BA, Boro Mitrovic

Design and Development of a Framework for Informal Caregiver Decision Support and the Evaluation in the AAL Domain

Master's Thesis

to achieve the degree of

Dipl.-Ing

Individual Master

Biomedical Electronics

submitted to the

Technical University of Graz

Supervisor

PhD, MSc, MPh, BEng, CEng, DipEd, MBCS, Assoc.Prof.Dr.Andreas Holzinger

Institute of Information Systems and Computer Media

Graz, January 20, 2015

BA, Boro Mitrovic

Design und Entwicklung eines Frameworks zur Entscheidungsunterstützung für pflegende Angehörige und die Evaluierung im AAL-Bereich

Masterarbeit

zur erlangung des akademischen Grades

Dipl.-Ing

Individuelles Masterstudium

Biomedical Electronics

eingereicht an der

Technischen Universität Graz

Betreuer

PhD, MSc, MPh, BEng, CEng, DipEd, MBCS, Assoc.Prof.Dr.Andreas Holzinger

Institut für Informationssysteme und Computer Medien

Graz, 20. Jänner 2015

Abstract

The development of Europe's population shows an aging community with a simultaneous decrease in mortality and fertility rates. Furthermore, an increased number of elderly prefer to live and receive care within their familiar environments at home. Most of this care is provided by caring family members. The management of both work and caregiving, at the cost of a shortened private life, imposes significant strain and stress on caregivers. In addition, caregivers can experience mental stress from constantly worrying about the well-being of the beloved person.

The work in this thesis provides a technical support for caregivers in these situation. For this reason, the cared-for person's home is outfitted with a solution for home automation, which enables the sending of requests to a caregiver. For the caregiver, a mobile solution was developed to receive these requests. The framework was designed to provide casual communication between cared-for person and caregiver. The design of the solution for the cared-for person involves the use of a small and convenient Advanced Risc Architecture (ARM) board, such as the Raspberry Pi and similar products. The general purpose of this thesis is the contribution of an experimental setup to the scientific world, in the field of Ambient Assisted Living. Therefore, preliminary research in the field of ARM technology in consideration of Ambient Assisted Living and home automation, and a prolonged investigation on market-available ARM boards, was performed. The researched related work confirms the decision to use ARM related hardware for Ambient Assisted Living.

For the home automation software, the feasibility of running the OSGi based HOMe Event Recognition system (HOMER) on ARM platforms was tested. To implement the concept for the cared-for person, an OSGi bundle was developed for the compatibility with the home automation software. In regards of the caregiver, a mobile solution to run on the Android Operating System (OS) was created.

The described concept was developed as a first prototype in the course of this thesis and is ready for a trial with end users. For future work, the results provide a fundamental basis for subsequent projects in the field of Ambient Assisted Living.

Keywords

Ambient Assisted Living, Android, Embedded systems, Home automation, Information technology, Mobile computing, Near Field Communication, OSGi, Telecommunications

ÖSTAT classification

202022 202003 202017 202038

ACM classification

Automation, Embedded systems, Mobile computing, OSGi Alliance, Telecommunications, Wireless devices

Kurzfassung

Die Entwicklung der europäischen Bevölkerung tendiert zu einer immer älter werdenden Einwohnerschaft, bei gleichzeitigem Rückgang der Mortalitäts- und Geburtenrate. Hinzu kommt, dass immer mehr ältere Menschen länger in ihrer gewohnten Umgebung verbleiben wollen und nicht unüblich dort von Familienangehörigen gepflegt werden. Der Balanceakt zwischen täglicher Arbeit und Pflege, ist eine erhebliche Belastung für pflegende Angehörige. Des weiteren können Sorgen um das Wohlbefinden der zu pflegenden Person zusätzlichen mentalen Stress ausüben. Im Rahmen dieser Masterarbeit wird ein Konzept vorgeschlagen, dass eine Unterstützung für die pflegenden Angehörigen bietet und zusätzlich die zu pflegenden Person im Alltag unterstützt.

