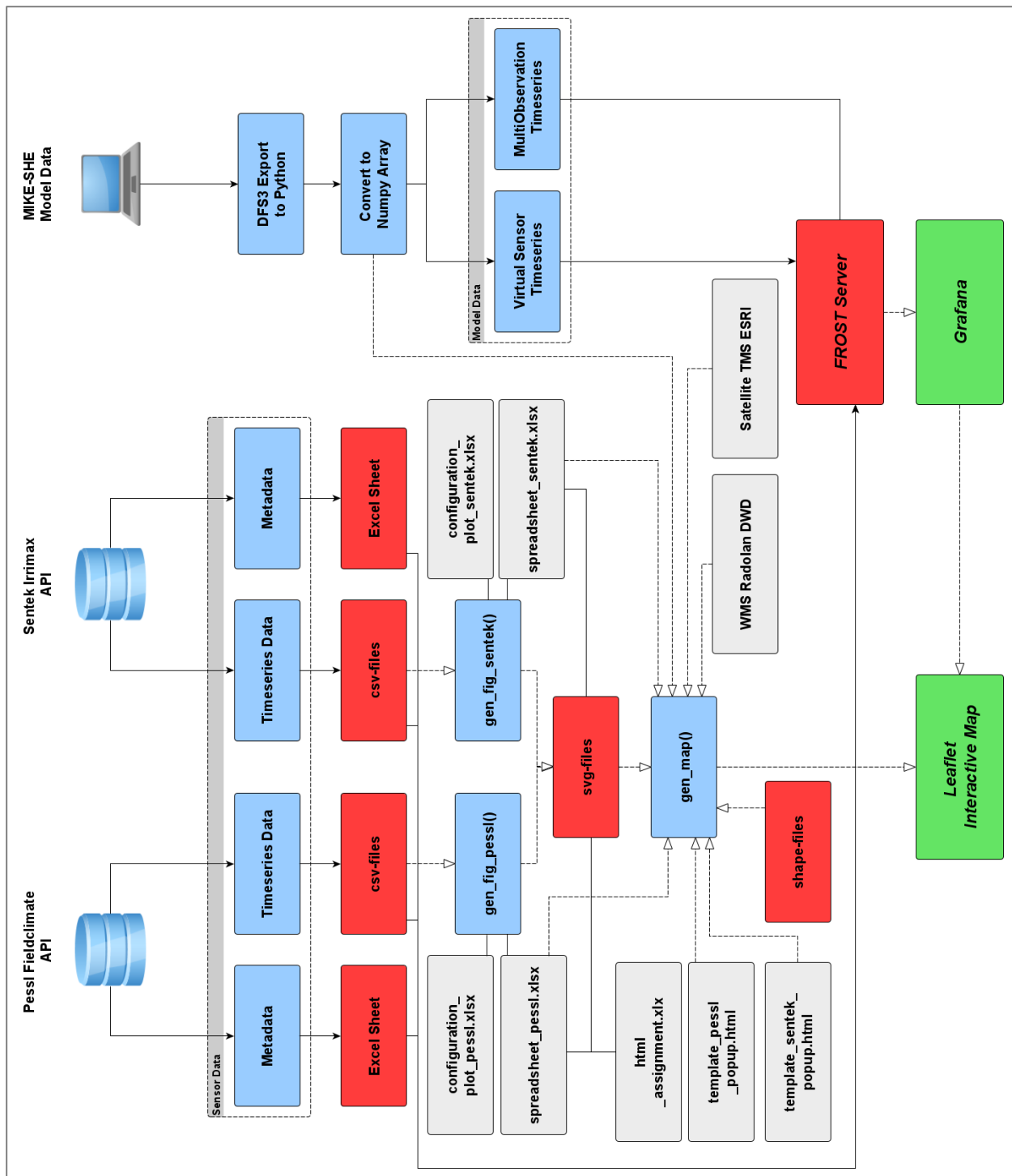


Figure 8: Overview of functions and relations for the visualization of sensor and model data. RED: data storage; GREEN: visualization tools; BLUE: data transformation/functions.

3.1.4.1 *Grafana*

Grafana was set up with Docker. When running the Grafana Docker image, some settings must be defined, such as what plugins to install and if embedment into other websites is possible.

Grafana offers a plugin that links the Sensor Things API as data source and, hence, depicts the data stored on the FROST server. Different dashboards have been created, for example, one for all sensor measurements of the Pessl Instruments climate station and one for a datalogger that covers plots of all salinity, soil temperature, and soil moisture in different depths. This way, the operator can see the spatial differences and draw conclusions from related parameters.

3.1.4.2 *Leaflet Interactive Map*

The leaflet map integrates shapefile polygons of the field and the individual sensors with their locations and popups with the data plots and last measured values. Within the popups the user can click on a hyperlink which connects HTML to each individual sensor with additional data. The model data is visualized as a chloropleth map with a time slider.

Additional supportive data was included in the Web-Application, like a rain radar that is updated hourly for whole Germany. The radar is a Web Map Service, provided by the German meteorological service on their Geoserver [69]. This is an example of WMS static images that can be implemented into the map. With the help of the existing script, in the future merely the source needs to be adjusted to enable embedment.

Furthermore, the user can switch between various Tile Map Service (TMS) base layers for reference and a satellite image layer provided by ESRI. This represents a way how to embed TMS layers. Other TMS layers can be imported in the future only by changing the source link.

The data plots are based on the timeseries data from the csv-files. Two main function can be differentiated in generation of figures and generation of the map. In the “gen_fig_senteK” function, for example, the csv-file data is exported and plotted. In the corresponding configuration excel file the user can define, what time interval to plot by choosing either a start and end time or activate automation, which enables the user to plot an amount of days before the current one. The paths of the csv-files and their generated svg-files with the plots are managed by the “spreadsheet_senteK/spreadsheet_pessl” excel files.

Once the svg-files are updated, the “gen_map” function can be called. This function has many inputs and is based on the folium library (see 2.1.3.6). HTML templates for the popups are linked to the function. On basis of these templates, further HTML templates are generated within the “gen_map” function for the hyperlinks in the popups that lead the user to further plotted data. The svg-file paths that should be plotted in the hyperlink are managed by the “html_assignment.xlsx” excel file. This approach is based on static plots that need to be updated. More desirable is an interactive plot, where the user can choose what data to see on click. Thus,

as a second approach Grafana interactive plots have been deployed into the map as inline iframe. However, no script has been created to automatize the assignment of Grafana dashboards to the corresponding sensors.

The last values of the timeseries that are shown in the popup are extracted on basis of the paths given in the “spreadsheet_sentek/spreadsheet_pessl” excel files. The model data is imported as Numpy array and converted to a chloropleth map.

3.2 Remote Control Concept

The objective was to make the system controllable remotely. A G.S.I Galcon Irrigation Control Unit is available on-site, located inside a small house close to the pilot field. The unit is depicted in Figure 9. It has three inputs to control the valves for watering. The aim was to enable a remote control of these. For this purpose, a connection had to be established between the control and the user.



Figure 9: G.S.I Galcon Irrigation Control Unit. [70]

Generally, the most suitable network depends highly on the given conditions. Since on agricultural land the conditions are usually not ideal due to lack of electricity and LAN on the fields, the focus was on LPWAN and cellular networks. Kais Mekki et al. concluded that for smart farming the NB-IoT is not a valid solution for the near future while Sigfox and LoRa are ideal [52]. However, in the long term, the federal government of Germany is planning to provide cellular network coverage for 99% of the German population [71]. Therefore, it is likely that many farms will also be covered in the coming years and hence, cellular networks and the NB-IoT technology are viable options, too.

