University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Implementation of a Mobile App to Access and Manage Wireless Sensor Networks

Michael Keller
Zürich, Switzerland
Student ID: 07-708-506

Supervisor: Dr. Corinna Schmitt, Dr. Thomas Bocek
Date of Submission: November 14, 2014

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Heutige Nutzer verlangen immer mehr nach Frameworks, sowohl für die Konfiguration und Verwaltung, wie auch für die Betrachtung ihrer kabellosen Sensornetzwerken. CoMaDa ist ein solches Framework, welches diesen Anforderungen gerecht wird, aber keine Betrachtung ausserhalb des Sensornetzwerkes zulässt. Aus diesem Grund wurde in dieser Arbeit die WebMaDa-Erweiterung für CoMaDa entwickelt, welche eine solche mobile Betrachtung ermöglicht. Dies wird mit Hilfe von einer Datenbank, einer Website, Zugangskontrolle und responsivem Webdesign erreicht, wodurch Sensornetzwerke überall und auf allen mobilen Geräten, unabhängig ihrer Grösse, betrachtet werden können. WebMaDa ist dabei genau so flexible und Hardware unabhängig wie CoMaDa, aber mit dem Vorteil, dass alle Nutzer mit den erforderlichen Zugriffsrechten ein Sensornetzwerk betrachten können.

Users request comfortable frameworks not only to configure and manage their wireless sensor networks (WSNs) but also to monitor them independent of the user's location. CoMaDa is a framework supporting the aforementioned two requirements but does not allow monitoring from outside the network using the Internet. Thus, the Web-based Mobile Access and Data Handling (WebMaDa) framework supports those mobility requirements and, therefore, incorporates online database storage, access control management, and visualization with responsive design for different screen sizes of mobile devices (e.g., smartphones, tablets). WebMaDa is as flexible and hardware independent as CoMaDa and additionally allows the user to monitor WSNs on mobile devices if they are the owner or have the corresponding rights.

Acknowledgments

First and foremost I want to thank my main supervisor Dr. Corinna Schmitt for her continuous feedback and support all the way from the early concepts of 'a mobile app to access and manage WSNs' to the final version of WebMaDa and this thesis. Without her, it would not have been possible. I also want to thank Professor Burkhard Stiller, head of the Communication Systems Group at the University of Zurich, for providing me with the chance to write this thesis. I have always enjoyed his lectures and I am happy to now conclude my academic career with a project in his group. Additionally, I want to thank my assistant supervisor Dr. Thomas Bocek, who has, especially in the beginning of my work, but also throughout the process, provided me with ongoing support with programming issues. Lastly, I want to thank my family and friends for their moral support. Specifically, I want to mention Mehmet Bekooglu, Marc Tobler, Sebastian Golaszewski, Jeton Memeti, and Pascal Muther who - while working on their own theses - helped me keep my motivation when I needed it the most.

Contents

Chapter 1

Introduction

1.1 Motivation

Wireless Sensor Networks (WSNs) have gained a lot of popularity in research throughout the past years as pointed out in [2]. Due to recent advances in wireless communication and electronics, it became possible to develop cheap and highly functional sensor nodes. Those tiny, sensing, data processing, and communicating components are used to build the modern WSNs whose capabilities are significantly superior compared to traditional, stand-alone sensors [2]. In such a WSN tens, hundreds or sometimes even thousands of sensors are deployed over an area of interest, either in random fashion (e.g., dropping them from an aircraft) or manually in strategic locations (e.g., for intrusion detection) [17]. Those sensors observe and measure ambient conditions in their environment, such as heat, pressure, light, sound, vibration, or presence of objects. The collected data and observed events are forwarded to so called sinks, that are post-processing the data and possibly invoking appropriate actions, like sending an alarm notification or forwarding it to analysis entities (e.g., labs, servers) [17].

Due to its popularity and advantages over traditional sensors, it is not surprising that research is proposing to apply WSNs in a great number of fields, such as weather monitoring, disaster management, intrusion detection, target tracking, and tactical surveillance [17]. Other application fields can be found in building automation and industrial control [7], medical monitoring [28], high fidelity electricity monitoring [23], smart grids [15], or earthquake detection [41].

While those examples of WSN research is directed at a specific problem or area of application, [37] has taken a more general approach. The authors are introducing the Configuration, Management, and Data handling (CoMaDa) framework, which can be used for WSNs independent of their area of application. It enables a user to program sensors, deploy WSNs, and visualize their measurements with heterogeneous hardware, meaning all kinds of different sensors can be used in the same WSN. Moreover, it has since been extended by a project, so that sensors use encryption before communicating their measurements wirelessly to protect the WSN from eavesdropping [33].

But advances in WSN research do not stop there. In recent years, thanks to the rise of mobile computing, WSNs were also extended with some degree of mobility. Examples of research with such mobile enabled wireless sensor networks (mWSN) can be found in [17], [5] or [31]. In [44], the authors even go further and introduce a SD card combined with a phone application that enables a phone to work as mobile sink and display WSN data. Until now, CoMaDa was lacking any mobility support as it can only display WSN readings on a rather powerful computer that is physically connected to the sink node of the deployed WSN.

1.2 Description of Work

This thesis is extending the CoMaDa framework in such a way that it supports mobility as well. That means a new, extended framework is created, which builds on CoMaDa's existing functionalities and extends it with the possibility to observe WSN data (e.g., the topology, sensor measurements, data packets) on mobile devices. Moreover, the extended framework uses the world wide web to transport the data onto mobile devices and, thus, allowing a user to monitor his WSN independent of his location. Because the Internet is used for data transport, the CoMaDa extension that is being implemented in this project and is described in this paper is referred to as the Web-based Mobile Access and Data Handling framework, or WebMaDa in short.

The main goal of this thesis is to extend the CoMaDa framework onto mobile devices in order to meet the requirements of a new, mobile generation of users. Apart from that, security is a rather big issue as WSNs can contain sensible information about an area and the people in them. This is why proper authentication and security mechanisms have to be considered as well, especially since the world wide web is used to transport data.

1.3 Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 outlines and summarizes scientific work that has been done on topics relevant to this thesis (e.g., WSNs, CoMaDa, mobility in WSNs, web based network management, security, and authentication). After that, the design decisions that are made for this project are outlined and justified in Chapter 3. Then, Chapter 4 shows the actual implementation of WebMaDa and its important components in detail, followed by an evaluation of the key elements of WebMaDa. Finally, a summary and conclusions together with ideas for future work is completing the thesis.

Chapter 2

Related Work

In this Chapter, an overview of the fields that are covering topics relevant to this work is given. By introducing related work, light will be shed on the topics of WSNs in general, the CoMaDa-Framework, mobile devices interacting with WSNs, web based network management, as well as security and authentication. This Chapter is not meant to give a complete listing on all papers in those areas, but to show the current state of those areas by pointing out the relevant research.

2.1 Wireless Sensor Networks

There have been many research projects in the recent years that concerned themselves with WSNs. In fact so many, that [2] felt the need to survey protocols and algorithms proposed for sensor networks in 2002 to 'provide a better understanding of the current research issues in this emerging field'. In 2008, [43] has surveyed the wireless sensing landscape again in order to update the findings by looking at more current literature. The fact that those two papers alone have been cited by over 5000 scientific publications ([22], [38]) goes to show that WSN research has truly become an important field in computer science.

Both [2] and [43] characterize and describe WSNs as a number of sensor nodes (a few up to thousands) densely deployed either inside or very close to a phenomenon that should be observed. They are working together to monitor a region to obtain environment data, and also use their processing abilities to carry out simple computations locally and transmit only the required and partially processed data. The position of sensors does not have to be predetermined: In an unstructured WSN, a dense collection of sensor nodes may be placed into the field in an ad-hoc, random manner. This kind of WSN will perform its monitoring and reporting functions unattended, as network maintenance (e.g., managing connectivity) is difficult with a great number of randomly distributed sensors. In case of structured networks, some or even all sensors are placed at their fixed, pre-determined location. This makes it easier to maintain and manage, as fewer sensors can be used at specific locations to guarantee coverage of the field's environment.

When looking at the bigger picture, WSNs are part of an Internet revolution that started in 1990s with industrial automation systems: The Internet of Things (IoT). According to [39], the idea behind IoT is that more and more embedded devices become IP enabled and, therefore, become an integral part of the Internet. Examples of such embedded devices and systems, also known as smart objects, would be mobile phones, personal health devices, home automation, industrial automation, smart metering, and environmental monitoring systems. It is estimated that the scale of IoT has the potential to reach trillions of devices becoming IP-enabled. Even though early research on WSN has considered their networks as completely isolated with no need for Internet compatibility or standards, this has changed more recently. Standards, marketable applications and Internet services became more important and lead the WSN community to become involved with IoT standards. [39]

[43] states, that there are five different types of WSNs: Terrestrial, underground, underwater, multi-media, and mobile WSN. The paper has gathered the definitions, challenges, and applications of each type of WSN, which can be seen in Figure 1:

	Terrestrial WSN	Underground WSN	Underwater WSN	Multi-media WSN	Mobile WSN
Definition	A network consists of hundreds to thousands of sensor nodes deployed on land	A network consists of wireless sensor nodes deployed in caves or mines or underground	A network consists of wireless sensor and vehicles deployed into the ocean environment	A network consists of wireless sensor devices that have the ability to store, process, and retrieve multi-media data such as video, audio, and images	A network consists of mobile sensor nodes that have the ability to move
Challenges	– In-network data aggregation to improve performance across communication, energy cost, and delay – Minimizing energy cost – Reduce the amount of data communication – Finding the optimal route – Distributing energy consumption – Maintaining network connectivity – Eliminating redundancy	– Expensive deployment maintenance, and equipment cost – Threats to device such as the environment and animals – Battery power cannot easily be replaced – Topology challenges with pre-planned deployment – High levels of attenuation and signal loss in communication	– Expensive underwater sensors – Hardware failure due to environment effects (e.g., corrosion) – Battery power cannot easily be replaced – Sparse deployment – Limited bandwidth – Long propagation delay, high latency, and fading problems	– In-network processing, filtering, and compressing of multi-media content – High energy consumption and bandwidth demand – Deployment based on multi-media equipment coverage – Flexible architecture to support different applications – Must integrate various wireless technologies – QoS provisioning is very difficult due to link capacity and delays – Effective cross-layer design	– Navigating and controlling mobile nodes – Must self-organized – Localization with mobility – Minimize energy cost – Maintaining network connectivity – In-network data processing – Data distribution – Mobility management – Minimize energy usage in locomotion – Maintain adequate sensing coverage
Applications	– Environmental sensing and monitoring – Industrial monitoring – Surface explorations	– Agriculture monitoring – Landscape management – Underground structural monitoring – Underground environment monitoring of soil, water or mineral – Military border monitoring	– Pollution monitoring – Undersea surveillance and exploration – Disaster prevention monitoring – Seismic monitoring – Equipment monitoring – Underwater robotics	– Enhancement to existing WSN applications such as tracking and monitoring	– Environmental monitoring – Habitat monitoring – Military surveillance – Target tracking – Underwater monitoring – Search and rescue

Figure 1: Five Types of WSNs [43]

In order to confirm that those areas of applications are not just a theory, but researchers actually are experimenting with sensor networks in many fields, some papers will addressed in more detail:
In [7] for example, the authors are investigating how ZigBee is used in industrial control and building automation. ZigBee is a standard for wireless sensor networks, providing a Network Layer, an Application Support Sublayer, an Application Framework, a ZigBee Device object, and security. The paper gives five examples where WSNs with ZigBee is already used or soon to be launched in industrial control and building automation.
The authors of [28] describe the development and prototype testing of a WSN personal health monitoring system called Medical MoteCare. With wireless body sensors attached

to the patient, MoteCare is monitoring vital signs such as temperature, heart rate or blood oxygen level. Network management tools and models are used to store and correlate the collected data and warn if thresholds are breached.
Another example is [23], which is focusing on a large scale wireless sensor network (called ACme) for high fidelity, building-wide, electricity monitoring within buildings. The authors of [23] test ACme by deploying it in their computer science department and running it. The conclusion is that even though there is still flaws in the prototype (e.g., overheating of ACme nodes), 'this paper shows that wireless sensor network technology has a great deal to offer [to] energy monitoring, management, and control applications.' [23]
Similarly, in [15] it is proposed to use WSNs to enhance generation, delivery, utilization, and various other aspects of today's electric power systems that supply buildings with electricity. This would make WSNs a vital component of the next-generation electric power system called the smart grid. The authors also tested the feasibility of wireless senors in real-world power delivery and distribution systems by deploying a WSN to an actual power system in field tests.
Another field in which researchers are using WSN is disaster management and prevention. [41] for example proposes a quality-driven approach to achieve real-time, stationary, long-living volcanic earthquake detection using wireless sensors. They have a big advantage over traditional volcanic instrumentation (e.g., broadband seismometers), which are expensive, power-hungry, bulky and difficult to install.
Those are obviously only a few of many possible examples on WSNs in different scenarios. For a more complete recital of WSN related research this thesis refers to [2] and [43].

Of course there are also many challenges when working with WSNs. Those range from general difficulties for all WSN, such as processing and power limitations of sensor nodes, security or compatibility of different hardware pieces, to specific difficulties within different types of WSNs, as shown in Figure 1. A framework that handles some of those difficulties, especially managing heterogeneous networks, is CoMaDa. Since the work of this thesis is based on the CoMaDa framework, it will be outlined in the next Section.

2.2 CoMaDa Framework

In this Section, the CoMaDa framework is illustrated in detail, following its description in [37]. This is the framework that is used and extended in this thesis to create the mobile app to monitor active wireless sensor networks under authorized access rules, called WebMaDa.

According to the authors, there is the problem that 'Many individual solutions, specializing in specific hardware/platforms or designed for specific tasks (e.g., simulation, visualization), exist and support the user in developing applications for specific platforms. Since the user should not be forced to switch to a different environment whenever he decides to use different hardware/platforms, the authors decided to develop a unifying framework that allows to easily adapt applications to new environments, including different hardware, node platforms and protocols, allowing the user to develop applications regardless of the underlying network architecture.'[37]

Also, CoDaMa's applications are operating in a centralized way, meaning they run on devices outside the WSN with much more resources at their disposal than nodes. Thanks to this centralization, logic can be applied that needs an overview of the whole WSN, which is hardly possible in a decentralized manner. CoDaMa is an adaptive graphical user interface (GUI), which includes management tools and can easily be adapted to different platforms, operating systems, and algorithms.

[37] describes a WSN as 'a set of constrained sensor nodes, which are capable of observing their environment (e.g., measuring the room temperature) and communicating with each other [...]. In modern days, sensor nodes are not only restricted to observing, but they are also capable of interacting with their environment (e.g., controlling a light switch).' The communication within the set of sensors and between the WSN, CoMaDa, and applications is illustrated in Figure 2. It can be seen how the physical WSN is connected to the machine running CoMaDa via the sink. On the other side, applications connect to and extend the CoMaDa framework, such as WebMaDa.

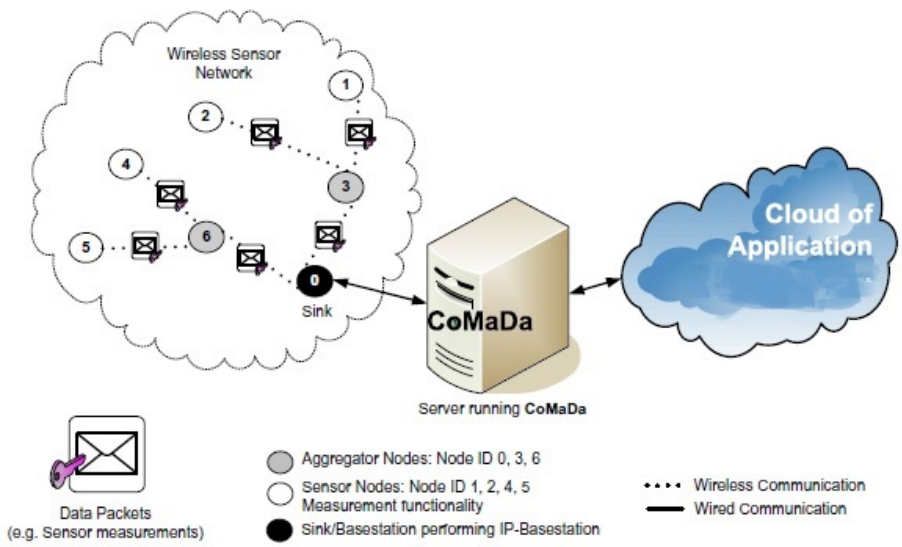


Figure 2: Communication Between Sensors, CoMaDa, and Applications [37]

In order for CoMaDa to be a framework, which offers an interface to WSNs that is independent of the underlying network hardware and software, the authors have implemented a layer of abstraction in form of a virtual representation of the network. This virtual representation is written in Java and defines every measurable aspect of the WSN that is connected to the framework. That is why it consists of the classes Topology, WSN, NodeCollection, Node, SensorDatum, Event, and NodeAction. An UML graph of the virtual WSN representation classes used for CoMaDa can be found in Figure 3.

The WSN class contains the topology of the network (represented in the topology class), that is the link between nodes, and a collection of nodes. The NodeCollection class is used to store the nodes instead of a common Java list object, so that a function to process different types of filtering can be provided. The node class represents the nodes in a WSN, storing an identifier, and some additional information. It also contains a set of SensorDatum classes, which define the type, the value, and the unit of the measured

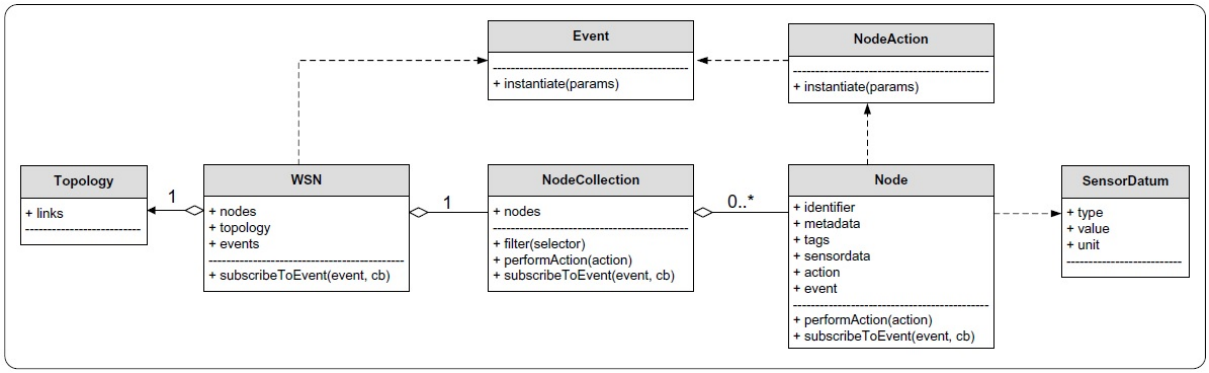


Figure 3: UML Diagram of CoMaDa's Virtual Representation [37]

data. The Node and the WSN class also each define a set of events, represented in the Event class, to which extensions to the framework (called Modules) can subscribe to get notification when certain events are triggered (e.g., measurement updated, low battery, topology has changed). Lastly, nodes contain a set of action classes, which represent actions that can be performed on the node (e.g., switching off the light). Events and actions can also be seen in Figure 4, where it is shown how they connect the virtual representation with applications. With those classes, the framework is able to create a virtual representation of the WSN including all nodes, their links, all measured data, and communicate with other parts through events and actions.[37]

As mentioned before, this virtual representation is included into CoMaDa as a layer of abstraction, between the part that defines the desired logic and the part that interacts with the underlying WSN. It allows for an user to design applications that work with any WSN by relying on the abstraction layer. Figure 4 gives an overview of that general structure of the CoDaMa framework. It can be seen more clearly how the actions and events that were shown in the virtual representation of Figure 3 provide an interface for applications.

