Technische Universität München
Department of Informatics

Bachelor's Thesis in Informatics

A Feedback System for Smart Buildings

Arved Baus

Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

A Feedback System for Smart Buildings

Ein Feedback-System für intelligente Gebäude

Author	Arved Baus
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Holger Kinkelin, Marcel von Maltitz, M. Sc. , Dipl.-Inf. Johann Schlamp
Date	15. July 2015

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, 15. July 2015

Signature

Abstract

In smart buildings multiple devices, sensors, and actuators are connected and orchestrated by services to work as a single entity. The overall aim is to improve and optimize our living and working environment. Today, one especially important goal is the reduction of the building's energy footprint. However, building automation alone is unable to exploit the maximum energy saving potential. Multiple studies stress that the involvement of the inhabitants plays an important role as energy consumption relies heavily on habits and social norms. This means that an interaction between a smart building's energy saving system and its residents becomes highly important to reduce wasteful energy consumption.

This leads to the question how this communication can be realized. This work presents design, implementation and evaluation of a prototypical interaction system between building and inhabitants. The core idea is to provide a user-friendly and non-intrusive mobile phone application together with a server backend. To ensure authenticity and secrecy of messages, the server backend is based on the MeasrDroid Framework. Although the primary use case considered in this work is energy saving, our system was designed to be adaptable to other use cases.

Considering the amount of data produced by a smart building and the different *interests* of different users, the selection of information sent to a user is highly relevant. Therefore, the Publisher/Subscriber pattern was applied: Users can subscribe to publishers providing only relevant information to them. Vice versa, different people with different *authorizations* live and work in a building. As a consequence, publishers are capable to offer different information streams depending on the role of the user to ensures that possibly secret information does not reach the wrong user. Furthermore, information is typically only interesting for a user if the information refers to the user's near surrounding. *Location* is therefore the third selector for targeting. As user's location is personal data, our system was designed to respect the user's privacy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Buildings consume more than 42 % of all electric energy worldwide and will be the biggest emitters of greenhouse gases by 2025, according to a research conducted by IBM. In developed countries statistics are even more devastating: In the US, buildings use more that 70% of all electricity from which 50% is wasted. This leaves room for further optimization: The carbon dioxide (CO_2) emission can be lowered by approximately 50 - 70% and the water usage by 30 - 50% through the usage of smart buildings. But today, smart buildings are mainly intended to protect their occupants and provide them a safe working and living environment. Thereby, they miss to address the mentioned need for optimization regarding energy consumption. This means that during the development of future buildings safety and the protection of the environment must be considered.
Smart buildings connect many services and devices to work as a single entity. To tap the full potential of smart buildings also the inhabitants must get involved in the process of lowering the environmental impact. [1]

1.2 Problem

According to 2.1 the smart building has to have not only the ability to collect an extended amount of data but also to process and present it to involve the user in the process of energy saving. To accomplish this goal a communication system has to be established that allows the building to communicate with the user to provide information about its surrounding. But at the same time the amount of collected personal data such as GPS position should be kept to a minimum.
In many cases the user can also provide additional information to the system and other users. The system must offer a way for user to share their knowledge through sending

proactive notifications. Nearly all users of the target group carry a smart phone and are also familiar with its usage. Mobile applications have in general a flat learning curve and the user does not have to carry another device with them. Also the distribution and update of a mobile phone application is cheap and does also scale well with an increasing number of users. There are of course other possibilities to establish a connection between users and building such as the installation of panels located at important points of the building or a web interface which can be used with nearly all digital devices. The disadvantage of all these approaches is that a user has to login first before she can start communicating.

1.3 Goals and Research Questions

Due to the mentioned advantages it can be said that a smart phone application offers an ideal channel for communicating. The research questions that arises in the context of smart phones in connection with smart buildings are diverse and span over multiple fields of expertise. This work is therefore restricted on elementary questions regarding the communication between the user and the smart building.

RQ1: Identification of participating parties and their needs The different parties that are involved in the communication and their needs must be identified. It should also be ensured that new participants can be added easily. This question is discussed in detail in chapter 3.1.

RQ2: Filter Information In order to motivate the user to participate actively in the communication, it should be avoided to overwhelm her with too many information. Therefore, it is necessary to research how important information can be filtered for the user. An approach for this question is presented in chapter 2.5.

RQ3: Protect User Privacy In order to filter information for the user data must be collected. This data can reach from GPS coordinations to personal preferences.

RQ4: Choice of Technologies An application is developed that illustrate the results to the mentioned questions above. To implement this application different technologies must be evaluated. Chapter 5 presents the resulting application in detail.

The primary field of application for the developed system is an university. Under this restriction certain assumptions are made such as rights and tasks of participating users. Nevertheless, the system is not restricted to this area, it can be easily customized for other working environments. An university provides also a vast variety of people with different tasks and responsibilities.

Table 1.1: Application of Waterfall Model

Phase	Explanation	Chapter
Requirement	All necessary requirements are derived from use cases and documented	3.3
Design	The software architecture of the system is designed	4
Implementation	Concrete Implementation of the system that has been designed in the previous phase	5
Verification	Evaluating the quality of the solution.	6

1.4 Application of the waterfall model

This work applies the waterfall model which is a widespread sequential design process in the field of software engineering. It is in general divided into five different phases: Requirements, Design, Implementation, Verification, Maintenance. The waterfall model clearly emphasizes the creation of documentation to receive maintainable and understandable source code. The phases must be conducted in this order because they build on each other. It is not possible to return to a phase that has already been finished. Table 1.1 presents the different phases of the waterfall model and their corresponding chapters in this work. The phase Maintenance is not part of this work due to the time constraints. [2]

Chapter 2

Background

This chapter lays the knowledge foundation for further discussions in the design chapter.

2.1 Smart Building

There is no clear definition of smart building. In the context of this work the definition of Smart building follows the “Smart 2020 report“. This report has been conducted by The Climate Group and “McKinsey & Company“. According to this definition the Information and Communication Technology (ICT) in a building has to fulfil all of the following points to be “smart“.

- **Standardise** ICT can provide information in standard forms on energy consumption and emission across multiple sectors
- **Monitor** Data can be collected in real time. The ICT is able to control the energy consumption
- **Account** The ICT provides useful information to the consumer in order to involve her in the process of energy saving.
- **Rethink** The user should rethink his behaviour due to the provided information
- **Transform** As a result of the described process the user should change her behaviour to save energy.

The focus lays on the raise of awareness concerning energy consumption among inhabitants. At the same time it misses to stress the already mentioned importance of privacy. Nevertheless in the context of this thesis a smart building has to have the potential to fulfil all of the mentioned points. [3]

2.2 Importance of Feedback for Influencing Users

Also the report "Schlussbericht Nutzverhalten beim Wohnen" from the city of Zurich emphasizes the influence of the user's behaviour on the energy consumption of a building. Furthermore, the report states that habits can be changed more easily than other factors which lead to a bad carbon footprint such as the location, size and equipment of the building. [4]
The report does also mention that the behaviour of the user is not only changeable but does also has a high potential for improvement: According to a survey of the german Bundesministerium für Umwelt (BMU) from the year 2008 three out of four interviewee neither has known their energy consumption nor their energy price. [5]
But saving money is - according to another survey conducted by the BMU in the year 2012 - the biggest motivation for people to rethink their habits regarding energy consumption. [6]

Summarizing it can be said that smart buildings satisfy a market with high potential. They try to close the gap between intention and a lack of knowledge through an educational approach. Nevertheless there are problems regarding the communication between user and smart building which will be discussed in the next chapter.

2.3 Google Cloud Messaging

Google Cloud Messaging (GCM) is the successor of Cloud to Device Messaging (C2DM) and was introduced in June 2012. It is a service which enables programmers to send data with maximal 4KB payload to Android, Chrome and iOS applications. It supports the HTTP as well as the XMPP protocol. It "handles all aspects of queueing of messages and delivery to the target Android application running on the target device"[1]. If an application uses GCM it does not need to be currently running on the device. A broadcast Intent is permanently running in the background on the android phone and listening to incoming messages.
GCM needs four credentials to send a notification to an application. The **Sender ID** is used in the registration process and defines the server that is permitted to send messages. This message has to include the **Sender Auth Token** in the header of the POST request. The target device is uniquely defined by the **Application ID**. Furthermore the Android app has to provide a **Registration Token** that allows it to receive messages by the server. When the intent receives a new message it will wake up the specified application and pass the data to a handler defined by the target application for further processing. The described architecture provides a communication layer which can be used by all applications on the same device. Therefore, it is only necessary to run one background

[1] https://developers.google.com/cloud-messaging/

thread listening on requests instead of a thread for every application. This leads to an more efficient battery and memory usage. The MeasrDroid Framework (2.4) does also use GCM for communication between server and client. At the time of writing the GCM service is free to use. Furthermore, Google does not limit the amount of sent messages. Nevertheless there are also several limitations: If a device is not reachable for the service messages will be stored on the GCM server to send them to the client at a later point of time. This storage can take maximal 100 messages addressed to one device. If this limit is outreached all cached messages will be discarded. If this has happened a report message will be sent to the client at the next time he is reachable. There is, as already stated, no limitation regarding the number of send messages. But Google throttles messages sent to a single device to prevent an attack through a huge number of messages. Google does not publish the exact number of directly receivable messages. But, according to several tests, 20 messages can be send from the same server and received on one client without any artificial delay. Furthermore, the size of a single message send to a device must not outreach four kilobytes. In this case the service refuses to send the message to devices. [7]

2.4 Measrdroid

The Measrdroid Framework is an Android Framework which collects technical data on mobile phones to use them in scientific research projects. The aim of this Framework is to analyse technical aspects of the internet to gain a better understanding of it. In the following the existing system will be presented and its usage for this project evaluated.

2.4.1 MeasrDroid Core

The MeasrDroid Core is an un-opinionated Android Framework which has vast capabilities. It allows the developer to integrate Google Cloud Messaging Service (see 2.3 for more information) easily into an application. In order to use MeasrDroid the main class of the application must inherit from the MeasrDroid base class and has to be marked with the *@MeasrDroidSetup* tag. Through this tag additional classes can use the capabilities of MeasrDroid. All available attributes and their meaning are listed in table 2.1.
There are no restrictions regarding the amount or combinations that can be used in an Android Application.

Table 2.1: MeasrDroidSetup Tag

Name	Meaning
gui_id	Multiple views use the MeasrDroid core. In order to map the applications with their measurements this id is necessary
name	Name of the application
services	Services which run in the background and that should be managed by the MeasrDroid core.
database	Provides a type safe interface to store settings. Noticeable is that MeasrDroid uses the *SharedPreferences.Editor* which is not able to store non primitive data types across user sessions[2].
sensors	Further sensors can be added to MeasrDroid. In order to activate external sensors the *S.ROOT.GUI.WILDCARD* needs to be set in the configuration of a measurement.
confManagers	The MeasrDroid core daily checks the *conf.droid* server for new configurations of the application. Configuration files provide a way to send seldom changing data to all clients.
GCMManagers	GCM messages which have an unknown format will be send to the GCMManagers which can process it further.
prefManagers	Through external Preference Manager it is possible to inject new preferences to the application. It will be called in case that the user clicks changes or persists any preference.
wizardPages	At the first start of an application a wizard is displayed which guides the user through the process of installation. The wizard can be extended with additional pages using this tag.

