der Bundeswehr
Universität München

User Identification for Ubiquitous User Interfaces Using Bluetooth Low Energy

Bachelor thesis

Christopher Lyko
1202408

Primary examiner:	Prof. Dr. Michael Koch
Secondary examiner:	Prof. Dr.-Ing. Mark Minas
Advisor:	Dr. Julian Fietkau
Deadline:	31 May 2023

Universität der Bundeswehr München
Fakultät für Informatik

Abstract

The Human-Computer-Interaction Group – led by Prof. Dr. Michael Koch at the Universität der Bundeswehr München – conducts research on ubiquitous user interfaces for community awareness – the so-called CommunityMirrors. Currently, the group strives for a practical solution to identify interacting users of the CommunityMirrors to personalize their experience. Therefore, this bachelor thesis discusses a solution to this problem using Bluetooth Low Energy which is a a low power consuming technology enabling wireless communication via radio waves between devices. Thereby, authentication and overall security is not explored. The technical fundamentals of Bluetooth Low Energy are elaborated and its possible uses in the context of user identification are discussed. One use is selected to implement a prototype. The prototype enables the user to be identified and displays their name on the interface. The prototype consists of an identity management system managing the digital identities of the users, an extension of the software of the CommunityMirrors to receive Bluetooth signals and an app for the user which transmits Bluetooth signals. The evaluation of the prototype confirms the practical usability of Bluetooth Low Energy in the context of user identification. But further research is necessary due to its inconsistency.

Contents

1 Introduction

The Human-Computer-Interaction Group (HCI-G) – led by Prof. Dr. Michael Koch at the Universität der Bundeswehr München – conducts research on ubiquitous user interfaces for community awareness – the so-called CommunityMirrors (CMs). The CMs supply nearby users with information they can interact with, which would be passively hidden in a database otherwise. Moreover, the users are animated to cooperate in front of the CMs due to their shared usage in (semi-)public environments (Ott, 2018, abstract).

Currently, the HCI-G strives for a practical solution to identify users in front of the CMs to personalize their experience. The goal of this bachelor thesis is to develop a prototype and integrate it into the software of the CMs. The CMs should also be able to display the names of the identified users.

The basis of the prototype is the usage of the technology Bluetooth Low Energy (BLE). Bluetooth technology was initially created to enable wireless communication between two devices through radio waves without the need for another intermediate networking equipment. It's first version is known as Bluetooth Basic Rate which offered a data exchange rate of 1mb/s between devices, followed later by Bluetooth Enhanced Data Rate with 2mb/s. In 2010 BLE was introduced and defines together with Bluetooth Enhanced Data Rate the Bluetooth Core Specification since its version 4.0. The additional features of BLE are a more efficient use of the device's power and next to 1:1 communication the possibility to realize 1:n and m:n communication between devices (*The Bluetooth Low Energy Primer*, 2023, p. 7).

The users have to be represented digitally, meaning each user who wishes to be identified needs a digital identity. A digital identity is a digital representation of an entity in a computer system. It consists of an identifier with attributes linked to it. "Identification of a user is the association of a personal identifier with an individual presenting attribute". (Camp, 2004, p. 35 - 36)

It is important to note that this bachelor thesis does not focus on authentication – the proof of the "association between an entity and an identifier" (Camp, 2004, p. 36) – and the prototype's security overall.

The thesis is structured this way: Chapter 2 covers (technical) fundamentals meaning the architecture of the CMs, the Bluetooth Low Energy Stack (BLES), parameters influencing the power consumption of BLE and direction finding using BLE. Chapter 3 analyzes related studies by the HCI-G and other institutions. In Chapter 4 the concept of the prototype is developed by conceptualizing an Identity Management System (IMS) which manages the digital identities.

Also, the different possibilities of using BLE are elaborated, compared and one is selected for the implementation. Chapter 5 focuses on the implementation of the concept. Therefore, the concrete implementation of the IMS, the user sided BLE logic and the CommunityMirror (CM) sided BLE logic are analyzed. In Chapter 6 the prototype is evaluated with up to three users. A final conclusion is done in Chapter 7.

2 (Technical) Fundamentals

This chapter covers (technical) fundamentals. It starts with analyzing the architecture of the CMs in whose context the prototype has to be developed. Afterwards, the striking parts of the BLES are covered.

2.1 The Architecture of the CommunityMirrors

Figure 1 shows the architecture of the project[1] containing the CMs.

The *CommunityMashup* integrates data from different social services and an individual content management system and transmits that data to the CMs.

A *CM*[2] consists of a (touch-) display that visualizes the data from the *CommunityMashup*, the hardware to run the software and the software itself – the *CommunityMirror Framework (CMF)*[3]. The users can interact with visualizations on the display via (touch) input.

Log files of the user interaction are saved on the *Community Logging Server*.

These log files are visualized on the *CommunityMirror Dashboard Web App* next to other data which is saved on the *CommunityMirror Dashboard Server*.

The software of the *CMs* – the *CMF* – runs on Windows and is implemented in Java using the Framework JavaFX[4]. JavaFX allows to create Java applications directly with a User Interface (UI). Therefore, the class *Stage*[5] of the Framework contains the UI of the application within one instance of the class *Scene*. The *Scene* manages a scenegraph which consists of instances extending the abstract class *Node*[6] which represents a UI element. This is shown in Figure 2 and Figure 3.

Figure 4 shows the part of the *CMF's* class diagram which is responsible for the data visualization on the CM's display using JavaFX. Therefore, the abstract class *Group* – a subclass of the class *Node* – is extended by the class *VisualComponent*. The *VisualComponent* is the abstract class which is extended by all kinds of visualizations. The figure also shows the visual representation of some of the extending classes on a screenshot of a *CM* display and the controllers controlling the visualized data of the child nodes of the *VisualComponents*.

[1] https://publicwiki.unibw.de/display/MCI/Gesamtprojekt
[2] https://publicwiki.unibw.de/display/MCI/CommunityMirror+-+Grundarchitektur+und+Wording
[3] https://athene2.informatik.unibw-muenchen.de/CM/communitymirrorframework3
[4] https://docs.oracle.com/javase/8/javafx/api/toc.htm
[5] https://de.wikipedia.org/wiki/JavaFX#/media/Datei:Javafx-stage-scene-node.svg
[6] https://de.wikipedia.org/wiki/JavaFX#/media/Datei:Javafx-layout-classes.svg

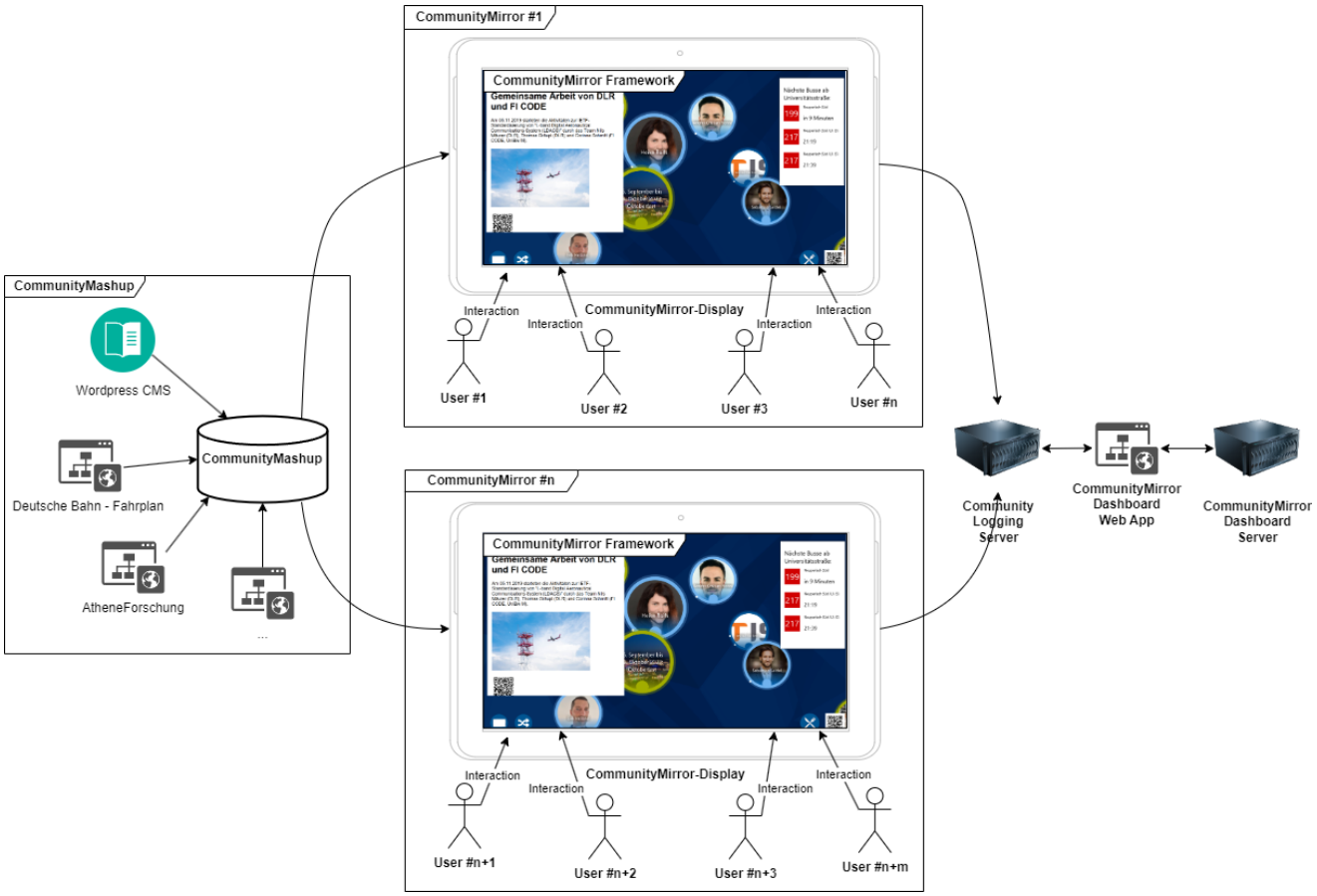

Figure 1: The architecture of the project containing the CMs (`https://publicwiki.unibw.de/display/MCI/CommunityMirror+-+Grundarchitektur+und+Wording`)

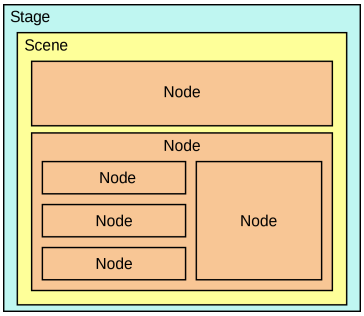

Figure 2: JavaFX application structure (`https://de.wikipedia.org/wiki/JavaFX#/media/Datei:Javafx-stage-scene-node.svg`)

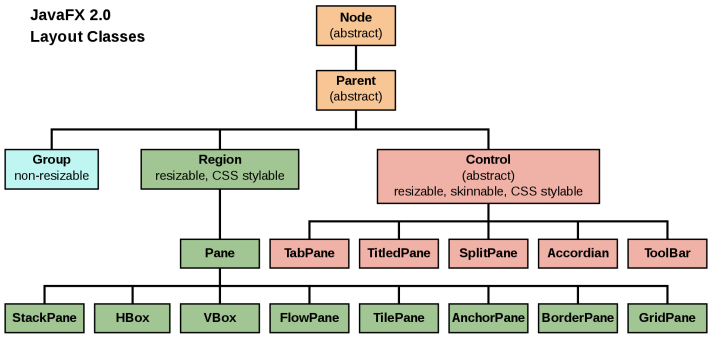

Figure 3: JavaFX Node(`https://de.wikipedia.org/wiki/JavaFX#/media/Datei:Javafx-layoutâĂŞclasses.svg`)