As can be seen, there are three components connecting the virtual representation with the real WSN: WSNdriver, Protocol stack, and WSNconnector. The WSNdriver provides access to the basic network functionalities, managing the communication between the network and the framework. Its received data is forwarded to the protocol stack, which is responsible for the different communication protocols that are used. When the protocol stack translated the data, it is passed on to the WSNconnector, which creates and updates the virtual representation. [37]

However, it is not possible to visualize and view anything about the WSN outside of the computer where CoMaDa runs on, because the virtual representation that it builds is only available locally, as indicated in Figure 2. Thus, mobility support is not possible with CoMaDa. The next Section addresses research that has experimented with incorporating mobility in various form, like smart phones, into WSNs.

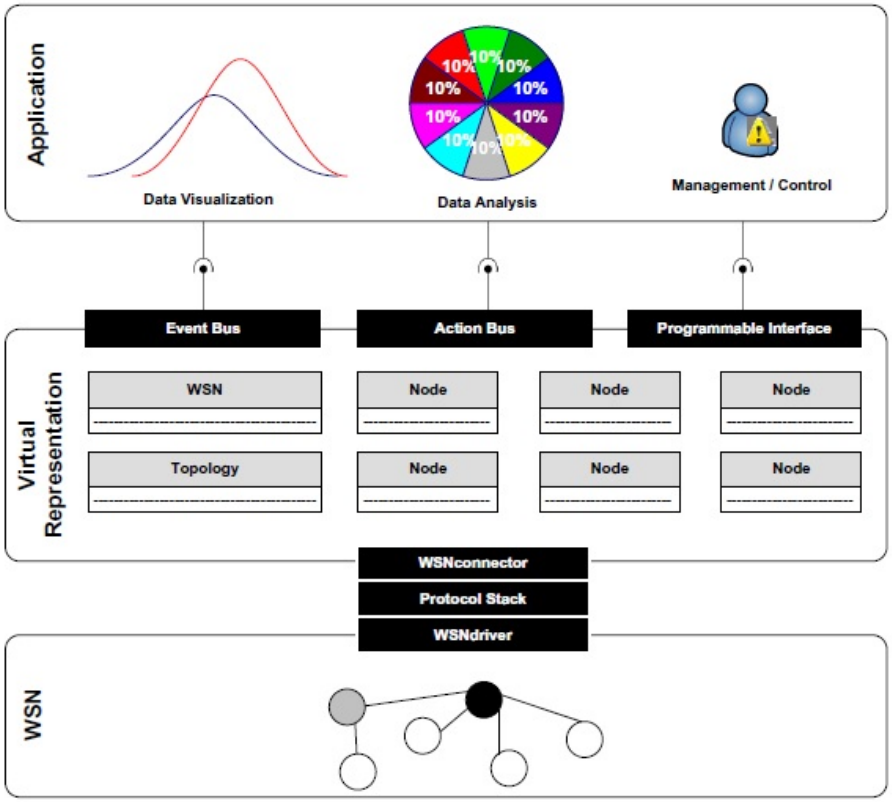


Figure 4: Structure of CoMaDa [37]

2.3 Wireless Sensor Networks and Mobile Devices

Mobility has been introduced into the research on WSNs in different shapes already. As mentioned in Section 2.1, [43] reports of a type of sensor network called mobile WSNs. They consist of multiple sensor nodes that have the ability to move on their own, and can interact with the physical environment they were deployed in. Like normal sensors, these nodes still have the ability to sense, compute and communicate data, but can additionally reposition and organize themselves within the network. Therefore, a mobile WSN could start with an initial deployment, from where nodes spread out to gather information. This data is sent to other nodes that are in range of the transmitter to be routed through the network. This means, that those kinds of mobile WSNs cannot use fixed routing or flooding to transport data, but need to have a dynamic routing mechanism. Figure 1 gives an overview of application areas and challenges of such mobile devices.

Another approach to introduce mobility into WSNs is using a mobile sink. The sink is a node in the network, where sensors route their measured data for post processing [17]. [17], [5] or [31] have suggested to create mWSNs, meaning that the sink moves within the network. This is motivated by a fundamental problem arising in WSNs with static topology: The energy consumption among sensor nodes is not uniform. The reason for that is sensors forwarding their data to other sensor in range that are closer to a sink, who then forward it further until the sink can be reached. Therefore, nodes close to the sink do more communication and use up their batteries faster. This is problematic as those

nodes close to a sink are important for routing data to its destination, so if too many go offline, a whole Section of the WSN could get disconnected. The papers on mWSN are proposing mobile sinks to solve this issue. The idea is that if sinks move within the network, then nodes could take turns with being close to the sink, which would balance the load and energy consumption would be more uniform among nodes.

Those fields of research do include mobile devices in WSNs, though not the ones typically owned by ordinary people, who would want to check their sensor information, such as smart phones or tablets. And if they are included, it is mostly phones that are used as mobile sink device ([5], [31]) but not for providing a graphic interface enabling users to read measurements or see network topology and statistics.
[44], however, goes one step further. It also presents a solution enabling smartphone-based mobile sinks for WSNs, called uSink. But the paper also introduces a middleware on mobile phones called uSinkWare, that provides functionalities such as sensor detection, topology monitoring, routing, sensor data collection, and sensor control. The advantage of using a mobile phone, which is also a sink to monitor the WSN, is that its mobility helps with load balancing as described in the mWSN literature mentioned above. However, there are also certain drawbacks with this approach: Not all mobile phones can be used with uSink, as it requires a slot for a SD card, it cannot run on iPhones OSX operating system, and it needs to be a high-end smartphone to have enough performance. But even new phones have limited computational power in comparison to a PC, which means that some algorithms might not run smoothly. As [44] puts it: 'In a result, low complexity algorithm should be explored in the implementation of phone based mobile sink'. Also, the mobile phone needs to be inside the actual network in order to read and display measurements or network status, as it is the sink.

This Section shows that there is quite a bit of research that concerns itself with mobility in WSNs. Mainly, this research focuses on moving parts within the actual sensor network, such as moving sensors or mobile sinks. Apart from [44], no relevant publications on frameworks or other tools is trying to visualize WSNs and their measurements on common mobile devices such as smartphones and tablets. And the tool design used by [44] limits mobile devices that can be used and also limits the mobility to movement within the WSN's range. There is currently no project like WebMaDa, which enables users to visualize and monitor their WSNs on all common, mobile devices independent of their location.

2.4 Web-based Network Management

The approach taken by WebMaDa is to visualize and access data of a WSN trough the Internet, which has already be done for other kinds of networks (e.g., networks of office computers). This practice is called web based network management, and there is some research on the topic (e.g., [42], [24]). Those contributions try to give an administrator the ability to configure and monitor his network over the Internet by using a web-browser, just like WebMaDa for WSNs. The authors of [20] were inspired by those contribution on web based network management and applied it to wireless sensor networks, which makes it relevant for this thesis.

In [20] the authors state: 'In general, it is widely accepted that web-based management should be used for managing and querying sensor networks'. Their reasoning is, that a web-based network management scheme has advantages thanks to the flexibility of the Internet, the facilitated development of web-pages and the easy access of it. For this purpose, they have designed and implemented a sensor gateway that is placed between a WSN and the Internet to connect those two with each other. However, the design of this sensor gateway does not just enable the connection of a sensor network to the web. It is also designed to achieve the following requirements: Sensor data aggregation, flexible query management between web and sensor network, and efficient access by the Internet. Queries in this case are requests send into the network with an ID, a question (e.g., what is the temperature), an area (e.g., in one room, on one floor), and the type of sensors that should answer, which then returns with the answer. All queries are disseminated into the WSN via a sink, and all returning responses from the network are aggregated there. The sensor gateway is, therefore, connected to the sink to forward the WSN data to the Internet, as modeled in Figure 5:

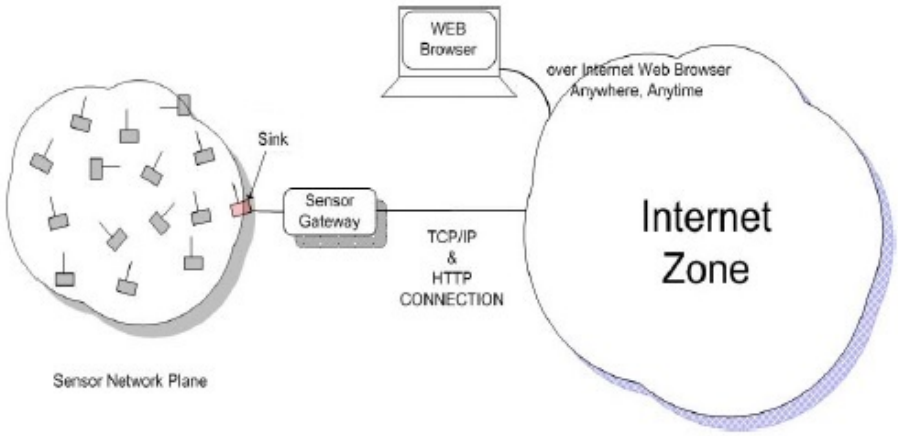


Figure 5: Overview of WSN set-up using a Sensor Gateway [20]

The gateway delivers queries that arrive at the sink from within the WSN to the web-page, and forwards queries it receives from the Internet to the sink to be disseminated into the sensor network. Also, the gateway employs a web-server internally, which means that the administrator can connect to it directly using its IP address or domain name to get to the management web-pages. Figure 6 shows in detail how the proposed sensor gateway is build.

The authors have defined a layered architecture in the sensor gateway, for efficient connection of the WSN with the web as well as for processing and transmitting sensor queries. There are basically two parts in the gateway; the one modeled on the right of Figure 6 is the Internet access part with a TCP/IP stack and a small web-server, the one on the left is the WSN access and management part with 3 layers. The Data Analysis Process (DAP) is responsible for authenticating a user, processing, and forwarding queries received from the web-browser and showing the replies from the WSN on the web page. The Sensor Query Management Layer (SQML) is transforming web-queries into properly formed sensor queries and vise verso. Finally, the Sensor Data Aggregation Layer (SDAL) is

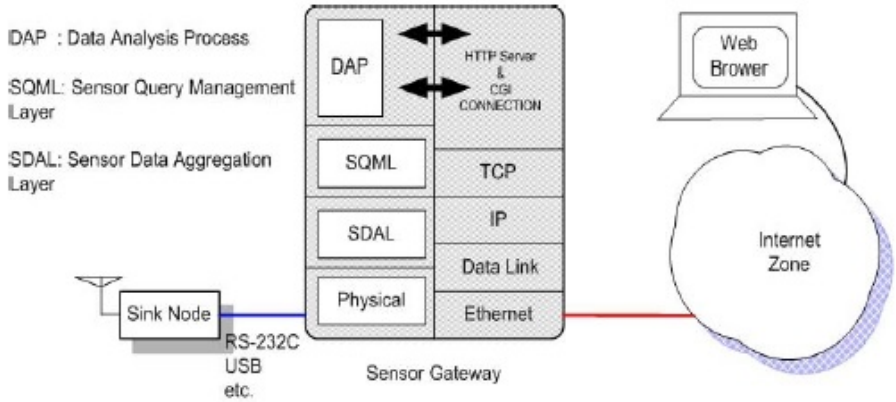


Figure 6: Detailed architecture of a Sensor Gateway [20]

responsible for disseminating the query that was transformed in SQML all over the WSN, and for aggregating the replies from the nodes. [20]

While the proposed sensor gateway of [20] does enable viewing of WSNs on all common, mobile devices, which can be used independent of their proximity to the actual network, it is limiting users. They are required to buy hardware in form of the Sensor Gateway and need to have an Internet connection with static IP to access it. Furthermore, the project does not help with programming nodes and also does not support heterogeneous WSNs like WebMaDa does thanks to the underlying CoMaDa.

2.5 Security and Authentication

A big issue with WSNs is the need for secure communication. Data that is measured and communicated by network nodes may contain private information about the user. For example with WSNs used in building control, the state of light switches or sound measurements in residences could reveal when somebody was home and in which rooms that person stayed. Or when using a WSN to monitor a patients vital signs, which could indicate what health issues the patient is suffering from. The problem is that, because of the nodes wireless way of communication, everyone is able to get a receiver and intercept messages. Also it is easy for users to send fake messages to sensor nodes. That is why each message needs to be signed to proof it originated from a trusted node and encrypted to hide its content from unauthorized parties. A further challenge is the limited resources and computing power of sensor nodes, which is limiting the computational complexity of signing and encryption mechanisms. That is why much research has been conducted on securing communication between nodes in a WSN, which generated many solutions for securing WSNs, such as SPINS [35], TinySec [25], Sizzle [16], Tiny 3-TLS [8], MiniSec [30], or TinyDTLS [26].

Security within the actual sensor network will not be discussed any further, as it does not really relate to this work. Securing the inter-sensor communication is done by the CoMaDa framework when programming nodes with selected code supporting security

features. However, the web extensions to CoMaDa proposed in this thesis will create more attack points for intruders. As the graphical representation of the WSN is copied to a website, literature on secure Internet access and web security becomes relevant. The author of [40] details problems and solutions of Internet security. In sum, it can be seen that there are three main entry points for attackers, which need to be secured. Those are the following:

1. User authentication through a log in mechanism needs to be present to enables different levels of access. This requires saving username / password combinations in the database, which should be done by hashing the password first to ensure nobody can get them even if the database is compromised.

2. Traffic between the web-server and clients, who access it (in this case the browser of end-users as well as frameworks uploading WSN data), needs to be secured. The standardized way to do so is using SSL/TLS so encrypt the HTTP messages which are send fourth.

3. Accessing and querying the database needs to be secure. One way to do that is for websites (and in this case also frameworks) to access the MySQL host via a SSH-Host that is encrypting th communication.

When designing and implementing WebMaDa, those security risks have to be considered and appropriate countermeasures as described in [40] have to be implemented.

Chapter 3

Design Decisions

The design of a system to visualize WSNs on mobile devices calls for decisions in three general fields. One is which mobile devices, or rather what mobile operating system(s), should be supported. Then, of course, it needs to be established how the system could be build, what the advantages and disadvantages are, and which should be chosen. Last but not least, authentication and security mechanisms need to be designed, which in this case also lead to the designing of an user account generation mechanism. This Chapter will reveal the decisions made in those areas that lead to the creation of WebMaDa.

3.1 Mobile Operating System Choice

The first decision that had to be made when planning on extending the CoDaMa framework for mobile devices is which Operating System (OS) should be focused on. It has been decided that the development will be construed for the Android platform. According to the International Data Corporation (IDC), it is the by far most used OS on mobile devices with almost 80% of the shipped devices in 2013 running Android, and 85% in 2014 [21]. Also, it is the OS of the authors personal mobile phones and, therefore, more accessible than the alternatives. On top of that, it would require a Mac to develop on the second biggest mobile operating system, iOS, for using the App Store and cost 99$ a year [3].

But as it will become clear in the next Section, the best approach to make the WSN framework accessible on mobile devices is one that should work independent of its OS. Never the less, it will mainly be tested on Android based devices, with only one or two tries on an OSX based device (the IPad).

3.2 Design of the App Architecture

This Section contemplates and discusses different approaches that describe how WebMaDa could have been designed and created. By analyzing the advantages and disadvantages of

different possible methods, a very well suited way of implementing a WSN representation for mobile use will be concluded and described.

3.2.1 Re-Implementation with Android Software Development Kit

The probably most obvious and straight forward approach to make a WSN accessible on an Android based mobile device would be to simply re-implement CoMaDa's functionality in an App. The Android Software Development Kit (SDK) would be well-suited to do so. It uses Java Classes for the back-end, just like CoMaDa, and XML to create the User Interface showing the back-end. Therefore, the exact same classes and methods could be used to represent the given WSN with its Nodes, Events, and Actions. Like CoMaDa, the App would be a framework that offers an abstract interface to WSNs that can be shared between different types of networks and is independent of the underlying hardware or software. The only thing that changes would be the front-end: Instead of generating HTML-files and streaming them over the local host, it would make more sense to use XML to present and interact with the Java representation of the WSN.

The biggest advantage of this approach would be the simplicity, as the whole back-end is already programmed in Java and could mostly be copy-paste into the mobile App SDK. Also beneficial would be that the User-Interface would have the native Android App look and feel, and it would not change the way users interact with the original CoMaDa.

However there are some tremendous disadvantages to that approach as well. As Hunt and Thomas state in [18]: 'We feel that the only way to develop software reliably, and to make our developments easier to understand and maintain, is to follow what we call the DRY principle: Every piece of knowledge must have a single, unambiguous, authoritative representation within a system'. Copy-Pasting code is, therefore, against that established principle of Don't Repeat Yourself (DRY), since it requires to make changes to the Framework on both places whenever the code is serviced or extended. This problem is especially relevant in this case, since the framework is still developed further and there are many new features to be implemented. So it would be a huge drawback to re-implement everything that already exists in CoMaDa. This would get even worse if the framework should also be implemented for different mobile device OS, which would also be written in a different SDK and would need to be adjusted separately every time there is an update. Another downside of reimplementing everything in the App is, that mobile devices would need to download all information on the WSN to build the Java class representation. This would lead to much data traffic and, therefore, to high battery drain which, depending on the users mobile plan, might even become costly.