Based on these findings, two distinct approaches were chosen to create a resilient, complementary, and redundant system. The first implementation was based on the cellular network, more specifically, on the second-generation mobile GSM technology. Additionally, in case that a farm is not in the range of the cellular network, a LoRaWAN microcontroller and a LoRaWAN Gateway were installed to establish a low power wide area network. In the following chapter the hardware for the two approaches are described and it is elaborated how the systems were established and configured. A detailed concept scheme of the setup is shown in Figure 10.

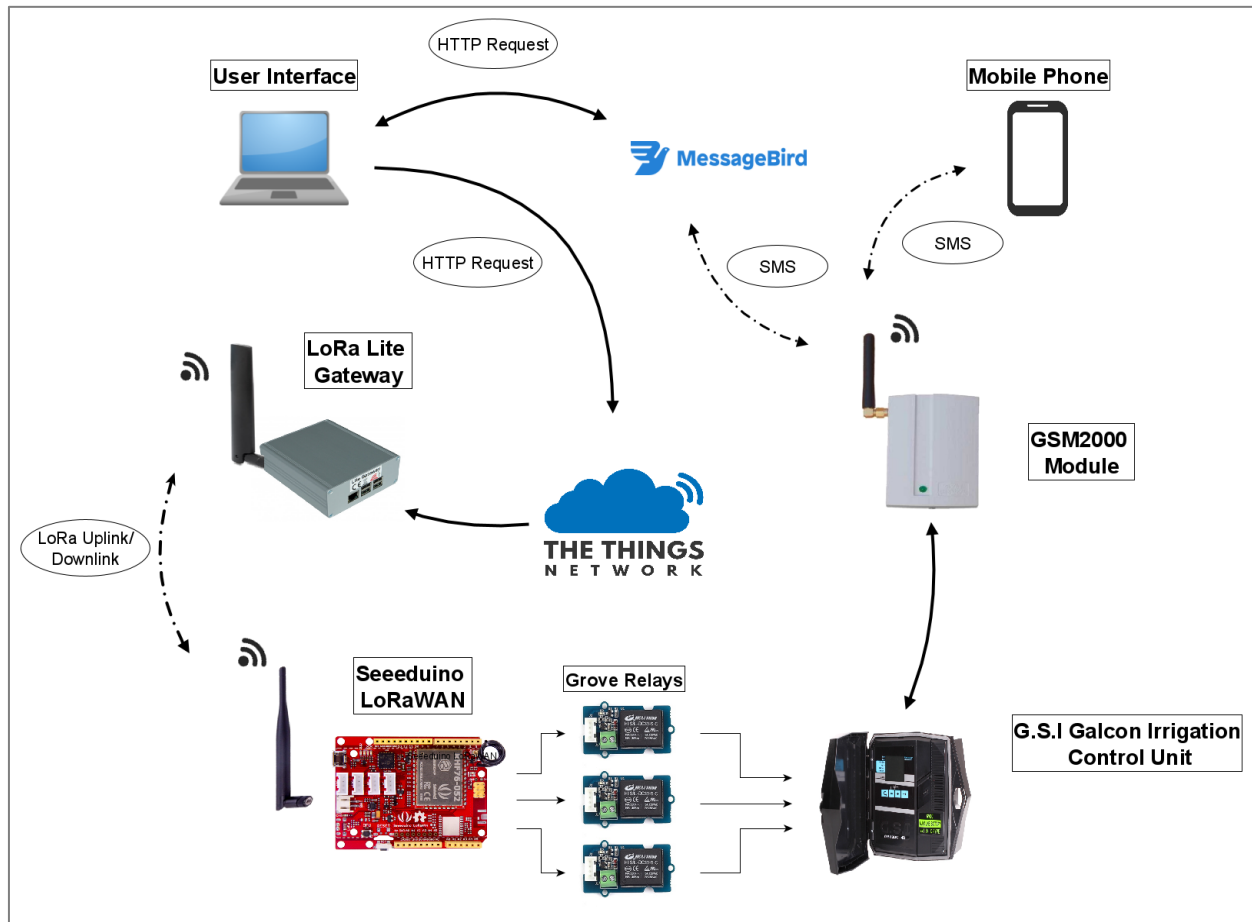


Figure 10: Remote Control System Implementation Scheme.

3.2.1 Cellular Network (2G)

3.2.1.1 Hardware

A commercial GSM remote control module was used to enable cellular network connection. The hardware is shown in Figure 11. The module is designed for monitoring of events and operation of remote-control systems. It enables the user to send and receive Short Message Service (SMS) notifications and CLIP calls in any GSM mobile phone network. The GSM2000 module has an integrated GSM transceiver chipset, four control inputs, and four relay outputs that operate in pulse mode. This means they can be either switched on for a programmable time or the state can be permanently changed. Access can be limited to a certain amount of phone numbers to improve IT-security and malicious usage. For the uplink (confirmation SMS) up to six numbers can be chosen. Phone numbers can be added and deleted remotely with a master number that is set when the device is configured. Moreover, a limit can be set for the maximum amount of SMS per day to avoid high bills in case of errors and disturbances in general. [72]



Figure 11: Danitech Alarm GSM2000 Remote Control Module universally programmable via PC. [72]

3.2.1.2 Setup and Configuration

The GSM module requires a SIM-card that is put into the slot shown in the top center of the left photo in Figure 11. The card must either have the PIN-code “1234” or no PIN at all.

The configuration is done with a software that comes with the GSM module. In the software, the user can change all possible settings, which include the configuration of the inputs and outputs, the definition of the phone numbers allowed to receive and send SMS, and some general settings. The software user can define an access code, activate status SMS, performance tests, and more.

As mentioned before, the outputs can be set to bistable, which enables an on and off switch via SMS, or to monostable, which means that the SMS must contain a time specification. Another option is to define a constant time within the software or set it to operation mode “any”, which means all parameters are defined in the SMS text. An example would be “OUT1 2:30 OUT2 T”,

which sets the output 1 to HIGH for 2 hours and 30 minutes and the output 2 to HIGH permanently with only one SMS. The producer provides a manual for the use of the configuration software with more detailed instructions. [73]

It is not desired that the operator needs to send a manually typed SMS for the control of the water system, hence, the Messagebird API was used, to make the communication programmable. Messagebird provides their users with their own library to communicate with their server [51]. The code to send a SMS with a specific message is implemented in the “Downlink_TTN.ipynb” file.

3.2.2 LoRa Network

As a second option, the LoRaWAN technology was applied. Several devices were needed to establish a LoRa network, such as a microcontroller board, a LoRa gateway, two SMA antennas, and three relays. As shown in Figure 10, the microcontroller board was programmed to send uplink and downlink payloads to the gateway. As mentioned in the theoretical part, TheThingsNetwork community supplies publicly available gateways. Unfortunately, it was observed that no gateway was available within the range of the pilot field. Therefore, a Lora Lite Gateway, which is described more detailed in the hardware chapter, was set up. Nevertheless, the server of TheThingsNetwork was used to communicate with the operator. Via HTTP requests, the operator can send downlink signals to the API of the TTN, which forwards it to the microcontroller. The microcontroller was programmed to listen to the downlink signals after every uplink and then sets the states of the relays according to the signal. It was elaborated in chapter 2.3.2 that the amount of downlink signals should be kept as low as possible. Since it is essential for the control of the pump that it receives a signal, no way led around sending downlinks. The problems that arise with this will be further elaborated in the discussion part.