2.4.2 Backend Infrastructure

The measrdroid application for example is an application which allows the user to visualize all taken measurements. [8]
In the following the core components of the existing Measrdroid infrastructure and their tasks are presented.

- **C3PO** *C3PO* is the core Server of the MeasrDroid backend. It is the only server which has direct access to the database of measurements. Due to security reasons is *C3PO* not connected to the internet. *C3PO* is involved in every communication between the Android Client and the backend. The core assumption of Measrdroids security system is that any server but *C3PO* is possibly compromised. The security architecture of MeasrDroid foresees that any communication involving *C3PO* must have also been started by it.

- **Satellite Server** *C3PO* does only provide core features. To extend the capabilities of MeasrDroid new Satellite Servers have to be deployed. MeasrDroid does regularly check specific directories of all satellite servers to receive data.

- **gcm.droid** The *gcm.droid* is capable to send messages to devices with the given gcmId.
- **Push.droid** *C3PO* pushes messages to the *Push.droid* server which sends it to the clients using Google Cloud Messaging.
- **Upload.droid** All measurements taken by the clients are sent to the *upload.droid* server. *C3PO* does regularly pull all received messages from this server and stores it in its database.

The presented components use Python and Bash Scripts to communicate with each other over the file system.
Figure 2.1 shows the current architecture of the MeasrDroid project in connection with an Android Application that uses the *MeasrDroid Core*.

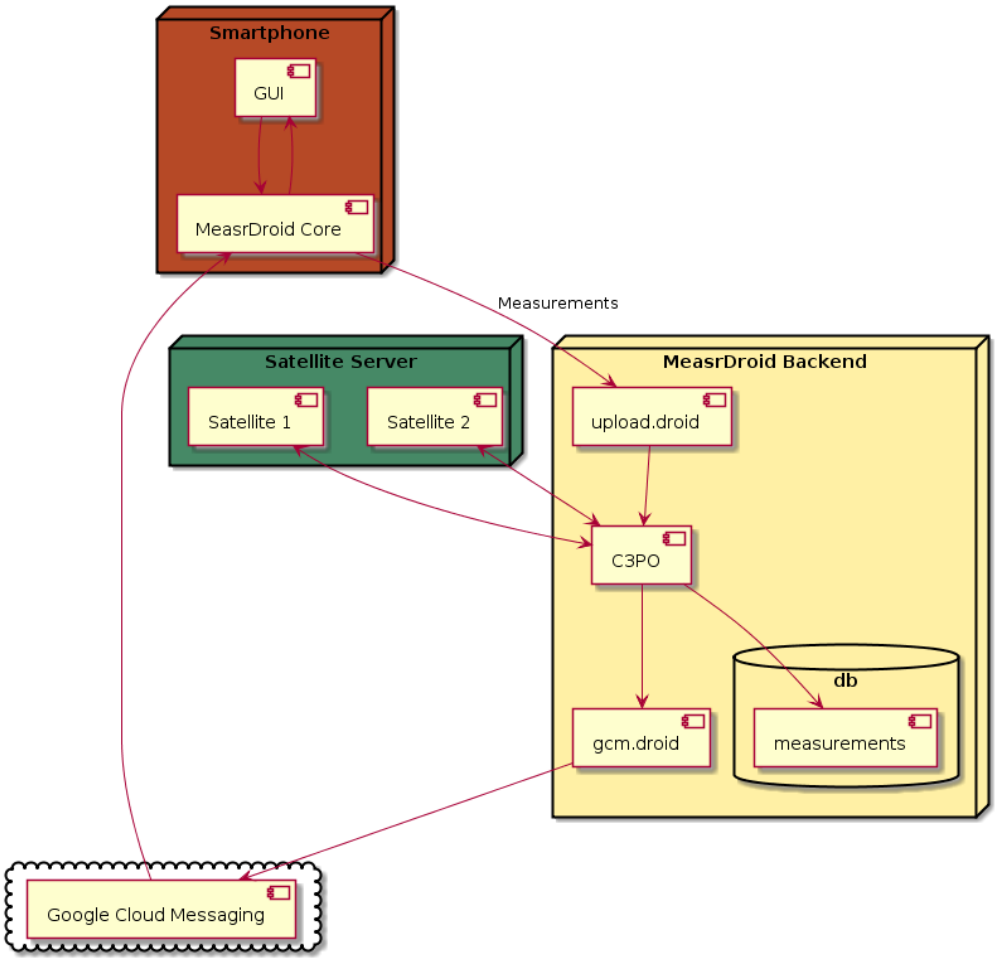


Figure 2.1: Component Diagram

2.4.3 Properties of MeasrDroid

MeasrDroid has a variety of advantages, some of those are:

- **Documentation** During the development over ten Bachelor and Master theses had been written about the different parts of the system. All components are therefore well documented.
- **Flexibility** MeasrDroid can read multiple sensors of a device. Not all of them will be useful for this project. But MeasrDroid offers also the opportunity to activate and deactivate sensors flexibly.
- **Easy Integration** The MeasrDroid core can be easily integrated into an existing application. Few changes are necessary to use the provided capabilities and start measurements.
- **Security** MeasrDroid provides an existing security system. All messages send to the application must have been signed by *C3PO*. This prohibits the misuse of the established communication channel.
- **Existing backend** MeasrDroid does already have an existing and tested backend architecture with Google Cloud Messaging integration. The use of MeasrDroid does therefore decrease the workload significantly.

But the use of MeasrDroid has also some disadvantages:

- **No real-time Applications** Due to the described architecture no real time Application can be implemented. *C3PO* syncs after a certain time interval with its servers. Due to the amount of processed data, the time steps cannot be set arbitrary small.
- **Restricted to Android** MeasrDroid is at the time of writing only available for Android smart phones. Even if Android has possessed the highest market share among all mobile operating systems with 78% in the first quarter of the year 2015, the user base of the other systems is noticeable. It is unlikely that future versions of the MeasrDroid core will support other mobile operating systems. [9]

2.5 Publisher/Subscriber pattern

The Publisher/Subscriber pattern (also called Observer pattern) intents to define a one-to-many dependency so that when one object changes state, all its dependents are notified and updated. At the same time a tight coupling between the publisher and subscriber should be avoided because this reduces the reusability of the components. A Publisher offers therefore an API to register and unregister publishers.

If a Publisher sends a notification it pushes it to all subscribed publishers. The Publisher/Subscriber pattern can be divided into three subclasses depending on their way to handle an update of their status:

- **Push Notification** Every Time the publisher update its status every subscriber is notified. The actual update of the status is not contained in the notification.
- **Push-Update Notifications** Like *Push Notification*, but the new status is send with the notification.
- **Pull Notification** The subscriber is notified about any status change, it autonomously ask the subscriber for status changes.

Chapter 3

Analysis

In this chapter the environment of a typical university is analysed. At first, the roles of possible users are identified. Afterwards, typical use cases that involve these roles are presented from which the functional and non-functional requirements are derived.
The use cases and roles are restricted to the environment of an university as this is the target platform for the application. This environment serves as an example to derive requirements of the application. The university offers an environment where users with a variety of different tasks and authentications work together and communicate with each other, similar as in a factory or office building. It is not the goal to develop an application specific to the needs of a university.

3.1 Roles

A variety of people with different roles are interacting with the system and with each other. In the following their different interests of roles and their tasks are described:

- **Facility Manager** The facility manager has vast responsibilities. They reach from repairing the elevator to salting the pavement on cold days. He wants to be informed about any malfunctions and their location. His goal is to provide fast and reliable support.
- **Student** The student works in the building and is also interested in its environment. Therefore he is willing to contribute to the maintenance of the building. Nevertheless he wants to spend as less time as possible on this task to concentrate on his actual work. He has also minor authorities and in most cases no administrative tasks.
- **Professor** The professor does research and and holds lectures. He has great interest in the university and major authorities. He often has extensive administrative tasks.

During the development of the application all of these interests should be considered. It is noticeable that this is only a small subset of all roles which are present in a building. Further ones could be a normal employee or a member of the cleaning staff. Nevertheless the presented selection illustrates the differences between different roles.

3.2 Use Cases

In the following use cases are presented to show the reward of using the presented system. All use cases follow the same structure: After a short introduction into the domain of the problem a possible solving strategy is presented.

3.2.1 First Aid

A student hurts himself during a Basketball game and needs first aid. The system offers an opportunity to call for first aid. Some fellow student volunteer and provide their help for such a case. The urgent need for help makes this use case time-critical.

Use Case 1	**Emergency**
Scope:	Local
Primary Actor:	Students
Roles and Interests:	• Student: Fast and reliable first aid • Volunteer (fellow student) : Location of the accident
Preconditions:	Students have the App installed and fellow students are registered as volunteers
Postconditions:	Volunteer is informed about the position of the student that needs help
Main Success Scenario: 1. Students call the system for help. 2. Volunteers next to the location of the accident receive a notification with exact GPS coordination.	
Frequency of Occurrence:	Medium

3.2.2 Proactive Notification

Due to the limitations of a smart buildings, there are events which cannot be recognised automatically. The system offers therefore an option to make notifications initialized by a user. Not only the user who is responsible to solve this issue is informed but also other users that are also affected.

Use Case 2	**Proactive Notification**
Scope:	Local
Primary Actor:	Facility Manager, Student, Professor
Roles and Interests:	• Student, Professor: Does not want to slip on the slick pavement • Facility Manager: Provides a safe working environment
Preconditions:	A student steps out of the train and slips due to a slick pavement.
Postconditions:	The facility manager and everybody near to the location are informed about the slick pavement.

Main Success Scenario:

1. A student observes the slick pavement and sends a notification to the system.
2. The System informs the facility manager who salts the pavement.
3. Until this happens all passants will be asked to walk carefully through a notification sent by the system.

Frequency of Occurrence:	Medium

3.2.3 Interest Groups

An interest group exists of people with an equal opinion about one or multiple topics. If a user is unhappy about his environment due to a certain circumstance, it is very likely that there are other users who share his opinion. The system supports therefore the establishment of interest groups. In an interest group users of different roles can participate.

Use Case 3	**Interest Groups**
Scope:	Global
Primary Actor:	Student, Professor, Facility Manager
Roles and Interests:	• Student, Professor: Appropriate working environment • Facility Manager: Minimize Cost, Possible Improvements
Preconditions:	Multiple Students complain about headache after meetings in a certain room of the building, caused by a low percentage of oxygen in the air
Postconditions:	The Facility Manager is aware of the poor working environment

Main Success Scenario:

1. An student sends a notification to the system to complain about his environment.
2. Multiple students observe the same issue and form a group of interest.
3. The Facility Manager gets informed about the amount of complaints.