Because of those disadvantages the decision to look for an alternative way to create a framework for accessing and managing a WSN on mobile devices than reprogramming CoMaDa with a mobile SDK was made.

3.2.2 Creating Server-based Framework

Another approach to access WSNs on mobile devices, that does not have the before mentioned disadvantages, would be using a server. Instead of running the back-end that creates the WSN representation on the phone, it could be done on a server, which then makes the user interface (UI) available for mobile devices over the Internet. This would also be somewhat easy to realize with the existing framework, since the UI is consisting of HTML-Files that the back-end generates. So instead of CoMaDa running on local computers, it would be put on a server where it is constantly running, creating a Java representation of the WSN and showing it using HTML-files that are accessible over its Internet-domain. An added authentication mechanism would be necessary ensure that only authorized personal can view and interact with the sensor network.

This approach would eliminate all disadvantages that a mobile App has. The framework would not be rewritten in a different SDK, but just moved out of the local CoMaDa environment onto a server. Therefore, changes do not have to be written twice. When a new version of the framework is being developed, it would be put on the server exactly the same as it is being issued for computer use. Also the data transfer for mobile devices would strongly be reduced, as they do not need to collect all information on the WSN but only load the provided UI, reducing the amount of battery drained.

Unfortunately there are some problems with this approach as well: First of all it would require ownership of an own web-server before someone can use the tool if it is designed like this. Owning a server is not very common though. Most people that require a server for something (e.g., host a website) rent space on one that is provided by an Internet Service Provider (ISP). Then, the user would also be required to configure his server to be reachable from the outside, which is an additional effort. Finally, the server would need to be physically connected to the sink node of the WSN that should be displayed through the Internet. That means the server needs to be moved whenever the WSN should be placed somewhere else.

All those factors would limit potential users, as only a fiew people have the knowledge and resources to set up their own web-server at the place they want to deploy a WSN. Therefore, this approach would also not be optimal for accessing and managing a WSN on mobile devices.

3.2.3 The Hybrid Approach

As can be seen from the disadvantages of the previous approaches, the following challenges or problems arise: If every user generates their virtual representation of the WSN on their mobile devices, it would lead to the problem that there is a separate implementation for each mobile OS that needs to be adjusted when new versions are released. Also it would require a lot of resources from the mobile device. But if there is a central server where the virtual representation for mobile devices is made, each user would need to go through the trouble of setting up and running their own server. Therefore, the framework will be not implemented using either approach, but using a hybrid solution that removes both problems:

The idea of the hybrid approach is that every user is generating its own virtual representation of the WSN on a commonly owned machine, but not on the mobile device itself. This virtual representation should then be transported on the users mobile device through the Internet. To do so, the user will start the framework that is still running on his personal computer and represent the WSN on it. But instead of visualizing the WSN on the loacal machine only, it will also be placed on a web-server that can be reached over a domain. Then, the user can access the UI with any mobile device, given it has a web-browser installed on it. An authentication mechanism will make sure that only authorized users can look at the different WSN representations on this website, and that only the user who created one can make changes to it. For clarification, this approach has been visualized in Figure 7.

This way, there is no code duplicity, because the framework is still only running on the local machine, just with the possibility to view it from the outside. This means that future updates and extensions need only be programmed once since it is not repeated with different SDK. Furthermore, most of the calculation and data traffic will be done on a computer which will safe resources and battery of the mobile device. At the same time the mobile version will be available for all devices independent of their OS. And it can all be done with hardware that most people own and know how to use.

This approach, therefore, eliminates all the disadvantages of the previous ones, and is hence chosen for making WSN accessible and manageable on mobile devices. Because it is based on CoMaDa with the additional functionality of distributing the WSN data through the web, it is called WebMaDa as in Web-based Mobile Access and Data Handling Framework. There is only one small drawback using this method of implementation: Since the UI is displayed as a website, there is no native android look at feel to it. This could be solved easily though by programming an Android App that is connecting to the UI's domain and displaying its content (like for example the youtube-app for youtube videos). However, it has been decided that this is not a necessity and is, therefore, left for future work.

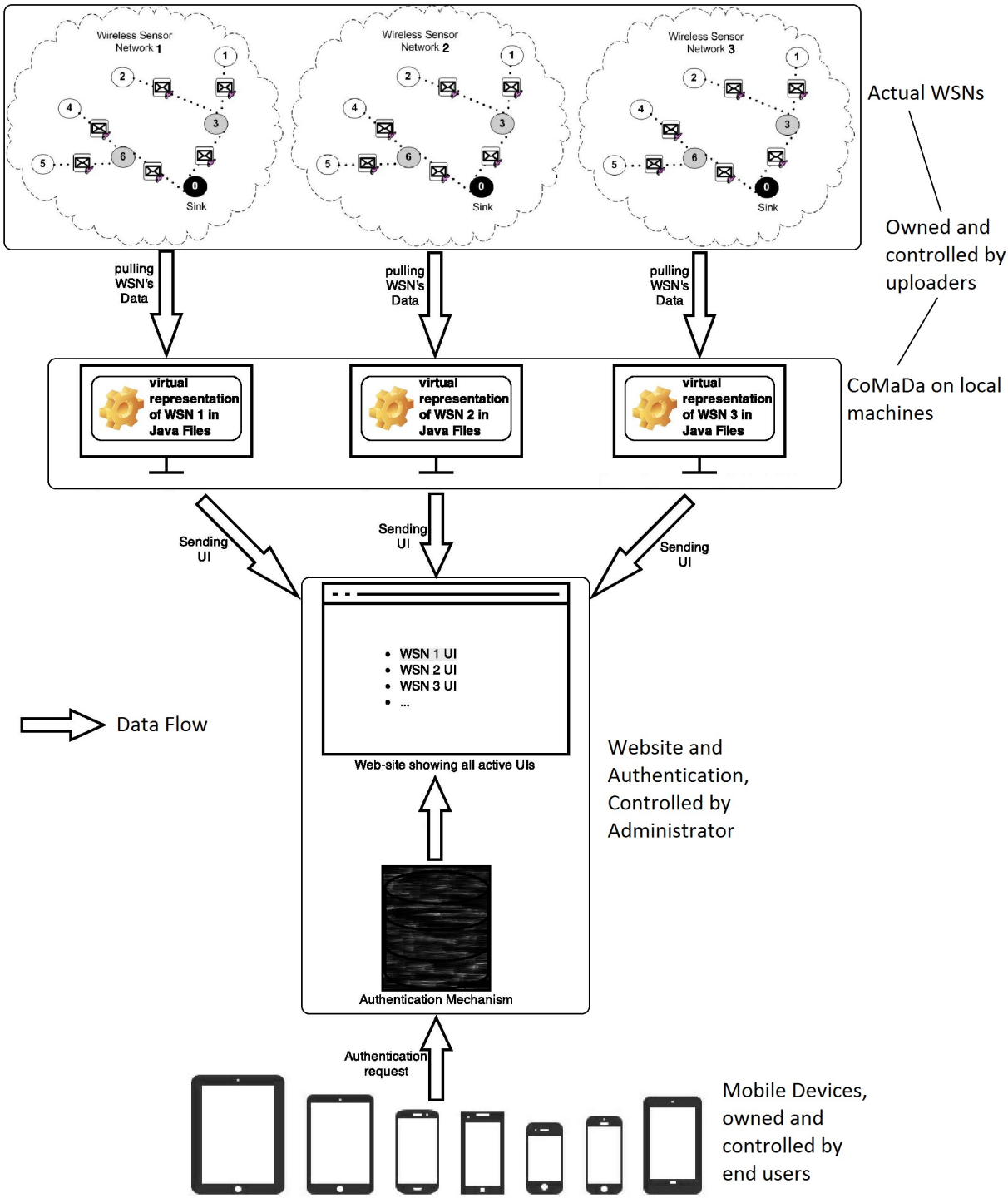


Figure 7: Preliminary Design of WebMaDa, with Authentication as a Black Box

3.3 Security and the Authentication Mechanism

When it comes to security and authentication (both with uploading WSNs and viewing them) the principles described in [40] are followed, which have been outlined in the previous Chapter, in Section 2.5. Particularly, the website is accessed trough SSL/TLS secured HTTP (called HTTPS), which encrypts the communication and adds a signature. That way, communication packages intercepted by third parties cannot be read by them as they do not have the necessary key. Also man-in-the-middle attacks, where a third party pretends to be the server and sends wrong answers back, will not be successful as the wrongly issued packets will lack the proper signature.

Also, the database used to store the network data is created on a server within the university network. The whole network infrastructure of the university is forming a private network, which connects to Internet using a NAT (Network Address Translation). Therefore, resources inside the university network can only be reached by addressing the NAT, which works with a firewall and only forwards credible requests to the database. However, the database cannot be specified as target directly but instead database queries are send in a SSH tunnel and have to be routed to a SSH host for decryption first.

Since the previously named approaches are ensuring that communication between the users browser, the framework, the website, and the database is secure, it is now important to make sure only authorized persons can use the tool. For this, user authentication mechanisms are required. This means that WebMaDa needs to incorporate user accounts with different rights that the user authenticate with before being allowed to view pages with critical information, or upload their sensor networks. Of course, there need to be multiple status types for different kind of rights. While this is pretty straight forward when it comes to upload rights, either a status allows for it or not, there are many different combinations possible for viewing rights: A user can be allowed to see all networks, or none, or any subset of networks. Also, the service needs to be designed in a way that gives users the possibility to request access to networks they are not allowed to see, and the WSN uploader (or administrators) will have to decide whether the request should be granted or denied. Therefore, the web-server will have three tables that are required for user authentication: One that holds the user accounts with its status (administrator, uploader or viewer only), one to save the view permissions, and one that holds view permissions that were requested but not yet granted (or denied). And, as seen in Section 2.5 the password for each user account needs to be hashed before it is saved.

3.4 User Account Generation

As seen in previous Section that users must identify themselves in order for the website to know which pages he can view and which are restricted to him, the question arises how to register new accounts for users.
For security reasons, this is implemented in a separate part and not included in the website. If the website would have a Subsection where new users can be registered, it would need to be protected so that only the administrator(s) can use it. Otherwise all

implemented security measurements would be pointless, as everybody could just generate his own account and view the representation of other peoples WSNs. But if the mechanism to create a new account is only accessible for someone that is logged-in as an administrator, then it cannot be used before the first account was generated. This means, that the website would have to generate a default administrator account. Because of that, the person setting up the WebMaDa website would need to take conscious steps to guarantee the safety of the service - namely logging in with the default account, then changing its login credentials and disable the websites mechanism to generate the default account. If those steps are not taken, anyone who knows the credentials of the default account can access every WSN representation, as well as generate, alter and/or remove user accounts at will. And even if those steps are taken when the framework is set up, account generation and deletion can still be misused by anyone who manages to get his hands on an administrator account.

Those problems do not exist when the whole account registration and removal mechanism is provided in a stand-alone tool. The person, who is setting up the framework does not need to change any credentials or disable the generation of a default account to keep the service secure. Also, with a stand-alone tool it is not as bad if a administrator account is compromised, because it cannot be used to create or delete any other accounts. The tool can, therefore, be seen as a (physical) key that gives access to insert into and delete from the USER table in the database. As long as it is only held by the actual system administrator, only he or she can generate and remove user accounts, independent of who might know the log-in off 'administrator' accounts.

Figure 8 shows an updated visualization of the approach taken to implement WebMaDa, including the user account generation tool which was part of the black box of Figure 7. On the left there is the WSN, which has a sink that is connected via USB with a computer that is running CoMaDa. The computer also uses the WebMaDa module, that receives WSN data from CoMaDa and uploads it onto the DB, which is signalized with arrow 1. The user management tool is another part of the WebMaDa landscape, which is inserting an deleting user accounts into the DB, as arrow 2 shows. User devices can send requests for authentication or content to the website (arrow 3), which then looks up the request in the DB (arrows 4 and 5), and then returns the result onto the users device (arrow 6).

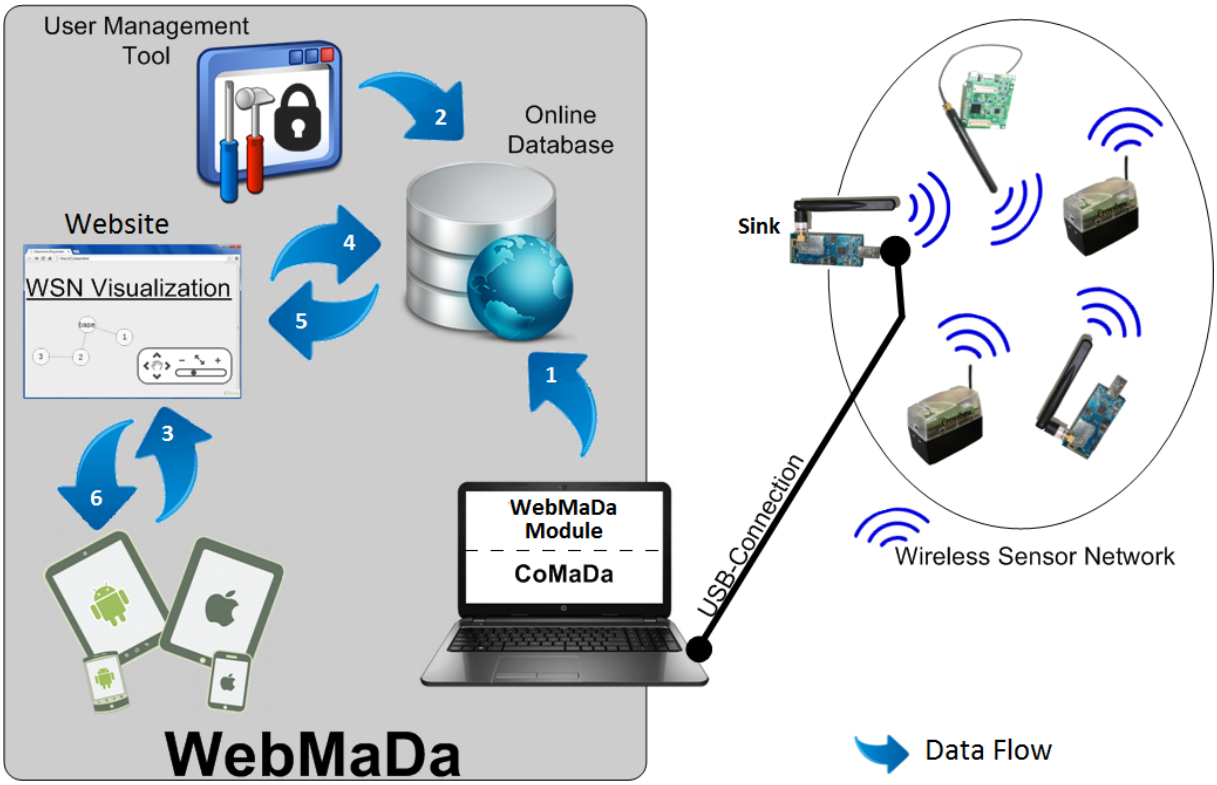


Figure 8: Design of WebMaDa, including Data Flow

Chapter 4

Implementation

In this Chapter, the implementation of WebMaDa is described. It follows the design decisions discussed throughout Chapter 3. The decisions made on how to visualize a WSN on a mobile device using the CoMaDa framework (e.g., the hybrid approach called WebMaDa) and on how to secure the tool from unauthorized access, have been implemented in a three-part architecture. Section 4.1 will introduce those three parts, highlighting what the part is used for, how it engages the made design decisions, by whom it will be used, and how it is set-up and used.
The subsequent Sections 4.2 to 4.4 will then give a more detailed examination of each part by describing how they were programmed. This includes showing some code fragments for illustration purposes, but not the whole program code, which can be found on the enclosed CD with installation guidelines in the appendix.

4.1 The 3-Part Architecture

As has been argued in Chapter 3, the best way for WebMaDa to be designed is in a way that requires three parts to be implemented, which work together in visualizing WSN data on mobile devices. The three parts that form WebMaDa are: The web-server with its database, a small application to manage authorization of users, and a module that is integrated in the existing WSN framework CoMaDa. Figure 8 shows an overview over the parts and their intercommunication as well as interactions with the external components (e.g., CoMaDa, the WSN and mobile devices).

4.1.1 Part 1: Web-Server and Database

This Section focuses on the first part of the WebMaDa architecture, which includes the web-server and database (DB). In the previously made design decisions, it has been argued and shown that the best way to visualize a WSN and its measurement on a mobile device using the CoMaDa framework, is the 'hybrid approach' (of Section 3.2.3). In that approach, the user runs CoMaDa on a local computer that is connected to the Internet and

the sink of his WSN. The sensor node information, network topology, and measurements are then send to a website where they are presented, and can be viewed with all kind of devices that can browse the web (e.g., tablets, smartphones, netbooks). This first part of the implementation is responsible for providing the WSN representation on a website. It is comprised of the following:

1. A MySQL database to hold the data about the WSN nodes, topology, and measurements,
2. some PHP files that contain PHP, HTML, and Javascript code that reads the data and visualizes it, and
3. a CSS file that defines the style of the websites relative to the screen size of the user.

This part is used by all people that get access from the administrator to view any sensor network representations on the website, by simply accessing it through a browser. The actual files containing the code however is only known to the administrator. He is also the one, who needs to set this part up by following these steps:

1. Create a new MySQL Database at any location with any name.
2. In this DB, create a total of four empty tables with the following names and structures:
   - ACTIVE_WSN, with columns USERNAME type varchar(50), and WSNLINK type varchar(30)
   - VIEW_RIGHTS, with columns USERNAME type varchar(50), and ACCESS type varchar(50)
   - REQUESTS, with columns USERNAME type varchar(50), and ACCESS type varchar(50)
   - USER, with columns USERNAME type varchar(50), PASSWORD type varchar(200), FIRSTNAME type varchar(30), LASTNAME type varchar(30), EMAIL type varchar(50), AFFILIATION type varchar(45), and STATUS type varchar(10)
3. Create a new SQL-user with a secure password that has rights to SELECT, INSERT, UPDATE and DELETE Data, as well as CREATE and DROP Structure on the DB.
4. Put the PHP and CSS files on a web-server, which can be either as a new website or in a sub-folder on an existing one.
5. Open the config.php file in a text editor and enter where the DB is hosted, the name of the DB, the name of the SQL-user that was created in step 3, and the SQL-user's password.

After this is done, the website is established and can be accessed with a browser. However, the part that visualizes WSNs cannot yet be accessed as it requires authentication and so far there is no user with the permission to access. Part 2 of the WebMaDa architecture will take care of that.