3.2.2.1 Hardware

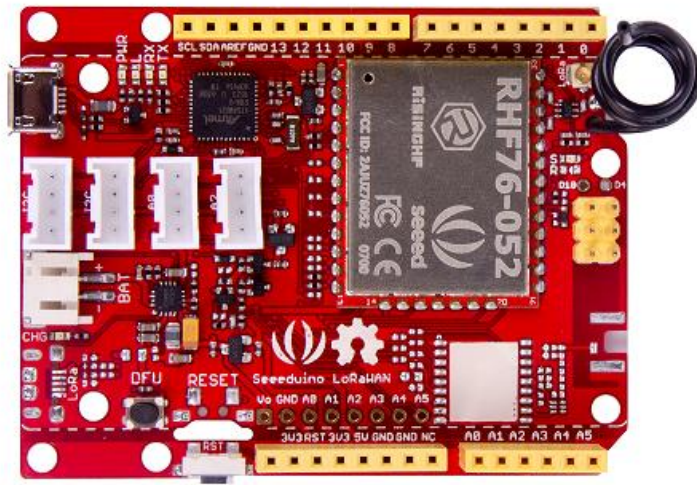


Figure 12: Seeduo LoRaWAN Microcontroller. [74]

The microcontroller that was used for the project is the Seeduo LoRaWAN microcontroller board. It has 32 kilobytes (KB) of RAM and a voltage of 3.3 V. It has 20 general purpose input/output (I/O) lines, two I2C connections, a wire antenna, and two serial connections. An LED indicates when the microcontroller is hooked up to a power supply. It is specifically designed to function on the LoRaWAN network, as it has a chip (RHF76-052) implemented for the LoRaWAN protocol. The microcontroller is compatible with Class A and C (see 2.3.2.3). Furthermore, it has a battery management chip embedded that makes charge of lithium batteries via USB possible and ensures long battery lifetime [74]. The GPS version was bought; hence, it is also possible to track the location of the microcontroller.



Figure 13: LoRa Lite Gateway

The LoRa Lite Gateway consists of an iC880A LoRaWAN concentrator, a Raspberry PI B+, and a sandwich board. According to the datasheet, it supports a radio frequency range of 863 MHz to 870 MHz, requires a supply voltage of 5V, and consumes depending on the operation mode up to 2300 mA. The transmission power is limited to a maximum of 20 dBm. It comprises four USB ports, an Ethernet port, a status LED, and requires a SubMiniature Version A (SMA) antenna. [75]

The Seeduino LoRaWAN microcontroller has a wire antenna integrated but to be on the safe side, an additional LoRa antenna kit was purchased (see Figure 14).



Figure 14: LoRa Antenna Kit. [76]

A connection between the control unit and the Seeduino Microcontroller is required to control the pump. Hence, three Grove Relays were added to the system. The operating voltage of the relays are between 3.3 and 5 Volt. An integrated light will switch on when a current is introduced. [77]



Figure 15: Grove Relay. [77]

Further, as described in the section about the LoRa Lite Gateway, an additional SMA antenna was needed. The Lora 868 MHz Antenna, shown in Figure 16, was chosen to be a good fit. It covers the frequencies from 860 MHz to 870 MHz and the antenna gain is 3 dBi. [78]



Figure 16: LoRa 868 MHz Antenne SMA. [78]

3.2.2.2 Setup and Configuration

For the setup of the Seeeduino LoRaWAN microcontroller the boards driver had to be installed. With the help of the board manager, which is integrated in the Arduino IDE, the Seeduino LoRaWAN board can be installed. Furthermore, the CayenneLPP and the LoRaWAN library needed to be downloaded and added to the sketch.

The microcontroller was programmed to send uplinks every 5 minutes using the Cayenne protocol. The uplink signals are assigned to the lowest payload possible. The DevEUI, AppEUI, and AppKEY are defined in the setup function for the communication with the TTN server. A high data rate is set by default to DR5 with a spreading factor of 7. Moreover, the adaptive data rate is activated, and the transceiver power is set to the maximum that is possible in the 868 MHz band.

All parameters can be adjusted, as it is described in section 2.3.2.4, if connection problems occur due to the range. Eight channels are set between 867.1 MHz and 868.5 MHz in steps of 0.2 MHz.

With every uplink, the controller also listens for downlinks. Hence, it can take a maximum of 5 minutes until the pump reacts to the sent downlink signal. The downlinks are sent via HTTP request, which are sent in the “downlink_ttn.py” python file. Two options were programmed: the first is an on-off code, which changes the state of the relay permanently. The other option is based on duration. In this case, the microcontroller switches the state of the relay to HIGH for as long as it is specified in the python code. The first option only works if the microcontroller has been flashed with the “Lora_downlink_bistable.ino” sketch, while the duration approach is relying on the “Lora_downlink_duration.ino” sketch. Implementing both options would cause a large payload, which creates the issues explained in section 2.3.2.4.

The API of the TTN requires a JSON object, which specifies the device ID, port, payload, and if a confirmation should be send. The payload has to be encoded in base64 to be accepted from the TTN server [79]. As described in section 2.3.2.5, the uplinks are encoded in hexadecimal and the downlinks are encoded in accordance to the encoding of the uplink. Hence, the server forwards the base64 encoded message it received from the python script, in hexadecimal encoding to the gateway, which forwards it to the microcontroller. Once the microcontroller checks for downlink signals, it will receive the hexadecimal encoded buffer and decode it to an integer. This number is used for the duration-dependent activation of the relay. The downlink and uplink signals can be tracked in the console of the TTN website [61].

To configure the LoRa Lite Gateway for the communication with the TTN server, the SD of the Raspberry Pi that is integrated in the gateway was flashed with a docker image. The image is provided by a Bavarian group named “Docker Pirates”. It was downloaded from their page: <https://blog.hypriot.com/downloads/>. To connect to the Raspberry Pi B+, an SSH connection was established in the local network. A log-in is required with the credentials provided on the GitHub page of the image provider [80]. Furthermore, a GatewayEUI is necessary for the registration of the Gateway in the TTN network [60]. The unique EUI was created by using the unique MAC address of the Ethernet interface, which is built into the Raspberry Pi, and adding 6 bytes to reach the required 64 bit length [60]. After this, the docker image was run on the Raspberry Pi, which commences a docker container, as explained in section 2.1.4. This container is restarted every time the Raspberry Pi is booted. It needs to be mentioned that a Raspberry Pi in general must not be cut off from electricity without shutting it down, as this can cause the corruption of the SD. A detailed description of the setup of the gateway using the described image can be found on the wiki-site of the Chair of Geoinformatics of the TU Munich [81]. After the gateway is up and running, it needs to be registered on the website of the TTN with the GatewayEUI that was created.

4 Discussion

4.1 Data Workflow

4.1.1 Data Storage and the SensorThings API

The use of standards for the storage, communication, and exchange of data generally bears great potential. A study of the U.S. Department of Commerce Technology Administration suggests that 15.8 billion US dollars are spent per year due to interoperability issues in the U.S. capital facilities industry [82]. If all parties used standards, industries will be not only more feasible but also long-term storage and accumulation of data will be made easier. Furthermore, services and applications can be based on standards, which enables plug-and-play uses. One example of such a service used in this work is Grafana, which provides a plugin to incorporate the Sensor Things API standard as data source. Moreover, standardization generally makes it easier for developers to use data for further programming, increases productivity, and connect applications [83].

One objective of the study project was to build a data workflow that enables efficient management of both sensor and model data with the aim of implementing a standard. The available sensor data contains only one-dimensional data (timeseries). This was ideal for the use of the Sensor Things API standards, which is meant for IoT applications.