Das Konzept besteht aus einer Heimautomatisierungsanwendung für die zu pflegende Person, die es ihr ermöglicht, Anfragen an eine Bezugsperson zu senden. Die Bezugsperson wird mit einer mobilen Lösung ausgestattet, die die Anfragen der zu pflegenden Person empfangen kann. Die Kommunikation soll eine zwanglose Verständigung zwischen pflegender Person und Bezugsperson ermöglichen. Das Konzept für die zu pflegende Person beinhaltet die Verwendung einer ARM Plattform, wie dem bekannten Raspberry Pi Board und ähnlicher Hardware. Allgemein kann der Beitrag dieser Arbeit als Versuchsaufbau im Bereich des Ambient Assisted Living betrachtet werden. Zu diesem Zweck wurden vorausgegangene Forschungen auf dem Gebiet der ARM-Technologie im Zusammenhang mit Ambient Assisted Living und Heimautomatisierung analysiert. Des weiteren fand eine Analyse, der am Markt verfügbaren ARM Boards statt. Die Recherche von ähnlichen und vorausgegangene Arbeiten bekräftigte die Verwendung von ARM gestützter Hardware für den Ambient Assisted Living Bereich.

Als Software für die Hausautomatisierung auf der Seite der zu pflegenden Person wurde das auf OSGi basierende HOMe Event Recognition System (HOMER) verwendet. Vorausgehend wurde die Möglichkeit der Ausführung der verwendeten Software, auf den ausgewählten ARM Plattformen erprobt. Aufgrund der OSGi basierenden Struktur der Software wurde ein OSGi Bundle für die Umsetzung des Konzepts für die zu pflegende Person, entwickelt. Hinsichtlich der Bezugsperson wurde eine Mobile Lösung auf Basis des Android Betriebssystems entwickelt.

Das beschriebene Konzept wurde im Verlauf dieser Master Arbeit zu einem ersten Prototypen entwickelt und ist bereit für erste Testversuche in einer realen Anwendung. Hinsichtlich künftiger Projekte die auf dieser Arbeit aufbauen, wird eine fundamentale Grundlage bezüglich der Anwendung von ARM Technologie im Bereich des Ambient Assisted Living geliefert.

Schlüsselwörter

Ambient Assisted Living, Android, Embedded systems, Heimautomatisierung, Informationstechnik, Mobile computing, Near Field Communication, OSGi, Telekommunikation

ÖSTAT Klassifikation

202022, 202003, 202017, 202038

ACM Klassifikation

Automatisierung, Embedded systems, Mobile computing, OSGi Alliance, Telekommunikation, Drahtlosegeräte

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, January 20, 2015

_____

Boro Mitrovic

Acknowledgements

My gratitude goes to Professor Andreas Holzinger from the Institute of Information Systems and Computer Media at the Technical University of Graz, for the possibility to write this thesis and his enthusiasm towards my work.

Further, my appreciation goes to the Biomedical Systems unit of the Health & Environment Department of the Austrian Institute of Technology (AIT) in Wiener Neustadt, for the opportunity of this master thesis. Special thanks go to, Dipl. Ing. Martin Morandell, for his guidance and support, as well as to Dipl. Ing. Jonathan Steinhart, for the revision of my thesis and Dipl. Ing. Emanuel Sander, for his support during this master thesis and his friendship over many years. Moreover, I want to express my appreciation for the possibility to take part in the RelaxedCare project. I enjoyed working with the RelaxedCare team, whereat I would like to acknowledge the AAL Joint Programm and the national funding authorities and R&D programs in Austria (bmvit, program benefit, FFG), Switzerland, Slovenia and Spain, which co-founded the RelaxedCare project.

My deepest gratefulness, is for my family. I especially want to thank my parents, for giving me the opportunity to study and their loving support. Moreover, I want to express my gratefulness to my sister, for her caring and open ear. I also owe special thanks to all of my friends. Thanks to them, I can look back to wonderful memories, special moments and bonding, during my student life.