4.1.2 Part 2: User Administration Tool

The second part of the WebMaDa architecture is necessary to ensure that only selected individuals are able to view the WSN representations on the website. Since everybody is able to use a web-browser to connect to the site, a log-in is required to identify oneself as someone who was given access. The log-in mechanism compares the entered credentials (username and password) with the entries in the table USER, which was created in step 2 of setting up the DB (see Section 4.1.1). In order to make that mechanism more safe from attackers, it has been stated in the design decision that passwords are not saved themselves in the DB, but the results of a hash function of the passwords. This is the purpose of the second part: It asks for a username and a password that should be added to the list of authorized users, and then save the username and the hash of the password in the database table.

Obviously, this part is only used by the administrator and should not be given to anyone else. Before it can be run like any Java application, the config class needs to be edited by stating where the DB is hosted, the name of the DB, the name of the SQL-user that was created on the DB and its password. When running, it asks if a new user should be registered or if an existing one should be removed. First, a new user has to be added for the administrator that is setting up the whole thing. For this, all text fields nedd to be filled out and Admin has to be chosen as status. After that, more users can be added (and removed again) using this application whenever the administrator wants to grant (or deny) access to someone.

Now that a user has been created, it is possible to access the part of the website that visualizes WSNs, by logging in with the newly created user. The page should then display a message stating that currently no WSN is activated and displayed. This is going to change once the third part has been set up.

4.1.3 Part 3: WebMaDa Module

The third part of the WebMaDa architecture is needed to integrate WebMaDa as a module into the CoMaDa framework. It is responsible for sending all data of the connected WSN to the database, so that it can be visualized by the first part. Thus, it is responsible, together with part 1, to realize the 'hybrid approach' that was chosen in the design decisions (see Section 3.2.3). It consists of a few Java classes that are integrated into the pre-existing WSN framework as a module that runs in its own thread and extends the framework.

This part is used by the administrator to connect a WSN and push the representation on his website. The extended CoMaDa can be shared with other people which grants them access for visualizing their own WSN on the same website. However, by sharing this part the administrator also gives away the address, name, and SQL-user of the database that was established in the first part. That is a problem if the user might misuse this information to access the database and read or even change the data of active WSNs, or mess with the registered users (though they cannot read any password as they are hashed

and, therefore, not generate new users either, it would be possible to delete existing users). If the administrator wants to make sure that a user getting this part cannot easily access the database, code obfuscation mechanisms have to be used. Code obfuscation means that the code is re-written in a way that it still does the same thing but is as unreadable and confusing for humans as possible. That way, it is almost impossible to realize what the program is doing or what is saved in which variable, including the connection details to the DB.

In order to set-up this part for the first time, the config class needs to be provided with the following information: Where the DB is hosted, the name of the DB, the name of the user that was created on the DB, its password and the link to the website. After this is set up, the framework is run almost in the same way as before. The only difference is that a valid username and password, as created by the second part, needs to be entered with every startup. Then, the local CoMaDa UI is used to program nodes and establish the tunnel to the sink the same way as it was done before.

When all those parts are set up and a WSN connection is established, the website will show a representation of that WSNs topology and nodes (the sensors) as well as the measurements of the different nodes. Sections 4.2 to 4.4 will go into detail on how the parts are programmed in order to provide the described functionalities.

4.2 Implementation Details of Part 1

The implementation of the front-end, which is kept and accessed on the web-server, is consisting of 26 PHP files. Apart from that there are some CSS files for style and Javascript files for drawing and other effects on the website. Together, those files contain all the code that is visualizing wireless sensor networks, managing security, authentication and access management, styling the page in a way that it looks good on all screen sizes, and also providing a 'hook' for the third part (WSNFramework Module) of CoMaDa to upload data. This Section will provide a more in depth look at code and mechanisms that accomplish all of that.

4.2.1 WSN Visualization on Website

The website is conceived in such a way that it can be used by multiple people, who want to represent their WSN online at the same time. That way, the web extension of CoMaDa cannot only be used by individuals to view their own WSN on their (private) website, but also by groups of users, who all want to display their WSNs on the same site. A possible use case could be swiss skiing regions that upgrade their slopes with sensors (measuring snow amount, temperatures or the like) and all display the readings from their ski area on one central info-site. Figure 9 shows how such a list could look like. But of course the networks do not need to be related in any way, as long as they are deployed using CoMaDa, it is possible to visualize them together on a site.

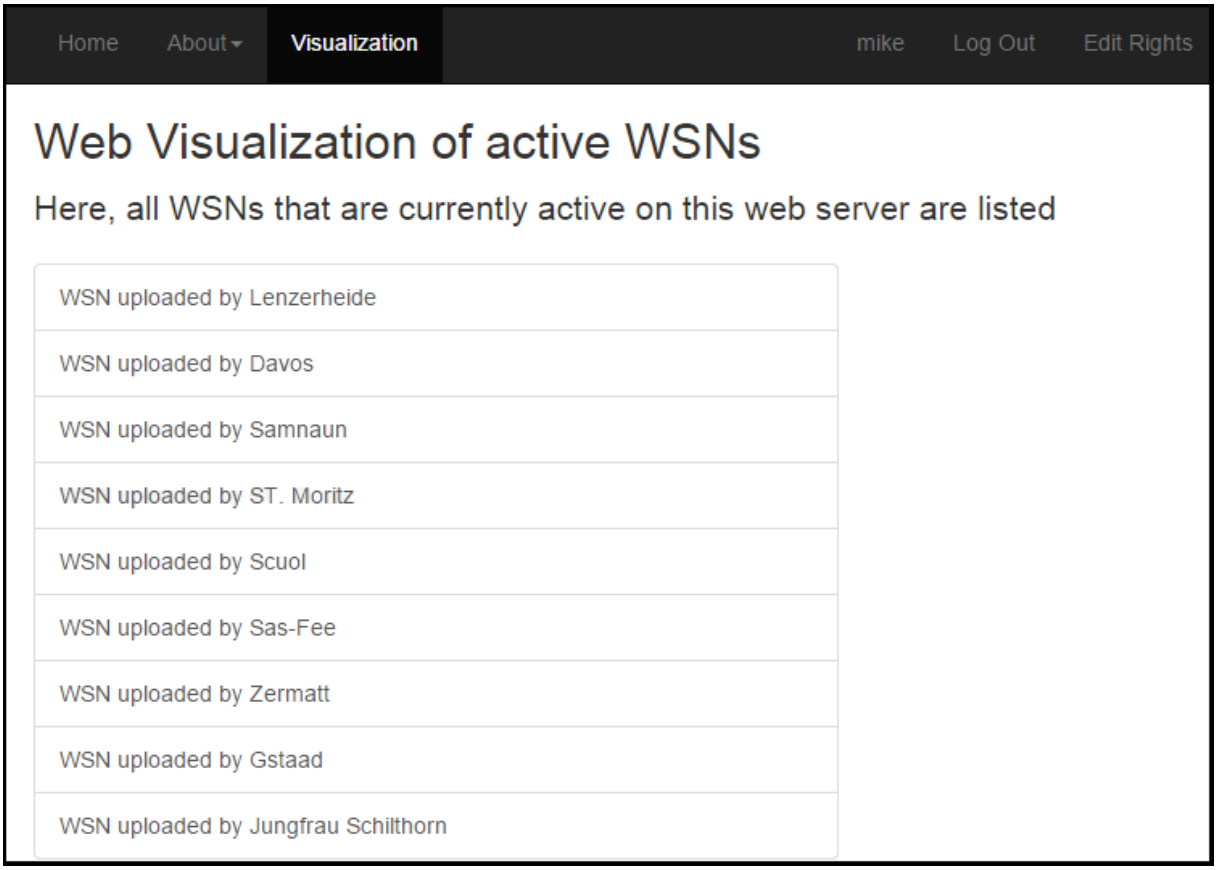


Figure 9: Exemplary look of the visualization page

It works like that: When a user connects his WSN to the site, it gets a random, 10 character string assigned, which is henceforth referred to as *[random]*. The username of the account that is used to publish the WSN and *[random]* are saved in the DB table ACTIVE_WSN. When navigating to the menu Visualization on the website, visualization.php will query all entries in that table and list them unless there are no entries in the table (in that case the user gets informed that there are currently no WSNs to be shown). Each element links to the file visualize.php with his specific *[random]* as added GET variable, which will look like this: domain.com/visualize.php?wsn=*[random]*. As shown in Figure 10, the user can choose between three ways of visualizing the selected WSN: Displaying the raw packets as they arrive in CoMaDa, showing the topology of the network with current measurements, or visualizing data in a stream over time.

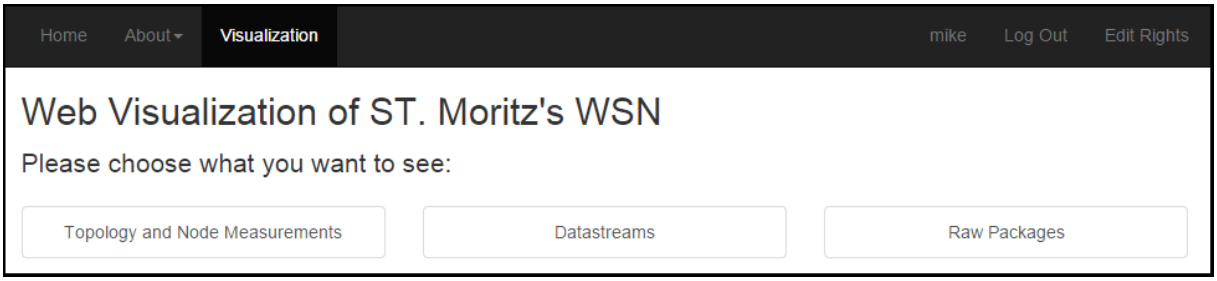
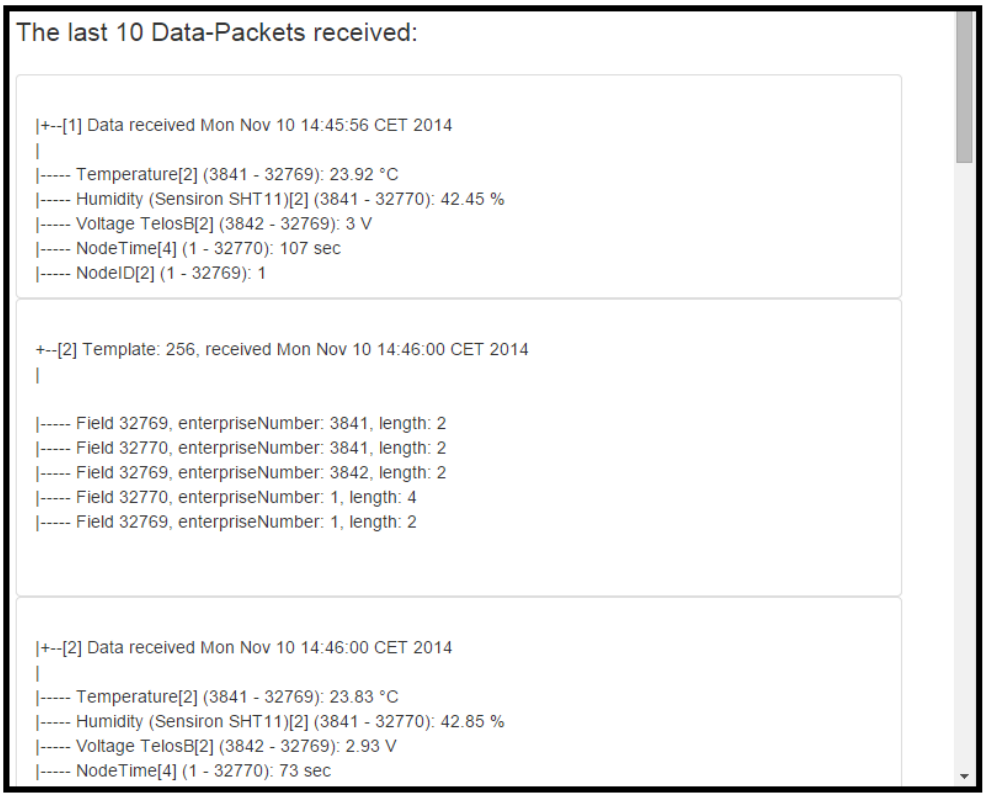


Figure 10: Menu of different things that can be displayed about a WSN

Visualizing the packets is very straight forward: For every WSN that is represented on the site, a table called *[random]*_PACKAGES has been created. Each row in this table contains one already formated packet-string in the first column and the time stamp of that packet in the second. When this visualization option is chosen, the page retrieves all entries from the table where *[random]* corresponds with the one submitted as GET variable and simply displays them, ordered by the timestamps. Figure 11 shows the PHP code on the top (a) and the visualization it leads to on the bottom (b). If the table is empty, then a message stating that no data has been received yet is shown.

```
20 try {
21     $protocolResult = $pdo->query('SELECT `TEXT` FROM `'.$link.'_PROTOCOL` ORDER BY timestamp ASC;');
22 } catch (PDOException $ex) {
23     echo 'An Error occured!';
24     echo $ex->getMessage();
25 }
...
42 <?php
43   if($protocolResult->rowCount() > 0) {
44     echo '<div class="list-group">';
45     echo '<p class="lead">The last 10 Data-Packets received:</p>';
46     while($row = $protocolResult->fetch(PDO::FETCH_ASSOC)) {
47       echo '<div class="proto-wrap">
48               <p class="list-group-item">'.$row['TEXT'].'</p>
49             </div>';
50     }
51     echo '</div>';
52   } else { //if no data yet
53       echo '<p class="alert alert-info">No data received yet<p>';
54   }
55 ?>
```

a) PHP Code for Packet Visualization



b) Packet Visualization on Website

Figure 11: The Visualization of Data Packets on the Website

When a user chooses to display the networks topology with current measurements, information from two database tables are needed. Those have also been created for each WSN specifically when it was connected to the website, and just like the table holding packets they start with the *[random]* that is associated with the WSN. One of the tables called *[random]*_TOPOLOGY holds, as the name suggests, the topology of the whole network. Each row in the table represents one edge or link in the network, which contains three attributes: The source, which is the sensor node sending information, the target which is the node that the data is sent to and the weight, which is how strong the connection is. Since each sensor in the network routs its data to exactly one other node (the sink node if in reach, otherwise a node closer to the sink) the source column of the topology table provides a complete, non-redundant list of all network nodes. The other relevant table for displaying the network topology with current measurements is called *[random]*_DATA. It holds all current measurements of all nodes that measured something. Each row in that table consists of one single piece of data with a field identifying the sensor who measured it, and one field each for stating the type of measurement (e.g., temperature, humidity) the value that has been measured, and the unit the measurement is in (e.g., degrees or percentage). Together those two tables contain a list of all sensors in the network, all measurements those sensors made and the paths that are taken to route the measurements to the sink.
When the option to visualize the network topology with measurements is chosen, the PHP code is first accessing the source column of the topology table in order to get a list of all nodes in the network, which are put into an array. Then, all edges between the nodes are retrieved (also from the topology table) and put in another array. Those are then encoded using the Javascript Object Notation (JSON) and passed on to the Javascript code. This code is using the open source project Cytoscape Web, a reusable component that allows for embedding networks within HTML via its Javascript API [6], for drawing a visual representation of the nodes and edges. The code used to achieve the described data gathering and passing is shown in Figure 12.

Additionally, the PHP code retrieves all sensor measurements from the data table and puts them in a two dimensional array. The indices of the outer array are the IDs of the nodes, and the array at each index contains the measurements that are associated with that node. That collection is also JSON encoded and passed to the Javascript code. There, a so called onclick function is waiting for the user to click on one of the nodes or edges, and then displays either the direction and wight of the edge or the measurements of a node. Figure 13 shows how this looks like on the website. The upper part visualizes the network topology. If one of the nodes or edges is clicked, the bottom part shows the node's measurements or the edge's weight.

In order to visualize a stream of data over time, a lot of data needs to be saved. For example, if a user wants to create a datastream for just one measurement of one of his sensors (e.g., temperature of node 1), which is updated only once a minute, that equals over half a million data points (1*60*24*365=525600). This dramatically increases if multiple users on the same site want to visualize datastreams, if a user wants to monitor more than one value, if the value should be updated more often than once a minute and/or if the WSN is running for longer than a year, all of which is very likely. That is why a third party services has been used to upload the data that is used to create and visualize continuous streams of data, called Xively [29]. While the backend, meaning

```php
//try to access the topology of the wsn that should be visualized (identified by $_GET)
try {
    $topology = $pdo->query("SELECT * FROM `".$link."_TOPOLOGY`");
} catch (PDOException $ex) {
    echo "An Error occured!";
    echo $ex->getMessage();
}
//create 3 array, one to store the nodes (sensors) of the network, one for the links between them
$nodeArray = array();
$linkArray = array();
$nodeArray[] = array("id" => "none", "label" => "base");
//all sensors connect to exactly one other device (sensor, agregator or base)
//-> for each sensor there is exactly 1 link in topology where the sensor is the source
//-> iterating over topology links (from db), adding one node (with id in source column) and one topology link
while($row = $topology->fetch(PDO::FETCH_ASSOC)) {
    $nodeArray[] = array("id" => (string)$row['SOURCE'], "label" => (string)$row['SOURCE']);
    $linkArray[] = array("id" => (string)$row['SOURCE']."to".(string)$row['TARGET'],
                         "source" => (string)$row['SOURCE'],
                         "target" => (string)$row['TARGET'],
                         "weight" => (string)$row['WEIGHT']);
}
//create a json object that represents the wsn by including the nodes found and links between them
$network_json = array(  //-> will retreived by javascript code for visualization
        "dataSchema" => array(
                "nodes" => array(
                            array("name" => "label", "type" => "string")
                        ),
                "edges" => array(
                        array("name" => "source", "type" => "string"),
                        array("name" => "target", "type" => "string"),
                        array("name" => "weight", "type" => "string")
                        )
                ),
        "data" => array(
                "nodes" => $nodeArray,
                "edges" => $linkArray
                )
        );
```

Step 1: Gathering Data

```php
<script type="text/javascript">
window.onload=function() {
    drawTopology(<?php echo json_encode($network_json); ?>, <?php echo json_encode($measurements); ?>);
};
</script>
```

Step 2: JSON encoding and passing to Javascript

```javascript
function drawTopology(network, measures) {
    // id of Cytoscape Web container div which will be replaced with graph
    var div_id = "cytoscapeweb";
    // get the network model object from the php code above
    var network_json = network;
    // get the node measurements array that was made in the php code above
    var measurements = measures;
    // create and configure Cytoscape Visualization Object (var vis)
    var options = {
        swfPath: "swf/CytoscapeWeb",
        flashInstallerPath: "swf/playerProductInstall"
    };
    var vis = new org.cytoscapeweb.Visualization(div_id, options);
    // callback when Cytoscape Web has finished drawing, attached to Cytoscape
    // Makes nodes clickable and displayes the measurements if clicked
    vis.ready(function() {...}); //full code on CD

    vis.draw({ network: network_json });
    vis.zoom(3);
};
```

Step 3: Drawing the Topology

Figure 12: Code for Drawing WSN Topologies

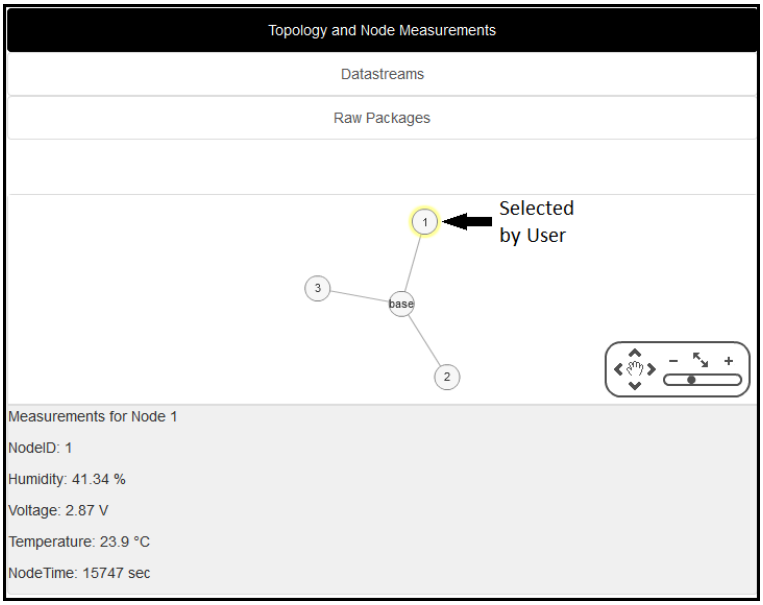


Figure 13: The Visualization of a WSN's Topology with one Sensor Selected

CoMaDa with the WebMaDa extension, takes care of supplying Xively with the data of the measurements that users want to visualize as a stream of data, the visualization site just needs to grab the datastreams and display them. Doing so is relatively easy: Xively offers the possibility to directly get all the data of one measuring unit as a .png image from their server. The URLs to the pictures of the datastreams that have been created by the owners of a WSN are also saved to a table that starts with *[random]* assigned to the WSN, followed by _DATASTREAMS. Therefore, all that the site visualizing datastreams needs to do is retrieve all rows in that table and display the image those links return. The look of that can be seen in Figure 14. However, one has to authenticate with Xively before the images, or any other information, can be retrieved. That is why each image request to the given URL is first manually extended using PHP with the Xively-API Key, which is done with the code shown in Figure 15.