Frequency of Occurrence:	Medium

3.2.4 Confirmation of Malfunctions

Intelligent buildings are only able to detect malfunctions of a building with a certain percentage. Also proactive notifications of a user can be misleading. Therefore it is necessary to check the validity of both. The system should therefore notify a user whose location is near to the malfunction.

Use Case 4	**Confirmation of Malfunctions**
Scope:	Local
Primary Actor:	Facility Manager, Student

Roles and Interests:	• Student: Quick fix of issues • Facility Manager: Location, no misleading information
Preconditions:	The smart building recognizes an anomaly in the power consumption of the coffee machine.
Postconditions:	The Facility Manager repairs the coffee machine.

Main Success Scenario:

1. The System sends a notification to a nearby user to check whether the coffee machine works properly.
2. The user discovers that the coffee machine is off due to a short circuit.
3. The user sends a confirmation with a short description.
4. The Facility Manager gets a notification with the location of the coffee machine and the task to fix it.

Frequency of Occurrence:	High

3.2.5 Reduce Energy Consumption

Intelligent buildings are already possible to detect wasteful use of energy. The system should for example be able to notify a user which is located near to a display that has been left on after work. This use case and possible solutions are explained in the Bachelor's thesis of Lawrence Krug. [10]

Use Case 4	**High Energy Consumption**
Scope:	Local
Primary Actor:	Student, Professor, Facility Manager
Roles and Interests:	• Student, Professor, Facility Manager: Reduce energy consumption
Preconditions:	The smart building detects a turned on display despite the fact that the user has already left the building
Postconditions:	The display will be turned off

Frequency of Occurrence: High

3.3 Requirements

From these different use cases multiple functional and non-functional requirements can be derived.

3.3.1 Functional Requirements

R1: Publisher/Subscriber The system should be implemented with the Publisher/-Subscriber pattern that is presented in chapter 2.5. Therefore, every user receives only a subset of all sent messages. Through this approach different use cases like 3.2.3 are possible and fit in a natural way into the existing system. Every interest group is a publisher which sends its messages to everybody who has subscribed to the topic. The Publisher/Subscriber pattern is described in chapter 2.5. In this case the *Push-Update Notification* version of the pattern should be applied.

R2: Role-based Channels The system should allow communication channels for different users and different roles. The owner of the channel can decide which roles can subscribe to her channel and which cannot. Through this design decision the channel owner has full control about the access rights.

R3: Location Awareness Many informations that a smart building provides are only specific to a location. If, for example, somebody has left his computer turned on after leaving the building, then this is only important to people which are still in the same part of the building. Therefore, a location aware system is needed to filter the information for the user.

3.3.2 Non-functional Requirements

R4: Security The proper functioning of the system relies heavily on the absence of misleading information. It should be therefore impossible for any user to inject such information or trick the system in any other malicious kind.

R5: Privacy Due to requirement *R3* and the implicit dealing with the location of the user the question after the privacy of the user arises. It should not be possible for an attacker to associate the position with personal information of a user or to even recreate a movement profile. The goal of the system is to expose and collect as less data as possible.

R6: Real Time Interaction The importance of a fast interaction with the user is stressed by the use cases in 3.2.1 and 3.2.2 . Especially during an emergency it is important to inform all affected persons as soon as possible. For the use cases presented in 3.2.3 and 3.2.4 the response of the system in real time is less important.

R7: Extensibility The architecture should allow to extend and improve the system and its functions easily.

R8: Usability The advantage of using a smart phone application for interaction is that most users are already familiar with the system and do not need special training. In order to keep this advantage the Graphical User Interface should follow design patterns the user is already familiar with.

Chapter 4

Design

This chapter derivatives the architecture from the requirements presented in chapter 3.3. At first the different components of the system are identified. Following this, benefits of using the MeasrDroid Framework that is described in chapter 2.4 are evaluated. Afterwards the abstract architecture and the flow of information between the different components is described. The following chapter builds on the insights of this chapter to explain concrete details regarding the implementation of the application.

4.1 Architecture

The architecture can be divided into three main components.

- **Mobile Application** An Android Application which allows the user to send and receive notifications. It should be able to receive messages through Google Cloud Messaging and send responses back to the server. Requirement R3 states that the application must be aware of the current location. No special training should be necessary to use this application according to requirement R8.
- **Communication Layer** The communication layer provides a channel of communication between application and server. Google Cloud Messaging is an ideal service to send messages from the server to the client as discussed in 2.3 and should therefore be used.
- **Logic Layer** The Logic Layer consists of multiple Publishers according to requirement R1. These publishers are independent of each other and use the infrastructure provided by the framework to communicate with the Android clients.

Figure 4.1 shows a component diagram of the presented architecture. The complexity of the diagram increases in the course of this chapter. Each component can be easily

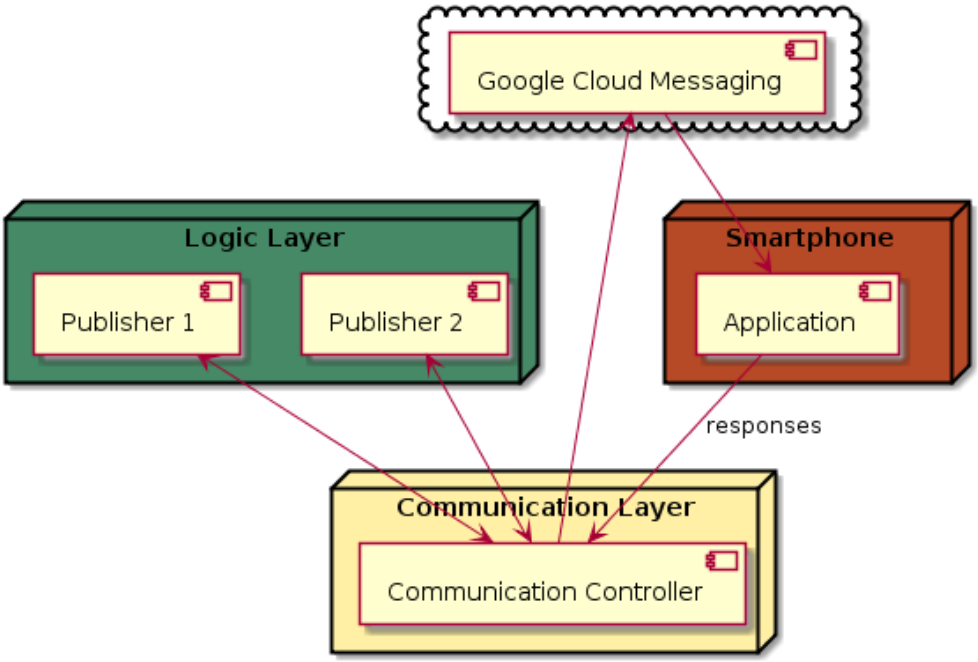


Figure 4.1: Component Diagram

identified in the different diagrams by its unique color. Publishers can only communicate with clients through the Communication Layer. This enables the fulfilment of requirement R4 and R5 (3.3.2).

4.2 Publisher

The developed framework should provide an API which enables developers to integrate new publishers easily into an existing system. The complexity of logic a publisher provides differs. A Publisher provides multiple tags users are able to subscribe to. In the following a few possible publishers and their tags are presented:

- **Alert system** In the use case described in chapter 3.2.2 a user can send a message proactively and warn other users about possible dangers such as a slick pavement.
- **Interest Groups** The use case in chapter 3.2.3 provides an example where user can form groups of interests. Every interest group would form a tag user can subscribe to if they are interested.
- **Energy Manager** In 3.2.5 an use case has been presented where the system does automatically detect high energy consumption. The developed framework could be used to communicate with a user who is located in the near surrounding of a

device that wastes energy. Different part of the building could be represented by tags. The user can subscribe to frequently visited parts of the building.

4.3 Contextual Evaluation of the MeasrDroid Framework

The MeasrDroid Framework, its architecture and advantages such as disadvantages have been already discussed in 2.4. It will now be evaluated as a foundation of the architecture presented in 4.1. MeasrDroid provides a core component which allows to use a variety of measurements. [11] Sensors can be enabled and disabled independently of each other. That means a granular control over the collected data what is in favor of the user's privacy. MeasrDroid has also an already existing Google Cloud Messaging integration. The disadvantage of using MeasrDroid is that it is not possible to implement applications who rely on a time critical connection to the server. MeasrDroid's security approach forbids sending requests to *C3PO*. This leads to an architecture where any component communicating with MeasrDroid puts data in a specified directory and relies on MeasrDroid to fetch and further process it. Due to the amount of collected data the time interval between two fetches must not be arbitrary small.
From a pragmatic point of view the advantages of using MeasrDroid for this project outreach the disadvantages. Due to the limited time and resources of this work the usage of MeasrDroid lightens the workload tremendously out of the discussed reasons. The *MeasrDroid Core* component can be used on the client side. The capabilities of *MeasrDroid's* backend system around *C3PO* matches also partially the requirements of the *Communication Layer*.

4.4 Additions to MeasrDroid Framework

To realise the architecture described in chapter 4.1 utilizing MeasrDroid, the existing system must be extended.

4.4.1 Additions to the Backend

Figure 4.2 shows the necessary additions to the MeasrDroid Framework on the server side. A new satellite server must be developed which offers a RESTful API for Publishers. The Publisher uses this API to communicate with the client. Furthermore, Publishers can provide a Web Interface for more sophisticated interaction with the client such as presenting graphs or complex input forms. The satellite server keeps track of all registered services and offers a selection to the user that depends on her role. This satellite server is called *Publisher Manager* in the context of this work.