The model data, however, which has the proprietary format DFS, is not so easy to handle. For further use of the model data it is desirable to store it in a raster format within the framework of an OGC standard. It was discovered that the Sensor Things API is not the right standard to achieve this. In theory, a large array could be stored with the help of the MultiDatastream extension. However, the observed property must be defined for every array index. The properties of the raster data matrices of the model output are usually all the same, which would mean that every array index would have the identical observed property. This is not what the Sensor Things API was established for.

A viable alternative for storing raster files is PostGIS raster [84]. PostGIS is not OGC compliant per se but implements the “Simple Features – SQL -Types and Functions 1.1” specification [85]. The FROST server is already based on a PostgreSQL and PostGIS data structure and delivers a standardized web service interface by implementing the Sensor Things API. The PostGIS raster extension could be established and the standardized web interface service could be extended. To make storage possible, an interface for the transformation from the Data File System format to PostGIS raster would be needed.

The implemented approach of storing the data as virtual sensors is also desirable. By reducing the dimensions to two, comparison between model data and physical measurements are made easier and model data can be put into an IoT environment. It is necessary that for every estimated MIKE-SHE virtual sensor one Things and one Sensors entity must be defined. This has been solved by creating identical instances, but the issues can be solved in more elegant ways. An alternative approach would be to assign the entities in the same way as the datalogger data was interpreted. This means that one model node would be defined as a Thing, which has one location but different

virtual sensors in different depths. By applying this, the semantics between sensor and model data would be consistent.

If needed, the MultiDatastream extension can be applied to store additional data in an array-based result, such as depth information or more than one observed property. Since the data loggers were equipped with three sensors for every depth, it would be also a possibility to assign these values to merely one sensor entity, which delivers result arrays that contain all three values for salinity, soil moisture content, and soil temperature that are measured by the individual sensors. By applying this, the individual sensor characteristics are lost, but the amount of sensor entities would decrease, which makes it more manageable and easier to access the data. However, this may not necessarily comply with the ideology of the Sensor Things API.

In general, it must be clear to all users how all entities are defined for different situations. For example, in the case of the Irrimode project dataloggers with various sensors were put into the soil. Every datalogger was interpreted as an individual thing entity. If somebody else interprets the three sensors in one depth as a thing instead of the whole datalogger, it will lead to misunderstandings and errors. Especially when arrays are used, as described in the example above, it can cause problems when people are not aware that there are three sensors behind one result array. It is one step to standardize the storage format and web service interface, but the semantics must be standardized, as well. In my opinion, the Thing entity is not clearly defined in the scope of the Sensor Things API standard and leaves space for interpretation [34].

4.1.2 Alternative Standards

The Sensor Things API is a standard specially designed for the web development community and offers a standardized way to describe, interact and store sensor observation data. For IoT applications the model data can be transformed according to the implementation of this project. However, a lot of metadata is lost by using the Sensor Things API because no entities exist to store the water modeling specific metadata. One option to handle this problem is to extend the Sensor Things API standard, which is possible according to the document [34].

Another option is the use of alternative standards. The OGC established a standard tailor-made for the water sector. The so-called Water Modeling Language (WaterML) provides encoding of water observation data and comprises several parts, such as: Timeseries, Ratings, Gaugings and Sections, and Surface Hydrology Features [29]. Furthermore, the standard contains another modeling language for groundwater applications (Groundwater Modeling Language – GWML2) and a section on water quality data (WaterML-WQ) [29]. The WaterML is, equivalent to the Sensor Things API, based on the O&M conceptional model. It is implemented as application schema in XML following the rules of the Geographic Markup Language (GML) [17, 77]. This enhances consistency and interoperability between the WaterML and other standards. Moreover, it harmonizes with the Australian Water Data Transfer format, the WaterML 1.0 from the United States and the German XHydro format [78, 79]. By applying this standard, further information of the model could be stored.

If in the future it is also desired to store the model data as raster files within the scope of the OGC standard suite, the use of an alternative standard is recommended. The OGC established a Web Coverage Service Interface (WCS) standard that defines the retrieval of geospatial data as so-called “coverages”. These coverages represent temporal and spatial variation phenomena, more precisely, regular and irregular grids, point clouds, and general grids [89]. This means that spatial and temporal information are related to potentially multi-dimensional properties. The MIKE-SHE model DFS files fall under the category of information that can be described by the coverages. The Web Coverage Service Interface standard consists of a core that every WCS implementation must support and various extensions [90].

4.2 Technical Improvements

Apart from conceptional improvements, technical issues must be discussed, as well. So far, there is no automated time schedule for when the scripts are run. However, it makes sense to set a time, for example, at night when all data is updated. The “schedule” module for python provides functions to make this possible [91].

Furthermore, I recommend that the scripts that update the csv-files and the FROST Server database are coupled, so that no csv-files can be updated without a POST request to the FROST Server. This way, it is ensured that no errors occur due to asynchronous updates.

The csv-files and excel serve as manual interface for data validation and definition of metadata. This bears advantages and disadvantages with it. Most people are familiar with excel and csv files, which makes the data workflow applicable for vast amount of people. However, one can argue that it is not state-of-the art technology and excel brings some problems with it. One major issue I faced was that when updating values in excel files, other cells which contain functions that point to the updated cell, were read as none type. This issue was solved by creating a function that opens the corresponding file, saves it, and closes it again. Although this works, it is not an efficient way when it comes to computational calculation time. Therefore, the alternative option to make a direct POST request to the FROST server has been programmed. This enables a translation from the API data of Pessl Instruments to the Sensor Things API standard on the fly. This approach limits the user’s expenditure to a minimum. In both cases, the scripts rely on the consistency of the API that provides the data.

4.3 Visualization

The visualization was done before the FROST Server had been setup because of an internal schedule of the company. The ideal case, however, is that the leaflet map is generated on basis of the data exported from the FROST Server. Furthermore, the model data was only visualized for one day and cannot be changed by the user yet. An interactive plot is more desirable where the user can change parameters and see model and sensor data for specified time periods. The sensor data time plot interval can be changed within the configuration excel sheets or interactively when using the Grafana dashboards. As mentioned, no automation script exists yet to assign the sensors to a specific Grafana plot when creating the map. Such a function will be necessary to create a more interactive user experience. The user should further have the chance to compare the model data to the physically measured timeseries. This could be also realized with Grafana by adding the model data into the existing sensor data plots.

4.4 Remote Control Concept

Two approaches were chosen suitable for the remote control of the on-site control unit. One was based on the second-generation cellular network. While the 2G network can be described as one of the most successful cellular networks, the amount of users has declined in recent years and some countries have started to shut down their 2G network due to new evolving technologies such as UMTS, LTE, NGMN, etc. [41, 42]. Therefore, the GSM module must be replaced by newer technologies, which are based on the fourth and fifth generation networks. As elaborated, the state of Germany is planning to provide cellular network coverage of 99% of the population in Germany and currently invests in 5G technology [71]. The responsibility for optimization lies here with the state. [94]

The other approach made use of the LoRa technology as one of the available options for low power wide area networks. A Seeduino LoRaWAN microcontroller board has been programmed to send uplinks and to receive downlinks. The signals are forwarded by a Lora Lite Gateway, which was setup to communicate with TheThingsNetwork server. To sum up, a duration-based communication from a user interface to the control-unit was established. The main problem with this approach lies within the delay of signals that originates from the uplink/downlink principle of LoRa. The microcontroller only checks for downlinks with every five minutes uplink. Hence when considering airtime, and computational time, it can take up to a maximum of approximately 6 minutes until the activation/deactivation of a relay occurs. A reduced uplink time interval would improve this at the expense of the battery lifetime. Nevertheless, the possibility of an immediate shutdown is not given and in case of an emergency, this option is crucial. One option to implement this is the “interrupt” function. This provides a way to break the script and execute an alternative function [94]. With this an emergency button can be defined, which can be activated at any time.