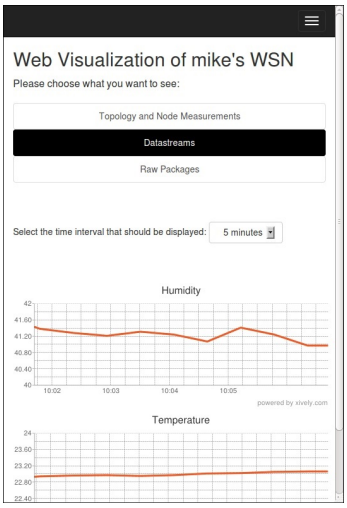


Figure 14: The Page Visualizing the Images Retrieved from Xively

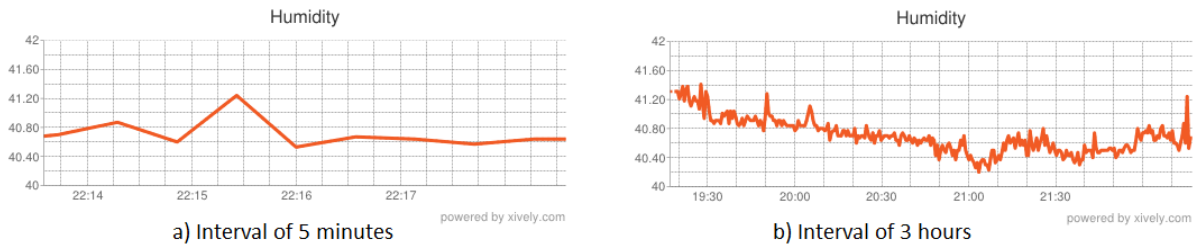
```php
<?php
  $datastreamResult = $pdo->query('SELECT * FROM `'.$link.'_DATASTREAM`');

  if($datastreamResult->rowCount() > 0) {
      $_SESSION['url'] = array();
      $i = 0;
      while($streamRow = $datastreamResult->fetch(PDO::FETCH_ASSOC)) {
      $_SESSION['url'][$i] = $streamRow['URL'].'&duration='.$interval;
      echo '<div class="img-wrap">
              <img class="img-responsive" src="imageGenerate.php?num='.$i.'" />
            </div>';

      $i = $i+1;
      }
  } else {
      echo '<p class="alert alert-info">No data received yet<p>';
  }
?>
```

imageGenerate.php:

```php
<?php
header("Content-type: image/png");
require_once 'config.php';
require 'myFunctions.php';

$url = $_SESSION['url'][$_GET['num']];

$opts = array(
        'http'=>array(
                'method'=>"GET",
                'header'=>"Accept-language: en\r\n" .
                "Cookie: foo=bar\r\n" .
                "Connection: close\r\n" .
                "X-ApiKey: hQ8a-ARXAJgyCGZqygfWTnsCRNiSAKxmanRicWQ3UklSQT0g\r\n"
        )
);
$context = stream_context_create($opts);
$file = file_get_contents($url, false, $context);

echo $file;
exit();
?>
```

Figure 15: Code Used to Access the Xively Images After Adding the Key Into the Header

The site displaying those images comes with the option of setting a different interval for the picture, meaning how far back the data is displayed on it. Xively makes it easy to accomplish that, it only requires that the link specifies an interval inside the GET (e.g., exampleurl.com?interval=10minutes). Therefore, the interval entered on the page is simply added to the link and then the pictures are loaded again which will automatically adjust the scale of the image, as can be seen in Figure 16.



a) Interval of 5 minutes

b) Interval of 3 hours

Figure 16: Scaling of Images by Xively

4.2.2 Security, Authentication, and Access Management

As mentioned earlier in Section 4.1.2, creating new user accounts and deleting them is done by a independent piece of software and not included on the website, due to security concerns. Therefore, adding and removing users will be discussed in the following Subsection, as it is not part of the website. The things done for security and access control through the web page include authentication, editing user rights, and requesting access to restricted WSNs.

Authentication is important, because deploying a network of sensors in a area has the potential to reveal private information about that area and the people in it, and it should, therefore, not be shown to anyone. However, the Internet and this website can be reached by everyone, so the page is restricting access to certain parts for users that are not logged in or do not have necessary permissions.
The different level of permission is modeled by one of five status that is assigned to each user account within the STATUS column of the USER table. The keywords in this column that are accepted by WebMaDa are the following:

1. Admin - An account that has Admin status is used by an administrator of the website, who can view all published WSNs and also authenticate with the framework to upload one or more WSNs. Additionally, accounts with this status can change the rights or status of all other registered accounts (that do not have the status Admin themselves).

2. All Rights - Accounts with this status are allowed to view all published WSNs and also upload sensor readings using the framework. Its almost the same as an Admin, but without the possibility to change other peoples status.

3. View All - This account type lets the user view all published WSNs on the website, but does not grant the right to use the framework for uploading a WSN to the website.

4. Upload - Accounts with this status are allowed to publish their WSN on the website, but are restricted in viewing the networks of others. A separate table specifies whose networks they are allowed to see. The own network is alway accessable to those users.

5. View Some - This status only allows the user to see some networks, but not upload his own. Again a separate table specifies whose networks a user with this status is allowed to see.

A user of this website shows his level of access by logging-in using the link in the navigation bar. Alternatively, it is also possible to navigate to a Section of the site that is only available to users that have logged in (e.g., visualization). That will cause a reroute to the log-in page if the user is not yet authenticated, as is shown in Figure 17. For more user convenience the page remembers where the user wanted to navigate to before being rerouted to the log-in form, and goes there if the authentication was successful and the rights associated with the account are sufficient for that part of the site.

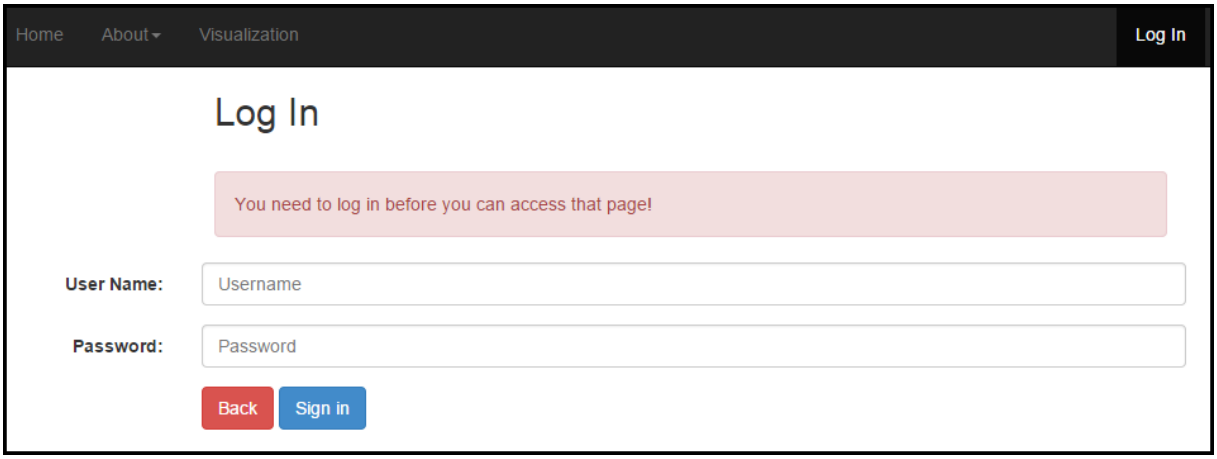


Figure 17: Log-In Page

The authentication process is coded in PHP, and compares the credentials the user has submitted in the log-in form with the USER database table. For this, the table is queried for all rows where the entry in the column USERNAME is equal to the username the user stated. Since that column is a (unique) key of the USER table, the query can return either one row as result if it was found or an empty result but not multiple rows. If the query returned something, then the next step is to compare the password that is in the query result row with the password specified by the user. For this, the user inputed password needs to be hashed first because only the hash of the password is saved in the USER table. The precise method used to do so is explained in more detail below, in the Section about the user administration tool. If the hash of the stated password equals the one retrieved from the database, the authentication was successful. In that case, a new session for the browser is created on the server using PHP. In this, the username and his status is saved. The code that does this is shown in Figure 18, but the code that does the actual evaluation of the password is omitted, as it is very long and complicated. For those interested, it can be found in the source code (file myFunctions.php) or on the website of Defuse Computer Security R&D, who owns it [36].

If a user tries to access a restricted page the server first checks if there is a session set for that browser. If that is the case the database table USER is queried to see if the username, that is saved in this session, is still present in the table. This is done to get rid of users that have been banned, meaning that had their account deleted. The reason is simple: A session stays in place for as long as the browser is not closed. Usually that means it is not long lasting (unlike cookies) but it can be cheated. Theoretically, a user could log-in as long as he has an account and then never close his browser. If the owner of the service wants to remove that user by deleting his account with the user administration tool, he does not automatically loose the session. It only means that he is not able to log-in with that account any more, as the query for his username would then return an empty result. By checking that the user in the session is still in the database every time he tries to load restricted pages, this security issues is solved. If the username in the session cannot be found in the database any more, the session is closed (which has the same effect as logging out) and the user is forwarded to the start page of the website.
In case the username found in the session still exists in the database, it is checked if he

```
if(isset($_POST["userName"]) && $_POST["userName"]!= "" && isset($_POST["userPW"]) && $_POST["userPW"]!= "") {
    $usersFound = $pdo->query("SELECT * FROM `USER` WHERE `USERNAME`='".$_POST["userName"]."';");
    //see if user with provided name exists in DB
    if($usersFound->rowCount() == 1) {
        $row  = $usersFound->fetch(PDO::FETCH_ASSOC);
        //see if hash of password provided equal the one in the DB
        if(validate_password($_POST["userPW"], $row["PASSWORD"])) {
            session_start();
            $_SESSION["User"] = $_POST["userName"];
            $_SESSION["Rights"] = $row["STATUS"];
            header("Location: ".$location);
            exit();
        } else {
            $message = "Wrong Username / Password combination";
        }
    } elseif($usersFound->rowCount() == 0) {
        $message = "Wrong Username";
    } else {
        $message = "Fatal error - User Username exists ".$usersFound->rowCount()." times.";
    }
}
```

Figure 18: Code that Compares User Entered Credentials with Database Content

also has the rights to access that part of the site. Only after this has been confirmed, a restricted page starts to load as is shown in Figure 19 for a page that only administrators can see.

```
function requireAdminUser ($pdo)
{
    if(session_id() == '') {
        session_start();
    }
    if(isset($_SESSION["User"]) && $_SESSION["User"] != ""){
        $userRow = $pdo->query("SELECT * FROM `USER` WHERE `USERNAME`='".$_SESSION["User"]."';");
        if($userRow->rowCount() > 0) {
            $user = $userRow->fetch(PDO::FETCH_ASSOC);
            if($user['STATUS'] != "Admin") {
                header("Location: index.php?force=true");
                exit();
            } else {
                return $user;
            }
        } else {
            header("Location: logOut.php?force=true");
            exit();
        }
    } else {
        header("Location: logIn.php?loc=editRights.php&force=true&require=Admin status");
        exit();
    }
}
```

Figure 19: Code that Checks if a User is Logged In Before Displaying the Page

As mentioned in the enumeration in the beginning of this Subsection, the user accounts with Admin status are able to change the view rights of other users (shown in Figure 20). This is done with the website on a page that is only linked in the menu when someone is logged-in with an account that has Admin status. This page, called edit rights, is listing all non-Admin users when opened (of course after doing the procedure of checking if a session is set, if that user is still in the database and confirming he has admin status, as described earlier in this Subsection). The user can then choose one of those users by

clicking on it which will show his current rights. The admin can then change the whole status of the user (e.g., from View Some to Upload) or which networks he can view if he has limited view rights. As mentioned before, he cannot change the status of other Admins, they can only be removed by the site owner using the user administration tool.

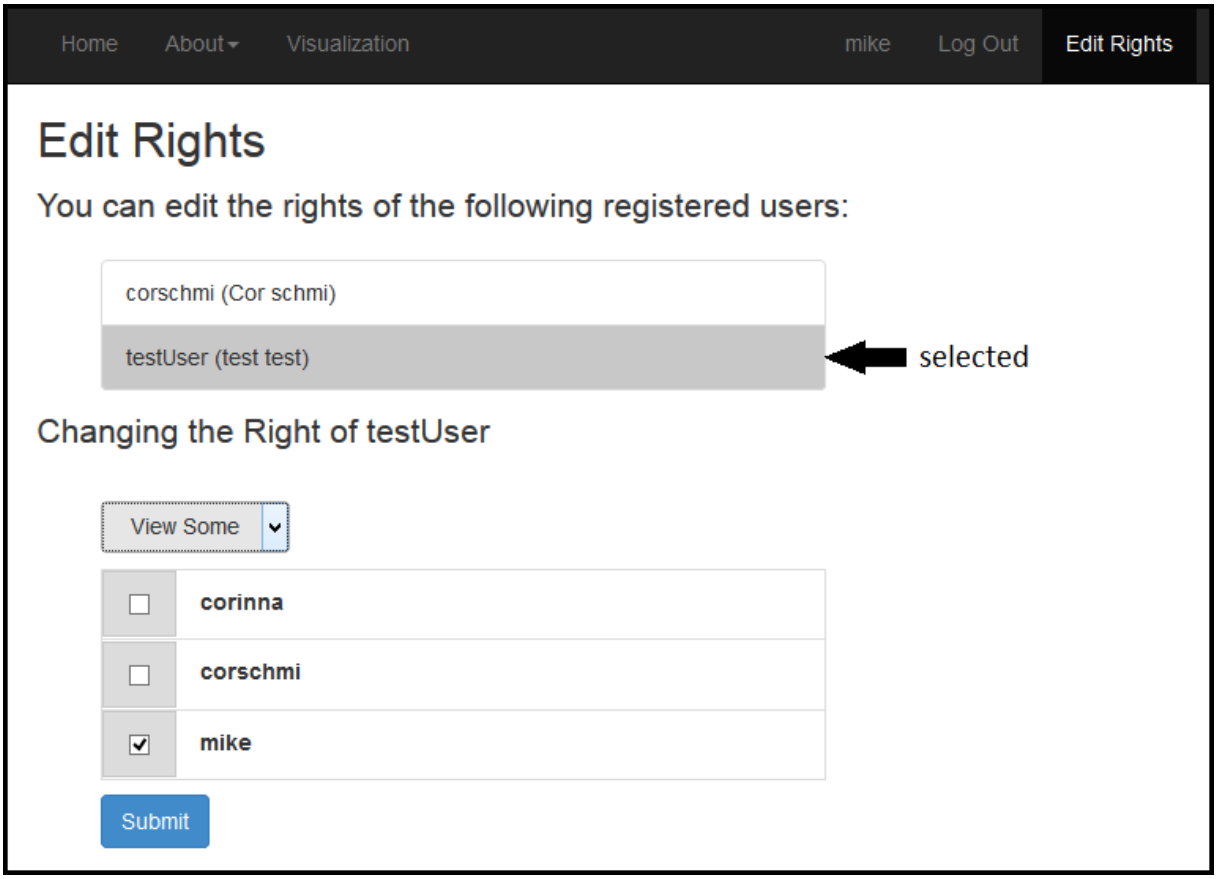


Figure 20: Edit Rights Page (only Available for Administrators)

Also, users with limited view rights can request the right to see other peoples networks. When they visit the visualization page and then click on a listed WSN that they have no access to (which indicated with a label and grey color), they are routed to the request access page instead the visualization of the network. If he chooses to go forward with his request, it is saved in the database. From that moment on all Admins will see a notification next to the navigation link that point to the edit page. On that page they will find a list of all open requests with an accept and deny button, to grant the requests or delete them from the database again. Also, the user whose network has been requested for viewing gets a notification next to the link leading to his profile page. There, he also gets the possibility to decide if the request should be granted for his networks or denied and removed. The whole process looks as illustrated in Figure 21.