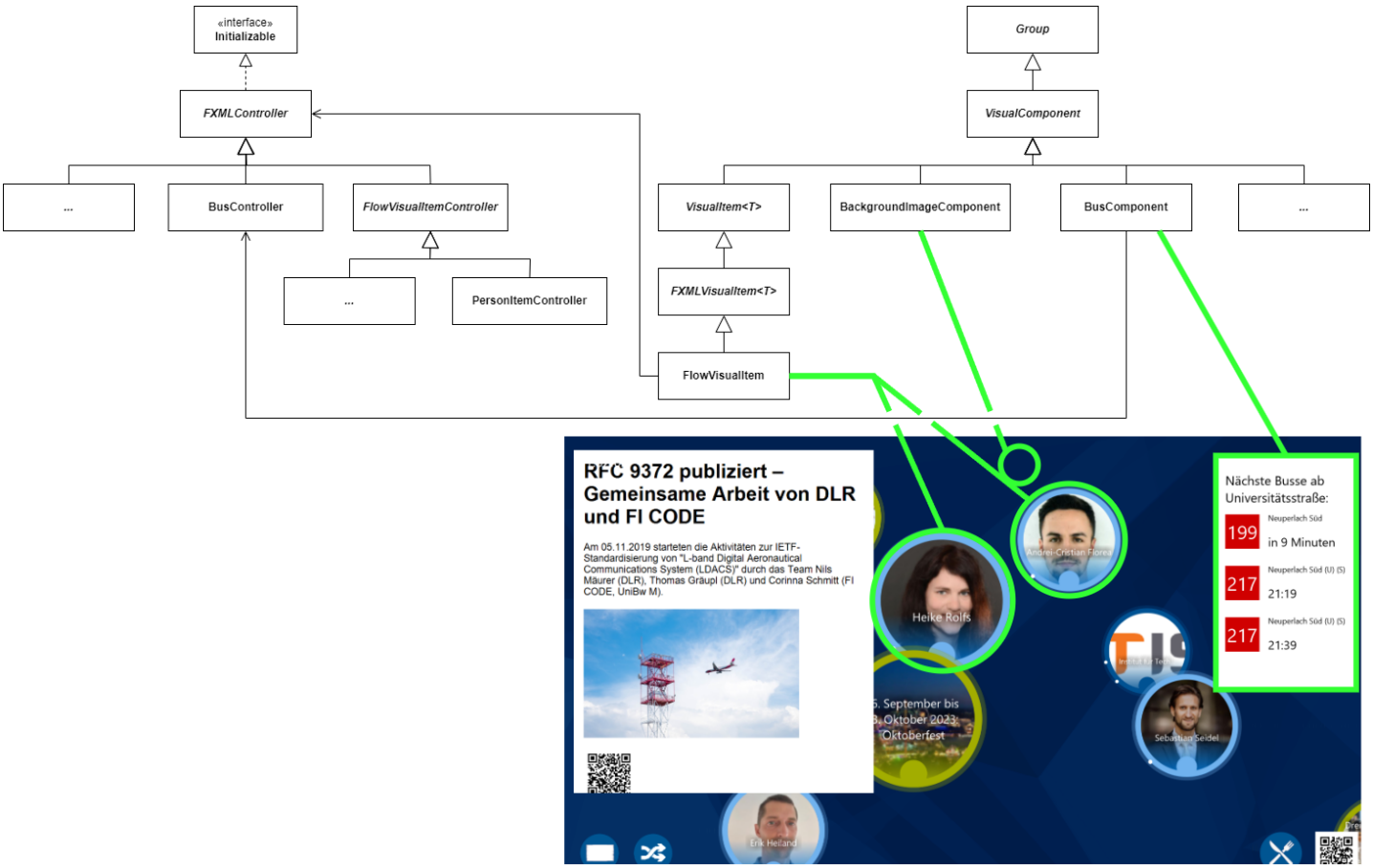

Figure 4: Parts of the class diagram of the CMF responsible for visualization and data control (`https://athene2.informatik.unibw-muenchen.de/CM/communitymirrorframework3`)

2.2 The Bluetooth Low Energy Stack

Figure 5 – the BLES – describes the high-level architecture of BLE. On the highest level the BLES differs between the *host* and the *controller*. Each of them contain different functional layers. The five *host* layers – shown in Table 1 – have to be implemented by the operating system of the device. (*The Bluetooth Low Energy Primer*, 2023, p. 10)

The *controller* is usually implemented on a external chip and manages the usage of radio waves. It executes the defined operations by the host and consists of two modules, shown in Table 2. (*The Bluetooth Low Energy Primer*, 2023, p. 10)

The host and the controller communicate through the Host Controller Interface (*The Bluetooth Low Energy Primer*, 2023, p. 10).

For a better understanding of BLE the following subsections provide a deeper analysis of the Physical Layer, the Link Layer, the Generic Access profile (GAP), the Attribute Protocol (ATT) and the Generic Attribute Profile (GATT).

Module	Key Responsibilities
Generic Access Profile	Highest control layer which defines the roles of the device
Generic Attribute Profile	Organization of attributes from the Attribute Protocol within services, characteristics and descriptors
Attribute Protocol	Client-server communication model
Security Manager Protocol	Protocol for pairing and key distribution
Logical Link Control and Adaption Protocol	Managing the execution of the different protocols

Table 1: The different layers of the *host* (*The Bluetooth Low Energy Primer*, 2023, p. 12)

Module	Key Responsibilities
Link Layer	Responsible for advertising, scanning and creating/maintaining connections within a state machine
Physical Layer	Defines the usage of the radio transmitter and receiver

Table 2: The different layers of the *controller* (the Isochronous Adaption Layer is ignored as it only matters for BLE Audio) (*The Bluetooth Low Energy Primer*, 2023, p. 12)

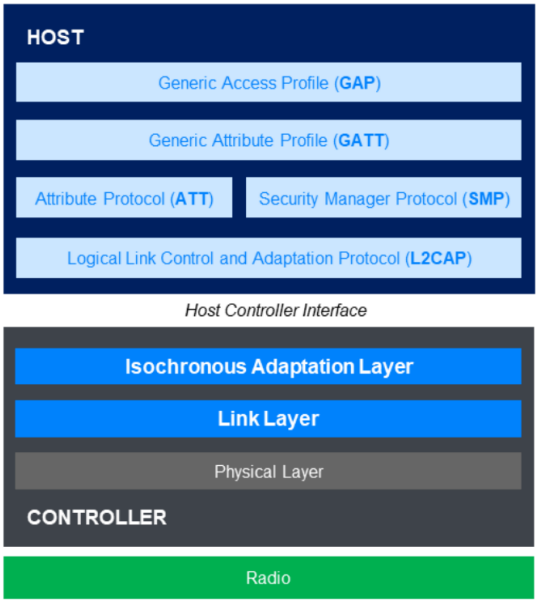

Figure 5: Bluetooth Low Energy Stack (*The Bluetooth Low Energy Primer*, 2023, p. 10)

BLE Packet

Preamble	Access Address	Protocol Data Unit (PDU)	CRC
1 Byte	4 Bytes	2-257 Bytes	3 Bytes

Figure 6: Link Layer Packet (*The Bluetooth Low Energy Primer*, 2023, p. 16)

2.2.1 The Physical Layer

The Physical Layer defines the usage of the device's radio transmitter and receiver. Therefore, digital data is encoded for the transmission and decoded for the receipt of the radio waves. The data is transmitted on at least one of 40 channels with a spacing of 2 MHz in the range of 2400 MHz to 2483.5 MHz in the 2.4 GHz band. The usage of these channels is defined by the Link Layer. (*The Bluetooth Low Energy Primer*, 2023, p. 13)

2.2.2 The Link Layer

The Link Layer defines the data packets that are transmitted over the air and the different modes of the transmission. Also, it contains a state machine – depending on which it operates in different ways. Moreover, it defines the usage of the radio channels of the Physical Layer. (*The Bluetooth Low Energy Primer*, 2023, p. 16)

Packets

Figure 6 shows the components of each data packet exchanged between devices over the air. It consists of the *preamble*, the *access address*, the *protocol data unit* and the *cyclic redundancy check*.

The *preamble* contains useful information for the receiver to optimize the receipt of the packet, for example by synchronizing on the frequency of the packet. The *access address* helps the receiver to evaluate the relevance of the packet by differentiating it from background noise. The *protocol data unit* contains the data to be exchanged. The *cyclic redundancy check* is a uniquely calculated value from the other data in the packet by the transmitter. The receiver also calculates that value from the data it received and compares it with the transmitted *cyclic redundancy check*. If both values are equal, the packet was correctly transmitted and received. (*The Bluetooth Low Energy Primer*, 2023, p. 17)

The Link Layer State Machine

The Link Layer operates depending in state machine which is shown in Figure 7.

The Link Layer can contain multiple instances of the Link Layer State Machine depending on the hardware and software of the device. These are the different states: *standby*, *advertising*, *initiating*, *connection*, *scanning* and *synchronization*.

In the state *standby* the device neither transmits nor receives packets.

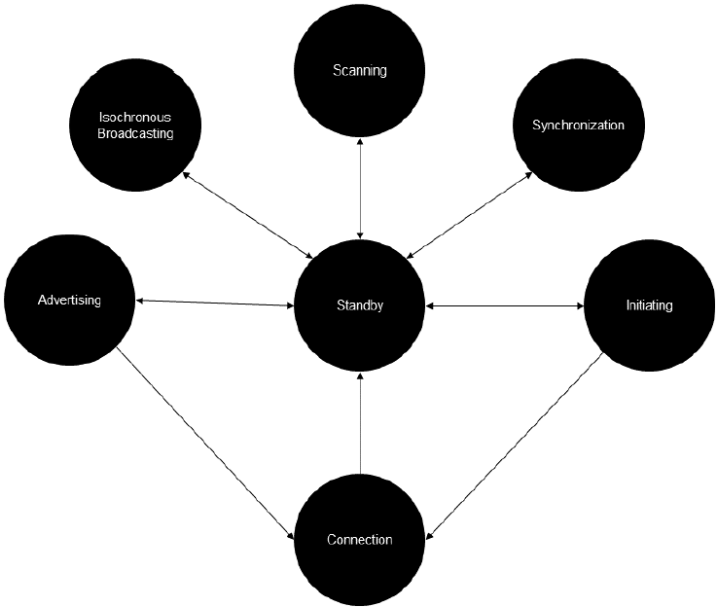

Figure 7: The Link Layer State Machine (*The Bluetooth Low Energy Primer*, 2023, p. 18)

In the state *advertising* the device sends out advertising packets in a predefined *advertising interval*. On the one hand *advertising* is used to indicate the existence of a connectable device to other receiving devices nearby. After an accepted connection request the devices is called *peripheral*. On the other hand *advertising* can be used to transmit data to unconnected devices nearby that are in the state *scanning* or *synchronization*. In that case the device is called *broadcaster*. Advertising packets are transmitted on the channels 37, 38 and 39. Each *advertising event* is timed depending on the *advertising interval*. These timings are made slightly irregular to avoid persistent collisions with packets of other advertising devices.

In the state *initiating* the device sends a connection request to a connectable advertising device. Once the connection is established the device is called the *central*.

In the state *connection* the device is connected with another device. Both devices communicate over encrypted data packets and mutually confirm the receipt of each packet. One of the devices is called *peripheral*, the other is called *central*. 37 of the 40 channels can be used by connected devices. A channel is deterministically selected before the transmission of a packet using a channel selection algorithm. The connection is sustained by exchanging packets in a predefined interval, called *connection interval*. If a connection event is not executed and the *connection supervision interval* is exceeded the connection is lost and both devices switch to their previous state.

In the state *scanning* the device listens for advertising packets from other devices. If the device receives advertising packets from a connectable device and successfully connects to it it is called *central*. If the device just listens to the advertising packets from non connectable devices or without the intention of initializing a connection it is called *observer*. The *observer*

Role	Description
Broadcaster	A device which advertises constantly. Usually, it is non connectable unless it also acts as a *peripheral*. It is equipped with a transmitter.
Observer	An *observer* scans constantly. It does not connect to other devices, unless it also acts as a *central*. It is equipped with a receiver.
Peripheral	A *peripheral* is a *broadcaster* that switched its state from *advertising* to *connection* during the connection process. It is equipped with a transmitter and a receiver.
Central	A *central* is an *observer* that switched its state from *scanning* over *initiating* to *connection*. During the process a encryption procedure is exchanged. It is equipped with a transmitter and a receiver.

Table 3: The four GAP roles (*The Bluetooth Low Energy Primer*, 2023, p. 71)

can perform two types of *scanning* – *passive* and *active scanning*. When *scanning* actively, the *observer* sends packets back to the *broadcaster* for example to request further information. Otherwise, the *observer* just listens passively for advertising packets.

In the state *synchronization* the device listens for periodic advertising packets by a particular device. Therefore, the *scanning* for packets is synchronized with the advertising events of the advertising device. These types of advertisements use 37 channels for the data transfer and are non-connectable. (*The Bluetooth Low Energy Primer*, 2023, p. 18 - 25)

2.2.3 The Generic Access Profile – the Roles

The GAP is the highest control layer and defines the roles of the device. These are the four roles that a communicating device can act as which were mentioned in the previous subsection: *broadcaster*, *observer*, *peripheral* and *central* (*The Bluetooth Low Energy Primer*, 2023, p. 71 - 72). These roles are further explained in Table 3.

2.2.4 The Attribute Protocol

The ATT is a client-server communication protocol. It is used by two connected devices for data exchange. Independently from the devices' roles after the connection process (*central* or *peripheral*) one device is called the *client* and the other is the *server*. The *server* contains data storing attributes the *client* can write to or read from depending on its permission when interacting with that attribute. The *client* can send read- and write-requests via packets to the *server*, which will send back a response, also in packet form (*The Bluetooth Low Energy Primer*, 2023, p. 61 - 63). Figure 8 represents a simple write request by the *client* to the *server* with the two possible outcomes success and failure.