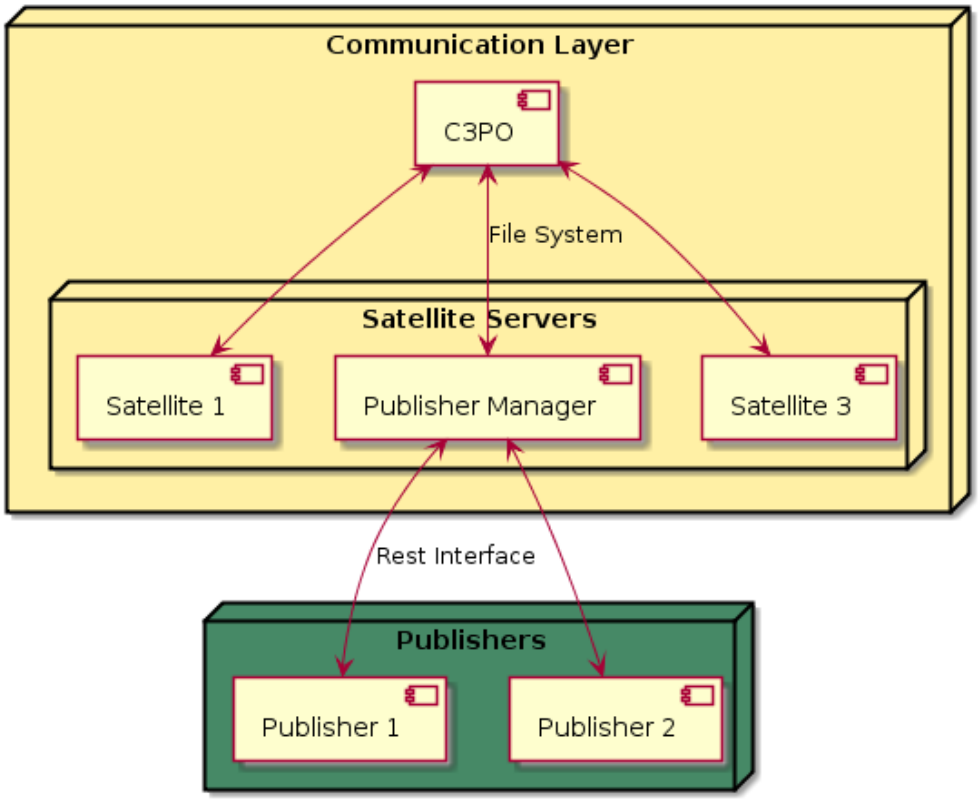


Figure 4.2: Component Diagram

4.4.2 Android App

The native Android application should also provide a native user interface to ensure a flawless user experience. In the final application the user receives a notification on his phone as displayed in figure 4.3 that indicate that a new message has arrived. The user can now either click on the notification to open the *Message View* instantly or he can open the application and take a look at all unanswered messages as shown in figure 4.4. The application does also provide a view that shows all available Publishers and their tags. It is depicted in figure 4.5. In order to subscribe or unsubscribe to a tag the user must click on the switch button in front of the tag. This also activates the tag in the proactive notification fragment as shown in figure 4.6 where user can send messages without having to have received one in the first place.

4.4.3 Summary

Figure 4.7 shows an overview about the whole architecture. Publishers communicate with a server called *Publisher Manager* to send requests to and receive responses from

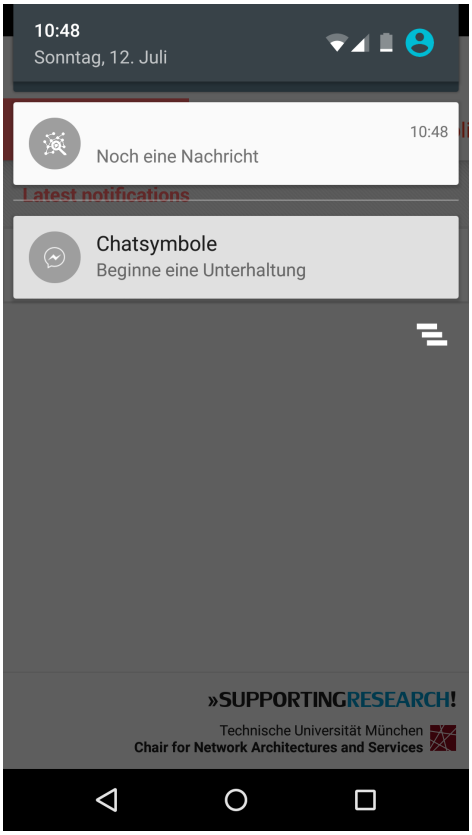


Figure 4.3: User Notification

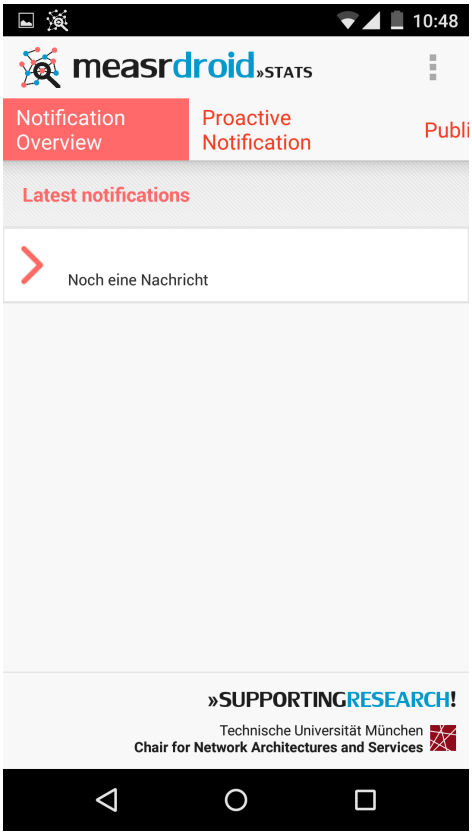


Figure 4.4: Notification Overview

clients. All Publisher together form the *Logic Layer*. A Publisher is a web server that communicates with the *Publisher Manager* through a REST Interface. The *Publisher Manager* persists all incoming requests and their responses in a database. This database does also contain all users of the system and their subscribed tags. The *Publisher Manager* sends any new request from one of its Publishers to *C3PO*.

4.5 Information Flow

In the following the communication between the different components of the system and the user will be explained in detail. The entire flow of information is displayed in 4.8 as an UML sequence diagram. The different communication “steps “are increasingly numbered by their time of execution. The description refers to this steps as an orientation.
In general, there are two different types of communication between publisher and client. On the one hand the client can start the communication proactively through sending a text message to the publisher. On the other hand the publisher can start the communication through sending a new message to the *Publisher Manager*. The first type works

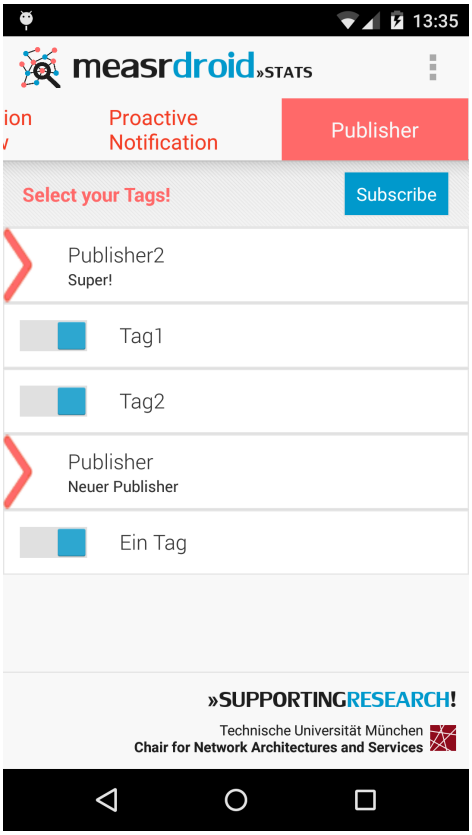


Figure 4.5: Publisher Overview

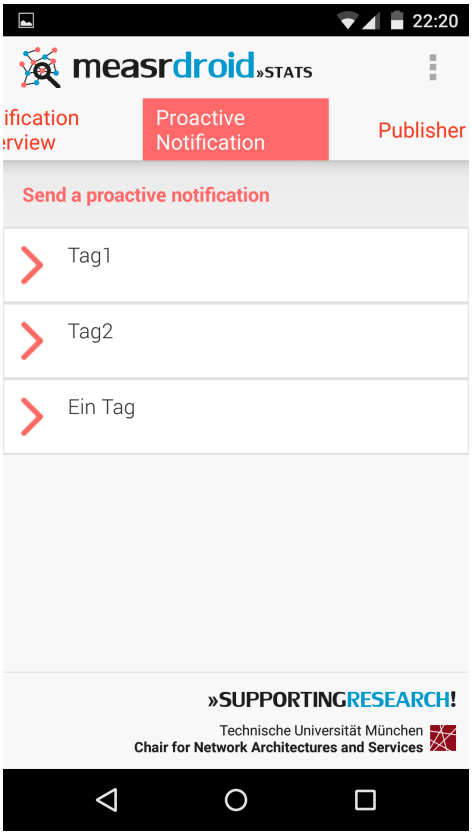


Figure 4.6: Proactive Notifications

similarly as the second one but without the initial communication between publisher and user. Therefore, only the second type will be discussed.

Step 1: The Publisher starts the communication through the call of a RESTInterface which the *Publisher Manager* provides. A detailed explanation if this Interface can be found in chapter 5.2.2. The message contains a text, the role of the receiver and optionally her GPS position and a radius.
Step 2 - 3: The *Publisher Manager* does now look up the GCM ids of all users who have subscribed to the tag and also have a role that matches the requirement. Every device gets an unique GCM id for every installed application which uses this service. MeasrDroid polls regularly all new messages from a directory of the *Publish Manager* and sends it via Google Cloud Messaging to the client. To send messages the *push.droid* server is used. It is not displayed in the diagram for the sake of simplicity.
Step 4 - 6: The *MeasrDroidCore* parses and analyses the received message. In case that the message has an unknown structure, the *MeasrDroidCore* sends it to the GUI. If the publisher has specified a user location through GPS coordinates, the GUI starts a MeasrDroid Measurement to check if the client is within the radius. Every executed Measurement will be uploaded to C3PO to an unknown point of time. This constraint

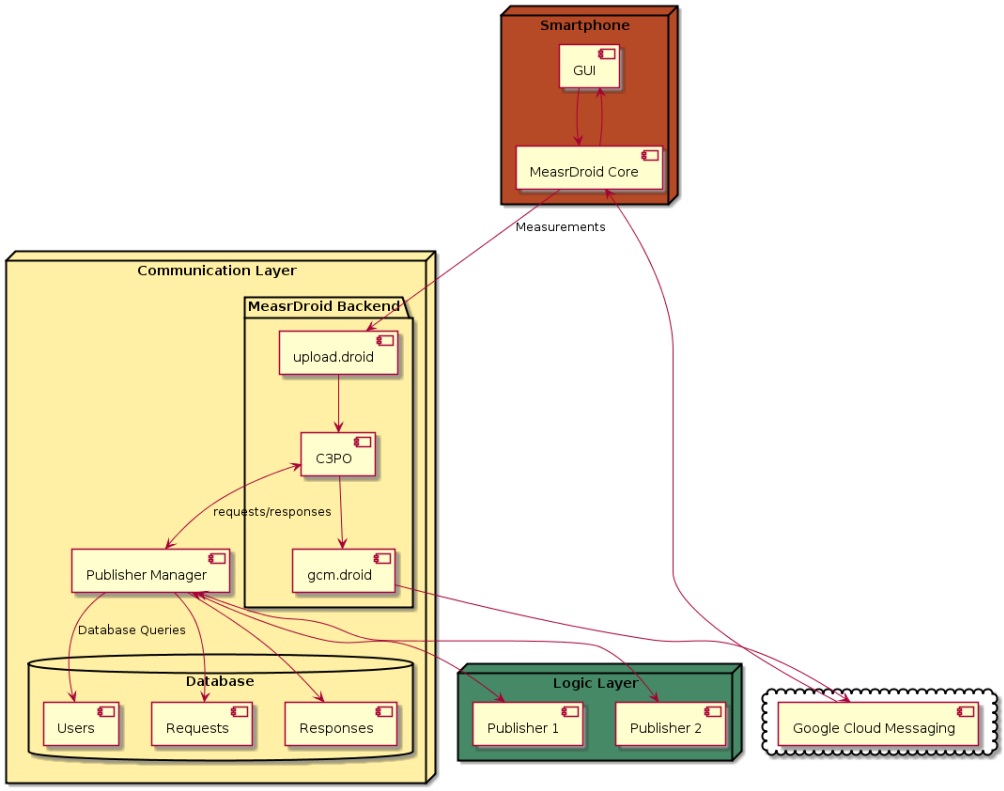


Figure 4.7: Architecture Overview

is by design and cannot be deactivated.
Step 7 - 11: The GUI has registered a handler on the measurement to receive the results, too. In case that the user is within the given radius he will receive a notification on his mobile phone. Now, he is able to give a feedback through the interaction with a *Message View*. His input will be handled has a normal measurement and instantaneously uploaded to *C3PO*.
Step 12 - 13: *C3PO* forwards all incoming measurements which have been taken on a device with *gui_id=5* to the *Publisher Manager* which stores every response in a database. Afterwards, it calls the REST API of the *Publisher* which has initialized the communication in the first place with the new user response.