4.2.3 Styling and Phone Compatibility

Apart from the actual functionality (visualizing sensor network readings) and access control, the styling of the website is a major issue. Of course it is always important to make

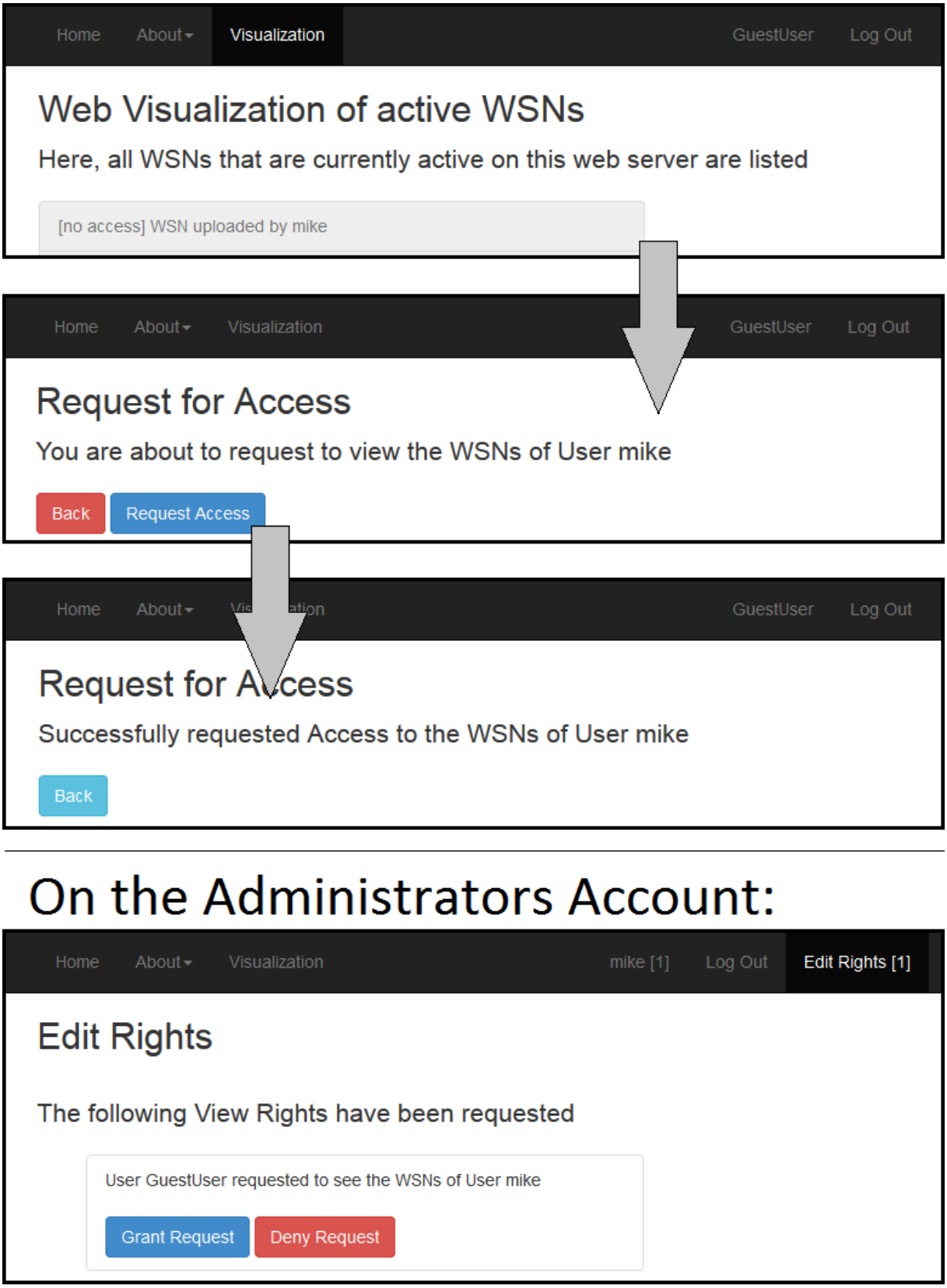


Figure 21: Process of Requesting Access to a WSN

an user interface appealing and intuitive to use, but it is a more challenging task in this situation. The reason is that the website is meant to be viewed on mobile devices, which have different screen sizes and resolutions. This means that there cannot be just one good looking style that is used, but multiple styles are required for multiple screen sizes. This concept is called responsive web design, which refers to a website with a design that responds to the screen size it is viewed with for a more satisfying user experience [10]. In order to achieve that on the introduced website an open source project by twitter called bootstrap has been used [4]. The result is that the style changes depending if the page is viewed on a smartphone, on landscape mode, on a tablet, on netbooks or on normal sized screens. This can be seen in Figures 22, 23, 24, and 25, which all show the exact same page in the same browser but with a different window size. That way it is always easy to use and looks well.

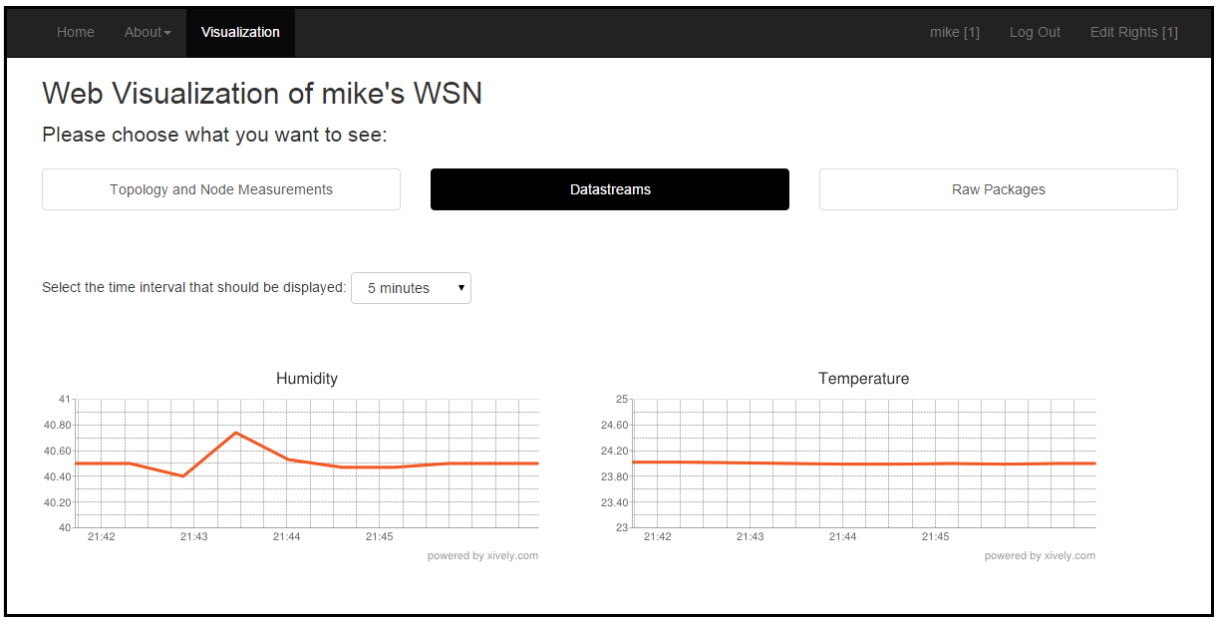


Figure 22: Datastream Visualization on a Big Screen

Another advantage of using bootstrap to help with styling is that it is used by many other websites as well. That means it has been well tested and should not contain serious bugs any more. Also the user is already familiar with the look and feel of elements on the page (e.g., navigation bar, buttons, lists), which might enhance user experience even further.

4.2.4 Hook for Data Upload

While the previous Subsection explained all important aspect of the website when interacting with it, there are still four important files on the server that have not been introduced yet: updateDatastream.php, updateNode.php, updateProtocol.php, and updateTopology.php. Those files are not part of the user interface, but work in the back. They are called by the WebMaDa module together with POST variables for uploading network information onto the database. The reason for uploading data in such a manner is presented in Section 4.4.2, when the framework module is illustrated in detail. How the

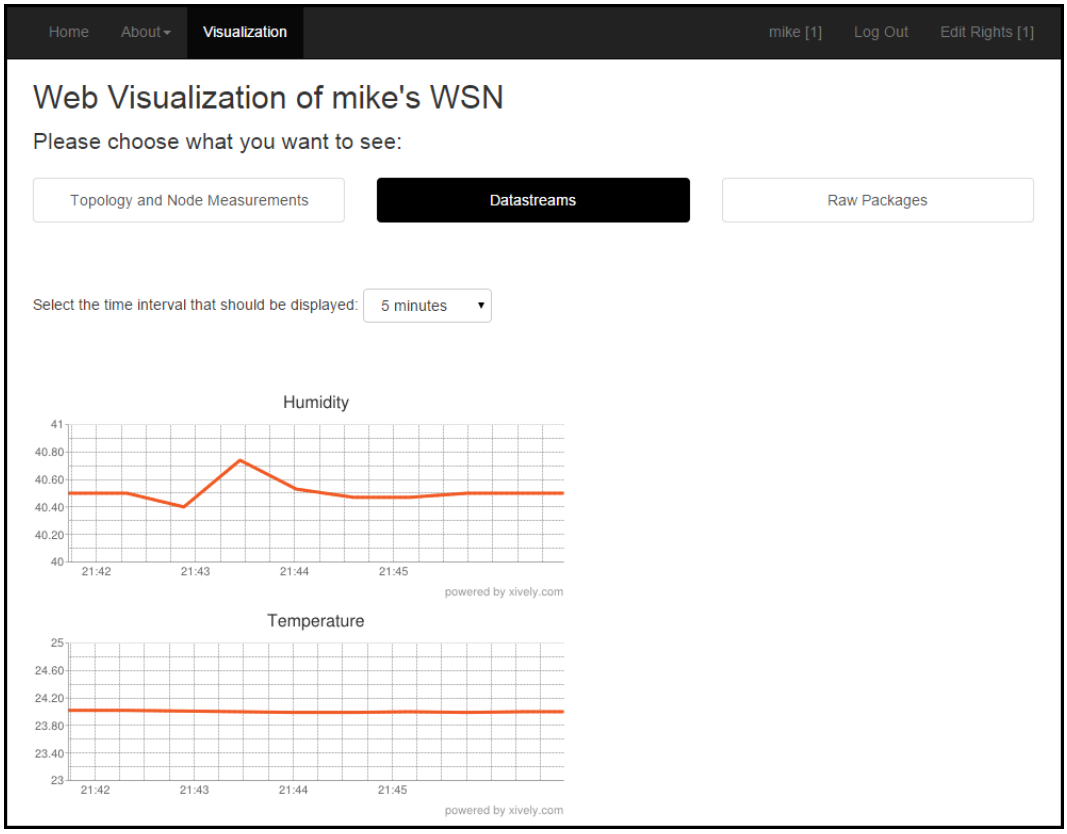


Figure 23: Datastreams on a Smaller, Netbook-Sized Screen

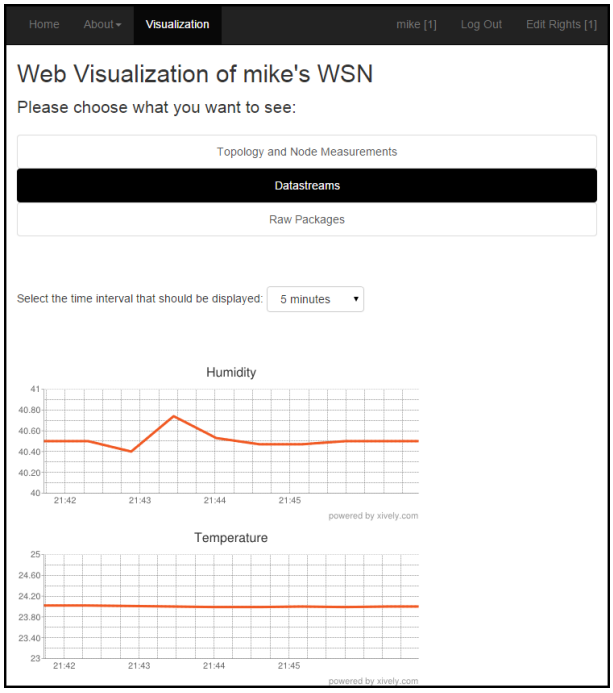


Figure 24: The Datastrems on a tablet sized screen

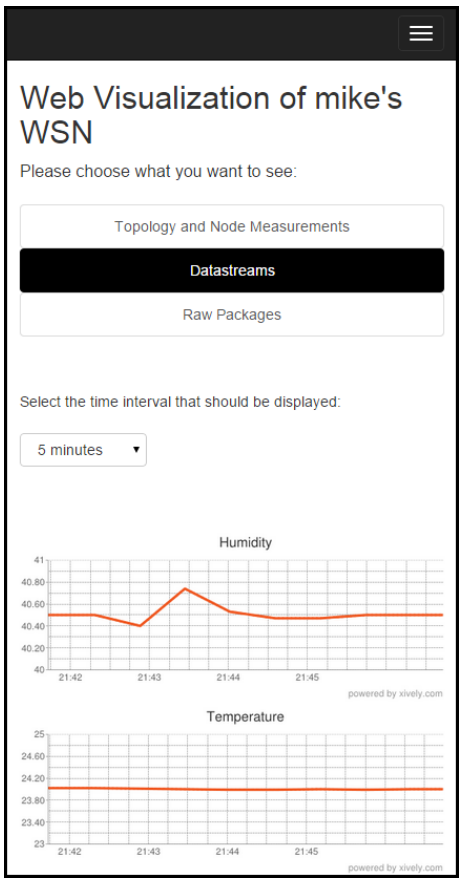


Figure 25: Datastrem on a Small Phone-Sized Screen

data encapsulated in POST variables is saved in the database is very straight forward. Each of those files first check if a username and password has been send. If so, the website checks if that username is in the database and if the password matches it, which is done the same way as with logging-in. Further, it is verified that the network, which is getting a data upload (identified by *[random]* that is also send) belongs to the user that has just been authenticated. This is required for security, as it is quite easy to manually send POST request to pages on the Internet even if they do not come with a HTML form (e.g., with the Firefox plug-in Live HTTP Headers [9]). Without authentication, everyone who knows about those four files and the name of the post variables they require could manipulate and falsify any WSN that is represented on the page.
If the verification succeeded and the authenticated user is owning the sensor network, then the database is updated. The content of the remaining POST variables define if rows are added, removed or updated and what the new values are. An example of how this looks like on the topology updating code can be seen in Figure 26.

```php
require_once 'config.php';
require 'myFunctions.php';

if(isset($_POST['username']) && isset($_POST['password']) && isset($_POST['dblink'])
   && hasUpdatePermission($pdo, $_POST['username'], $_POST['password'], $_POST['dblink'])) {
    if(isset($_POST['task'])&&isset($_POST['dblink'])&&isset($_POST['source'])&&isset($_POST['target'])&&isset($_POST['weight'])) {
        if($_POST['task'] == "add") {
            try {
                $pdo->query("INSERT INTO `".config::$mysqldb."`.`".$_POST['dblink']."_TOPOLOGY` (`SOURCE`, `TARGET`, `WEIGHT`)
                             VALUES ('".$_POST['source']."', '".$_POST['target']."', '".$_POST['weight']."');");
                echo "added - no error";
            } catch (PDOException $ex) {
                echo "An Error occured!";
                echo $ex->getMessage();
            }
        } elseif ($_POST['task'] == "remove") {
            try {
                $pdo->query("DELETE FROM `".config::$mysqldb."`.`".$_POST['dblink']."_TOPOLOGY` WHERE `SOURCE`='".$_POST['source']."'
                             and `TARGET`='".$_POST['target']."' and `WEIGHT`='".$_POST['weight']."';");
                echo "removed - no error";
            } catch (PDOException $ex) {
                echo "An Error occured!";
                echo $ex->getMessage();
            }
        }
    }
}
```

Figure 26: PHP Code on the Server that is Used to Upload Topology Data

4.3 Implementation Details of Part 2

In this Section, the stand-alone tool, which is adding and removing user accounts to and from the websites database is elaborated in detail. The tool is written in Java and comes with a very simple user interface, that represents and shows the workflow of creating or deleting a user. There are two slightly different workflows for making new accounts depending on the status. If the user has limited view rights, an additional step is necessary to ask whose WSNs that user should be allowed to see. Figure 27 shows that workflow, where step 3 only appears if an account with limited access rights is created.

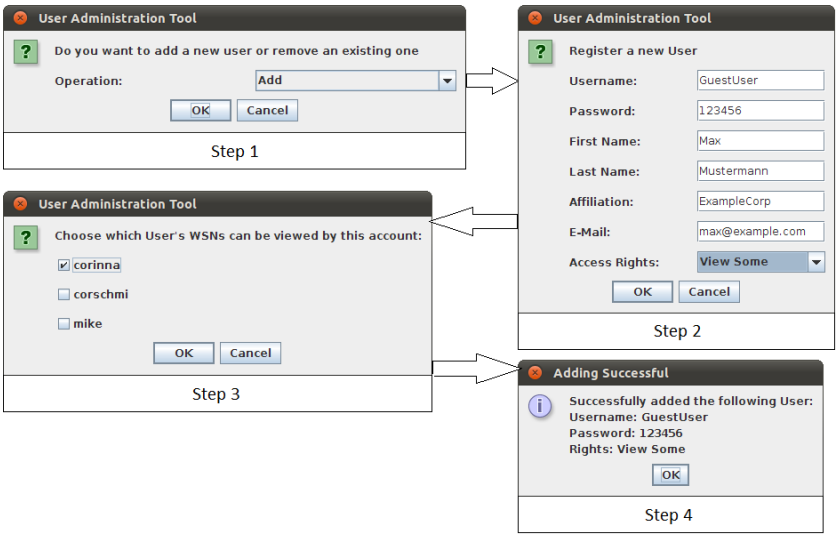


Figure 27: Steps Involved in Creating an Account Using the User Administration Tool

When a user is added to the database, the information that has been provided is pushed into the database, just the way it has been entered into the field. The only exception is the password, which is not saved itself for security reasons: If the database should

be compromised and an authorized person is able to access that table, he cannot read any passwords, because they are not saved. Instead the database saves the result of a Password-Based Key Derivation Function, the PBKDF2. First proposed by B. Kaliski, it derives a key from a password, a (pseudo) randomly generated salt, the iteration count and the defined key length [19]. The author proposed a key generation process in PBKDF2 that consists of these four steps:

1. Generate the salt and define the iteration count
2. Select the length of the resulting key (in octets)
3. Apply the key derivation function to the password, salt, iteration count and key length
4. Put the key together from function results and return it

For this tool, the salt has been generated using the Cryptographically Secure Pseudo-Random Number Generator offered by the java.security.SecureRandom library. The mechanism used for message authentication is called Keyed-Hashing for Message Authentication (HMAC, as defined in RFC 2104 [27]). The HMAC used in this tool is based on SHA1, as PBKDF2 with HMAC and SHA1 is already provided by Java when requesting a SecretKeyFactory [34]. After the password key has been generated, the Java API called Java DataBase Connector (JDBC) is used to save the user information and key into the database. More information on database access through JDBC will be given in the following Section.