Moreover, the *server* can notify the *client* for example when an attribute's value changes.

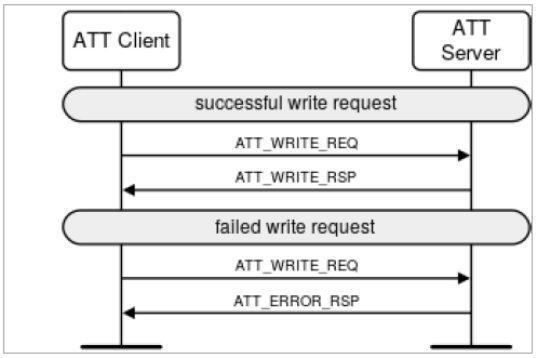

Figure 8: A write-request by the *client* (*The Bluetooth Low Energy Primer*, 2023, p. 63)

Figure 9: A notification by the *server* (*The Bluetooth Low Energy Primer*, 2023, p. 63)

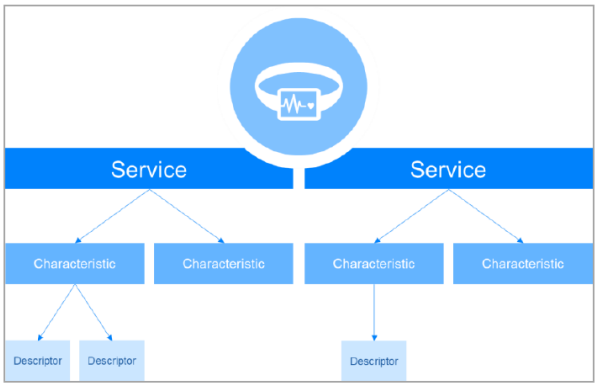

Figure 10: The hierarchic structure of the GATT (*The Bluetooth Low Energy Primer*, 2023, p. 67)

This is shown in Figure 9.

2.2.5 The Generic Attribute Profile

The GATT defines *services*, *characteristics* and *descriptors* based on the attributes of the ATT. A *service* groups characteristics and is identified by its Universally Unique Identifier (UUID). A *service* provides a context which is usually related to a capability of the device. A *characteristic* stores a certain type of value, is contained by at least one service and can contain descriptors. Also, permissions define how the user may interact with the value. A *descriptor* is contained by a *characteristic* and describes the *characteristic*. (*The Bluetooth Low Energy Primer*, 2023, p. 67)

Figure 10 summarizes the hierarchic structure of the GATT.

Advertising Interval in ms	Average current in mA
20	1.34
100	313.493
250	159.389
500	86.203
1000	55.291
10240	27.962

Advertising Data Length in Bytes	Average current in mA
0	1.438
15	2.264
19	1.677
22	1.860
31	2.161

Connection Interval in ms	Average current in uA
20	244.198
100	111.722
250	64.926
500	43.394
1000	37.217
4000	25.406

Peripheral Latency (Skipped Connection Events)	Average current in uA
0	69.968
9	29.968
18	24.345

Transmission Power Level in dBm	Average current in uA
8	140.67
4	110.598
0	121.504
-4	105.664
-8	106.244
-12	101.446
-16	99.89
-20	99.843
-40	99.719

Table 4: Average power consumption depending on the following parameters: *advertising interval* (the average current for an advertising interval of 20 ms seems to be an inconsistency), *advertising data length*, *connection interval* and *peripheral latency* (Jaimin, 2021)

2.3 Power Consumption of Bluetooth Low Energy – Influencing Parameters

This section shortly evaluates the BLE parameters which have an impact on the power consumption of the device. There are impacting parameters when advertising, when being in a connection and general radio parameters. Therefore, a study on the influence of these parameters on the power consumption was conducted by the company BuildStorm (Jaimin, 2021). The results are shown in Table 4.

2.3.1 Advertising Parameters

The *advertising interval* is the time between two consecutive advertising events. This parameter can be set from 20ms to 10240ms. The study by BuildStorm concludes: the lower the *advertising interval* the higher the power consumption.

The *advertising data length* is the second parameter and is limited to a maximum of 31 bytes. As expected, the power consumption rises – but only minimally – with a greater data length. (Jaimin, 2021)

2.3.2 Connection Parameters

The *connection interval* is the time between two consecutive connection events. This parameter can be set from 7.5 ms to 4 s. Here, the power consumption rises with a lower *connection*

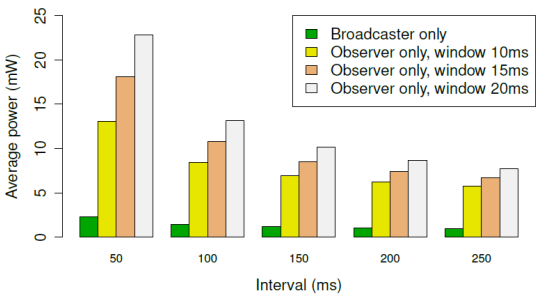

Figure 11: Average power consumption of broadcaster and observer with different advertising intervals, scanning intervals and scanning windows (Montanari et al., 2017)

Power Levels (l)	Transmission Power (ρ) (dBm)	Range (m)
0	10 dBm	200
1	4 dBm	70
2	0 dBm	50
3	−4 dBm	40
4	−8 dBm	30
5	−12 dBm	15
6	−16 dBm	7.0
7	−20 dBm	3.5
8	−40 dBm	1.0

Figure 12: Transmission power levels and their transmission ranges (Qureshi et al., 2018)

interval.

Another influential parameter is the *peripheral latency* which specifies the number of connection events that a *peripheral* is allowed to skip. The device may wake before the next connection event to participate in a transaction if it has something to exchange with the *central*, for example when it contains a GATT server and wants to notify the *central* about a value change of a *characteristic*. If the slave latency rises the power consumption decreases. (Jaimin, 2021)

2.3.3 Scanning Parameters

Influential scanning parameters are the *scanning interval* and the *scanning window*. The *scanning interval* is the time between two scans and the *scanning window* is the duration of a scan.

The influence of these parameters is not analyzed in the study of BuildStorm (Jaimin, 2021) but the power consumption of an *observer* and a *broadcaster* is measured in a study on human proximity detection using BLE (Montanari et al., 2017). These results are shown in Figure 11.

The measured power consumption is given in megawatt. The unity watt is the product of ampere and voltage. Under the assumption that both – *observer* and *broadcaster* – have the same voltage we can get the ratio between the ampere values of *observer* and *broadcaster*. With that ratio, the power consumption of an *observer* can be estimated.

2.3.4 Transmission Power Level

The *transmission power level* directly impacts the power consumption. Here applies: if the transmission power level rises, the the power consumption also rises but only minimally (Jaimin, 2021). Figure 12 shows the transmission ranges depending on the *transmission power level*.

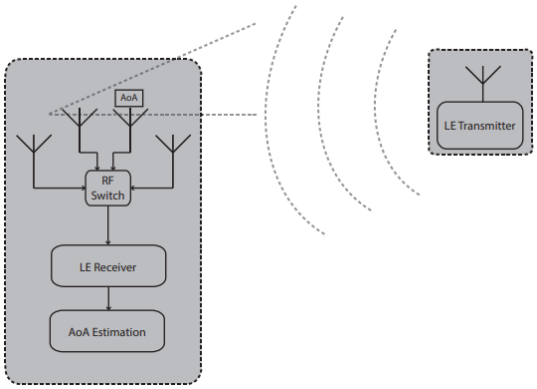

Figure 13: AoA (*Bluetooth Core Specification*, 2019, p. 281)

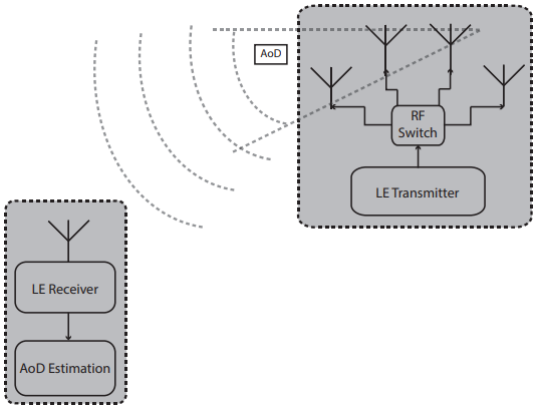

Figure 14: AoD (*Bluetooth Core Specification*, 2019, p. 283)

2.4 Direction Finding Using Bluetooth Low Energy

A device can calculate the direction of another device with two different methods: *Angle of Arrival (AoA)* and *Angle of Departure (AoD)*.

2.4.1 Angle of Arrival

The receiving device has an array of receiving antennas. By switching between these antennas during the receipt and calculating the phase differences of the packets at the different antennas the *AoA* can be estimated (*Bluetooth Core Specification*, 2019, p. 281). This is shown in Figure 13.

2.4.2 Angle of Departure

The transmitting device uses an array of antennas for transmission. The antennas switch during transmission. The receiving device uses a single receiver, calculates the phase differences of the receipt packets and estimates the *AoD* (*Bluetooth Core Specification*, 2019, p. 282 - 283). This is shown in Figure 14.

3 Related Work

This chapter covers related work of the HCI-G and other institutions in the context of user identification and BLE.

3.1 Previous Studies at the Human-Computer-Interaction Group

Between 2009 and 2010 students conducted research on user identification and authentication at the HCI-G using Radio-Frequency Identification (RFID) and fingerprint sensors. Two RFID readers were used – one for mid range and another one for long range reading – and each user was equipped with an individual signal sending RFID card for identification. Unfortunately, this turned out to be unreliable and not user-friendly as the card had to be aligned in a certain position towards the readers. (Ott, 2018, p. 433 - 437)

Besides, BLE as another possibility for user identification was already explored by members and students of the HCI-G. In the paper *Activity Support for Seniors Using Public Displays: A Proof of Concept* a system is designed and prototyped which consists of networked public displays to support senior users, particularly in the context of outdoor pedestrian movement (Fietkau & Stojko, 2021). Therefore, the user is equipped with a BLE broadcasting device whose MAC address was previously registered in the network. Each display acts as an *observer* which listens for advertisements and their MAC address. If a registered user is nearby a display, personalized data is displayed.

Also, Singh wrote a master thesis with the topic *Designing a Mobile Identification and User-Profile Solution for an Urban IoT Network* where he proposed a user identification and authentication system (Singh, 2018). Figure 15 shows the proposed architecture. In this context the smart urban object can be a CM, for example. The identification and authorization works via an extensive backend consisting of an authentication server, central server and a resource server. Singh proposes the use of BLE beacons – another name for a *broadcaster* – whose advertising packets are supposed to be received by the user who wishes to be identified. With this information the user is identified and authenticated centrally by the smart urban object they are closest to. Here, the objectives of this bachelor thesis differ. Authentication is not explored and the identification of the user is done decentrally only using BLE. Also, the use of the CM as a *broadcaster* is newly evaluated.

Moreover, Eisenlohr wrote his bachelor thesis about the concept and implementation of an application for identification using mobile devices and BLE (Eisenlohr, 2021) with the ultimate

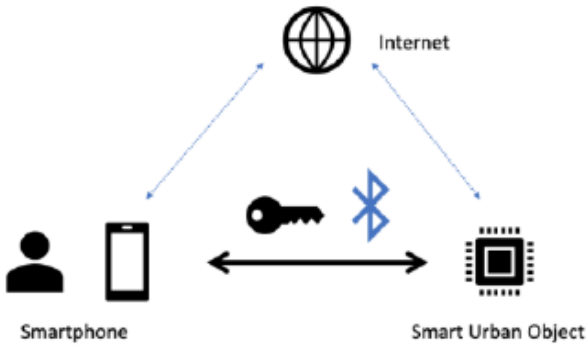

Figure 15: Singh's proposed architecture (Singh, 2018, p. 36)

goal to implement a working prototype for users to be identified by a CM – similarly to this bachelor thesis. Therefore, he chooses to assign the user the role of the *central* and the CM the role of the *peripheral* with the justification that the user should be the one to initialize the identification process. The CM implements a GATT server with a *service* for identification. The *service* contains three writeable characteristics, one for the user's name, another one for the user's password and one more to define the user's request. Here, the user has to specify if they wish to be registered in the IMS of the CM or if they are already registered and wish to be identified (Eisenlohr, 2021, p. 25). The CM sided logic is implemented as a separate application which is not integrated into the CMF. Thus, this bachelor thesis ultimately aims for the implementation of a prototype which can be used in production. This includes the integration of the CM sided logic into the CMF. Also, the assignment of the GAP roles is reevaluated more closely.

3.2 Other Studies

Other studies on user identification with BLE are rare to find. Current research mainly focuses on security issues regarding BLE and also its use for proximity detection, localization and how to improve its consistency in that use case.