My last, but dearest gratefulness, is for the person, which holds the one special place in my heart. Aleksandra, I am grateful that you were at my side through the ups and downs of this journey and I hope that you will be by my side, for the adventure which lies ahead of us.

Boro Mitrovic

Graz, January 20, 2015

Table of Contents

1. Introduction and Motivation for the Research in the Field of AAL

A major impulse for this project is the fact that the European population is growing older, precipitated by the simultaneous decline in mortality- and fertility rates. The percentage of individuals over 65 years is estimated to be 30%, with those 80 years and over predicted to reach 11.4% by 2050. These statistics lead to the implication that the demand for care of elderly people will rise drastically (Bolin et al., 2008). Another study shows that the demographic development displays an increase in proportion of the older population, together with an increase of single households (Giannakouris, 2008).

A further fact is that 80% of people in Austria who are in need of care receive this care by informal caregivers (ICs) at home, with 30% of these ICs being employed. Including ICs who worked up until needing to provide care at home increases this to 56% of all questioned caregivers. (Pochobradsky et al., 2005). Managing the care of loved ones while working or travelling large distances puts an enormous burden on the caregiver. Furthermore, worrying about the care recipient triggers additional mental stress.

A solution may be found in the approach of Ambient Assisted Living (AAL). AAL involves products and services, as well as concepts, to increase and secure the quality of life, through the use of information and communication technology. It should provide autonomy for the care recipient and nourish the interaction with their social environment (Georgieff, 2008).

Caregivers who consider technology as a support in caregiving, expect to reduce the stress and make care giving logistically easier through the applied technology. In addition, they expect to make the assisted person (AP) feel safer and to increase the

feeling of being effective, as well as to save time (National Alliance for Caregiving, 2011). An unobtrusive way of keeping the IC informed about the current situation of the AP, might be in the best interests of both parties. The proposed system should provide information for the IC without the need of a phone call or visits, since repeated checks on the well being of the AP can become a nuisance. The system should be able to unobtrusively and automatically monitor the condition of the AP. It should remove the burden and interruption in the daily life of the IC and at the same time reduce the need of additional caregivers, in addition to upholding a constant and yet flexible relationship between the AP an the IC.

The RelaxedCare project RelaxedCare (2014), aims to meet these expectations. An additional objective is to maintain an open system and therefore allow outside developers and organizations to extend the platform and add new sensors. The idea is to offer a system in a box, which does not need any effort from the user to collect data or to require an expert to interpret the output. Although the RelaxedCare system provides a connection between ICs and APs, it is crucial to point out that it is not intended as an emergency application. The design of the project bases all provided services upon existing AAL platforms, such as the Home Event Recognition (HOMER) system (Morandell et al., 2014). HOMER is a software platform, which provides a variety of functions for the control of a smart home. Furthermore, the software is free for developers, as it is intended to be open source.

As explained the RexlaxedCare project intends to connect elderly persons with ICs and with their social environment under the above stated premisses. The present contribution to the RelaxedCare project, as well as one of the challenges of this thesis, was an experimental set-up, based on Advanced Risc Machine (ARM) architecture boards, like the well known Raspberry Pi, which could fully run the aforementioned HOMER platform.

With the increasing penetration of technology into private spheres, this specific technology must satisfy the various needs of the end user group in order to be accepted by this group and therefore to be successful (Röcker et al., 2011).A reasonable level of acceptance can only be expected if the offered technology respects privacy limits, and provides an unobtrusive and invisible design (Ziefle et al., 2011). The beneficial small sizes of the boards enable the designers to integrate the system into everyday objects. The users may have a higher acceptance of new products if the objects look

familiar, e.g. a vase.

A further interest of this thesis was to develop a system for a scenario, where the care recipients can send a request (for example "I want to go shopping") to the care giver, without the need of a phone call or to write a text message by themselves. The system should work with the HOMER platform on the ARM based boards, and on the IC side an Android phone should be used. The action required to send the request by the AP, should be designed in a simple manner and consume very little time, as well as to be flexible to fit the specific need of every care recipient.