4.6 Privacy

User Privacy (Requirement R5) plays an important role if sensible data is collected. In the case of the presented system the response to messages and the location of the user are informations that should not be exposed to unknown parties. Nevertheless, it is necessary to store this information as pointed out in chapter 3.3. To address this

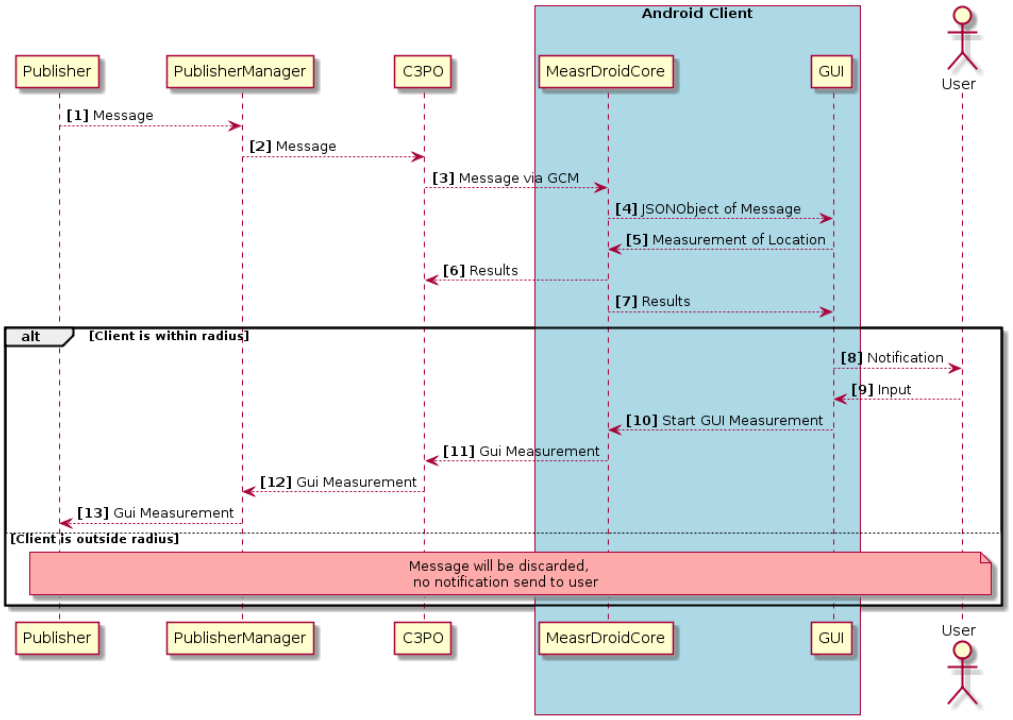


Figure 4.8: Sequence Diagram

issue the MeasrDroid Framework is used. *C3PO* signs all outgoing messages with a key that only the *MeasrDroid Core* component is able to decrypt. The result of every measurement is stored in an encrypted file in the file system of the device. This file is only readable by *C3PO*, not even the device itself is capable to decrypt it. In order to avoid possible security holes the application must not copy and store any measurements that are reached to it through a Handler.
The results are then uploaded to *C3PO* by *MeasrDroid* at an unknown point of time.

Chapter 5

Implementation

This chapter describes the implementation of the Publisher/Subscriber system that has been presented in chapter 4. Each component and its concrete implementation is presented in detail. The focus lays hereby on the communication between the different components and the use of MeasrDroid.

5.1 Android Application

Through the developed smart phone application the user should be able to communicate with the system. MeasrDroid is used to transport information from the server to the user and vice versa.

5.1.1 Integration of Google Cloud Messaging

The *GoogleCloudMessagingManager* will be called if the core receives a message that it is unable to parse. In general there exist two types of messages the server can send to the client.
On the one hand there are *PublisherLists* which contain a new list of publisher including their tags and IDs. If a new TagList arrives it will be stored to the local file system in JSON using the *DatabaseManager*. Afterwards a new *NEW_PUBLISHER_LIST* event is broadcasted using the *LocalBroadcastManager* provided by the Android SDK. A broadcast can be received by every process that listens to the specified event using a *BroadcastReceiver*. Local broadcasts can only be received by the same application which also sent it. A normal broadcast, in contrast, allows the communication between different applications on the same smart phone. All types of broadcasts are send asynchronously by default. The *PublisherFragment* listens to the *NEW_PUBLISHER_LIST* event and reads the list of publishers from the file system to update the user interface in case of an incoming

event. The structure of a Publisher list is shown in 5.1

Listing 5.1: Tag List

```
{
     "messageType": "TagList",
     "tags": [
          {
               "_id": "...",
               "description": "...",
               "title": "asdd"
          }, ...
      ]
}
```

On the other hand there are publisher notifications which represent a message sent by a Publisher to the *Publisher Manager*. The structure is displayed in listing 5.2. The *publisherMessageId* is set by the publisher to keep track of its own messages. The location is an optional attribute which allows to address only users which are in a certain area. To decide whether a user is in an area or not the MeasrDroid core has to take a GPS measurement, because the MeasrDroid server does not know the exact position of its users. If this attribute is not specified or the radius is set to zero, no measurement is conducted and the message is directly prompted to the user. The message is the part of the notification which will be displayed to the user. Title and text are mandatory fields, whereas the url is optional. This attribute indicates that there is a website that provides information that the user has an interest in.

Listing 5.2: Publisher Notification

```
{
  "publisherMessageId": "0",
  "messageType": "PublisherNotification",
  "location": {
    "latitude": ...,
    "doc_type": "GeoLocation",
    "radius": ...,
    "longitude": ...
  },
  "message": {
    "title": "...",
    "url": "...",
    "text": "...",
    "viewId": ...,
  },
  "publisherId": "..."
}
```

5.1.2 Measurements

MeasrDroid is only capable to send measurements from an Android client to a server. The result of a measurement is stored in the JavaScript Object Notation (JSON). After a measurement has been taken its result will be immediately decrypted and cached in the local file system of the device. This prevent attackers who have access to the device from reading any measurements taken by the user. Only *C3PO* has the key to decrypt measurements. To save mobile data value MeasrDroid prefers to send measurements only if the device is connected to a WI-FI network. In case the user is not connected to such a network for a time, that has been specified by the user in advance, it uploads all cached measurements at once. This means in order to use the MeasrDroid infrastructure, any communication between client and server has to be represented by a measurement. Through the sensors attribute of the *@MeasrDroidSetup* tag external sensors such as the *GUISensor* can be registered. If a measurement with enabled external sensors is started the *GUISensor* adds every new user response and the current role of the user to the measurement. Furthermore, MeasrDroids default behaviour to cache measurements until the device is in a WI-FI network has to be bypassed. Therefore the *ForcePublishManager* has been implemented. It delegates all calls to his own instance of a *PublishManager* but without checking whether the user is connected to a wireless or mobile network or whether his donated limit of traffic has already exceeded. After every interaction between the user and the application which should be stored on the *Publisher Manager*

such as subscribing to new tags the *ForcePublishManager* must be called to upload the new measurements as soon as possible. *Message Views* must store the user input in the *NotificationDB* in an object of the response class. The JSON representation of the response class is represented in 5.3. It stores the original notification and the response as a string, because no more sophisticated response structure is needed for the current *Message Views*.

Listing 5.3: JSON of Response class

```
response: {
        notification: {
                ...
        }
        response: "..."
}
```

5.1.3 Location Awareness

Requirement R3 demands that the application must be aware of its location. If a message arrives that is only interesting for users within a certain area, the application conducts a measurement using MeasrDroid with activated location sensors. The message provides longitude, latitude and radius to determine the area. After finishing the measurement the application evaluates the result on the device. The haversine formula is used to calculate the distance between the current location and the specified one. It states that between two points on a sphere the central angle between them is given by equation (5.1).

$$haversin(\frac{d}{r}) = haversin(\phi_2 - \phi_1) + cos(\phi_1)cos(\phi_2)haversin(\lambda_2 - \lambda_1) \tag{5.1}$$

Equation (5.2) show the definition of the haversin function.

$$haversin(\theta) = sin^2(\frac{\theta}{2}) = \frac{1 - cos(\theta)}{2} \tag{5.2}$$

Table 5.1 shows the explanation of the different symbols

After the application of the inverse haversine function the distance between the two points depending on their GPS coordinates can be derived as shown in equation (5.3). [12]

$$\begin{aligned} d &= 2r\ arcsin(\sqrt{haversin(\theta_2 - \theta_1) + cos(\theta_1)cos(\theta_2)haversin(\lambda_2 - \lambda_1)}) \\ &= 2r\ arcsin(\sqrt{sin^2(\frac{\theta_2 - \theta_1}{2}) + cos(\theta_1)cos(\theta_2)sin^2(\frac{\lambda_2 - \lambda_1}{2})}) \end{aligned} \tag{5.3}$$

Table 5.1: Symbol Meaning

Symbol	Meaning
d	distance between the two points
r	radius of the sphere (in this case 6378.1 kilometres)
θ_1, θ_2	latitude of point 1 and 2
λ_1, λ_2	longitude of point 1 and 2

5.1.4 Role Selection

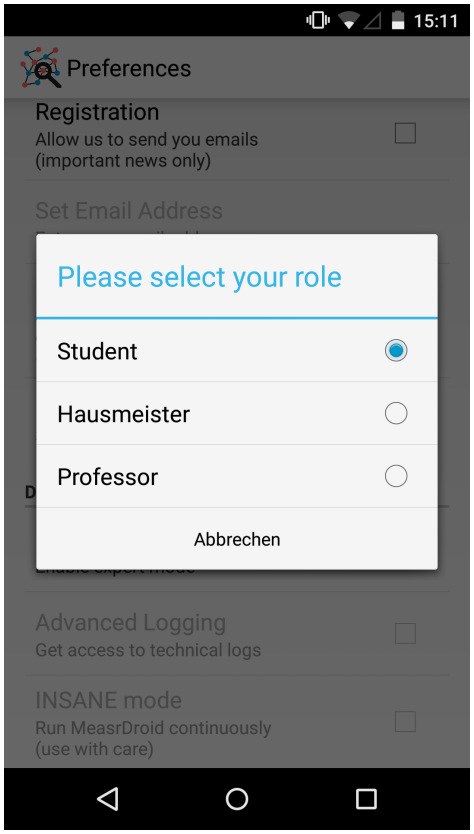
The application allows users to select their role. There are three different roles provided by the application: Student, Professor and Facility Manager.
An application that uses Measrdroid must display its wizard to inform the user about the fact that measurements are taken. Measrdroid provides the possibility to show further wizard pages to the user during the installation process. In general there are two different types of wizard pages: Obligatory and mandatory. The role selection is a mandatory page. The *RolePreferenceManager* tracks every change of this setting and must be set as an Preference Manager in the *@MeasrDroidSetup* tag. If the user persists his selection the inherited *preferencePersisted* method will be called, which ensures to call the *ForcePublishManager* to upload the new role preference as soon as possible.
A picture from the application that shows the role selection page is displayed in figure 5.1.