There is one study on using BLE for the identification of small wooden boats (Saputra et al., 2022). Therefore, each boat is equipped with a BLE transmitter which is used as a *broadcaster*. The marine inspector at a port acts as an *observer* and scans for the packets. On receipt they filter the Bluetooth MAC Address of the *broadcaster* from the packet. They search for the Bluetooth MAC Address in a list and inspect the boat themselves. The result of the inspection together with the corresponding Bluetooth MAC Address is saved on a server.

As the Operating System (OS) of most smartphones and computers randomizes the BLE MAC address to improve privacy of users, there is a lot of research on digital fingerprinting for BLE as a mean of authentication but also to make aware of potential bypasses of the randomization and loss of privacy (Akiyama et al., 2021; Guillaume Celosia, 2019; Nilsson, 2022;

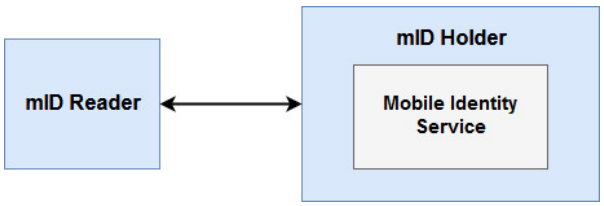

Figure 16: The identity reader and the identity holder containing the identity *service* (Sakkopoulos et al., 2019, p. 4)

Nilsson & Yan, 2021). This can be achieved through the exploitation of hardware imperfections, privacy issues in the GATT and by analysing the Received Signal Strength Indicator values of received advertising packet in static systems. With these methods classifications can be created to differ between the transmitting devices.

Another paper proposes an approach for exchanging mobile identity data between two devices with BLE (Sakkopoulos et al., 2019). It is differed between two roles: the identity reader and the identity holder. The holder is a *peripheral* with a GATT server containing a *service* for the identity. The reader is a *central*. The overall idea is visualized in Figure 16.

4 Concept

In this chapter the concept of the prototype is developed. The basic process which needs to be fulfilled by the prototype is the transmission of the user's personal identifier of their digital identity via BLE to the CM which links that identifier to the attributes of the digital identity and displays the attributes – the user's first name and last name. Also, the user should be able to control when the CM stops displaying the attributes – from now on called detachment. Therefore, the user requires wearable hardware that is able to perform BLE operations while consuming little power. Due to its complexity, direction finding using BLE is not conceptualized.

First, an IMS – managing the digital identities of the users – is conceptualized. After that, the usage of Bluetooth Low Energy is discussed by evaluating all different combinations of the device roles that are defined by the Generic Access Profile of the BLES and choosing one combination. Afterwards, the BLE related program flow of the chosen combination is created. Last but not least, the updated architecture is created.

4.1 Identity Management System

A system which manages the digital identities representing the users is required. Therefore, a database management system needs to be implemented that manages the digital identities. First of all, the digital identity representing a user has to be defined. For the prototype this digital identity is kept simple. It consists of the user's personal identifier, first name and last name. The Bluetooth MAC address of the user's device could be used as the personal identifier as it is done in related works (Fietkau & Stojko, 2021; Saputra et al., 2022). But this would limit the identification to a specific device and moreover many devices randomize the transmitted Bluetooth MAC address. Thus, a UUID is generated and used as the user's personal identifier for the prototype. A UUID is a 128 bits long universally unique identifier which is generated decentrally without any centralized registration process (Leach et al., 2005).

Secondly, a database schema needs to be defined. This is shown in an Entity-Relationship Diagram in Figure 17.

The database management system has to enable the CM to select the entry from the user table corresponding to the personal identifier of the user. Also, the user should be able to insert, update and delete the entry representing their digital identity. These are the required SQL commands:

- SELECT first_name, last_name FROM users WHERE user_id = '?';

Users	
PK	user_id VARCHAR(36)
	first_name VARCHAR(20) NOT NULL
	last_name VARCHAR(20) NOT NULL

Figure 17: The Entity-Relationship diagram for the IMS

- REPLACE INTO users (user_id, first_name, last_name) VALUES ('?', '?', '?');
- DELETE FROM users WHERE user_id = '?';

As all CMs and users need access to the IMS, the system has to run on a central server. The communication with that server can be realized with Representational State Transfer (REST).

4.2 Choosing the Bluetooth Low Energy Roles

Next, the usage of BLE has to be discussed. As already mentioned the GAP defines four possible roles of a device using BLE: *central*, *peripheral*, *observer* and *broadcaster*. The communicating devices for now are simply referred to as the user and the CM. These are the possible role combinations:

$$
\begin{array}{lcccc}
Role_{Device} & Central_{User} & Peripheral_{User} & Observer_{User} & Broadcaster_{User} \\
Central_{CM} & & C1 & & \\
Peripheral_{CM} & C3 & & & \\
Observer_{CM} & & & & C2 \\
Broadcaster_{CM} & & & C4 &
\end{array}
$$

Table 5 shows the chosen values of the BLE parameters for each combination and the estimated power consumption of each combination per user and CM.

Each combination is tested on its implementability. Therefore, two Node.js modules – *bleno* and *noble* – are used to test the essential CM sided functionality. Node.js[1] is an open-source, cross-platform JavaScript runtime environment. The module *bleno*[2] is used to implement a *peripheral* or a *broadcaster* and *noble*[3] to implement a *central* or *observer*. Another promising BLE Application Programming Interface (API) is contained by the Universal Windows Platform. The Universal Windows Platform[4] is an API by Microsoft to create Windows Apps.

[1] https://nodejs.org/en
[2] https://github.com/noble/bleno
[3] https://github.com/noble/noble
[4] https://learn.microsoft.com/en-us/windows/uwp/get-started/universal-application-platform-guide

	C1		C2		C3		C4	
	User	**CM**	**User**	**CM**	**User**	**CM**	**User**	**CM**
Advertising Interval in ms	100	-	100	-	-	100	-	100
Advertising Data Length in Bytes	16	-	16	-	16	-	-	16
Connection Interval in ms	20	20	-	-	20	20	-	-
Peripheral Latency (Number of Skipped Connection Events)	0	-	0	-	-	0	-	-
Scanning Interval in ms	-	100	-	100	100	-	100	-
Scanning Window in ms	-	15	-	15	15	-	15	-
Transmission Power Level in dBm	-16	-16	-16	-	-16	-16	-16	-16
Advertising/Scanning:Connection-Ratio	20:80	90:10	-	-	20:80	90:10	-	-
Average Power Consumption in mA	**63**	**1971**	**315**	**2190**	**438**	**283**	**2190**	**315**

Table 5: Power consumption of user and CM depending on the GAP roles (Estimated by using the data from 2 2.3)

Unfortunately, the API is very roughly documented[5] and therefore not used for this bachelor thesis. On the user side the Android Apps *BLE Peripheral Simulator*[6] and *BLE Scanner*[7] are used.

4.2.1 C1 – CommunityMirror as Central and User as Peripheral

The CM takes the role of the *central* and the user the role of the *peripheral*. Thus, the user constantly sends advertising packets which the CM has to scan for and connect to. This is shown in Figure 18. Once a user is connected the CM would require – in context of Figure 18 – three more instances of the Link Layer State Machine to enable the other users to identify. According to the mentioned paper on mobile personal information exchange over BLE (Sakkopoulos et al., 2019), the GATT server for identification should be implemented on the user side as the user is the identity holder and the CM the reader. The server would have to offer an identity *service* with a readable *characteristic* for their personal identifier, another one for their first name and one more for their last name. If the IMS does not contain an entry with that personal identifier, a new entry is inserted. Otherwise, if the user changed their attributes, the entry is updated. A detachment can be triggered by manually disconnecting the *peripheral* or when the *connection supervision interval* is exceeded, for example when the user moves out of range.

Noble is used to simulate the CM sided BLE logic.

```
const noble = require('noble');
noble.on('stateChange', (state) => {
    if (state === 'poweredOn')
        noble.startScanning([], true);
    else
```

[5] https://learn.microsoft.com/de-de/windows/uwp/devices-sensors/bluetooth-low-energy-overview
[6] https://play.google.com/store/apps/details?id=io.github.webbluetoothcg.bletestperipheral&hl=en&pli=1
[7] https://play.google.com/store/apps/details?id=com.macdom.ble.blescanner&hl=de&gl=US

```
        noble.stopScanning();
});
noble.on('discover', (peripheral) => peripheral.connect());
noble.on('connect', () => noble.startScanning([], true));
noble.on('disconnect', (peripheral) => {});
```

Once the Bluetooth adapter is powered on the scanning for advertisement packets is started. When a *peripheral* is discovered the CM connects to it. On a connection event *noble* stops the scanning by default. Thus, the scanning is manually started again. Unfortunately, *noble* does not support that and throws an error. Moreover, disconnect events are only fired when the CM itself triggers the disconnection and not when the *connection supervision interval* is exceeded. Thus, this combination is not suitable for the implementation of the prototype with *noble*.

4.2.2 C2 – CommunityMirror as Observer and User as Broadcaster

This combination is implemented in two of the related works (Fietkau & Stojko, 2021; Saputra et al., 2022). The CM acts as an *observer* and the user as a *broadcaster*. The user constantly sends advertising packets containing their personal identifier which the CM scans for. This is shown in Figure 18. When receiving a packet the CM identifies the user if they are registered in the IMS for example via REST. Therefore, the IMS is checked for the personal identifier and the user's first name and last name are displayed. When the CM receives no advertising packet from that user for some time afterwards the detachment is triggered. Simultaneous identifications are possible this way as no connections are formed but constant collisions could happen on the three advertising channels depending on the packet density. Practically, this should not be a problem when scanning for up to 50 advertising devices at least. When the advertising interval is perfectly configured it takes an observer only about 2.25 seconds to receive an advertising packet from each of the 50 devices (Shan et al., 2016).

The implementability of this combination is already indirectly tested in 4.2.1 where the discovery of *peripherals* worked fine.

4.2.3 C3 – CommunityMirror as Peripheral and User as Central

This combination is implemented in Eisenlohr's bachelor thesis for one identification at a time (Eisenlohr, 2021). The user acts as the *central* and the CM acts as the *peripheral*. Now, the CM sends the advertising packets and the user scans for them. This is visualized in Figure 19. As in C1, the user contains a GATT server with the identification service which the CM can read from. Simultaneous identifications again require multiple implementations of the Link Layer State Machine. The detachment can again be triggered manually by the user or by exceeding the *connection supervision interval*.

Bleno is used to simulate the CM sided logic.

```
const bleno = require('bleno');
```

Figure 18: Bluetooth Low Energy Usage – Combination 1 / 2

```
bleno.on('stateChange', (state) => {
    if (state === 'poweredOn')
        bleno.startAdvertising(...);
    else
        bleno.stopAdvertising();
});
bleno.on('accept', () => bleno.startAdvertising(...));
bleno.on('disconnect', (peripheral) => {});
```

Once the device is powered on the *bleno* starts advertising. When *bleno* accepts the connection request of a *central* the advertising is stopped and needs to be manually started again. Unfortunately, this is not supported and causes an error. Also, the disconnection event is only fired when *bleno* itself triggers the disconnection from the *client*. So exceeding the *connection supervision interval* can not be detected. Thus, this combination is not implementable with *bleno*.

As a side note, *bleno* interprets the peripheral role also as the role of the GATT server which means the identification *service* would need to be implemented by the CM which was also done by Eisenlohr (Eisenlohr, 2021). This would require additional logic to handle simultaneous requests by users to that server.