When setting up a gateway with TheThingsNetwork, it is usually made publicly available. If a gateway is used by many users and generally for many downlinks, it will cause further delays and

uplinks will be blocked more often. Hence, a reliable remote-control system should not be integrated in the public network. TheThingsNetwork also offers a way to connect to a private back-end [95]. However, TTN has been designed for the public network and documentation on handling private networks is limited [61]. One alternative is ChirpStack, which was designed specifically for users that want to setup their own private LoRa network and servers [96].

ChirpStack is open-source, commercially usable (MIT licensed) and offers a ready-to-use solution with a Web-interface and APIs [96]. The technology is based on four major components: LoRa Gateway Bridge, Network Server, Application Server, and Geolocation Server [96]. Different configurations are possible, for example, installation of all these four components on separate servers or setup of all components and dependencies directly on a gateway [96]. A detailed documentation is available on the ChirpStack website [96]. With the help of this service, a private network with a local backend can be established. In my opinion, this is a better solution for the remote control of agricultural control-units than using TTN.

Ideally, the microcontroller board must be programmed only once and can be run for several years depending on the battery lifetime. In case the code needs to be changed, the developer must collect the microcontroller from site. A single-board computer (e.g., Raspberry Pi) offers an alternative. This credit-card sized computer offers a feasible way of running codes and Arduino IDE sketches with low energy consumption. It can be connected to the microcontroller board and programmed remotely [97]. Thus, the combined approach is an option for a more flexible remote control with higher controllability.

An operating system has advantages over microcontrollers boards but comes at the price of elevated power consumption. Hence, a single-board computer could replace the microcontroller completely offering more flexibility and further network options. The Raspberry Pi 4 [98], for example, comes with Gigabit Ethernet, onboard wireless networking and Bluetooth. The Raspberry Pi Zero W [99], which consume much less power, also provides a wireless LAN and Bluetooth connection. In the microcontroller board industry, devices with WLAN and Bluetooth connections have been developed, too (e.g., ESP32). In conclusion, there are many alternatives for the hardware setups with different networks. According to the available network, demanded flexibility, power consumption, and general preferences of the farmers, a few setups must be established that can replace each other.

5 Outlook

In the scope of this project a full data workflow was created. This enables the request from two different data sources, the import of MIKE-SHE simulation data, and storage of all data within the scope of the IoT based Web Interface Standard SensorThings API. Another script exports the timeseries data from the FROST server for further use.

In the future other add-ons may be desirable and other data sources will be required to extend the workflow. Figure 19 shows a potential extended data workflow. As depicted in the figure, one addition to the data workflow could be a validation script preceding the data storage to prevent subsequent problems. Furthermore, developers may want to implement scripts for statistical analysis, geoprocessing, or data fusion. These can be built on the output of the FROST export script and may add further information to the database of the FROST server.

In this work, it has been illustrated how to access two different third-party APIs. Unfortunately, as long as external providers do not employ standards in their APIs, individual script will be required to convert information from other sources into the format that the further data workflow is built on. The scripts that request and transform data from the Pessl Instruments and Sentek Technologies API handle xml and json format. Thus, these scripts can be used as base for further acquisition of data. However, in the future other technologies are likely going to play a part in smart farming systems, such as image processing and drone flight data. Script will be required that integrate such data in the FROST Server. The SensorThings API supports data from remote sensing, which makes the established data workflow suitable for these applications.

The overall aim of the Irrimode project is the digitization and optimization of the irrigation system. The first step has been accomplished in the framework of this study project. However, the development of an optimization algorithm lies still ahead. The optimization can be accomplished in different ways. Analytical solutions as well as machine learning algorithms are viable options.

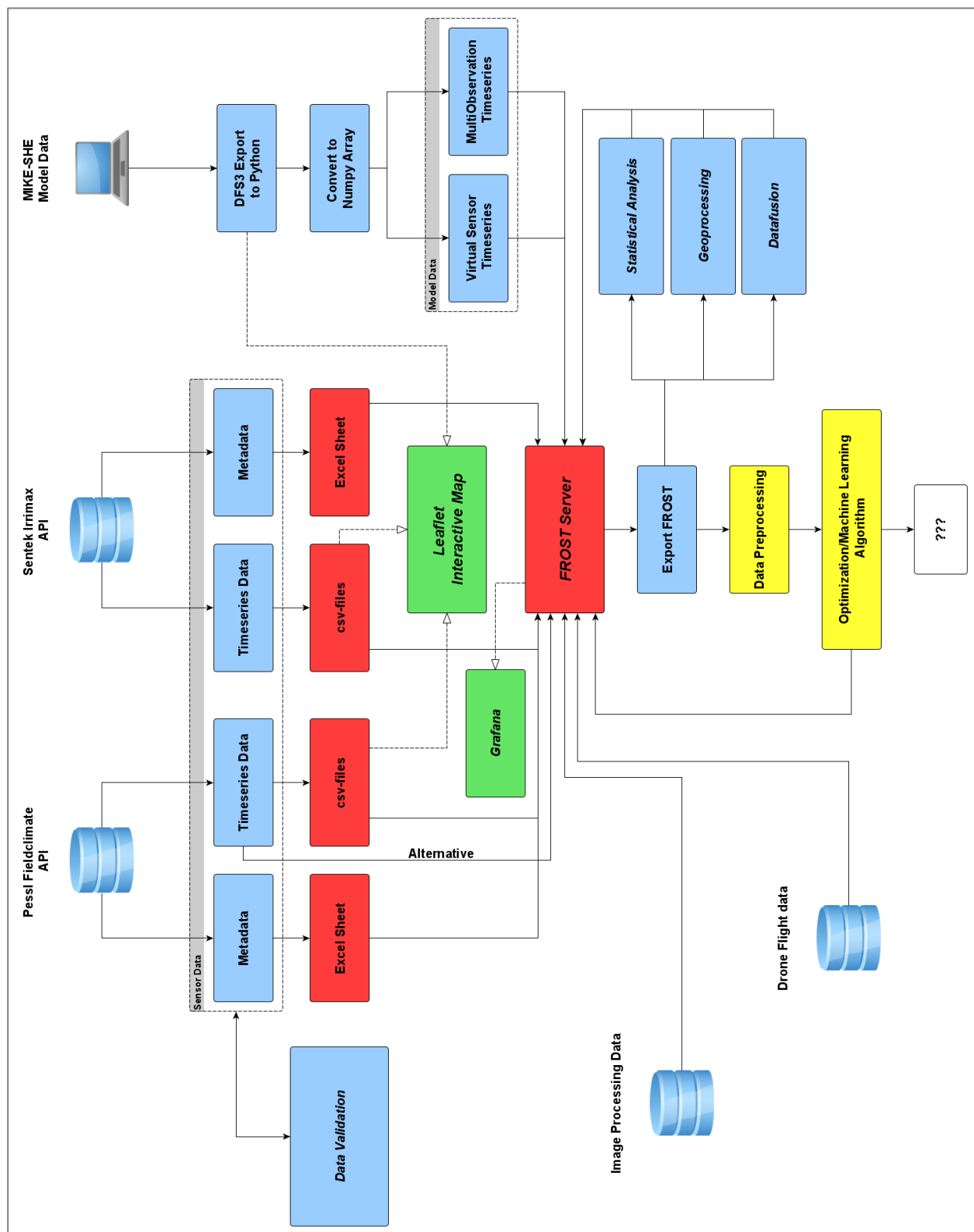


Figure 179: Possible extensions of data workflow. [78]

Another goal could be to create a machine learning algorithm as supplement to the MIKE-SHE model to ensure real-time simulation. The pilot crop field uses drip irrigation for watering. For this application, the created model is fast enough because the watering happens rather slowly, roughly 2 mm per hour. For other farms, which use irrigation methods with a higher flux rate, the numerical model is too slow for the optimization in real-time. Furthermore, simulations of different scenarios are beneficial for optimization. This requires high computational time, which makes it not suitable for real-time application. To resolve this problem a machine learning algorithm could offer a solution, which generally needs less computational time. The algorithm would replace the MIKE-SHE model and predict soil parameters.