2. Background

In the field of AAL there are a myriad of possibilities how to engineer solutions for the needs of elderly people and with the development of new technologies the scope of this possibilities widens. The RelaxedCare project aims to be open and thus gives developers the opportunity to implement such new technologies or to use existing solutions in a different context for AAL tasks. RelaxedCare implies a number of tasks, or scenarios which have the purpose to help with actions the AP wants to perform or to support the caregiver.
One of these mentioned scenarios is the "Assistance Request scenario", which was developed and taken to a first prototype in the course of this master thesis. The upcoming part focuses on the concept of the Assistance Request scenario and provides the background for further chapters.

2.1 Assistance Request Scenario

The Assistance Request scenario provides a casual communication between an AP and an IC. The idea is to offer the AP a possibility to express a request for an assistive action the AP needs help with, or wants to be performed. The type of request is of a not urgent nature. For instance the AP could express the wish to go shopping or to be called. The pool of requests, an AP can resort to, should depend on the needs of the AP, anticipating that the amount of requests as well as the requests themselves should be flexible.
The way of communication should not force any participant to answer immediately. In addition, the AP should be able to express the request without the need of a phone call or to write a text message. Furthermore, the device, which enables the

communication with the IC, should integrate in the existing interior. The device should not interrupt the everyday life of the AP and the act of sending a request should not exert any strain on the AP.
The communication design should consider a distraction free notification of incoming request on the IC's side as well. In addition, the IC should be able to choose when to receive requests. The design should be mobile, which enables the IC to receive requests anywhere. The choice of reacting on a request should freely depend on the IC's decision. For convenience, the IC should be able to have an overview of all the requests the AP expressed.

Use Cases

The use cases, shown in Figure 2.1, should provide an easy to understand overview, for the Assistance Request scenario. As already mentioned, the decision of receiving