4.2.4 C4 – CommunityMirror as Broadcaster and User as Observer

This combination is mentioned in Singh's master thesis, but in the context of proximity measurement (Singh, 2018). The user acts as the *observer* and the CM as the *broadcaster*. Thus,

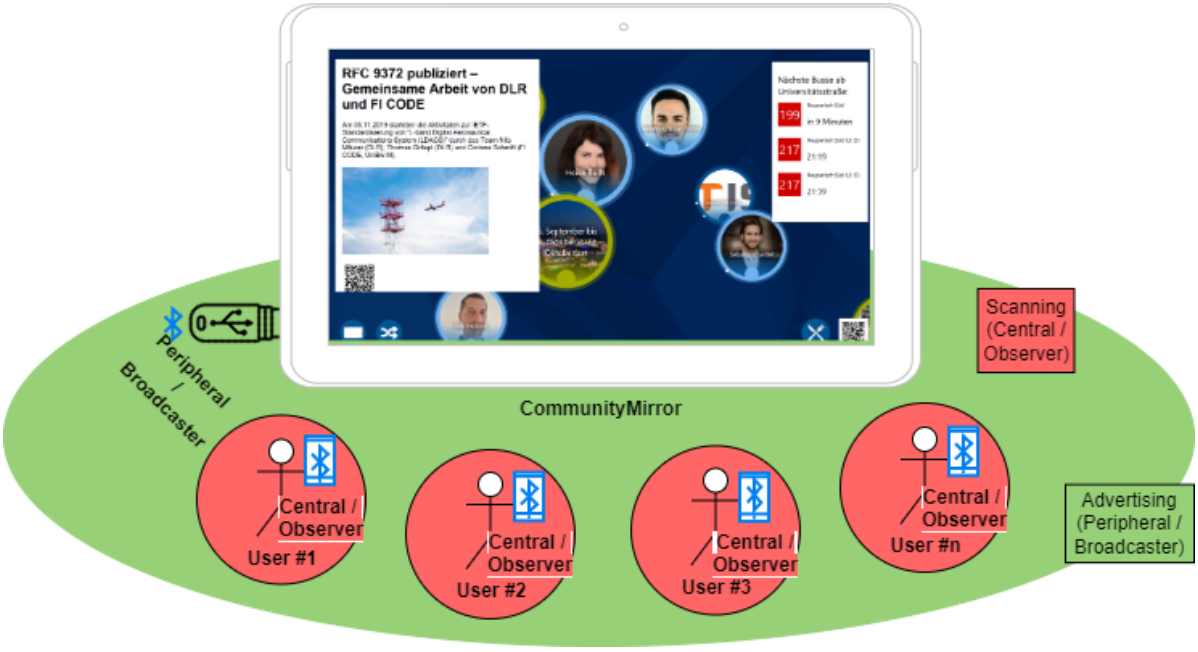

Figure 19: Bluetooth Low Energy Usage – Combination 3 / 4

the user listens for advertising packets transmitted by the CM. To send back a packet containing the personal identifier the user would have to perform *active scanning*. Unfortunately, the detachment is difficult this way. Firstly, the user could trigger the detachment manually and send another response packet to the CM. Secondly, if the user forgets to do so and moves out of the receival range for the CM's advertising packets they can not trigger a detachment via BLE anymore. Another form of communication for example via REST would be necessary. Simultaneous identifications would be possible but depend on signal density and collisions when sending the packet with the personal identifier. This is also visualized in Figure 19.

Bleno provides functionality to implement a *broadcaster*, but none to react to answering packets of *observers* that perform *active scanning*.

```
const bleno = require('bleno');
bleno.startAdvertisingIBeacon (...);
```

Unfortunately, none of the advertising packets can be received by the *BLE Scanner*.

4.2.5 Comparison and Selection

Last but not least, one of the four combinations has to be chosen for the next chapter on the prototype's implementation. Therefore, Table 6 compares the different combinations.

The implementability is fundamental for the implementation. Only C2 meets that requirement which makes the choice trivial. Nevertheless, C2 is not the optimal solution as it consumes way more power than C3, provides only limited bidirectional communication, no encrypted

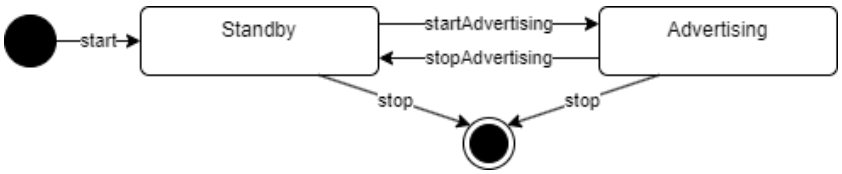

Figure 20: User – State pattern for broadcasting

communication and another form of communication has to be used for the registration in the IMS.

4.3 User – Broadcaster

The user acts as a *broadcaster*. Therefore, a device with a Bluetooth transmitter – running the advertising script – needs to be chosen.

4.3.1 Choosing the Broadcasting Device and Operating System

It is most handy for the user if they do not have to use an additional device. Thus, the smartphone seems to be the best choice as approximately 86 percent[8] of the world's population own a smartphone. Most importantly, modern smartphones implement the modules of the BLES and provide the necessary hardware.

The mainly used OS are Android by Google – used by about 69 percent of all smartphone users – and iOS by Apple – used by around 30 percent.[9] The Flutter[10] Development Kit could be used for the development of an application which runs on both OS. Unfortunately, Flutter does not have an API or library[11] to use the smartphone as a *broadcaster*. Thus, Android is chosen as the OS which runs the advertising script.

4.3.2 Advertising Script

The advertising script is executed according to a state pattern depending on which the user advertises data or is in standby. The user has full control over the state changes. Figure 20 shows the state diagram.

4.3.3 Android Application

The advertising script is implemented within an Android application. Next to implementing the state pattern, the application should enable the user to write their digital identity to the IMS, edit it and delete it.

[8] https://www.bankmycell.com/blog/how-many-phones-are-in-the-world
[9] https://gs.statcounter.com/os-market-share/mobile/worldwide
[10] https://flutter.dev/
[11] https://leancode.co/blog/bluetooth-low-energy-in-flutter

	Simultaneous identifications	Power consumption of the CM	Power consumption of the user	Communication only via BLE	Bidirectional communication	Encrypted communication	Hardware requirements	Software requirements	Implementable with *bleno* or *noble*
C1	Depending on the amount of Link Layer State Machine instances	197 1mA	63 mA	Yes	Yes	Yes	CM: receiver and transmitter with a controller that supports multiple instances of the Link Layer State Machine; user: receiver and transmitter	CM: software stack which supports multiple instances of the Link Layer State Machine; user: basic software stack	No
C2	At least 50 simultaneous identifications possible: depends on finding the best advertising interval to avoid packet collisions	2190 mA	315 mA	No, registration in IMS can not be done over BLE	Limited: the CM could perform *active scanning*	No	CM: receiver; user: transmitter	Basic software stack	Yes
C3	Depending on the amount of Link Layer State Machine instances	283 mA	438 mA	Yes	Yes	Yes	User: receiver and transmitter with a controller that supports multiple instances of the Link Layer State Machine; CM: receiver and transmitter	CM: software stack which supports multiple instances of the Link Layer State Machine; user: basic software stack	No
C4	Unlimited	315 mA	2190 mA	No, detachment can not be done when user moves out of range	Limited: the user performs *active scanning*	No	CM: receiver; user: transmitter	Basic software stack	No

Table 6: BLE usage – Comparison of the combinations

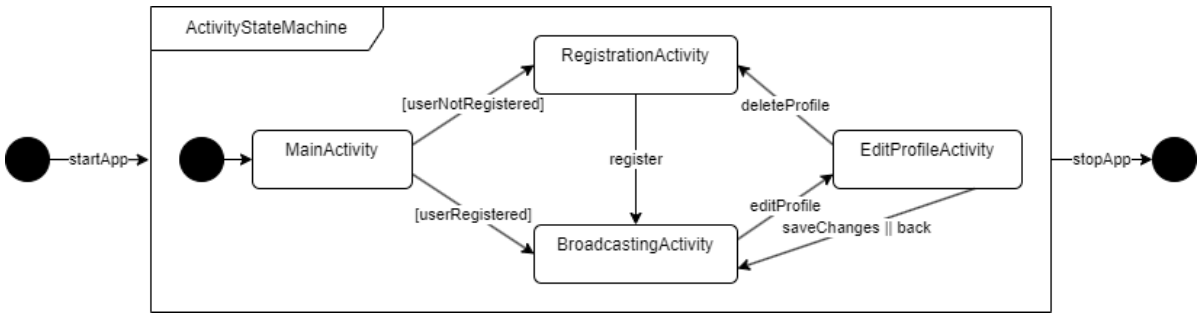

Figure 21: State diagram representing the activities and their flow

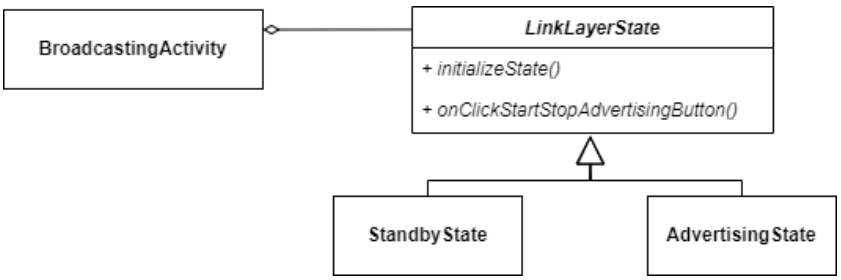

Figure 22: State class diagram executing the advertising script

Like JavaFX uses the class *Scene* to manage a scene graph, Android provides the class *Activity*[12] to control the UI elements. The state diagram in Figure 21 shows the different required activities and their flow.

The *MainActivity* is the starting point of the app. From here it is decided whether to switch to the *RegistrationActivity* or the *BroadcastingActivity* where the user controls the advertising script. Also, the user can switch from the *BroadcastingActivity* to the *EditProfileActivity* to edit or delete their profile, depending on which they return or switch to the *RegistrationActivity*.

Moreover, the state pattern described in the previous subsection is conceptualized within the following class diagram in Figure 22.

The *BroadcastingActivity* is the context of the abstract class *LinkLayerState*. The subclasses of the *LinkLayerState* have to implement the methods *onClickStartStopAdvertisingButton()* – which is called from the context when the user wants to switch states – and *initializeState()* – which is called after a state switch.

The class diagram of the entire Android application can be seen in Figure 23.

The different activities extend the abstract class *AppCompatActivity*[13]. Moreover, the UI of the activities is shown. Additionally, the class *UserRepository* manages the communication with the REST API and is used by the *RegistrationActivity* and the *EditProfileActivity*. Moreover, the *LinkLayerState* has a reference to the *BluetoothAdapter*[14] of the device provided by the Android

[12] `https://developer.android.com/reference/android/app/Activity`
[13] `https://developer.android.com/reference/androidx/appcompat/app/AppCompatActivity`
[14] `https://developer.android.com/reference/android/bluetooth/BluetoothAdapter`

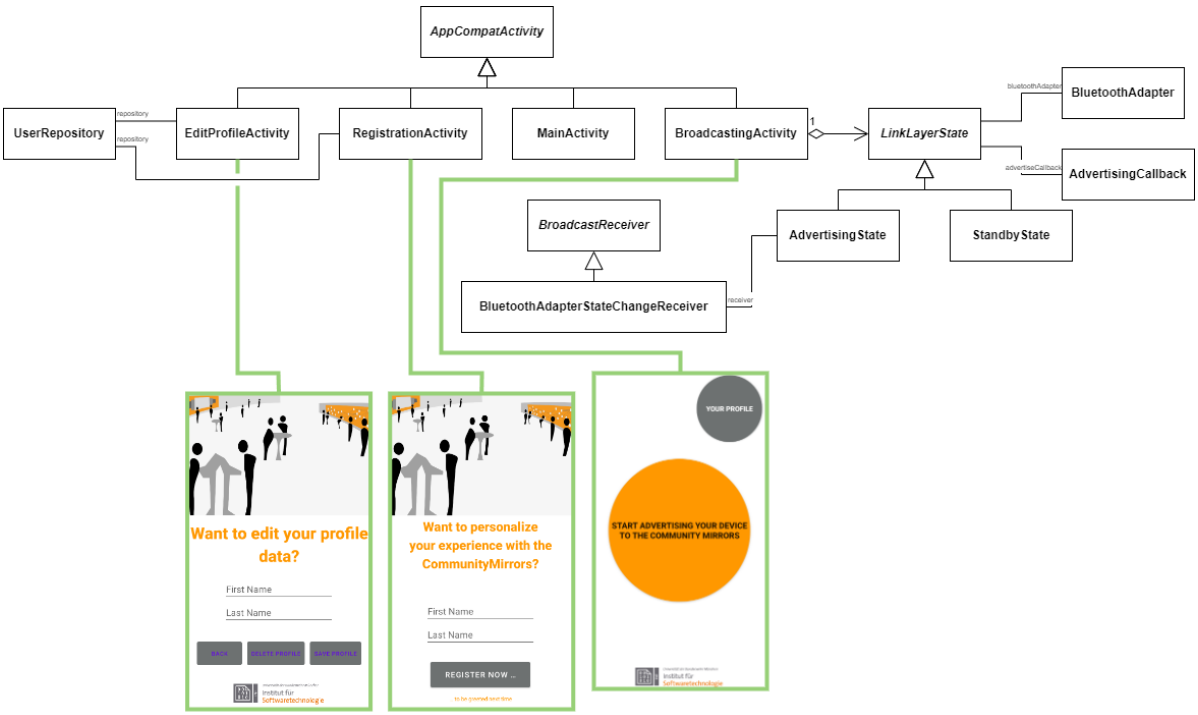

Figure 23: Class diagram of Android application

API. Also, a reference to the *AdvertisingCallback*[15] is required to start and stop advertising. Last but not least, the *AdvertisingState* registers a *BluetoothAdapterStateChangeReceiver*[16] which listens for the state of the *BluetoothAdapter*. If the *BluetoothAdapter* is disabled by the user, the *StandbyState* is initialized.

4.4 CommunityMirror – Observer

As mentioned before, the CM acts as an *observer*. Therefore, the CM requires a scanning script which needs to be integrated into the CMF.