6 Conclusion

The goal of this work was to establish a data workflow that offers a resilient and effective way of digitizing and storing data. By implementing the Sensor Things API standard, this goal was achieved. This Web Service Interface standard offers a well-conceived way of data storage and interaction for IoT applications with a high level of detail. If further algorithms are built based on this standard, developers will face fewer interoperability issues. Both sensor and model data have been implemented within the FROST server that implements the Sensor Things API. It has been discovered that it is possible to store model data in the framework of the Sensor Things API either as virtual sensors or with the help of the data array extension. Since the sensor and model data are stored in the same manner, comparison and further use are made easy. Moreover, it has been found that statistical data can be stored within the Sensor Things API by embedment of the ISO 19157 and the QualityML.

As it was expected, the storage of whole raster files does not make sense within the Sensor Things API standard. The Web Coverage Service standards of the OGC is more suitable for this purpose. The WaterML may be interesting in the water modelling sector to save model metadata that is otherwise lost.

One objective was to create a data structure that enables other developers to import the stored data. Hence, the workflow was extended by the means of an export script that enables requests of the standardized data.

The visualization was achieved through an interactive map that integrated interactive plots of the sensor and model data. This increases transparency and makes decision making easier for the operator. Furthermore, a graphical user interface for the data workflow has been established.

Two approaches were implemented to create a redundant and resilient system for the control of the on-site control unit. A GSM module enables the operator to send control-SMS in the second-generation cellular network, whereas a microcontroller board was programmed for the use of a LoRa network. In the future, the GSM module should be replaced by a technology that works on the fourth-generation network. The LoRa network was set up to be public. To make the remote control more reliable it should be employed in form of a private network.

7 References

- [1] 'Machine Learning in Agriculture'. [Online]. Available: <https://objectcomputing.com/industries/agriculture/machine-learning-in-agriculture>. [Accessed: 23-Dec-2019].
- [2] 'IrriMode Projekt'. [Online]. Available: <http://www.irrimode.de/>. [Accessed: 11-Dec-2019].
- [3] P. Keilholz *et al.*, 'Bewässerung 4.0: ein möglicher Ansatz zur weiteren Optimierung der Bewässerung', *Korresp. Wasserwirtsch.*, vol. 12, no. 9, 2019.
- [4] 'Pessl Instruments', *Pessl Instruments*. [Online]. Available: <http://www.pesslinstruments.com/>. [Accessed: 17-Sep-2019].
- [5] 'Mike SHE'. [Online]. Available: <https://www.mikepoweredbydhi.com/download/mike-2017-sp2/mike-she?ref=%7B40160C10-5509-4460-A36F-FA2759EAC02F%7D>. [Accessed: 17-Sep-2019].
- [6] 'DFS file system - MIKE for developers docs'. [Online]. Available: http://docs.mikepoweredbydhi.com/core_libraries/dfs/dfs-file-system/. [Accessed: 10-Dec-2019].
- [7] 'Welcome to Python.org', *Python.org*. [Online]. Available: <https://www.python.org/>. [Accessed: 17-Dec-2019].
- [8] 'Project Jupyter | Home'. [Online]. Available: <https://jupyter.org/>. [Accessed: 29-Nov-2019].
- [9] 'The Jupyter Notebook — Jupyter Notebook 6.0.2 documentation'. [Online]. Available: <https://jupyter-notebook.readthedocs.io/en/stable/notebook.html>. [Accessed: 29-Nov-2019].
- [10] 'Features - PyCharm', *JetBrains*. [Online]. Available: <https://www.jetbrains.com/pycharm/features/>. [Accessed: 29-Nov-2019].
- [11] 'NumPy — NumPy'. [Online]. Available: <https://numpy.org/>. [Accessed: 18-Dec-2019].
- [12] 'Python Data Analysis Library — pandas: Python Data Analysis Library'. [Online]. Available: <https://pandas.pydata.org/>. [Accessed: 18-Dec-2019].
- [13] 'Pandas ecosystem — pandas 0.25.3 documentation'. [Online]. Available: <https://pandas.pydata.org/pandas-docs/stable/ecosystem.html>. [Accessed: 18-Dec-2019].
- [14] 'GeoPandas 0.6.0 — GeoPandas 0.6.0 documentation'. [Online]. Available: <http://geopandas.org/>. [Accessed: 18-Dec-2019].
- [15] 'Requests: HTTP für menschliche Wesen — Requests 1.2.0 Dokumentation'. [Online]. Available: <https://de.python-requests.org/de/latest/>. [Accessed: 18-Dec-2019].
- [16] 'pickle — Python object serialization — Python 3.8.1rc1 documentation'. [Online]. Available: <https://docs.python.org/3/library/pickle.html>. [Accessed: 18-Dec-2019].
- [17] 'Leaflet — an open-source JavaScript library for interactive maps'. [Online]. Available: <https://leafletjs.com/>. [Accessed: 17-Sep-2019].
- [18] 'Folium — Folium 0.10.0 documentation'. [Online]. Available: <https://python-visualization.github.io/folium/>. [Accessed: 17-Sep-2019].
- [19] 'About Docker - Management & History', *Docker*. [Online]. Available: <https://www.docker.com/company>. [Accessed: 29-Nov-2019].
- [20] 'What is a Container?', *Docker*. [Online]. Available: <https://www.docker.com/resources/what-container>. [Accessed: 29-Nov-2019].
- [21] 'What is Docker?', *Opensource.com*. [Online]. Available: <https://opensource.com/resources/what-docker>. [Accessed: 29-Nov-2019].