5.1.5 Graphical User Interface

The Android application uses the ActionBar pattern to provide an intuitive user interface. An ActionBar allows the user to switch between different fragment through swiping left or right. A Fragment is an independent page. Subscribed tags, proactive notifications and received notifications are fragments in the developed application. The application uses *ActionBarSherlock*[1] for the implementation of an ActionBar. *ActionBarSherlock* is an extension of the native support library and supports all Android versions from 3.x. These are more than 94.1% [2]of all Android devices. Notifications from publishers are displayed in message views. Every type of notification has its own independent message view and activity to dynamically customize the view programmatically depending on the message. The built-in view of the application shows for example a button with "More Information" printed on it, depending whether the message provides an non-empty url attribute.
New Message Views can be added easily. The class *ViewEnum* maps the *viewId* attribute

[1]http://actionbarsherlock.com/
[2]https://developer.android.com

Figure 5.1: Screenshot: Role Selection



which is provided by incoming notifications with the corresponding Activity. The *NotificationOverviewFragment* and *NotificationManager* use this enum to display the right view for every incoming message. In case that a third view needs to be added, a new Android Activity and its correspond XML layout must be implemented. Furthermore, a new entry must be added to the *ViewEnum*. It is the task of the Activity to store every response in the *NotificationDB* as a response object and uploads it afterwards through a call to the *ForcePublishManager*.

5.2 Publisher Manager

The *Publisher Manager* acts as a connector between the Android clients and the different Publishers. It uses MeasrDroid to communicate with the clients and provides a Rest Interface for publishers. It does also expect the publishers to implement a callable REST Interface. The Publisher Manager uses the Python Pyramid Web Framework that emphasizes its small size and its customizability. [13]

5.2.1 Authentication

The Publisher Manager provides a variety of services. But its capabilities should not be exposed to everybody. The implementation of an authentication mechanism is therefore necessary. HTTP Basic authentication (BA) is one of the widely used authentication mechanism in the world wide web. The transferred username and password are encoded with Base64 but neither encrypted nor hashed. BA should therefore always used in connection with HTTPS. The credentials are cached in the browser for 15 minutes. BA has also the disadvantage that username and password must be given in every request which increases the danger of an attack and the logout is only possible if the user closes the window. The Publisher Manager uses therefore a hybrid solution. The user authenticates herself once with basic http authentication and receives a *auth_tkt* session cookie in return. If another service is called, the Publisher Manager uses this cookie for further authentication.
At the time of writing there are two different permission types implemented to illustrate the access control. The user admin has the right to call interfaces with the permissions view and edit. The user viewer belongs to the group of spectators and has only view rights. The admin group has full control about the publisher and is allowed to perform every possible action.

5.2.2 RESTful Service

For the implementation of a RESTful Service the *Publisher Manager* uses the cornice REST framework. It provides useful helpers to build and document services. It also has a sensible default behaviour that follows best practices such as raising a Http Bad Request error if a non standard conform JSON object has been provided.[3] Furthermore, cornice takes care to check if the cookie of the user matches the specified permission.

The table 5.2 provides a full overview about the provided REST Interface and an short explanation for every address. All addresses are relative to the base url of the *Publisher Manager*. The addresses */publisher/register* and *publisher/sendMessage* expect complex data in the body of the request encoded as JSON. These services are therefore described in detail in the following chapters.

The *Publisher Manager* does also expect all of its Publisher to provide a REST Interface. Table 5.3 shows an overview of the expected interface.

[3] https://cornice.readthedocs.org/en/latest/

Table 5.2: REST Interface - Publisher Manager

Address	Type	Permission	Meaning
/publisher/register	POST	edit	Add a new Publisher to the Publisher Manager. Returns the id of the new Publisher
/measrdroid/upload	GET	None	Called by *C3PO* to indicate that new measurements are available
/publisher/sendMessage	POST	edit	Send a new message from a publisher to the subscribed clients
/publisher/info	GET	view	Get information about all stored Publishers
/request/info	GET	view	Get all requests send by one publisher. The url parameter publisherId specifies the said publisher.
/publisher/delete	POST	edit	The id of the publisher must be given in the body in the field publisherId
/role/info	GET	view	Show all available roles
/login	POST	None	Returns an *auth_tkt* session cookie
/logout	POST	None	Removes the session cookie from the browser

Table 5.3: REST Interface - Publisher

Address	Type	Meaning
/receiveMessage	POST	Signals that a new respond for this Publisher has arrived. The complete response object is provided as JSON in the body of the request.

5.2.2.1 Sending a Message

In listing 5.4 the JSON structure is shown that the publisher/sendMessage address requires. The *viewId* identifies the *Message View*. In the current implementation only the *viewId* one is occupied. It is noticeable that the provided *viewId* must be identical with the corresponding Message View Id that is stored in the *ViewEnum* class of the Android Application. It is the task of the programmer to ensure that these ids match. The *publisherMessageId* is given by the publisher to map the request and response of its messages. Longitude, latitude and radius specify a certain areal. These parameters are optional. The provided radius is interpreted in meter. The *url* attribute points to a website with further information. This field is also optional.

Listing 5.4: sendMessage

```
{
        viewId: "..",
        publisherMessageId "..",
        roleIds: ["..", ..],
        tagIds: ["..", ..],
        textMessage "..",
        longitude: "..",
        latitude: "..",
        radius: "..",
        url: ".."
}
```

5.2.2.2 Registering a new Publisher

To register a new Publisher a request with the body displayed in listing 5.5 must be sent to publisher/register. *Tags* is a list with the name of all Tags that the Publisher will manage. The *url* attribute describes the base url of the publisher. The Publisher Manager expects the Publisher to provide a RESTInterface under this url. Name and description are plain strings. The roles attribute is a list that expects object from type role.

Listing 5.5: register

```
{
        tags: ["..", ..]
        url: "..",
        description: "..",
        name: "..",
        roles: ["..", ..]
}
```

5.2.3 Communication with Android Clients

Due to MeasrDroids strict security architecture no direct communication with Android Clients is possible. Every measurement taken by the MeasrDroid Core is encrypted with a key so that only *C3PO* is able to decrypt the measurement. Also messages send to the client must be signed by *C3PO*. *C3PO* stores all incoming measurements that belong to *gui_id=5* in the directory /srv/measrdroid/incoming/ of the *Publisher Manager* and calls the /measrdroid/upload service to signal the presence of new data. MeasrDroid uploads

all new measurements every five minutes. The format of incoming measurement is displayed in listing 5.6.
In the error attribute every deactivated sensor is listed. Every Android client has a unique client_id. The client_id is a hash value over multiple hardware characteristics of the device. It is therefore possible to recognize clients even after a reinstallation of the application. MeasrDroid conducts a first measurement after the user has finished the installation of the application. This is the only time that also the gcm_id is included in the measurement. It is therefore important to store the gcm_id and the corresponding gcm_id. The core_ver specifies the version of the MeasrDroid Core. MeasrDroid puts all information provided by non-built-in components in the *gui* field of the JSON. The *data* field contains all data from external sensors. Currently, just the *GUISensor* is registered as external sensor and the *RolePreferenceManager* as preference manager. A response from the client contains a text and the appropriate tag. Also the subscribed Tags and role are sent to update the user entry in the database. The measurements are sorted by their timestamp and parsed in this order. This way the newest role selection will not be overridden by an older one. This is important because multiple measurements of role selections can arrive on the *Publisher Manager* at the same time as already mentioned.

C3PO checks the directory */srv/measrdroid/outgoing/* every three minutes for new messages. MeasrDroid expects messages in the structure as displayed in listing 5.7 and will discard any message that does not follow this specification without any error. The payload is in the context of this work a message that can be parsed by the *GoogleCloudMessagingManager* class of the application (see also 5.2)

Listing 5.6: Incoming Messages

```
{
        "error" : {
                ...
                },
        "result" : {
                "client_id" : "...",
                "conf_id" : "..",
                "core_ver" : 11,
                "device_id" : "...",
                "gui": {
                        "data" : {
                                "GUISensor" : {
                                        "responses" : "[{
                                                "response" : "...",
                                                "tags":[{"title": "..",
                                                "_id": ".."}, ...
                                                ]}
                                        }]",
                                        "subscribedTags" : "[
                                        {"title" : "...", _id : "..."},
                                            ...
                                        ]
                                }
                        },
                        "settings" : {
                                "role" : {
                                        "user_role" : "..."
                                }
                        }
                },
                "gui_id" : 5,
                "timestamp" : "...",
                ...
        }
}
```

Listing 5.7: Outgoing Message

```
{
        ids: ["..", ..]
        payload: {
                ...
        }
}
```

5.2.4 Database

The *Publisher Manager* must persist data to keep track of incoming and outgoing messages and the user base. In the following the decision for CouchDB and its integration is explained in detail.

5.2.4.1 CouchDB

The usage of a NoSQL database is a sensible choice for the *Publisher Manager*. The reason therefore lays in the fact that only JSON is used as a format for communication within the MeasrDroid Project. NoSQL databases do not have a relational database schema in contrast to relational databases. CouchDB belongs to the class of document-oriented databases which is a subclass of NoSQL databases. Those databases get the type of an entry from the data itself instead from an explicit entry in the database. Every document in the database has an unique id field. CouchDB does also provide a REST API for updating and querying the database. To integrate CouchDB into the Python Pyramid Web Framework the couchdbkit[4] database driver has been used. It provides many features such as schema definitions and representation of database entries as Python objects that are similar to dictionaries. Couchbkit is available on the Python Package Index (pip) and licensed under a Creative Commons Attribution 2.0 License[5].