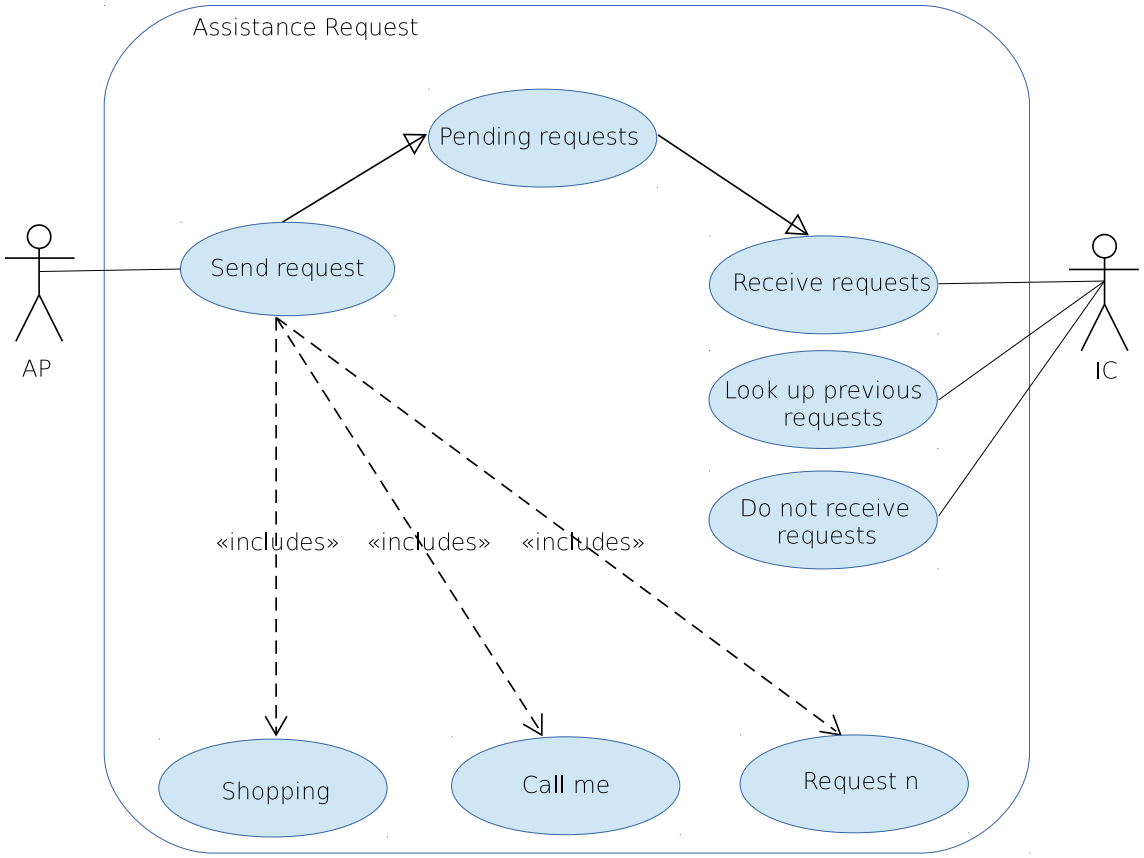


Figure 2.1: The figure illustrates the use case for the Assistance Request scenario.

a request should be determined by the IC . This shows the need for an intermediate layer, between the AP and the IC, to hold the requests till the IC chooses to permit the reception of requests.
To briefly summarize the scenario, on the one hand, the AP is able to send requests to the IC, while on the other hand the IC can determine when the requests are received. As well has the IC the possibility to look up the previously received requests.

2.2 Requirements for Thesis

The previously discussed demands for the Assistance Request scenario, lead to the upcoming challenges, dealt within the scope of this thesis.

2.2.1 Concept development

The introducing work, performed within the course of this thesis, was the development of a concept for the Assistance Request scenario. It was predefined that the HOMER platform has to be applied as the home automation software. In conjunction with the stated requirement, to fit into the interior of the AP's home, a small device had to be selected capable of running the HOMER platform. The concept of a request input for the AP, had to be developed as well as a hardware, which meets the demands for this concept.
For the receiving side, a concept for a mobile solution had to be created, which fits the formerly mentioned demands for the IC.

2.2.2 Selection of hardware platform

A hardware platform for the AP was needed, powerful enough to execute the HOMER platform and at the same time to provide small dimension, to enable designers to integrate the hardware into everyday objects. For the selection of a hardware platform the research of existing hardware possibilities had to be performed. The research was focused on process power and connection standards as well as extensibility. Furthermore, hardware for the request input, which features a compatibility with the selected hardware platform, had to be found.

2.2.3 Implementation of the additional components of the HOMER software

The implementation of the HOMER software, preceded a preliminary preparation of the selected hardware platforms. A low level software implementation for the request input hardware was needed to make the input hardware operational with the selected hardware platforms.
To add the Assistance Request scenario to the existing HOMER software an additional software bundle had to be developed. As the finishing work a mobile client on the IC side of the Assistance Request scenario had to be written.

3. Related Work

Projects similar to RelaxedCare, which aim to assist the elderly and provide status updates for ICs, are described in the work of Roush (2009) and Hanson et al. (2011). Further similar solutions are GrandCare Systems (2014); Rest Assured® (2014); Live!y (2014); SimplyHome (2014); Sonamba (2014) as well as Pers+™ (2014) and Doumas (2014). However as stated in the work of Morandell et al. (2014), these solutions differ in functionality, cost and availability. Furthermore, a deeper insight on the respective solutions, in regards to technical specification is hard to obtain, since the mentioned solutions are strictly of commercial nature and thus such information is apparently proprietary.

As this thesis serves as an research and experimental platform for the RelaxedCare project, it contributes an experimental set-up for the Assistance Request scenario. The work done in this thesis has two crucial points. The first point is the ARM board, which is used as the control unit on the AP side and the second point is the Android application on the IC side. Thus this chapter will focus on related work on this two topics, in the AAL and home automation domain.

3.1 ARM in Home Automation and AAL

An earlier example for home automation and AAL can be found in the work of Chan et al. (1995). The author describes a smart home environment, where different types of sensors are used for data acquisition. The collection and analysis of the acquired data is executed through a PC.