4.4.1 Scanning Script

The CM scans for advertising packets. Therefore, the following diagram in Figure 24 – a mixture of a state diagram and an activity diagram – will be implemented. In the state scanning the CM constantly scans for advertising packets. Once a packet is received the packet is checked for a personal identifier. If no personal identifier could be found in the packet the CM simply continues scanning. Otherwise, it is checked if a user with that personal identifier is already identified. If not, the IMS is searched for the user's entry and their name is depicted on

[15] `https://developer.android.com/reference/android/bluetooth/le/AdvertiseCallback`
[16] https://developer.android.com/reference/android/content/BroadcastReceiver

the display of the CM. Also, the user is added to the identified users. The user's personal disconnection timer is reset if they were already identified before. The user is detached if the timer runs out.

4.4.2 Integration into the CommunityMirror Framework

The CMF needs to be expanded which can be seen in Figure 25. The class *UsersComponent* displays the identified users on the display. Also, it controls the execution of the scanning script with the class *UsersScanningProcess*. Both these classes have a reference to the class *UsersController* where the identified users are added to the visualized data.

4.5 Updated Architecture

Figure 26 shows the updated architecture of the CMs. The IMS, the *observers* and the *broadcasters* are added.

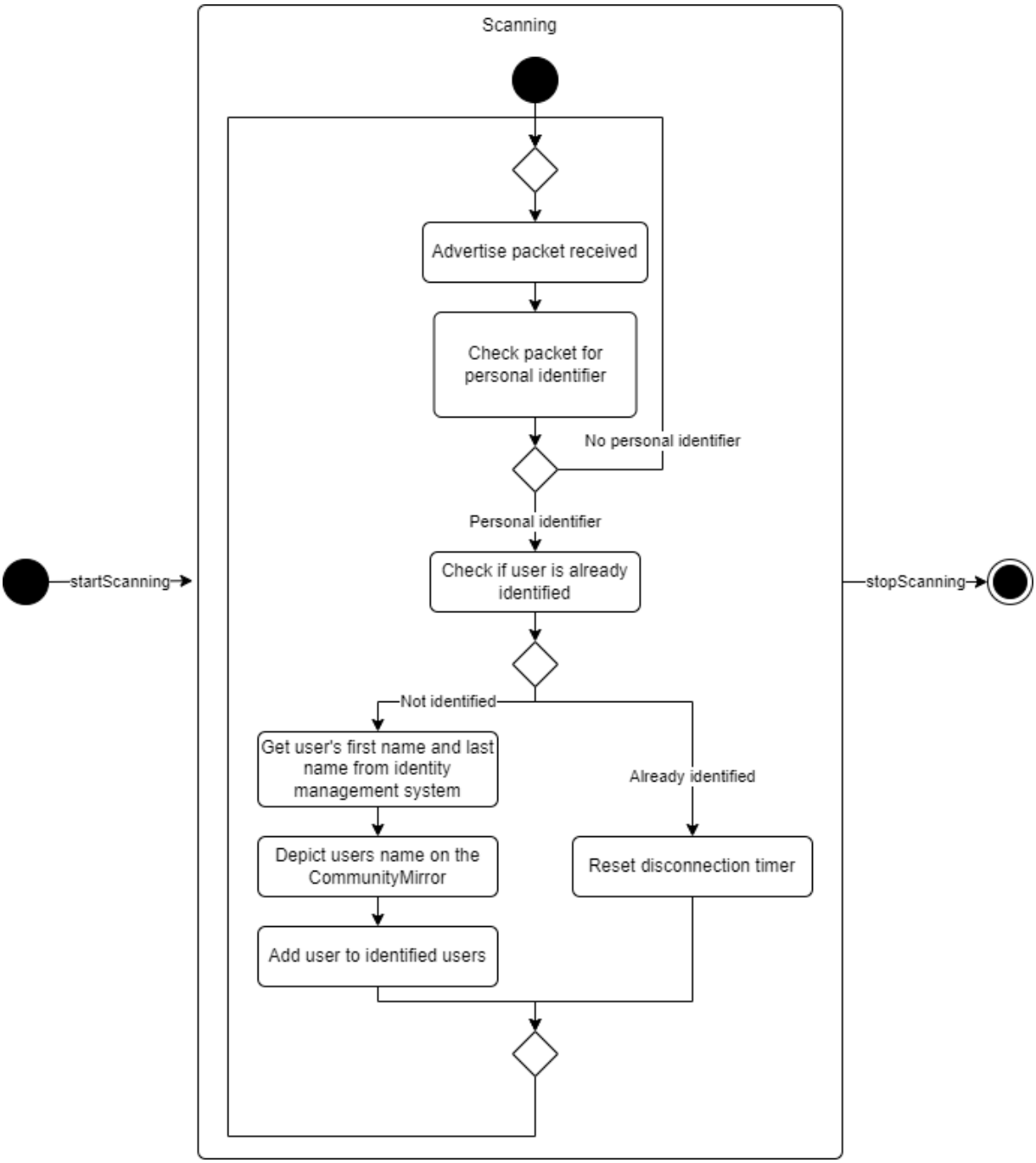

Figure 24: Scanning script

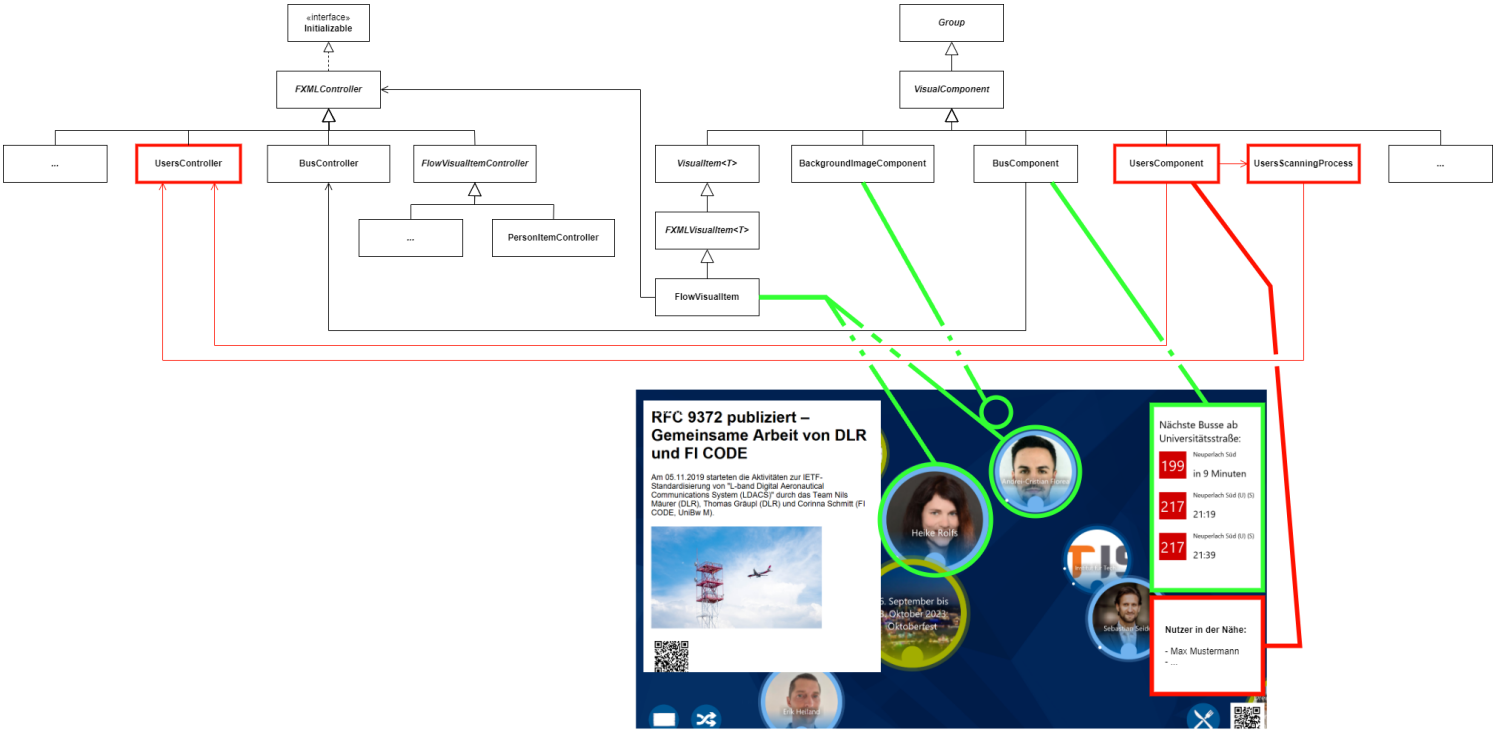

Figure 25: Updated class diagram of the CMF responsible for visualization and data control

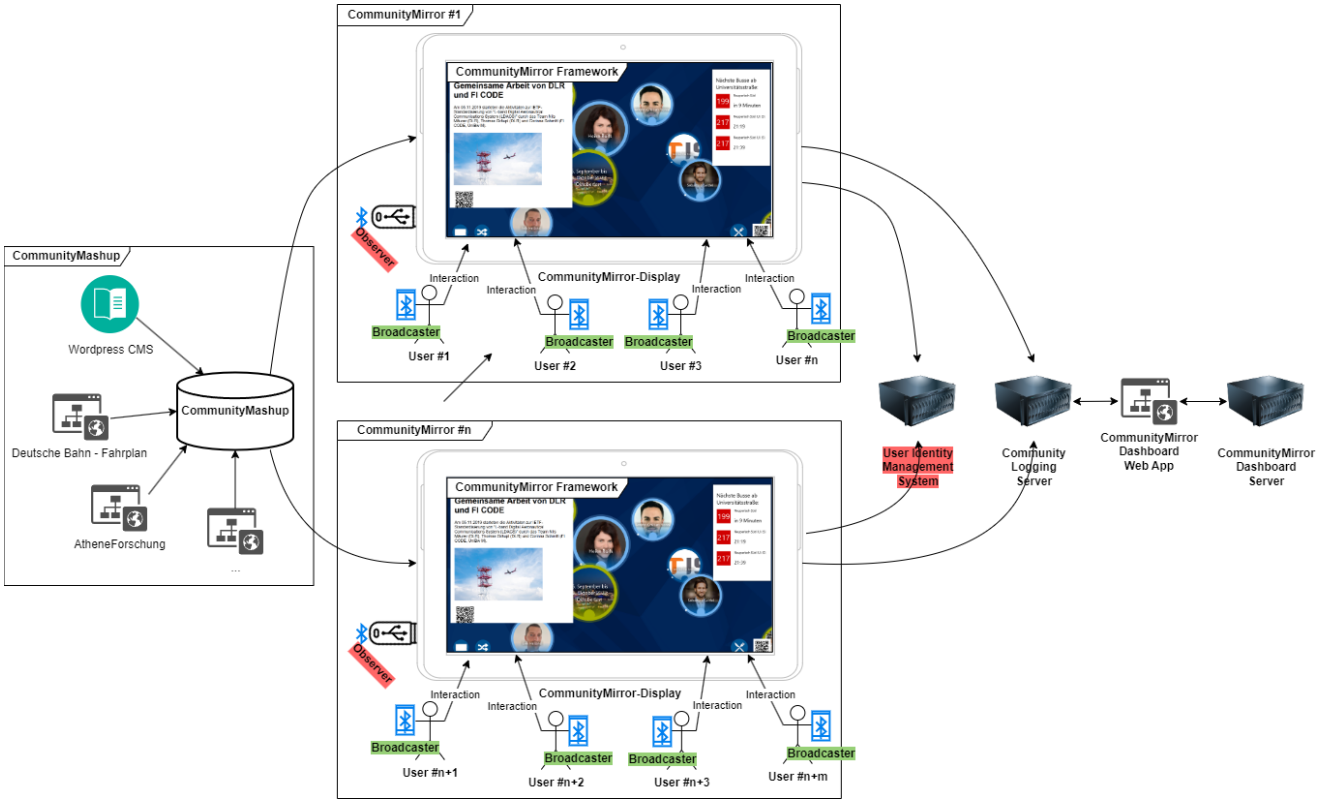

Figure 26: Updated architecture of the CMs

5 Implementation

This chapter covers the implementation of the concept. It starts with the implementation of the IMS, is followed by the implementation of the user sided BLE logic and ends with the implementation of the CM sided BLE logic. Different packages from the Node.js[1] package manager are used. Also, an application is developed for the user running on Android.

The source code can be found under `https://athene2.informatik.unibw-muenchen.de/user-recognition-for-communitymirrors`.

5.1 Identity Management System

The digital identities are stored in a SQLite database and can be accessed and manipulated via REST.