- [22] 'Was ist Docker?' [Online]. Available: <https://www.dev-insider.de/was-ist-docker-a-733683/>. [Accessed: 29-Nov-2019].
- [23] 'Docker Container - an overview | ScienceDirect Topics'. [Online]. Available: <https://www.sciencedirect.com/topics/computer-science/docker-container>. [Accessed: 29-Nov-2019].
- [24] 'GNU tar 1.32: Basic Tar Format'. [Online]. Available: https://www.gnu.org/software/tar/manual/html_node/Standard.html. [Accessed: 17-Dec-2019].
- [25] 'Docker Documentation', *Docker Documentation*, 28-Nov-2019. [Online]. Available: <https://docs.docker.com/>. [Accessed: 29-Nov-2019].
- [26] 'Docker registry first steps | OCTO Talks !' [Online]. Available: <https://blog.octo.com/en/docker-registry-first-steps/>. [Accessed: 29-Nov-2019].
- [27] 'Grafana Features | Grafana Labs'. [Online]. Available: <https://grafana.com/grafana/>. [Accessed: 03-Dec-2019].
- [28] 'OGC® Standards and Supporting Documents | OGC'. [Online]. Available: <https://www.opengeospatial.org/standards/>. [Accessed: 02-Dec-2019].
- [29] 'OGC® WaterML | OGC'. [Online]. Available: <https://www.opengeospatial.org/standards/waterml>. [Accessed: 02-Dec-2019].
- [30] B. Brodaric, 'OGC WaterML 2: Part 4 – GroundWaterML 2 (GWML2)', 06-Mar-2017. [Online]. Available: <http://docs.opengeospatial.org/is/16-032r2/16-032r2.html>. [Accessed: 27-Sep-2019].
- [31] A. Bröring *et al.*, 'New Generation Sensor Web Enablement', *Sensors*, vol. 11, pp. 2652–99, Dec. 2011, doi: 10.3390/s110302652.
- [32] M.C. Kastelijns, 'Making Sense of Standards -An Evaluation and Harmonisation of Standards in the Sensor Web'. [Online]. Available: http://www.gdmc.nl/publications/2016/Making_Sense_of_Standards_Sensor_Web.pdf. [Accessed: 27-Sep-2019].
- [33] Michael Jacoby, 'OGC SensorThingsAPI and FROST Server', 29-Oct-2019.
- [34] S. Liang, C.-Y. Huang, and T. Khalafbeigi, 'OGC SensorThings API Part 1: Sensing', *SensorThings SWG*, 26-Jul-2016. [Online]. Available: http://docs.opengeospatial.org/is/15-078r6/15-078r6.html#table_3. [Accessed: 04-Sep-2019].
- [35] 'MQTT'. [Online]. Available: <http://mqtt.org/>. [Accessed: 18-Dec-2019].
- [36] 'OASIS Open Data Protocol (OData) TC | OASIS'. [Online]. Available: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=odata. [Accessed: 18-Dec-2019].
- [37] ITU-T, 'Y.2060 : Overview of the Internet of things', 2012. [Online]. Available: <https://www.itu.int/rec/T-REC-Y.2060-201206-I>. [Accessed: 02-Dec-2019].
- [38] OGC and ISO 19156:2001, OGC 10-004r3 and ISO 19156:2011(E), 'OGC Abstract Specification: Geographic information — Observations and Measurements'. 17-Sep-2013.
- [39] 'Eclipse Whiskers', 18-Jan-2019. [Online]. Available: <https://github.com/eclipse-archived/whiskers.js>. [Accessed: 02-Dec-2019].
- [40] 'gost/home', 31-Oct-2019. [Online]. Available: <https://github.com/gost/home>. [Accessed: 02-Dec-2019].
- [41] 'FraunhoferIOSB/FROST-Server', *GitHub*. [Online]. Available: <https://github.com/FraunhoferIOSB/FROST-Server>. [Accessed: 29-Nov-2019].
- [42] *mozilla-sensorweb/sensorthings*. mozilla-sensorweb, 2019.

- [43] International Organization for Standardization, 'ISO 19157:2013 Geographic Information - Data Quality'. 2013.
- [44] 'DQ Element - Federation of Earth Science Information Partners'. [Online]. Available: http://wiki.esipfed.org/index.php/DQ_Element. [Accessed: 02-Dec-2019].
- [45] 'QualityML'. [Online]. Available: <http://www.qualityml.org/>. [Accessed: 02-Dec-2019].
- [46] 'UncertML : Exchanging uncertainty since 2006.', 29-Oct-2016. [Online]. Available: <https://web.archive.org/web/20161029215725/http://www.uncertml.org/>. [Accessed: 02-Dec-2019].
- [47] 'OGC Releases Discussion Papers: Uncertainty Markup Language and OGC WCS Extension for netCDF Weather Data | OGC'. [Online]. Available: <https://www.opengeospatial.org/node/1002>. [Accessed: 27-Sep-2019].
- [48] 'OGC seeks public comment on SensorThings API standard for Internet of Things | OGC'. [Online]. Available: <https://www.opengeospatial.org/pressroom/pressreleases/2238>. [Accessed: 29-Nov-2019].
- [49] C.-L. Tseng *et al.*, 'Feasibility study on application of GSM–SMS technology to field data acquisition', *Comput. Electron. Agric.*, vol. 53, no. 1, pp. 45–59, Aug. 2006, doi: 10.1016/j.compag.2006.03.005.
- [50] A. Acker, 'The Short Message Service: Standards, infrastructure and innovation', *Telemat. Inform.*, vol. 31, no. 4, pp. 559–568, Nov. 2014, doi: 10.1016/j.tele.2014.01.004.
- [51] 'Communication, solved. - MessageBird'. [Online]. Available: <https://www.messagebird.com/en/>. [Accessed: 04-Dec-2019].
- [52] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, 'A comparative study of LPWAN technologies for large-scale IoT deployment', *ICT Express*, vol. 5, no. 1, pp. 1–7, Mar. 2019, doi: 10.1016/j.icte.2017.12.005.
- [53] 'Sigfox Buy'. [Online]. Available: <https://buy.sigfox.com/buy>. [Accessed: 17-Sep-2019].
- [54] 'Classes', *The Things Network*, 29-Nov-2019. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/classes.html>. [Accessed: 04-Dec-2019].
- [55] '(PDF) Impact of Spreading Factor Imperfect Orthogonality in LoRa Communications'. [Online]. Available: https://www.researchgate.net/publication/319486965_Impact_of_Spreading_Factor_Imperfect_Orthogonality_in_LoRa_Communications. [Accessed: 19-Dec-2019].
- [56] 'Modulation & Data Rate', *The Things Network*, 29-Nov-2019. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/modulation-data-rate.html>. [Accessed: 04-Dec-2019].
- [57] 'LoRaWAN frequencies', *Man Mobile Machine*. [Online]. Available: <https://www.manmobilemachine.com/blank-1/2018/06/06/LoRaWAN-frequencies>. [Accessed: 04-Dec-2019].
- [58] S. F. Barrett and D. J. Pack, *Microcontrollers fundamentals for engineers and scientists*, 1. ed. San Rafael, Calif.: Morgan & Claypool, 2006.
- [59] European Telecommunications Standards Institute, 'ETSI EN 300 220-1', Jan-2012. [Online]. Available: https://www.etsi.org/deliver/etsi_en/300200_300299/30022001/02.04.01_40/en_30022001v020401o.pdf. [Accessed: 04-Dec-2019].