4.4 Implementation of Part 3

As mentioned in Subsection 4.1.3, the pre-existing CoMaDa code was extended by a new module, which is detailed here. It essentially builds the connection between the framework, which is connected to the actual WSN and building a virtual representation of it, and the website. This connection works in one direction only: It is connecting the framework to the website to transport measurements and other data from the framework onto the online database, but not from the website or database into CoMaDa. There are two consecutive phases in which the module is actively doing something: First, the Module starts up and establishes that connection to the database when CoMaDa is started. Then it idles while the user is programming his sensors and deploying the network. When the network is in place and the readings from it are coming in, the module goes into the second phase, which is uploading the data to the database it is connected to. In the following, those two phases are illustrated in detail.

4.4.1 Startup Phase

When the CoMaDa Framework is starting it creates a new instance of the web module, which runs in its own thread. When initialized, the module prompts for a username and

a password, which have to be entered by the user who started the framework. Then, a connection to the database specified in the config file is established. For this, the java.sql.DriverManager is used to get a connection, which is saved, or the module throws an error if the connection could not be established. Typically, cause for an error at this stage is a lack of Internet connectivity or a wrong variable int the config file.
Then, the module uses the established connection to check if an account with the provided username can be found in the database, if the password matches that account and if it has a status with the permission to upload sensor network readings. If this is not the case the connection will be closed again and the module is not doing anything anymore, while the rest of the framework is running as usual. In case the user has authenticated successfully, the database is prepared to host the WSN. For this, a random String of characters with given length is produced, which will be used as unique identifier for this network (henceforth called *[random]*). Therefore, the module compares it to the identifiers already taken, which are listed in the ACTIVE_WSN database table. If it was already taken, a new one is generated and compared to the table, until one has been made that is not used yet. Then, *[random]* is saved together with the username of the person that authenticated inside the ACTIVE_WSN table. Then, the following four tables are generated which will be holding all the data and measurements of this sensor network:

1. Table *[random]*_DATA will hold all sensor measurements of the network. It comes with columns NODE_ID (identifying which node measured it), DATUM_ID (identifying the sensor of the node who made the measurement), NAME (name of that sensor, if any), TYPE (what has been measured, e.g., temperature), VALUE (saves the actual measurement) and UNIT (saving what unit has been measured in).

2. Table *[random]*_TOPOLOGY will hold information on what sensor routs his data to which other node until it reaches the sink. This table is made with the columns SOURCE (to save the ID of the transmitting sensor), TARGET (to save the ID of the node the sensor transmits to) and WEIGHT (the strength of the connection between source and target).

3. Table *[random]*_PROTOCOL will hold raw packets as they come into the framework from the sink. It has the columns TEXT (to save the actual packets in) and timestamp (which will be filled by default with the time of saving the packet).

4. Table *[random]*_DATASTREAM will save information on active datastreams that are uploaded on Xively and where to get the picture from. It contains the columns FEED_NAME (the name of the feed that holds the stream represented in each row), STREAM_ID (the ID of the datastream that the picture will show, usually those are descriptors like temperature) and URL (the web link from where the picture showing the datastream is fetched).

The module then creates an instance of a class called MyShutdownHook, which is running in a separate thread. As the name suggests, it doesn't do anything until the framework is shut down. When this happens, the hook instance is trying to connect to the database where it is dropping the four tables that have been made for this WSN, as well as deleting its entry from the ACTIVE_WSN table.

Finally, the module subscribes to events that are relevant to the representation of this network. Events are fired by CoMaDa, which is interacting with the sink and building a virtual representation of the WSN. The WSNTopologyUpdatedEvent, for example, is fired every time after the framework has received an update on the topology status and saved it. This event includes an instance of the topology class when it is fired, which includes all updated topology links. By subscribing to that event, the module is defining which of its methods is called when the event is fired. Also, the topology included in this event will be passed on to the called method as parameter, which will then upload the information to the database. This subscription is done for a number of other events as well, which are fired when node measurements have changed, raw or processed packets are saved or datastreams are updated. After finishing those tasks, which will only take a few seconds while the framework is starting up, the module waiting for the WSN to be set up.

4.4.2 Running Phase

After the nodes are configured, the sink is connected to CoMaDa, and the sensors are deployed and turned on, data starts to arrive. This is when the WebMaDa module becomes active again. The incoming data is caught, interpreted by CoMaDa and then send to the module via events, as it always does for communicating with applications (see Figure 4).
WebMaDa then uploads the data to the different tables that have been generated when it was started. It does that by sending POST requests that contain the data, as well as the username and password that the user has entered when starting the framework, to the server (namely the four files that have been described in the 'Hook for data upload'-Subsection). So unlike the user administration tool or the start-up process where the database is accessed and updated by directly accessing it (using the JDBC), the data upload is routed through the server that hosts the website. The reason for that is to ensure data integrity: If a user accesses for example the measurements of one node it could happen that the website is reading the node measurement at the same time than the framework is updating them. This could lead to wrong statements about the sensor, for example if the node time is updated by the framework before the data gets pulled by the website, but the temperature value after, then it will show a certain temperature at a time when it was not measured. This is prevented by uploading the data via the server. If data is requested by the website at the same time when the framework is sending an update using the website, the server where the website is hosted handles one request first and queues the other until its done. That way, the user can be sure that all data he is shown was measured at the same time.

Of course, sending database updates via the website increases the servers workload. That is why the methods, which are called when a event is fired, are only updating if something actually changed. This is saving a lot of updates, as most of the sensor data never changes. For example a node that measures temperature and humidity keeps five fields: For the node ID, the voltage, the run-time, the temperature and for the humidity. Each of those fields has six pieces of data: The ID of the node that measured it, an ID for this field, the name, the data, the value, and the unit its measured in. If the WSN is running without

being specially altered, only the value of the temperature-, humidity-, and runtime-field is changing. That means that almost always, just three out of all 30 pieces of data need to be send to the server for updating. This is why the module keeps track of the last state of the sensor and checks which of those fields have changed since last time. In the case of the temperature and humidity measuring node 90% of the workload can be prevented by sending only those values for update that changed.

Chapter 5

Evaluation

The purpose of this thesis was to build a mobile application, that will visualize all aspects and data of wireless sensor networks that were set up using the CoMaDa framework, similarly to how the framework already visualizes it locally. Therefore, the main goal of this thesis was introducing mobility into the visualization of a sensor network as done with the developed solution WebMaDa. In this Chapter, it is assessed how well this goal has been reached. After that, the design and implementation of the tool is evaluated through highlighting some aspects with either negative or positive implications on the tool. In the end, there will be also a statement on the topic of performance and scalability.

5.1 Mobility Goal

It can be stated without a doubt that the system implemented in WebMaDa fulfills the main goal, which was enabling mobile observation of sensor network readings based on the CoMaDa framework. Due to the design of the tool, meaning the 3 part architecture introduced in Chapter 4, sensor readings are available through the Internet and are, therefore, accessible all over the globe. And due to the responsive implementation of the web user interface of the system, it can be conveniently used on all devices that can connect to the Internet through a web-browser, independent of their screen size and resolution. This can be best seen in the figures 22, 23, 24, and 25, which show the same page of the WebMaDa website in the same browser with a different size. Furthermore, the system uses CoMaDa's virtual representation as source to get and visualize WSN data. This means, that CoMaDa is still used to program nodes and gather sensor readings when utilizing WebMaDa for mobile access. Therefore, WebMaDa users can still profit from all other applications that were implemented for CoMaDa and also from CoMaDa's user interface for easy node programming and heterogeneous network support. So all in all it can be stated that all the requested properties have been met by WebMaDa.

5.2 Tool Design and Implementation

The design and implementation of WebMaDa can be described as a general success. The key was extending the pre-existing framework in a way that it also shows sensor networks on phones, instead of programming a stand alone mobile phone app with some of the frameworks functionalities. Latter would have been operating system dependent like every app, and would have only lead to mobility within the reach of the network, as the phone app would have needed to connect to the sink of the network which only has a limited range.
By extending CoMaDa in the way it has been described in this thesis, all mobile devices can be used to view a sensor network, from everywhere in the world, and mobile users profit from all improvements whenever the framework is extended or otherwise advanced, without someone needing to separately improve or enhance a stand-alone tool.

However, by making data of sensor networks available through the Internet, new requirements to authentication and security arise. Those issues have been attended well in the system set up, following the recommendations of [40] by using HTTPS communication to the web-server and SSH tunnel to reach the MySQL database. Also, the implementation of the website and tool pays attention on how passwords are to be stored and evaluated, following the guidelines of [36]. This is to say that necessary precautions for security have been taken and state of the art security mechanisms are in place. However, this should not be mistaken for guaranteed, 100% water proof security as this is simply not possible. However, the tool makes it as hard as possible to be compromised which should discourage big, coordinated, resource and cost intensive attacks, especially when considering that hacking this tool will not result in monitory gains as this would be the case when hacking a E-Banking application for example.

Nonetheless, there is one weak point in the implementation of the frontend: The application used to draw the WSNs topology, cytoscape web [6], uses the Abode Flash multimedia and software platform to display the graphs it generates. However, Google has decided to move away from flash, which means it is not pre-installed on newer android devices and it has recently also been removed from the Google play store [32]. The Adobe Flash Player can still be downloaded from Adobes website and installed manually, but it requires some steps to be taken [1]. Alternately, Android users can download a variety of browser that include Flash from the play store, such as Photon Flash Player & Browser [13], FlashFox [12], Flash Browser [11], or Puffin Web Browser [14]. The fact that there are many Flash browsers and instructions on how to install Adobe Flash Player shows that it is still a technology that is being used, even on mobile devices. However, it is being replaced slowly and this will require a revision of that part of the website in the future.

With the exception of using Adobe Flash it can be said that the tools design and implementation is evaluated as being state of the art, safe, and smart.

5.3 Performance

There are two parts, which have the potential to cause performance issues: One is the MySQL database server, which is receiving queries from the website visualizing the networks and from the frameworks that are currently uploading data onto the server. The other is the computer (and its Internet connection) where the framework runs on.

Testing the server capacity can be done with either uploading a lot of data by connecting a few very big networks or a lot of small ones. A few very big networks will cause only a small amount of tables to be generated on the database, but some of them (the topology and data ones) will contain a lot of rows. With many small WSNs attached, no table single will contain many rows, but there will be a lot of tables on the database. In both cases there will be the same amount of uploads, given the view big WSNs do as many sensor measurements as the many smaller ones.

For evaluating the capacity of the uploading framework, a big network needs to be connected to one computer, which is running WebMaDa and CoMaDa. Two things will stress the setup with a big WSN connected: One the one side, there is a lot of data that is being routed to the underlying CoMaDa framework, which could generate problems with the sink's capacity, or the virtual representation that is created could grow to large. On the other hand, there is the stress put on the WebMaDa extension, which is caused by many measurements coming in, by having to react to many incoming events and upload a lot of data.

Therefore, WebMaDa's capacity has been tested using one big WSN, as it evaluates if the server and the local computer can handle it. Unfortunately, the number of nodes that were available for a first performance test were limited. That is why it has been tested with a WSN size of 13 measuring sensors at most. Figure 28 shows how this WSN of 13 nodes is being displayed on a small screen size, where it still fitted nicely, while Figure 29 shows the same network in the CoMaDa user interface. Those figures show how well WebMaDa can visualize the topology (which is the most challenging as it requires the most space) of a bigger WSN within a small space.

The performance of the computer, which was connected to the WSN and the web-server was excellent using that many sensors. But of course the framework should be tested with much bigger WSNs in future projects to ensure a good performance.

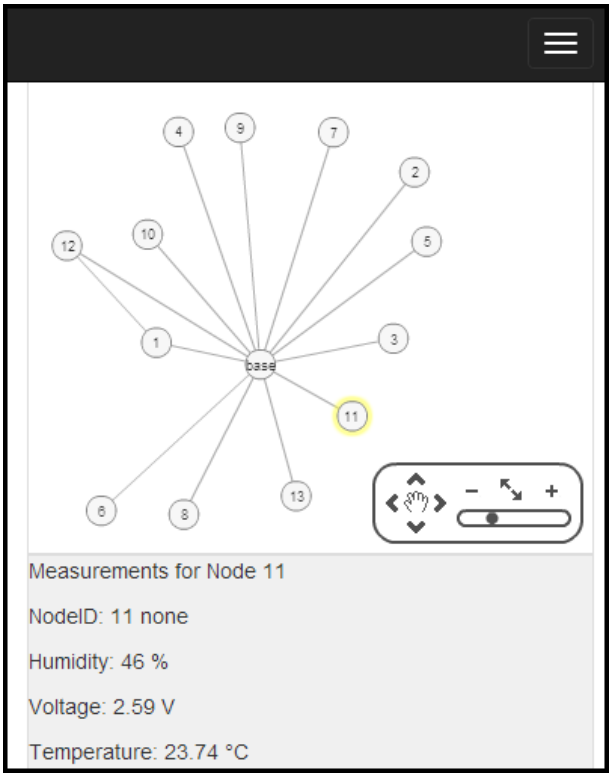


Figure 28: The Topology View of a WSN with 13 sensors on a Small Screen

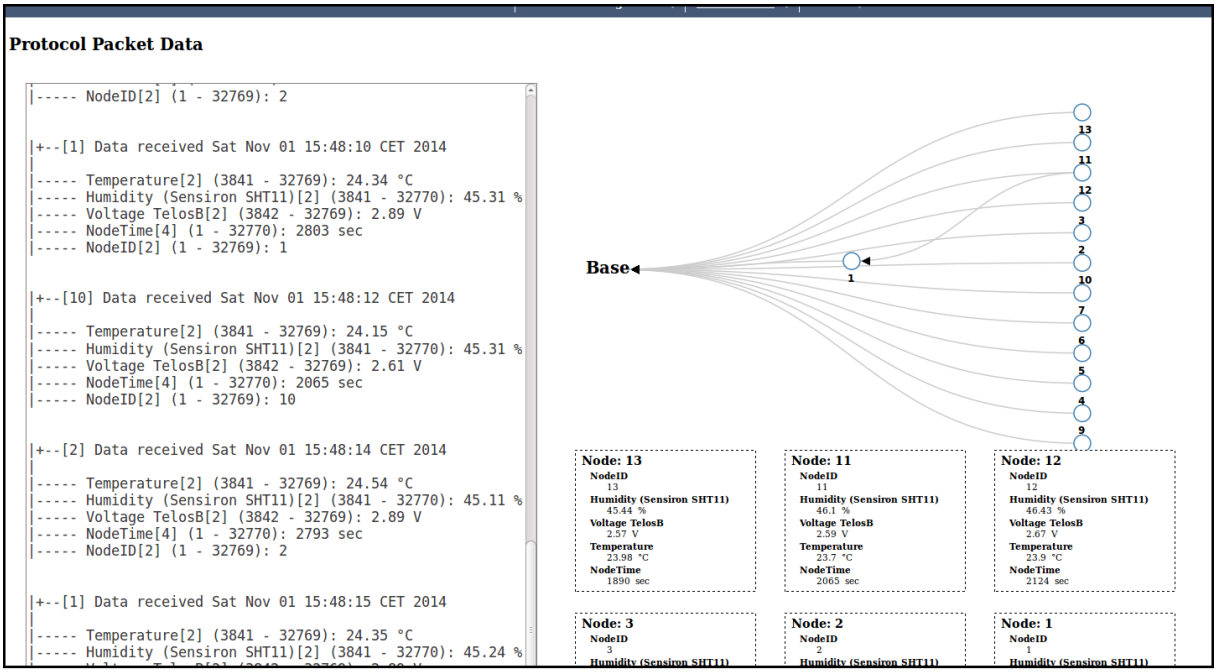


Figure 29: The same Network as in Figure 28, visualized by CoMaDa

5.4 Requirements

The requirements of using WebMaDa depends on what part is being used. There are essentially three different use cases that require different preconditions, which are listed

in the following:

- The end user - This is a person who is interested in viewing the representation of at least one WSN that is provided by WebMaDa. In order to do so, he or she needs three things: An Internet connection, any device that can be connected to the Internet that comes with a web-browser, as well as the Adobe Flash Player installed on that device. Those are the only things that the end user needs to provide himself, and is most likely already in his possession. However, he also needs to be granted access to the website hosting WebMaDa, but this cannot be obtained by him or her, but must be given by the system administrator.

- The WSN uploader - This is the category of users who upload and visualize their own WSN on the Internet using WebMaDa. They obviously need wireless sensors in order to build the WSN that they want to host, but this is of course required by everyone wanting to visualize a WSN, no matter what framework is used. Other than that, only a computer with Internet access and the WebMaDa framework is required. Again those items are most likely in the possession of everybody that would want to visualize a WSN, with the exception of WebMaDa which obviously needs to be downloaded first. Like the end user, any WSN uploader also needs access to the website that WebMaDa is configured to connect with, which he cannot get himself but needs to be given to him by the administrator.

- The WebMaDa provider - Those are the users that provide and administer a website together with a database for hosting online WSN representations. Of course they need access to a server where they can set up a website and MySQL database. Furthermore they require the WebMaDa website code and the user administration tool in order to set up accounts for the people that should get access to the website (e.g., the end users and WSN uploaders). With that, WebMaDa can be set up following the instructions of Section 4.1. While access to a server is not something many people already have, it is easy to get, as many ISPs provide that for a small fee.

It can be concluded that using the WebMaDa framework does not require a lot of infrastructure to be used. Especially end users and WSN uploaders only need to have things that most people have access to already. This is another reason to use WebMaDa.

Chapter 6

Summary and Conclusions

This thesis has introduced the Web-based Mobile Access and Data Handling (WebMaDa) framework. It is an extension of the CoMaDa framework, which means it has to be used in order for WebMaDa to work. A user, therefore, profits from all its key features and advantages, like creating a WSN independent of underlying hardware, easy programming of nodes, or profiting from other applications that are interacting with CoMaDa as well. On top of that, WebMaDa allows for visualizing and monitoring those WSNs on mobile devices all around the globe through the Internet.