5.2.4.2 Data Schemas

Couchdbkit allows to define CouchDB data schemas. Even if CouchDB allows the storage of JSON files that do not have any schemas, the application of schemas has advantages. In a data schema fields can have restrictions such as minimal or maximal size. Furthermore, they can be marked as required or optional. This approach allows the automatic validation of input data at storage time which leads to a more consistent database. The *Publisher Manager* has schema definitions for Roles, Tags, Users,

[4] http://couchdbkit.org/
[5] https://creativecommons.org/licenses/by/2.0/

Locations, Messages, Publishers, Requests and Responses. Listing 5.8 shows the schema definition of the user model. The *gmcId* and *clientId* are stored as Strings. One user can subscribe to multiple tags, the *tagIds* attribute is therefore a list of strings. Due to its small size, the role attribute can be stored as a *SchemaProperty* that copies this database entry into the document. In this case no second lookup of the id is necessary.

Listing 5.8: User Schema

```
class User(Document):
    gcmId = StringProperty(required=True)
    role = SchemaProperty(required=False, schema=Role)
    tagIds = StringListProperty(required=False)
    clientId = StringProperty(required=True)
```

5.2.4.3 Querying

CouchDB exposes an API to write views for querying the database. A view consists of a map and eventually a reduce function. The map function executes an operation on the whole dataset such as a selection. Listing 5.9 shows a map function that selects all documents with type user. The select function of other document schemas are analogously.

Listing 5.9: Map Function: User Select

```
function(doc) {
  if (doc.doc_type == "User"){
    emit(doc._id, doc);
    }
}
```

Listing 5.10 shows the selection of all users that have the same *clientId* as the key variable. The web server applies this view and passes the key through the following command from a Python application: *User.view('user/findUserByClientId', key=client_id)*.

Listing 5.10: Map Function: FindBy ClientId

```
function(doc) {
    if(doc.doc_type == "User") {
        key = doc.clientId;
        value = doc;
        emit(key, value);
    }
}
```

Through a reduce function it is possible to apply a function over the complete dataset. Nevertheless, this is not necessary in the case of the *Publisher Manager* web application.

5.2.5 Frontend

The *Publisher Manager* provides a frontend for administrative tasks such as adding new Publishers to the database or delete old ones. The frontend of the server consists only of static files, no rendering of documents is executed on the server. To provide an interactive user experience the JavaScript MVC framework AngularJS is used. AngualarJS is capable of two way data binding. This means, if the user changes data on the frontend the data does also change in the underlying service and vice versa. AngularJS provides a built-in HTTP service that is capable to call the REST Interface of the *Publisher Manager* via AJAX. The presented approach has many advantages. On the one hand it lowers the used performance on the backend, because no rendering must be executed. On the other hand the needed bandwidth is reduced, due to the fact that the *Publisher Manager* delivers only JSON files instead of complete HTML documents with redundant information after the page has loaded the first time. AngularJS has a strong ecosystem that provides access to many extensions. Those are installed with the frontend dependency manager bower.

5.3 Demo Publisher

To demonstrate the capabilities of the system a *Demo Publisher* has been implemented. The server uses the Express web framework for Node.js. Node.js utilizes the Google V8 runtime engine to execute JavaScript code outside the browser. The usage of Node.js has many advantages such as code sharing between client and server and non-blocking input and output calls. The Node Package Manager (npm) distributes thousands of production ready Node.js packages what accelerates the development of applications. [14, Chapter 8]
Node.js is therefore an ideal fit for the development of the *Demo Publisher*. The usage of two different web technologies for *Publisher Manager* and *Publisher* shows also that both architectures are completely independent from each other and only rely on the specified REST Interface.
Also the web interface of the *Publisher Demo Server* has been implemented with angularJS. Furthermore the socket.io package is used to establish a two-way socket connection between client and server.[6] This means, that the server can start the communication with the client or broadcast new events to all listening clients. This leads to a reduction of bandwidth in comparison to other approaches such as calling the REST Interface after a certain time interval to check for updates.

[6] http://socket.io/

The *Demo Publisher* uses bower as frontend package manager. To add a new dependency to the application a new entry must be added to the *bower.json* file. Bower is available on the Node Package Manager and licensed under the MIT License.[7]
To automate repetitive tasks such as minification, compilation, unit testing and linting the JavaScript task runner Grunt is used. Grunt is also available on the Node Package Manager and licensed under the MIT License.[8]

[7] http://bower.io/
[8] http://gruntjs.com/

Chapter 6

Evaluation

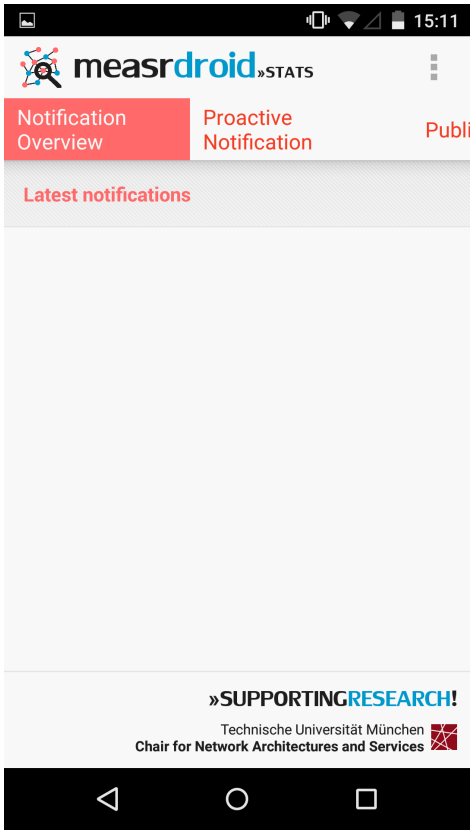
In the following the developed system is evaluated under different aspects.

6.1 Usability

The usability of the application (R8; c.f. section 3.3.2) is an important requirement. New users should be able to use the application without special training. To reach this goal the application follows conventions that the user are already familiar with if they have minimal experience in the use of mobile phone applications.
The application uses the ActionBarSherlock library as described in chapter 5.1.5 to apply the ActionBar interface pattern. The action bar of the Application is divided into three different action bars. The main action bar displays the MeasrDroid branding and indicates a menu through three points in the upper right corner. Under the main action bar the top bar displays three different Categories: Notification Overview, Proactive Notification and Publisher. The user can switch between them through swiping left and right. The bottom bar displays another branding for the Chair for Network Architectures and Services of the Technische Universität München. Summarizing it can be said that the application follows the Google Style Guide for mobile applications. [15] Figure 6.1 shows the ActionBar of the final application.

The developed application uses the same colour scheme and layouts as the MeasrDroid App does which is available on the Google Play Store. On this way a recognizable brand is established.

Figure 6.1: Screenshot: ActionBar



6.2 Latency

Latency is the time interval between a request and the matching response. Only through a low latency Real Time Applications are realisable (R7; c.f. section 3.3.2). In the following the latency of the developed system is analysed. If a Publisher sends a request to its clients, it sends the message to the *Publisher Manager* with a call of its REST Interface. The *Publisher Manager* validates the new message and puts it into a certain directory in the file system. *C3PO* checks this directory for new new messages every three minutes. Afterwards, MeasrDroid signs the message so that the *MeasrDroid Core* is able to approve the message. *C3PO* pushes all new messages to *gcm.droid* every minute where messages are send from to clients via GCM. The time GCM needs to deliver a message highly depends on the network connection of the client. The client needs also one to two minutes to conduct a GPS measurement that is only necessary if the message depends on the location of the user. The response of the user is uploaded to *upload.droid* immediately after the input. *C3PO* checks *upload.droid* every three minutes for new messages, puts them every three minutes in a special directory of the *Publisher Manager* and calls a REST API to signal the incoming of new messages. An overview about the

described process and times is depicted in figure 6.2.
In the worst case a message needs ten to twelve minutes from the publisher to the client and back, assuming that the user responses immediately to the new request and GCM delivers the message without an additional delay.
A main part of the high latency is caused by the usage of the MeasrDroid backend around *C3PO*. Without the use of MeasrDroid the latency is the sum of usual network latency and the time for a GPS measurement.

6.3 Privacy

Privacy is also an important requirement for the developed application (R5; c.f. section 3.3.2). The application uses the MeasrDroid system to take measurements and to transfer information from the client to the server and vice versa. On the one hand *MeasrDroid Core* ensures that every incoming message has been decrypted by *C3PO*. On the other hand *C3PO* accepts only messages that have been signed by the *MeasrDroid Core*. Thus, Publishers that use the *Publisher Manager* to communicate with the client do not need to establish their own secure connection. Every publisher is as trustworthy to the client as MeasrDroid itself.
The determination of the user position happens on the device itself as explained in chapter 5.1.2. Therefore, no active tracking of the user position is necessary.

6.4 Extensibility

Extensibility is according to 3.3.2 requirement R7. The system consists of Publisher and Subscribers. The architecture must ensure that both parties can be extended by new functionality easily. On the one hand Publishers can be added, edited and removed easily through the *Publisher Manager*. There are no restrictions regarding technologies for a Publisher, the developer is free to choose a language and framework she feels comfortable in. It is also possible to integrate third party services into a Publisher. In general, it is not necessary to have knowledge in the development of Android applications to use the presented framework.
The developed Android application on the other hand can be enriched with new *Message Views*. The process to do so is described in detail in chapter 5.1.5. New *Message Views* can enrich the user experience trough new interactive content. Furthermore, the system does also provide the capability to send an url to the user that links to a website with further information.

All in all it can be said that the developed system fulfils this requirement. The REST API of the *Publisher Manager* provides a flexible way to communicate with the clients

Table 6.1: Evaluation Summary

Name	Explanation	Evaluation
Publisher Subscriber	Pattern has been fully implemented	++
Role-based Channels	Role based messages and channels are possible, but the user can freely choose her role without authentication.	+
Location Awareness	The Android Clients are capable to measure the exact location with MeasrDroid.	++
Security	It is the task of the Publisher to ensure the absence of false information.	∘
Privacy	MeasrDroid ensures the integrity of user information. The API does not expose any user locations to the Publishers.	++
Real Time Interaction	MeasrDroids backend infrastructure is only capable to deliver and receive messages with a high latency (10 - 12 minutes)	- -
Extensibility	New Publisher and Subscriber can be added easily. The addition of new roles is slightly more complicated	+
Usability	The Application follows best practices and common design patterns to provide an intuitive user experience.	++

with as less restrictions as possible. Also the usage of JSON as primary communication format is well chosen due to its widespread distribution it has parsers in nearly all common languages. The Android application on the other hand can be extended with new views.

6.5 Summary

The table 6.1 gives a quick overview over the results of this chapter. The requirements have the same order as in chapter 3.3.