A major difference in comparison with current solutions, is the standalone philosophy. Faster traffic speed and the idea of cloud computing made a big impact on

home automation and AAL. Further, progress in the field of ARM architecture led to ARM single-board computer with enough process power for home automation solutions, with the benefit of a small size. Today the computational effort done by the above mentioned PC could be easily managed by a Raspberry Pi.

It is possible to use ARMs as the control unit for home automation applications, like in the work of Liu (2006). The proposed home automation solution uses a processor based on an ARM7 core for the main control unit. However the electrical appliances can only be turned on or off. Further the absence of a wireless implementation to connect the electrical appliances with the main unit is a big disadvantage, although the system provides a simple installation and a cost effective design.

A remote video surveillance concept, based on ARM, for home automation was designed in the work of Mei et al. (2011). An ARM is used as the control unit for the video surveillance and in advance to send images to the habitants mobile phone. The system is capable of detecting a moving target and to inform the habitant of an intrusion.

According to the author the disadvantages are the robustness of the system, in regards of target-detection, when the target's moving speed sharply changes and again the absence of a wireless implementation.

In the paper of Kovacshazy et al. (2013) a prototype for an IT smart building monitoring solution was tested. Several sever rooms where monitored and supervised in regards to energy, temperature and humidity as well as the operational state and occupation of the server room. The computation of the data was performed among other solutions on a Raspberry Pi (ARM11) and a Beaglebone XM (ARM8). The design was tested and a benchmark was applied for the different concepts.

A wireless implementation of a smart home based on Zigbee and ARM is proposed in the study of Yi et al. (2013). The used environmental sensors and actuators are connected through a Zigbee network with the ARM9 based control unit. In addition, a GSM module is used to provide a possibility to check on the home status. The user therefore sends a short message with a predefined format to the system, which in response sends a status short message back to the user.

Another relevant project is described in the paper of Igarashi et al. (2014). The design uses a Raspberry Pi as the main control unit for home automation. As the Raspberry Pi can lack in process power for resource intensive applications, the paper

proposes a cloud processing solution.
All used applications are developed as full applications and as applets. The applets is described as a resource saving version of the full application. As an example a full application would use video processing for face recognition to unlock the front door, while the applet only requires an authentication using a key pad.
If the full application or the applet is used, depends on the situation and availability of process power. The full application may be executed on the Raspberry Pi if enough process power is available and in case previous applications consume already a vast amount of process power, the applet is executed on the board or the full application in the cloud.
Which version of an application is executed and where it is executed is determined by the System-Wide Scheduler. The described paper illustrates an interesting combination of cloud computing and the use of a small convenient ARM board.
The control unit in the work of Jain et al. (2014) is likewise realized through a Raspberry Pi. An experimental setup is described, where home appliances, simulated by LEDs, are controlled through email traffic. The user sends an email with a specific subject, which is recognized by the system. The email is read by a developed algorithm, which executes the corresponding instructions. The major benefit of the proposed solution is the highly cost efficient design.
Another low cost solution in the field of home automation is proposed in the paper of Prasanna and Ramadass (2014), where speech recognition is used to control home appliances. The applied speech recognition is executed offline through the implementation of the Hidden Markov Model Tool-kit on a Raspberry Pi. Through a GSM mode the recognized commands are transmitted to a microcontroller, which operates the home appliances. The authors contemplate the work as support for the elderly and impaired.
All so far mentioned papers show that ARMs are able to handle the computational load necessary for home automation, which endorses the decision, proposed in this thesis to use ARM boards as the main core unit, on the AP side.
Beside the use as core units, ARMs are often used as gateways for home automation and AAL solutions, as proposed in the paper of Cubo et al. (2014); Gebhardt et al. (2014) and Alhamoud et al. (2014), as well for supportive services, like a mobile wearable device with a pico projector for augmented reality, designed in the work of

Saracchini and Ortega (2014).
Another technology that has a high potential for AAL applications is the operating system Android. The diversity of available hardware platforms and a high popularity as well as the openness of the software is beneficial for the AAL domain.