WebMaDa incorporates three parts to introduce mobility to the visualization of WSNs: One is a website, which provides the user interface where, after successful authentication, multiple WSNs can be viewed. It shows the WSN topology, sensor measurements, raw data packets and datastreams over time. Another part is a Java module that is running alongside the CoMaDa framework, which is feeding information to the web UI. It establishes a connection to the database and then uses event subscriptions to get notice whenever CoMaDa is receiving or calculating data on the connected WSN, which are uploaded onto the database, so it can be visualized by the website. The last part is a stand-alone Java application, which is used to generate or remove user accounts that are used to identify when viewing WSNs on the website or uploading them with the module.

Thanks to solid design decisions and implementation of WebMaDa it does not only fulfill its purpose of visualizing WSNs on mobile devices. It also enables complete mobility in viewing the network, enables for pooling multiple WSN representations on one location, incorporates best practices when it comes to security and access control, and it will automatically profit from future improvements and enhancements made to the underlying CoMaDa framework. It can, therefore, be concluded that WebMaDa works as intended and is a good option for programming, deploying and monitoring a WSN.

Even though the product of this thesis can be described as a success, WebMaDa is only a first version that has been developed within a limited amount of time. Future WebMaDa updates and extensions can address the following things, which did not fit in the volume of this thesis.

The first issue that should be solved is the way WebMaDa is the drawing of WSN topologies. The third party library that is used to generate the topology (cytoscape web) uses

Flash as data format on the website. As it has been shown in Chapter 5, the Flash standard is slowly disappearing from mobile devices and should, therefore, be replaced on the website. This requires using another library for graph visualization which does not use Flash, or program everything from scratch.

Another area with potential for further improvement is the interactions of user accounts on the website. So far, the website includes mostly functionalities that are required from user accounts, such as authentication through log in, access restrictions, rights management by administrators, and requesting access. There is, however, much more that could be done with user accounts. For example would it be possible that users can give names to the WSNs that they are uploading, which could give further information on the WSN (e.g., location, purpose). Also, it might be desirable to give each uploader of WSNs the possibility to see who has access to them, and manage all rights to their networks, not just requests. Apart from that, there are multiple other functionalities that could be incorporated which already exist on other platforms that use authentication (e.g., commenting and linking other WSNs, adjusting of account information by their users, messaging others, making friends).

To conclude this thesis, it can be said that the things, which could be done in future work are either to ensure WebMaDa's usability in the future (removing Flash) or to make the use of the service more appealing to end users (more account functionalities). But for now, WebMaDa incorporates everything needed to work properly and safely. At the moment, WebMaDa is running on the website http://tinyurl.com/WebMaDa where it can be tried out. But keep in mind that CoMaDa is needed on the local machine also for displaying your own WSN.

Bibliography

[1] Adobe. Manually Install on Android Devices. `http://helpx.adobe.com/flash-player/kb/installing-flash-player-android-devices.html`, 2014. Last accessed: 11.11.2014.

[2] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. A Survey on Sensor Networks. *Communications Magazine, IEEE*, 40(8):102–114, 2002.

[3] Apple. Apple Developer Support - Program Enrollment. `https://developer.apple.com/support/ios/enrollment.html`, 2014. Last accessed: 08.11.2014.

[4] Bootstrap. Bootstrap About Page. `http://getbootstrap.com/about/`, 2014. Last accessed: 09.11.2014.

[5] Canfeng Chen, Jian Ma, and Ke Yu. Designing Energy-Efficient Wireless Sensor Networks with Mobile Sinks. In *Proceeding of the 4th ACM Conference on Embedded Networked Sensor Systems, SenSys*, 2006.

[6] Cytoscape. Main Page of Cytoscape Web. `http://cytoscapeweb.cytoscape.org/`, 2014. Last accessed: 09.11.2014.

[7] David Egan. The Emergence of ZigBee in Building Automation and Industrial Controls. *Journal of Computing and Control Engineering, IET*, 16(2):14–19, 2005.

[8] Sepideh Fouladgar, Bastien Mainaud, Khaled Masmoudi, and Hossam Afifi. Tiny 3-TLS: A Trust Delegation Protocol for Wireless Sensor Networks. In *Proceedings of the Third European Conference on Security and Privacy in Ad-Hoc and Sensor Networks*, ESAS, pages 32–42, Berlin, Heidelberg, 2006. Springer-Verlag.

[9] Mozilla Foundation. Live HTTP Headers Page for Download and Information. `https://addons.mozilla.org/de/firefox/addon/live-http-headers/`, 2014. Last accessed: 10.11.2014.

[10] Brett S. Gardner. Responsive Web Design: Enriching the User Experience. *Connectivity and the User Experience*, 11(1):13, 2011.

[11] Google. Flash Browser. `https://play.google.com/store/apps/details?id=com.whattheapps.fbrowser`, 2014. Last accessed: 11.11.2014.

[12] Google. FlashFox - Flash Browser. `https://play.google.com/store/apps/details?id=mobi.browser.flashfox`, 2014. Last accessed: 11.11.2014.

[13] Google. Photon Flash Player and Browser. https://play.google.com/store/apps/details?id=com.appsverse.photon, 2014. Last accessed: 11.11.2014.

[14] Google. Puffin Web Browser Free. https://play.google.com/store/apps/details?id=com.cloudmosa.puffinFree, 2014. Last accessed: 11.11.2014.

[15] Vehbi C Gungor, Bin Lu, and Gerhard P Hancke. Opportunities and Challenges of Wireless Sensor Networks in Smart Grid. *IEEE Transactions on Industrial Electronics*, 57(10):3557–3564, 2010.

[16] Vipul Gupta, Matthew Millard, Stephen Fung, Yu Zhu, Nils Gura, Hans Eberle, and Sheueling C. Shantz. Sizzle: a Standards-Based End-to-End Security Architecture for the Embedded Internet. In *Third IEEE International Conference on Pervasive Computing and Communications, PerCom*, pages 247–256, 2005.

[17] Elyes Ben Hamida and Guillaume Chelius. Strategies for Data Dissemination to Mobile Sinks in Wireless Sensor Networks. *Journal on Wireless Communications, IEEE*, 15(6):31–37, 2008.

[18] Andrew Hunt and David Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Longman Publishing Co., Boston, MA, USA, 1999.

[19] Shakir M. Hussain and Hussein Al-Bahadili. A Password-Based Key Derivation Algorithm using the KBRP Method. *American Journal of Applied Sciences*, 5(7):777, 2008.

[20] Kwang-il Hwang, Jeongsik In, NhoKyung Park, and Doo-seop Eom. A Design and Implementation of Wireless Sensor Gateway for Efficient Querying and Managing through World Wide Web. *IEEE Transactions on Consumer Electronics*, 49(4):1090–1097, 2003.

[21] IDC. Press Release on Mobile OS Sales. http://www.idc.com/getdoc.jsp?containerId=prUS25037214, 2014. Last Accessed: 07.11.2014.

[22] IEEE. Citations of A Survey on Sensor Networks. http://ieeexplore.ieee.org/xpl/abstractCitations.jsp?tp=&arnumber=1024422&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D1024422, 2014. Last accessed: 13.11.2014.

[23] Xiaofan Jiang, Stephen Dawson-Haggerty, Prabal Dutta, and David Culler. Design and Implementation of a High-Fidelity AC Metering Network. In *International Conference on Information Processing in Sensor Networks, IPSN*, pages 253–264. IEEE, 2009.

[24] Hong-Taek Ju, Mi-Joung Choi, and James W Hong. An Efficient and Lightweight Embedded Web Server for Web-Based Network Element Management. *Int. Journal of Network Management*, 10(5):261–275, 2000.

[25] Chris Karlof, Naveen Sastry, and David Wagner. TinySec: A Link Layer Security Architecture for Wireless Sensor Networks. In *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, SenSys, pages 162–175, New York, NY, USA, 2004. ACM.

[26] Thomas Kothmayr, Corinna Schmitt, Wen Hu, Michael Bruenig, and Georg Carle. DTLS Based Security and Two-Way Authentication for the Internet of Things. *Ad Hoc Networks*, 11(8):2710–2723, 2013.

[27] Hugo Krawczyk, Ran Canetti, and Mihir Bellare. RFC 2104: HMAC: Keyed-Hashing for Message Authentication. 1997.

[28] Elaine Lawrence, Karla Felix Navarro, Doan Hoang, and Yen Yang Lim. Data Collection, Correlation and Dissemination of Medical Sensor Information in a WSN. In *Fifth International Conference on Networking and Services, ICNS*, pages 402–408. IEEE, 2009.

[29] LogMeln. Xively About Page. `https://xively.com/about/`, 2014. Last accessed: 09.11.2014.

[30] Mark Luk, Ghita Mezzour, Adrian Perrig, and Virgil Gligor. MiniSec: A Secure Sensor Network Communication Architecture. In *6th International Symposium on Information Processing in Sensor Networks, IPSN*, pages 479–488, 2007.

[31] Jian Ma, Canfeng Chen, and Jyri P Salomaa. mWSN for Large Scale Mobile Sensing. *Journal of Signal Processing Systems*, 51(2):195–206, 2008.

[32] BBC News. Adobe Flash Player Exits Android Google Play Store. `http://www.bbc.com/news/technology-19267140`, 2012. Last accessed: 11.11.2014.

[33] Martin Noack. Optimization of Two-Way Authentication Protocol in Internet of Things. Master's thesis, Zurich, Switzerland, 2014.

[34] Oracle. Java Cryptography Architecture Standard Algorithm Name Documentation. `http://docs.oracle.com/javase/7/docs/technotes/guides/security/StandardNames.html#SecretKeyFactory`, 2014. Last accessed: 10.11.2014.

[35] Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. SPINS: Security Protocols for Sensor Networks. *Wireless Networks*, 8(5):521–534, 2002.

[36] Defuse Computer Security R&D. Salted Password Hashing - Doing it Right. `https://crackstation.net/hashing-security.htm`, 2014. Last accessed: 10.11.2014.

[37] Corinna Schmitt, Andreas Freitag, and Georg Carle. CoMaDa: An Adaptive Framework with Graphical Support for Configuration, Management, and Data Handling Tasks for Wireless Sensor Networks. In *9th International Conference on Network and Service Management (CNSM)*, pages 211–218, 2013.

[38] Scopus. Citations of Wireless Sensor Network Survey. `http://www.scopus.com/results/citedbyresults.url?sort=plf-f&cite=2-s2.0-46449122114&src=s&imp=t&sid=5EE007F0643E98CA9120296D9022F26F.53bsOu7mi7A1NSY7fPJf1g%3a30&sot=cite&sdt=a&sl=0&origin=inward&txGid=5EE007F0643E98CA9120296D9022F26F.53bsOu7mi7A1NSY7fPJf1g%3a2`, 2014. Last accessed: 13.11.2014.

[39] Zach Shelby and Carsten Bormann. *6LoWPAN: The Wireless Embedded Internet*, volume 43. John Wiley & Sons, 2011.

[40] William Stallings. *Cryptography and Network Security: Principles and Practice*. Pearson Education, 3rd edition, 2002.

[41] Rui Tan, Guoliang Xing, Jinzhu Chen, Wen-Zhan Song, and Renjie Huang. Quality-Driven Volcanic Earthquake Detection Using Wireless Sensor Networks. In *IEEE 31st Real-Time Systems Symposium, RTSS*, pages 271–280. IEEE, 2010.

[42] Ching-Wun Tsai and Ruay-Shiung Chang. SNMP through WWW. *International Journal of Network Management*, 8(2):104–119, 1998.

[43] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless Sensor Network Survey. *Computer Networks*, 52(12):2292–2330, 2008.

[44] Jinfeng Zhang, Canfeng Chen, Jian Ma, Nengqiang He, and Yong Ren. uSink: Smartphone-Based Moible Sink for Wireless Sensor Networks. In *Consumer Communications and Networking Conference, IEEE*, pages 90–95. IEEE, 2011.

Abbreviations

API	Application Programming Interface
CoMaDa	Configuration, Management and Data handling framework
CSS	Cascading Style Sheets
DB	Database
DAP	Data Analysis Process
DRY	Don't Repeat Yourself
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
ID	Identifier
IDC	International Data Corporation
IoT	Internet of Things
IP	Internet Protocol
ISP	Internet Service Provider
JDBC	Java Database Connector
JSON	JavaScript Object Notation
mWSN	Mobile Enabled Wireless Sensor Network
NAT	Network Address Translation
OS	Operating System
PBKDF2	Password-Based Key Derivation Function 2
PC	Personal Computer
PHP	PHP: Hypertext Preprocessor (formerly Personal Home Page Tools)
SD Card	Secure Digital Memory Card
SDAL	Sensor Data Aggregation Layer
SDK	Software Development Kit
SQL	Structured Query Language
SQML	Sensor Query Management Layer
SSH	Secure Shell
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Service
UI	User Interface
UML	Unified Modeling Language
URL	Uniform Resource Locator
WebMaDa	Web-based Mobile Access and Data Handling framework
WSN	Wireless Sensor Network

XML Extensible Markup Language

Glossary

Ad-Hoc Ad hoc is a Latin phrase meaning ”for this”. It generally signifies a solution designed for a specific problem or task, non-generalizable, and not intended to be able to be adapted to other purposes.[Wikipedia]

Array An array is a data structure that is saving mutiple data objects. Those objects can be arrays themselves.

Authentication Authentication is the process of confirming someones identity as the person he claims to be. It is necessary if only certain people have the rights to do something.

Authorization Authorization is the decision whether an entity is allowed to perform a particular action or not, e.g., whether a user is allowed to attach to a network or not.

Building Automation Building automation refers to the process of creating a ‘smart building’, which is a structure that uses sensors and other devices to detect and automatically influence the space within (e.g., motion detectors for turning lights on and off, air measurement units that control ventilation, ...)

Database A database is a collection of data that is organized within tables. Each table has multiple rows that each represent one object the table contains. Each row is further divided into columns which safe pieces of data.

to Deploy In the case of WSNs (which is how it is used in this paper), deployment refers to the process of setting up a WSN by distributing and activate sensors.

Disaster Management Disaster management is everything that is done to milden the effect of disasters like earthquakes, floods or many others. It includes trying to prevent disasters, predicting them, reduce its negative effects while happening and recovering from it.

Edge The edge of a network is what a link or connection between two elements in that network is called.

Encryption Encryption is the process of changing a understandable message or information so that it cannot be understood any more on its own. Only entities that are authorized to read the message by returning it to a readable form.

Framework A framework is a reusable set of code classes or libraries for a software system. In the scope of this thesis it is a set of code that is used to manage WSNs.

Hashing Hashing is the process of using a hash function on a piece of information, which equals a result that is rather easy to calculate. It is, however, almost impossible to revert the result of a hash function back to its original piece of information.

Industrial Control Industrial control includes every system that is in place to control and observe what in an industrial environment (e.g., machines, processes, ...)

Medical Monitoring Medical monitoring refers to the process of monitoring the health of a person through medical sensors and other equipment.

Network A network is a collection of entities that are somehow connected to each other. This happens in various domains (e.g., neural networks, business networks, telecommunication networks) but in this thesis, networks are collections of sensors that are communicating with one another.

Node A node is one entity of a network, that is connected to other nodes via edges which builds the network.

Query A query is a precise request for information in the form of a set of data, that is retrieved from a database using a query language (e.g., SQL)

Responsive Design Responsive design refers to designing a website in a way that it is adapting its layout depending on the size of the screen it is viewed with, in order to make the experience optimal for all sizes.

Sensors Sensors are the entities that form a WSN. The have sensing capabilities that can observe certain attribute of their environment, such as temperature, humidity or sound.

Server A server is a performant computer in a network with other computers (usually the Internet), that can be addressed with requests to which it responds.

Signed Signing a message or information is used to declare where that message or information originated. Therefore, signed content is content which has a verifiable source.

Sink A sink, which is also known as base station, is a entity in a WSN that gathers sensor information and does somthing with it (usually communicating it to a framework for visualization)

Topology The topology of a network is the collection of all edges, that is all established connections between network entities.

List of Figures

Appendix A

Installation Guidelines

Before installation can begin, WebMaDa needs to be integrated into CoMaDa. For this, both a working version of CoMaDa that is set-up properly and the WebMaDa Module code (enclosed on the CD) are required. Then, the following steps need to be taken to integrate WebMaDa:

1. Open the folder code -> WebMaDaModule on the CD. It contains all files necessary for the integration.
2. Create a new package within the CoMaDa-Code called 'de.tum.in.net.WSNDataFramework.Modules.Web'.
3. Copy the following eight files from the CD folder into that package: Config.java, HttpUtility.java, MyShutdownHook.java, PastDatumState.java, PastNodeState.java, WSNDataUploader.java, WSNSQLConnector.java, and WSNWebModule.java
4. Add all .jar files to the classpath of the working CoMaDa framework. If some .jar files are already in the classpath but in an older version, they need to be removed.
5. Add the file WSNDatastreamChangedEvent.java into CoMaDa's package called 'de.tum.in.net.WSNDataFramework.Events'.
6. Open the file WSNApp.java, which is located in the package 'de.tum.in.net.WSNDataFramework'. Right on the top there is a variable called 'providedEvents' which lists event classes. Add 'WSNDatastreamChangeEvent.class' to this list, separated with a comma like the others.
7. Open the file WSNCosmModule.java, which can be found in CoMaDa's package called 'de.tum.in.net.WSNDataFramework.Modules.Cosm'. Go to the method _onEvent(WSNNodeUpdateEvent eve) and add this line of code to the END of the TRY-block: this.app().fireEvent(new WSNDatastreamChangeEvent(_activeFeeds));
8. Open the file WSNFramework.java that is located in the default package. Add the following line of code in the TRY-block AFTER the WSNApp was instantiated and drivers & protocols were added: app.addModule(new WSNWebModule());

After this, WebMaDa has to be set-up and configured following the instructions of Section 4.1 in this thesis.

Appendix B

Contents of the CD

The CD contains the following:

- thesis.pdf - an electronic version in the PDF format of this paper.
- Zusfsg.txt - the german abstract that can also be found in the beginning of this thesis
- Abstract.txt - the english abstract that can also be found in the beginning of this thesis
- code - a folder containing following subfolders:
  - WebMaDaModule - folder containing all files that need to be integrated into CoMaDa to add the WebMaDa extension. The Appendix Chapter 'Installation Guidelines' describes how this is done.
  - WebMaDaWebsite - folder containing all files necessary to create a website for hosting WSN visualizations with WebMaDa
  - WebMaDaUserAdministration - folder containing a complete Java-Project that can be executed, which is needed to insert user accounts into a WebMaDa database