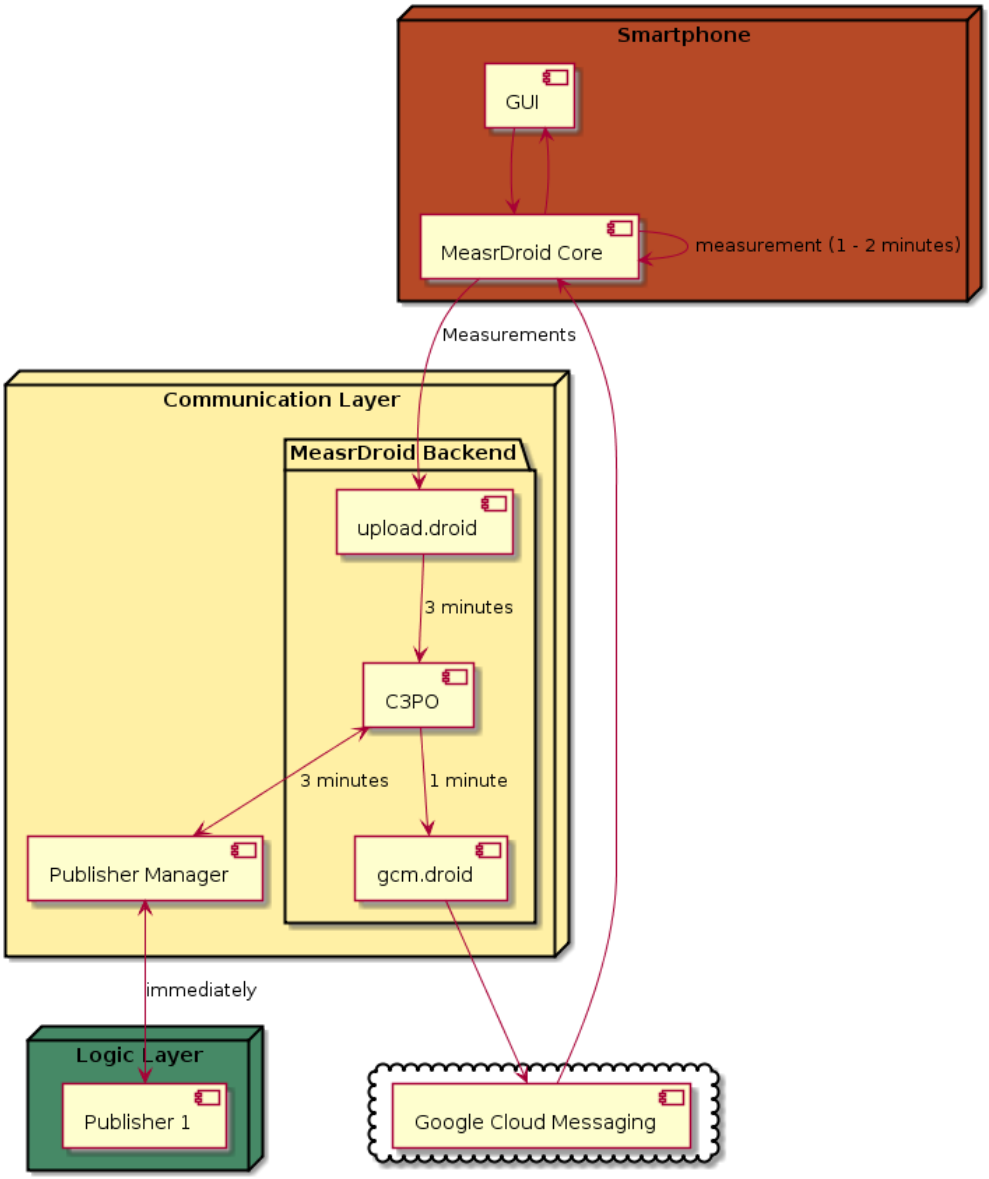


Figure 6.2: Overview Latency

Chapter 7

Conclusion & Outlook

The overall goal of smart buildings is the improvement and optimization of living and working environment. In order to achieve this goal connectivity is a crucial factor: A variety of devices and users with different tasks and authorization must communicate with each other. To motivate user to participate actively in the communication a user-friendly mobile phone application has been developed. New Users can be easily added to the system through the installation of an Android Application that guides them through all important steps of the setup process. (RQ1) In order to avoid an overwhelming amount of information the Publisher/Subscriber pattern has been applied to let the user freely choose between all available Publishers for her role. (RQ3) The described system allows to easily extend the system by further *Publishers* that use a REST API to send and receive messages form the clients. The developer of such a *Publisher* is not restricted to any technology and can use the system without any knowledge of mobile application development. New Publishers must be registered on the *Publisher Manager* and are then able to receive messages from and send to their user base.
The communication between subscriber and publisher is completely encrypted by the MeasrDroid framework. The API of the *Publisher Manager* exposes as less data as possible to its *Publishers* to ensure the integrity of the collected user data. As a result of this effort the creation of a movement profile is not possible. (RQ3, RQ4)
Also a demo Publisher is implemented that uses the Express Framework for Node.JS. This publisher provides an easy user interface to send messages manually to clients and serves as a foundation for further developments.

The presented system is a proof of concept that the different components of MeasrDroid can be used outside of their intended domain. Nevertheless, there are also aspects of the implementation which need further improvement to provide a solid solution that can be used in a real environment.
One important aspect of the system is the role system. The current system allows all users to select their role freely. Of course, this mechanism must be exchanged for a

service that provides a user authentication mechanism.
The system provides two *Message Views* at the moment. One to respond to incoming requests and another one for sending proactive notifications. The application can be extended in the future with new *Message Views*. A detailed explanation about the existing *Message Views* can be found in chapter 5.1.5.

In chapter 6 it has been shown that it take at least twelve minutes to get a response after the client has send a message. The reason therefore lays in the fact that the architecture of MeasrDroid is intended for non-time-critical applications and a high number of users. Its strict security policy prohibits the development of real time applications. In order to address this issue a new backend for the *MeasrDroid* Project must be developed. The new server must have a less strict security policy that allows the direct sending of data from the server to the client and vice versa. In this case no changes to the *MeasrDroid Core* component would be necessary.

Appendices

Appendix A

Glossary

- **C3PO** C3PO is the main server of the MeasrDroid backend. It is not connected to the Internet. C3PO is not callable from the outside, every communication must be initialized by it. Data must be transferred through the file system.
- **MeasrDroid Backend** The MeasrDroid Backend consists of multiple servers. It persists all client measurements in a database and provides further features such as configuration files for clients.
- **MeasrDroid Core** The MeasrDroid core is part of the MeasrDroid Framework. It can be used to develop Android applications which use the capabilities of MeasrDroid. It provides a programmatic way to take measurements using a variety of sensors.
- **Measurement** A Measurement is the only way for a client to communicate with the MeasrDroid Backend.
- **Message View** Publisher can specify a Message View. Message Views are native Android views and provide a way to extend the presentation of data on android clients. At the time of writing there are two views already implemented.
- **Publisher** A Publisher is a service communicates with users using the REST Interface of the *Publisher Manager*. It does not communicate with MeasrDroid directly.
- **Publisher Manager** The *Publisher Manager* takes messages from Publishers and uses MeasrDroid to deliver them to the clients.
- **Publisher Notification** A Publisher Notification is message sent from a publisher to a subscriber via the *Publisher Manager*. Whether a subscriber receives the message or not depends on her role and the subscribed tags.
- **Role** A Client can only have one role. Roles help the *Publisher Manager* to

understand which Publishers and Tags can be subscribed from Subscribers. The roles provided by the current implementation are facility manager, student and professor.

- **Subscriber** A subscriber is a user of the Android application. She has a role and can subscribe to multiple tags.
- **Tag** One Publisher can have multiple Tags, which can be attached to communication events. Tags help the *Publisher Manager* to send the right information to subscribers. The Subscriber can subscribe to tags which are grouped by their publishers.

Bibliography

[1] R. R. Brad Brech, "Smart cities series: Understanding the ibm approach to efficient buildings," pp. 1 – 5, 2011, accessed: 2015-07-14. [Online]. Available: http://www.redbooks.ibm.com/redpapers/pdfs/redp4735.pdf

[2] M. Rerych, "Wasserfallmodell : Entstheungskontext," 2007, accessed: 2015-07-14. [Online]. Available: http://cartoon.iguw.tuwien.ac.at/fit/fit01/wasserfall/entstehung.html

[3] C. Group, "Smart 2020: Enabling the low carbon economy in the information age," pp. 40 – 55, 2008, accessed: 2015-07-14. [Online]. Available: http://www.smart2020.org/publications/

[4] S. Z. A. für Hochbauten, "Schlussberihct nutzverhalten beim wohnen: Analse, relevanz und potenzial von massnahmen zur reduktion des energieverbrauchs (effizienz und suffizienz)," pp. 21 – 46, 2011, accessed: 2015-07-14. [Online]. Available: http://www.mehralswohnen.ch/fileadmin/download/1107_Bericht_Nutzerverhalten.pdf

[5] M. W. Sabine Veth, "Umweltbewusstsein in deutschland 2008," pp. 39 – 45, 2008, accessed: 2015-07-14. [Online]. Available: http://www.umweltbundesamt.de/sites/default/files/medien/publikation/long/3678.pdf

[6] D. K. S. Dr. Jutta Emig, "Umweltbewusstsein in deutschland 2012," pp. 42 – 50, 2012, accessed: 2015-07-14. [Online]. Available: http://www.umweltbundesamt.de/sites/default/files/medien/publikation/long/4396.pdf

[7] M. Faath, "Analysis of content delivery networks with an Android-based measurement framework," Master's Thesis, Technische Universität München, Munich, Germany, 2013, Advised by Dipl.-Inf. Johann Schlamp.

[8] "Measrdroid on the google play store," accessed: 2015-07-14. [Online]. Available: https://play.google.com/store/apps/details?id=de.tum.in.net.measrdroid.gui.stats

[9] IDC, "Smartphone os market share, q1 2015," 2015, accessed: 2015-07-14. [Online]. Available: https://www.idc.com/prodserv/smartphone-os-market-share.jsp

[10] L. Krug, "A System for giving Practical Advice for Energy Savings Based on Sensor Information," Bachelor's Thesis, Technische Universität München, Munich, Germany, 2015, Advised by Holger Kinkelin, Marcel von Maltitz.

[11] "Measurement overview," 2015, accessed: 2015-07-14. [Online]. Available: http://www.droid.net.in.tum.de/about/measurements

[12] U. S. C. Bureau, "Geographic information systems faq„" accessed: 2015-07-14, Repost, Linked has been removed. [Online]. Available: http://www.movable-type.co.uk/scripts/gis-faq-5.1.html

[13] "The pyramid web framework," 2015, accessed: 2015-07-14. [Online]. Available: http://docs.pylonsproject.org/projects/pyramid/en/latest/

[14] M. W. Tom Hughes-Croucher, *Node: Up and Running*, 2012.

[15] "Action bar," 2015, accessed: 2015-07-14. [Online]. Available: https://developer.android.com/design/patterns/actionbar.html