- [60] 'Addressing & Activation', *The Things Network*, 29-Nov-2019. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/addressing.html>. [Accessed: 04-Dec-2019].
- [61] 'The Things Network'. [Online]. Available: <https://www.thethingsnetwork.org/>. [Accessed: 29-Nov-2019].
- [62] 'Duty Cycle', *The Things Network*, 29-Nov-2019. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/duty-cycle.html>. [Accessed: 04-Dec-2019].
- [63] 'The Things Network'. [Online]. Available: <https://thethingsnetwork.org/airtime-calculator>. [Accessed: 04-Dec-2019].
- [64] 'Limitations', *The Things Network*, 29-Nov-2019. [Online]. Available: <https://www.thethingsnetwork.org/docs/lorawan/limitations.html>. [Accessed: 04-Dec-2019].
- [65] 'LoRaWAN® for Developers | LoRa Alliance®'. [Online]. Available: <https://loralliance.org/lorawan-for-developers>. [Accessed: 04-Dec-2019].
- [66] 'IPSO Smart Objects - OMA SpecWorks'. [Online]. Available: <https://www.omaspecworks.org/develop-with-oma-specworks/ipso-smart-objects/>. [Accessed: 04-Dec-2019].
- [67] 'Cayenne Docs'. [Online]. Available: <https://developers.mydevices.com/cayenne/docs/lorawan/#lorawan-cayenne-low-power-payload>. [Accessed: 04-Dec-2019].
- [68] 'MaxWin959/study_project', *GitHub*. [Online]. Available: https://github.com/MaxWin959/study_project. [Accessed: 05-Dec-2019].
- [69] German Meteorological Service, 'Wetter und Klima - Deutscher Wetterdienst - Leistungen - Analysen radarbasierter stündlicher (RW) und täglicher (SF) Niederschlagshöhen'. [Online]. Available: <https://www.dwd.de/DE/leistungen/radolan/radolan.html>. [Accessed: 11-Dec-2019].
- [70] 'Cloudbasierte Bewässerungssteuerung Galcon GSI, ANDO Technik'. [Online]. Available: <https://www.ando-technik.de/steuerungen/bewaesserungssteuerungen/galcon-gsi-bewaesserungssteuerung.html>. [Accessed: 11-Dec-2019].
- [71] 'Frequenzvergabe, Gesamtstrategie für flächendeckenden Mobilfunkausbau und Mobilfunkstandard 5G', *Startseite*. [Online]. Available: <https://www.bundesregierung.de/breg-de/themen/digital-made-in-de/frequenzvergabe-gesamtstrategie-fuer-flaechendeckenden-mobilfunkausbau-und-mobilfunkstandard-5g-1546640>. [Accessed: 17-Sep-2019].
- [72] 'Danitech Alarm GSM Modul Alarmmodul: Amazon.de: Elektronik'. [Online]. Available: <https://www.amazon.de/Alarm-Alarmmodul-Fernschaltmodul-universal-programmierbar/dp/B008JTQQDE>. [Accessed: 17-Sep-2019].
- [73] 'Alarm GSM Fernschalt Modul universal per PC programmierbar, 94,99 €', *Alarm GSM Fernschalt Modul universal per PC programmierbar, 94,99 €*. [Online]. Available: <https://adko.eu/Alarm-GSM-Fernschalt-Modul-universal-per-PC-programmierbar>. [Accessed: 11-Dec-2019].
- [74] 'Seeeduino LoRaWAN W/GPS'. [Online]. Available: <https://www.seeedstudio.com/Seeeduino-LoRaWAN-W-GPS-p-2781.html>. [Accessed: 11-Dec-2019].

- [75] 'LoRa Lite Gateway - Wireless Solutions by IMST GmbH'. [Online]. Available: <https://wireless-solutions.de/products/long-range-radio/lora-lite-gateway.html>. [Accessed: 12-Dec-2019].
- [76] 'LoRa Antenna Kit'. [Online]. Available: <https://www.seeedstudio.com/LoRa-Antenna-Kit-p-2787.html>. [Accessed: 12-Dec-2019].
- [77] 'Grove - Relay - Seeed Wiki'. [Online]. Available: <http://wiki.seeedstudio.com/Grove-Relay/>. [Accessed: 12-Dec-2019].
- [78] 'LoRa 868 MHz Antenne SMA Stecker 3 dBi omnidirektional mit Kippgelenk schwarz', *BerryBase*. [Online]. Available: <https://www.berrybase.de/audio-video/antennen/sma/lora-868-mhz-antenne-sma-stecker-3-dbi-omnidirektional-mit-kippgelenk-schwarz>. [Accessed: 11-Dec-2019].
- [79] 'HTTP | The Things Network'. [Online]. Available: <https://www.thethingsnetwork.org/docs/applications/http/>. [Accessed: 12-Dec-2019].
- [80] 'netceteragroup/rpi-ttn-gateway', *GitHub*. [Online]. Available: <https://github.com/netceteragroup/rpi-ttn-gateway>. [Accessed: 12-Dec-2019].
- [81] 'LoRaWAN Gateway - IMST LoRa Lite - Geoinformatik III – Geo Sensor Web - TUM Wiki'. [Online]. Available: <https://wiki.tum.de/pages/viewpage.action?spaceKey=geosensorweb&title=LoRaWAN+Gateway+-+IMST+LoRa+Lite>. [Accessed: 12-Dec-2019].
- [82] M. P. Gallaher, A. C. O'Connor, J. L. Dettbarn, Jr., and L. T. Gilday, 'Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry', National Institute of Standards and Technology, NIST GCR 04-867, Aug. 2004.
- [83] Deutsches Institut für Normung, Ed., *Gesamtwirtschaftlicher Nutzen der Normung: unternehmerischer Nutzen 1 ; Wirkungen von Normen ; Teil A: ökonomische Wirkungen der überbetrieblichen Normung ; Teil B: Verknüpfung der Ergebnisse*. Berlin: Beuth, 2000.
- [84] 'Chapter 9. Raster Reference'. [Online]. Available: https://postgis.net/docs/RT_reference.html. [Accessed: 13-Dec-2019].
- [85] 'OGC Implementing/Compliant Product Details | OGC'. [Online]. Available: <https://www.opengeospatial.org/resource/products/details/?pid=509>. [Accessed: 13-Dec-2019].
- [86] 'Geography Markup Language | OGC'. [Online]. Available: <https://www.opengeospatial.org/standards/gml>. [Accessed: 13-Dec-2019].
- [87] 'XHydro'. [Online]. Available: http://www.xhydro.de/index_en.html. [Accessed: 13-Dec-2019].
- [88] 'WDTF: Water Information: Bureau of Meteorology'. [Online]. Available: <http://www.bom.gov.au/water/standards/wdtf/>. [Accessed: 13-Dec-2019].
- [89] I. Standard, 'OGC Web Coverage Service (WCS) 2.1 Interface Standard - Core', *WCS SWG*, 16-Aug-2018. [Online]. Available: <http://docs.opengeospatial.org/is/17-089r1/17-089r1.html#1>. [Accessed: 23-Dec-2019].
- [90] 'WebHome < CoveragesDWG < OGC Public Wiki'. [Online]. Available: https://external.opengeospatial.org/twiki_public/CoveragesDWG/WebHome. [Accessed: 23-Dec-2019].
- [91] 'sched — Event scheduler — Python 3.8.1 documentation'. [Online]. Available: <https://docs.python.org/3/library/sched.html>. [Accessed: 24-Dec-2019].

- [92] 'Mobilfunk Generationen'. [Online]. Available: <https://www.startmobile.net/mobilfunk-generationen/>. [Accessed: 04-Dec-2019].
- [93] heise online, 'GSM-Zukunft in Deutschland, Österreich und der Schweiz ist offen', *heise online*. [Online]. Available: <https://www.heise.de/newsticker/meldung/GSM-Zukunft-in-Deutschland-Oesterreich-und-der-Schweiz-ist-offen-3582914.html>. [Accessed: 04-Dec-2019].
- [94] 'Arduino-Referenz'. [Online]. Available: <https://www.arduino.cc/reference/de/language/functions/external-interrupts/attachinterrupt/>. [Accessed: 24-Dec-2019].
- [95] 'Connect to a private back-end', *The Things Network*, 12-Dec-2019. [Online]. Available: <https://www.thethingsnetwork.org/docs/gateways/multitech/connect-private.html>. [Accessed: 25-Dec-2019].
- [96] 'ChirpStack, open-source LoRaWAN Network Server stack'. [Online]. Available: <https://www.chirpstack.io/>. [Accessed: 25-Dec-2019].
- [97] 'Tutorial: Remote Development on Raspberry Pi - Help | PyCharm'. [Online]. Available: <https://www.jetbrains.com/help/pycharm/remote-development-on-raspberry-pi.html>. [Accessed: 03-Sep-2019].
- [98] 'Buy a Raspberry Pi 4 Model B – Raspberry Pi'. [Online]. Available: <https://www.raspberrypi.org>. [Accessed: 04-Sep-2019].
- [99] 'Buy a Raspberry Pi Zero – Raspberry Pi'. [Online]. Available: <https://www.raspberrypi.org>. [Accessed: 04-Sep-2019].