5.1.1 SQLite Database

SQLite is a low scale database engine that stores the data in a single file. The package *sqlite3*[2] is used to allow bindings to the database from the Node.js script.

```
const sqlite3 = require('sqlite3');
const database = new sqlite3.Database('./users.db', sqlite3.OPEN_READWRITE, ...);
database.run('CREATE TABLE IF NOT EXISTS users (uuid VARCHAR(40) PRIMARY KEY,
    firstName VARCHAR(20) NOT NULL, lastName VARCHAR(20) NOT NULL)');
```

5.1.2 REST API

The package *express*[3] is used to create a REST API to remotely access and manipulate the data in the database.

In total, three endpoints are defined, one for posting a digital identity, another for getting a digital identity and the last one for deleting a digital identity:

```
app.post('/users/:uuid', (request, response) => {...});
app.get('/users/:uuid', (request, response) => {...});
app.delete('/users/:uuid', (request, response) => {...});
```

[1] `https://nodejs.org/en`
[2] `https://www.npmjs.com/package/sqlite3`
[3] `https://www.npmjs.com/package/express`

5.2 User – Broadcaster Advertising Script

This section focuses on the implementation of the advertising script. The following listing shows parts of the code of the *BroadcastingActivity*:

```
class BroadcastingActivity : AppCompatActivity() {
    private lateinit var currentLinkLayerState: LinkLayerState
    lateinit var startStopAdvertisingButton: Button
    ...
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_ble_advertisement)

        val bluetoothAdapter = getBluetoothAdapter()
        val advertisingCallback = AdvertisingCallback()
        currentLinkLayerState = StandbyState(this, advertisingCallback, bluetoothAdapter)
        ...
        startStopAdvertisingButton = findViewById(R.id.startStopAdvertisingButton)
        startStopAdvertisingButton.setOnClickListener {
            currentLinkLayerState.onClickStartStopAdvertisingButton()
        }
        ...
    }

    ...
    fun setAdvertisementState(linkLayerState: LinkLayerState) {
        currentLinkLayerState = linkLayerState
        currentLinkLayerState. initializeState ()
    }
    ...
}
```

The class is the context of the state pattern and therefore contains the *currentLinkLayerState*. Also, it provides functionality to switch the state. A listener is set to the *startStopAdvertising-Button* to execute the functionality provided by the *currentLinkLayerState* when the user taps the button.

The following listing shows parts of the code of the class *AdvertisingState*:

```
class AdvertisingState(broadcastingActivity: BroadcastingActivity,
                       advertiseCallback: AdvertiseCallback,
                       bluetoothAdapter: BluetoothAdapter)
    : LinkLayerState(broadcastingActivity, advertiseCallback, bluetoothAdapter) {
    ...
    override fun initializeState () {
        ...
        val advertiseSettings: AdvertiseSettings = AdvertiseSettings.Builder()
        .setAdvertiseMode(AdvertiseSettings.ADVERTISE_MODE_BALANCED)
            .setConnectable(false)
```

```
            .setTimeout(0)
            .setTxPowerLevel(AdvertiseSettings.ADVERTISE_TX_POWER_MEDIUM).build()

        val advertiseData: AdvertiseData = AdvertiseData.Builder()
            .setIncludeDeviceName(false)
            .setIncludeTxPowerLevel(false)
            .addServiceUuid(/* user's personal identifier */)
            .build()
        ...
        bluetoothAdapter.bluetoothLeAdvertiser.startAdvertising(
                advertiseSettings,
                advertiseData,
                advertiseCallback
        )
        ...
    }
    ...
}
```

Here, the *AdvertiseSettings*[4] and *AdvertiseData*[5] are defined and the advertising is started by the *BluetoothLeAdvertiser*[6].

The *AdvertiseMode* defines the *advertising interval*. Unfortunately, the documentation contains no information about the concrete values of the *advertising intervals* of the different *AdvertiseModes*. Here, it is set to *ADVERTISE_MODE_BALANCED*.

The *TxPowerLevel* is set to *ADVERTISE_TX_POWER_MEDIUM* whose dBm value is also not mentioned in the documentation.

Most importantly, the user's personal identifier is added to the advertised services.

The following listing shows parts of the code of the class *StandbyState*:

```
class StandbyState(broadcastingActivity: BroadcastingActivity,
                   advertiseCallback: AdvertiseCallback,
                   bluetoothAdapter: BluetoothAdapter)
    : LinkLayerState(broadcastingActivity, advertiseCallback, bluetoothAdapter) {

    override fun initializeState() {
        ...
        bluetoothAdapter.bluetoothLeAdvertiser.stopAdvertising(advertiseCallback)
        ...
    }
    ...
}
```

The function *initializeState* stops the advertising.

[4] https://developer.android.com/reference/android/bluetooth/le/AdvertiseSettings
[5] https://developer.android.com/reference/android/bluetooth/le/AdvertiseData
[6] https://developer.android.com/reference/android/bluetooth/le/BluetoothLeAdvertiser

5.3 CommunityMirror – Observer

This section analyzes the implementation of the observer script and its integration into the CMF.

5.3.1 Scanning Script

The scanning script is implemented using the Node.js package *noble*[7].

Two lists store the *identifiedUsers* and the *currentlyDiscoveredPeripheralIds* which contain the devices whose advertising packets are currently analyzed. Also, the class *User* – representing an identified user – is defined:

```
let identifiedUsers = [];
let currentlyDiscoveredPeripheralIds = [];

class User {
    uuid;
    intervalId;

    constructor(uuid) {
        this.uuid = uuid;
        this.intervalId = this.setDisconnectionInterval();
    }

    resetDisconnectionInterval() {
        clearInterval(this.intervalId);
        this.intervalId = this.setDisconnectionInterval();
    }

    setDisconnectionInterval() {
        return setInterval(() => {
            identifiedUsers = identifiedUsers.filter(user => user.uuid !== this.uuid);
            console.log('DISCONNECTED USER: ', this.uuid);
            clearInterval(this.intervalId);
        }, 2000);
    }
}
```

The *User* consists of an *uuid* – representing their personal identifier – and an *intervalId* – referencing their disconnection timer. Once the *User* is created the disconnection timer is started and set to two seconds. The timer can be reset with the function *resetDisconnectionInterval()*. If the timer runs out the *User* is filtered from the *identifiedUsers* and the disconnection is logged on the console.

This is the executed code when an advertising packet is received:

[7] https://github.com/noble/noble

```
noble.on('discover', (peripheral) => {
    const serviceUuids = peripheral.advertisement.serviceUuids;

    if (serviceUuids.length > 0 && !isCurrentlyDiscoveredPeripheral(serviceUuids)) {
        currentlyDiscoveredPeripherals.push(serviceUuids);

        for (let index = 0; index < serviceUuids.length; index++) {
            const peripheralServiceUuid = serviceUuids[index];
            const alreadyConnectedUser = identifiedUsers.find(user => user.uuid === peripheralServiceUuid);

            if (alreadyConnectedUser) {
                alreadyConnectedUser.resetDisconnectionInterval();
                currentlyDiscoveredPeripherals =
                    currentlyDiscoveredPeripherals.filter(services => services !== serviceUuids);
                return;

            } else {
                https.get(..., response => {
                    if (response.statusCode === 200) {
                        identifiedUsers.push(new User(peripheralServiceUuid));
                        console.log('CONNECTED USER: ', peripheralServiceUuid);
                        currentlyDiscoveredPeripherals =
                            currentlyDiscoveredPeripherals.filter(services => services !== serviceUuids);
                    }

                }).on("error", (e) => {
                    console.error(e);
                    if (index === peripheral.advertisement.serviceUuids.length - 1)
                        currentlyDiscoveredPeripherals =
                            currentlyDiscoveredPeripherals.filter(id => id !== peripheral.id);
                });
            }
        }
    }
});
```

It is checked first if the *peripheral* is currently already discovered to prevent the simultaneous execution of the observer script on the same *peripheral*. If not, it is added to the *currentlyDiscoveredPeripheralIds*. Now, the advertised service UUIDs are looped through and it is checked if a user with that personal identifier is connected. If so, their disconnection timer is reset. Otherwise, it is checked if a user with that personal identifier exists in the IMS. If that is the case, the newly connected user is logged on the console.

5.3.2 Integration into the CommunityMirror Framework

Next, the script needs to be integrated into the CMF.

The *UsersScanningProcess* is responsible for executing the scanning script on another thread with the class *ProcessBuilder*[8]. A *BufferedReader*[9] listens to the input stream of the process and thus receives the logs about connected and disconnected users. When a log is received a user is either added or removed from the *users* list in the *UsersController*. Also, the class provides functionality to stop the execution of the process.

```
public class UsersScanningProcess {
    private final UsersController usersController;
    private Process usersScanningProcessRef;

    public UsersScanningProcess(UsersController usersController) {
        this.usersController = usersController;
        this.startProcess();
    }

    private void startProcess() {
        final Thread usersScanningProcessThread = new Thread(() -> {
            final ProcessBuilder processBuilderUsersScanning =
                new ProcessBuilder("node", /* path to scanning script */);
            ...
            usersScanningProcessRef = processBuilderUsersScanning.start();
            Runtime.getRuntime().addShutdownHook(new Thread(usersScanningProcessRef::destroy));

            final BufferedReader bufferedReader =
                new BufferedReader(new InputStreamReader(usersScanningProcessRef.getInputStream()));
            String line;

            while ((line = bufferedReader.readLine()) != null) {
                if (line.startsWith("CONNECTED USER: ")) {
                    this.usersController.addUser(...);
                } else {
                    this.usersController.removeUser(...);
                }
            }
            ...
        });

        usersScanningProcessThread.start();
    }

    public void destroyProcess() {
        usersScanningProcessRef.destroy();
    }
}
```

[8] https://docs.oracle.com/javase/8/docs/api/java/lang/ProcessBuilder.html
[9] https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html

6 Evaluation

In this chapter the prototype – consisting of the IMS, the scanning script and its integration into the CMF and the user's Android App – is evaluated. Therefore, the evaluation process and its results are described. Also, the overall security of the prototype is evaluated.

6.1 Evaluation Process

The IMS runs on a server of the HCI-G which was configured with the help of Dr. Julian Fietkau the advisor of this bachelor thesis.

The CM is simulated by a Microsoft Surface Pro 6 running Windows 10 Pro Version 21H2. It is additionally equipped with the USB-BT400 Bluetooth Adapter by Asus. The surface runs the CMF with the integrated scanning script.

The users are simulated by three smartphones – see Table 7 – which run the Android application.

Figure 27 shows the floor plan of the test room. Six points are selected in the room from where it is tried with all smartphones to identify in three different scenarios:

- *Scenario 1*: No other user is identified
- *Scenario 2*: One other user is identified
- *Scenario 3*: Two other users are identified

Each smartphone sends the personal identifier for an interval of 30 seconds. In all these scenarios the *identification speed* is measured manually with a stopwatch and the *number of untriggered detachments* is documented.

6.2 Results

Table 8 shows the results of the evaluation.

	Smartphone-Model	**Android-Version**	**Android-API-Level**
Device A	Samsung Galaxy A52	13	33
Device B	Samsung Galaxy A32	13	33
Device C	Google Pixel 4a	13	33

Table 7: Technical specifications of the smartphones that simulate the users

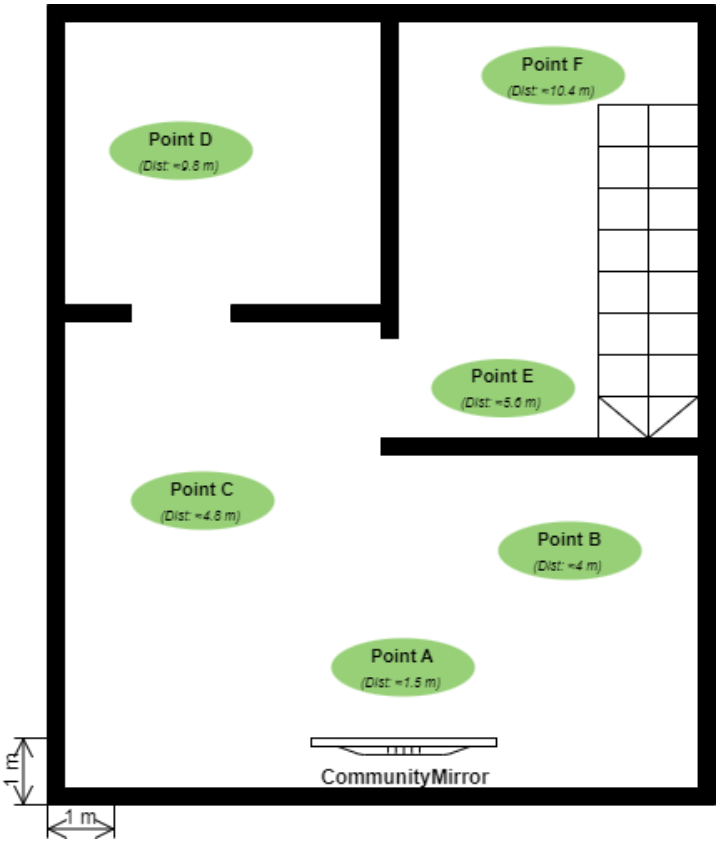

Figure 27: Floor plan of the test room with the position of the CM and the different points from where the user tries to be identified

Scenario 1																							
Device	**A**						**B**						**C**										
Point	**A**	**B**	**C**	**D**	**E**	**F**	**A**	**B**	**C**	**D**	**E**	**F**	**A**	**B**	**C**	**D**	**E**	**F**					
Identification Speed in sec	0.68	0.61	0.79	0.87	0.85	1.44	0.74	0.75	0.67	1.4	1.8	1.47	0.66	2.63	2.54	-	-	-	**1.12 (Avg)**				
Number of Untriggered Detach-ments	0	0	0	0	0	0	0	0	0	0	1	3	0	3	2	-	-	-	**0.5 (Avg)**				

Scenario 2																							
Device	**A**						**B**						**C**										
Point	**A**	**B**	**C**	**D**	**E**	**F**	**A**	**B**	**C**	**D**	**E**	**F**	**A**	**B**	**C**	**D**	**E**	**F**					
Identification Speed in sec	0.80	1.00	0.86	0.87	1.07	1.00	0.95	1.14	0.81	1.28	1.34	1.54	0.81	0.88	0.94	-	-	-	**1.01 (Avg)**				
Number of Untriggered Detach-ments	0	0	0	0	0	0	0	0	0	0	0	3	0	0	1	-	-	-	**0.26 (Avg)**				

Scenario 3																							
Device	**A**						**B**						**C**										
Point	**A**	**B**	**C**	**D**	**E**	**F**	**A**	**B**	**C**	**D**	**E**	**F**	**A**	**B**	**C**	**D**	**E**	**F**					
Identification Speed in sec	1.20	0.74	1.07	0.81	0.87	0.87	0.80	0.94	1.35	1.27	1.14	1.14	1.21	0.81	0.94	-	-	-	**0.93 (Avg)**				
Number of Untriggered Detach-ments	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	**0 (Avg)**				

Table 8: The results of the evaluation ("-" means that no measurements could be taken as the personal identifier could not be received)

What can be seen directly is that the personal identifier sent by device C from points D, E and F could not be received by the CM in all three scenarios. The reason for that is most likely low performing Bluetooth hardware. This might also explain the amount of untriggered detachments of device C from the three nearest points A, B and C.

The overall identification speed is mostly lower at the points A, B and C and higher at the points D, E and F, most likely due to the higher distance between these points and the CM. Also, there is a correlation between distance and the amount of untriggered detachments.

Testing in different scenarios did not have a huge impact on the measurements. But surprisingly the average identification speed and the average number of untriggered detachments declined slightly. Here the opposite behaviour would have been expected as there might be signal collisions on the three advertising channels. Obviously, collisions did not seem to have a manually measurable impact for three devices and the slight decline might be due to inconsistencies in the manual measurements.

During the evaluation the scanning script crashed approximately every 10 minutes. A restart was possible after invalidating the caches of the extended CMF using IntelliJ. The cause of the problem could not be found but most likely there is a problem with the used BLE API *noble* which was last updated in 2018.

6.3 The Security of the Prototype

Security is not taken into account during the development of the prototype. Nevertheless, it should be analyzed in case a secure version of the prototype will be implemented in the future.

The security issue of the prototype is the unencrypted and undirected transmission of the user's personal identifier during the identification. Spoofers can simply read the user's advertised personal identifier and misuse it to identify in the name of that user. The bachelor thesis of Teaca on the *Design of an encryption protocol for BLE advertising traffic* might be a good starting point for fixing that problem (Teaca, 2019).

Also, as the REST endpoints are not secured, spoofers can call these endpoints if they know the URL of the server which runs the IMS. With this information the user's digital identity can be manipulated and even deleted.

Last but not least, the REST endpoint for the insertion of a new user can be misused by flooding it with insertions which could result in a denial of service.

7 Conclusion

The HCI-G strives for a practical solution to identify interacting users of the CMs which are ubiquitious user interfaces for community awareness. Therefore, this bachelor thesis discusses the use of BLE for user identification in the context of the CMs and develops a prototype without focusing on authentication and security overall.

The context of the prototype, meaning the software of the CMs, the overall architecture of the CMs and the fundamentals of BLE – the BLES, the power consumption of BLE and direction finding using BLE – are discussed. Related work by the HCI-G and other institutions is analyzed and used as a foundation to conceptualize a prototype. The four different combinations to use BLE are elaborated from analyzing the GAP roles of the BLES and compared:

- C1: CM as *central* and user as *peripheral*
- C2: CM as *observer* and user as *broadcaster*
- C3: CM as *peripheral* and user as *central*
- C4: CM as *broadcaster* and user as *observer*

C1 and C3 turn out to be the most efficient combinations but unfortunately neither can be chosen for the implementation due to a lack of maintained and powerful BLE APIs for the OS of the CMs – Windows. Thus, the rather inefficient C2 is chosen for the implementation as the main priority of this bachelor thesis is the implementation of a working prototype. Changes to the architecture of the CMs and to the CMF are documented, as well as the implementation of a scanning script for the observer and an Android App for the *broadcaster*. Moreover, an IMS which manages the digital identities of the users is conceptualized and implemented. It can be accessed through REST. The evaluation of the prototype is done by simulating up to three identifying users at the same time. The evaluation shows the capabilities to identify all users reliably and quickly in a range of at least five meters. Nevertheless, the scanning script crashes about every ten minutes. The reason for that could not be found, but it is likely that the error lies in the used BLE API. Also, there are security issues as security is not focused during the development of the prototype.

From here on there are multiple research areas.

First of all, it could be reevaluated if the inefficiency of the implemented C2 is tolerable and if not C1 or C3 could be implemented. In that case it is strongly advised to use an external and

programmable micro controller with a powerful BLE API to execute the CM sided BLE logic. If it is wished to continue with C2 the crash of the scanning script needs to be fixed. Also, the security issues need to be fixed, meaning securing the REST endpoints and encrypting the communication via BLE.

Last but not least, direction finding methods using BLE could be additionally implemented to only identify users who are standing in front of the CM.

Acronyms

AoA Angle of Arrival. 13, 43

AoD Angle of Departure. 13, 43

API Application Programming Interface. 18, 19, 23, 25, 26, 30, 36, 39, 40, 41

ATT Attribute Protocol. 5, 9, 10

BLE Bluetooth Low Energy. 1, 2, 5, 6, 11, 12, 14, 15, 16, 17, 18, 19, 22, 24, 30, 39, 40, 41, 45

BLES Bluetooth Low Energy Stack. 1, 3, 5, 17, 23, 40

CM CommunityMirror. 2, 3, 14, 15, 17, 18, 19, 20, 21, 22, 24, 26, 27, 30, 36, 37, 39, 40, 41, 44, 45

CMF CommunityMirror Framework. 3, 5, 15, 26, 27, 29, 33, 34, 36, 39, 40, 43, 44

CMs CommunityMirrors. 1, 3, 4, 18, 27, 29, 40, 43, 44

GAP Generic Access profile. 5, 9, 15, 18, 19, 40, 45

GATT Generic Attribute Profile. 5, 10, 12, 15, 16, 19, 20, 21, 43

HCI-G Human-Computer-Interaction Group. 1, 14, 36, 40

IMS Identity Management System. 1, 2, 15, 17, 18, 19, 20, 23, 24, 26, 27, 30, 34, 36, 39, 40, 43

OS Operating System. 15, 23, 40

REST Representational State Transfer. 18, 20, 22, 25, 30, 39, 40, 41

RFID Radio-Frequency Identification. 14

UI User Interface. 3, 25

UUID Universally Unique Identifier. 10, 17, 34

List of Figures

List of Tables

Bibliography

Akiyama, S., Morimoto, R., & Taniguchi, Y. A Study on Device Identification from BLE Advertising Packets with Randomized MAC Adresses. In: In *2021 IEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*. Kindai Universityy. 2021. https://doi.org/10.1109/ICCE-Asia53811.2021.9641870.

Bluetooth Core Specification (v5.1). (2019). Bluetooth Special Interest Group. https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=457080

Camp, J. (2004). Digital identity. *IEEE Technology and Society Magazine*, *23*(3), 34–41. https://doi.org/10.1109/MTAS.2004.1337889

Eisenlohr, A. (2021). *Konzeption und Implementierung einer Anwendung zur Identifikation mit Mobilgeräten* (Bachelor thesis). Universität der Bundeswehr München.

Fietkau, J., & Stojko, L. (2021). Activity support for seniors using public displays: A proof of concept. In S. Schneegass, B. Pfleging, & D. Kern (Eds.), *Mensch und computer 2021 – tagungsband* (pp. 199–203). Association for Computing Machinery. ISBN: 978-1-4503-8645-6. https://doi.org/10.1145/3473856.3474002

Guillaume Celosia, M. C. Fingerprinting Bluetooth-Low-Energy Devices Based on the Generic Attribute Profile. In: In *2nd International ACM Workshop on Security and Privacy for the Internet-of-Things*. 2019. https://doi.org/10.1145/3338507.3358617.hal-02359914.

Jaimin, A. (2021). *BLE Power Optimization Parameters* (tech. rep.). BuildStorm. https://buildstorm.com/blog/ble-power-optimization-parameters/.

Leach, P. J., Salz, R., & Mealling, M. H. (2005). A Universally Unique IDentifier (UUID) URN Namespace. https://doi.org/10.17487/RFC4122

Montanari, A., Nawaz, S., Mascolo, C., & Sailer, K. A Study of Bluetooth low Energy Performance for Human Proximity Detection in the Workplace. In: In *2017 IEEE Conference on Pervasive Computing and Communications (PerCom)*. University of Cambridge and University of College London. 2017. https://doi.org/10.1109/PERCOM.2017.7917855.

Nilsson, D. (2022). *Identifying Bluetooth low Energy Devices via Physical-Layer Hardware Impairments* (Exam work). Uppsala Universitet.

Nilsson, D., & Yan, W. Identifying Bluetooth Low Energy Devices. In: In *21: Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*. Uppsala University. 2021. https://doi.org/10.1145/3485730.3492880.

Ott, F. (2018). *CommunityMirrors: Interaktive Großbildschirme als ubiquitäre Natural User Interfaces für Kooperationssysteme - Ein konzeptionelles Rahmenwerk soziotechnischer Gestaltungsparameter und Potenziale zur Verbesserung der peripheren Informationsversorgung in kollaborativen Wissensprozessen* (Doctoral dissertation) [retrieved on the 23rd May 2023]. Universität der Bundeswehr München. https://athene-forschung.unibw.de/doc/122536/122536.pdf

Qureshi, U. M., Umair, Z., Duan, Y., & Hancke, G. P. Analysis of Bluetooth Low Energy (BLE) Based Indoor Localization System with Multiple Transmission Power Levels. In: In *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)*. City University of Hong Kong. 2018. https://doi.org/10.1109/ISIE.2018.8433787.

Sakkopoulos, E., Ioannou, Z.-M., & Viennas, E. Mobile Personal Information Exchange over BLE. In: In *2018 9th International Conference on Information, Intelligence, Systems and Applications (IISA)*. University of Piraeus and University of Patras. 2019. https://doi.org/10.1109/IISA.2018.8633599.

Saputra, D., Gaol, F. L., Abdurachman, E., Sensuse, D. I., & Matsuo, T. (2022). Designing and testing of Bluetooth Low Energy (BLE) system for small wooden boat identification and e-certification. *SN Applied Sciences*. https://doi.org/10.1007/s42452-022-05021-z

Shan, G., Im, S.-y., & Roh, B.-h. Optimal AdvInterval for BLE Scanning in Different Number of BLE Devices Environment. In: In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS): Student Activities*. Ajou University. 2016. https://doi.org/10.1109/INFCOMW.2016.7562238.

Singh, R. K. (2018). *Designing a Mobile Identification and User-Profile Solution for an Urban IoT Network* (Master thesis). Technische Universität München.

Teaca, I. (2019). *Design of an encryption protocol for BLE advertising traffic* (Bachelor thesis). Vrije Universiteit Amsterdam.

The Bluetooth Low Energy Primer (1.1.0). (2023). Bluetooth Special Interest Group. https://bluetooth.com/wp-content/uploads/2022/05/The-Bluetooth-LE-Primer-V1.1.0.pdf

I hereby certify that I have written this paper independently, that no sources and aids other than those indicated have been used, and that all citations have been properly marked.

Furthermore I acknowledge having received the information overview concerning the usage rights regarding the Bachelor thesis. I grant Universität der Bundeswehr München the non-exclusive publication rights to my Bachelor thesis.

Neubiberg, 26.05.2023

.....

Christopher Lyko