3.2 Android in Home Automation and AAL

The work of Boulos et al. (2011) states that the amount of applications downloaded for smartphones, from 2009 to 2010 goes up from three hundred millions to five billion applications. The paper further describes an EU-funded project, the Enhanced Complete Ambient Assisted Living Experiment (eCAALYX), which is based on an Android application and a wearable smart garment with wireless health sensors, for older people with multiple chronic conditions.
The mobile Android platform acts as an middleware between the wearable health sensors and a health professional's internet site. Alerts and measurements obtained from the health sensors, as well as geographical location, from the smartphone's GPS, are transmitted by the android application to the internet site.
Furthermore, the application is capable of identifying higher level information from the provided sensor data, such as tachycardia and signs of respiratory infections, depended on established medical knowledge.
In the field of home automation the paper of Zhu et al. (2012) shows that it is possible to completely base home automation on Android. The paper suggests a design, where the user is enabled to control all home appliances through a mobile Android terminal, that communicates through a GSM network with a home info center, which likewise uses Android as the operating system. The home info center in succession operates the home appliances through a wireless module.
Continuing in the field of home automation, a KNX-based approach was developed in the work of De Luca et al. (2013), which is controlled by an mobile android device. The development of the android application considers a flexible design, which enables a technician to build the structure of the Android application to fit the applied sensors and actuator for the home automation system. The management of home devices by the android application can be achieved through the local network or remotely through the internet.

In addition, an Linux authentication server connected to an NFC reader is used for access control. The authentication is accomplished by approaching the smartphone to the NFC reader, which transfers the necessary information to the authentication server. The used information consists of a secret Personal Identification Number and information related to the smartphone (serial number of CPU, MAC address). The data is sent encrypted to the authentication server, which grants access through a KNX door actuator if the identity is successfully verified.

A different access to the management of a home automation system is intended in the paper of Santos-Perez et al. (2013). The authors present a prototype of an Embodied Conversational Agent (ECA) for the management of a home automation system. The ECA interface is developed on an Android device and involves automatic speech recognition, a conversational engine and a virtual head animation beneath other features. The prototype is capable of controlling a door lock, the temperature and the lighting in a home automation application.

A further interesting approach for home automation and Android is shown in the article of Gurek et al. (2013). The system mainly involves a local hardware, a web server and the mobile smart device running Android. Furthermore, the architecture is designed to support multiple users to control appliances through a developed Android application. Interesting is the choice for the web server, since it implies the Google Cloud Messaging service to establish the communication between the local hardware and the mobile Android device. The advantage of the free Google platform is that it supports the cost efficiency of the proposed system.

The idea of an low cost solution for home automation to fulfil the needs of elderly and disabled in home, is attempted in the approach of Ramlee et al. (2013). The concepts scope implements Bluetooth technology for remote access from a pc or an Android device to control home appliances. The system is installed beneath the existing electrical switches on the wall. Through a Graphical User Interface on the Android device or PC it is possible for the user to control the installed switches. In addition installed sensors provide readings for temperature and humidity, which are in real-time indicated on the Graphical User Interface. So far the system only displays sensor values without a regulating consequence. This can bee seen as an disadvantage, since the user still needs to switch home appliances manually on or off to change for instance the temperature.

The goal of the AAL project described in the paper of Junqueira Barbosa et al. (2014) is to deliver information from a user to a caregiver acquired from environmental sensors, to process the received data and present the information in an easy and understandable form. The platform chosen for display is beneath a web-page an Android application. As already mentioned the Android application displays the context of the received sensor data, although it makes sense, that some sensors like the SOS button are displayed straightly, with the highest priority. Furthermore, the ZigBee protocol is used for the sensors to communicate with a base station, which is connected to a Raspberry Pi acting as a middleware.
The presented related work for Android in the field of AAL and home automation shows, that the Android operating system features all requirements to prevail in this field of study.

3.3 Relevant Information from Related Work

The information excerpted from the referenced related work can be pointed out as the following enumeration shows.

- The related work shows that it is possible to use ARM single board computers as the main control unit for home automation and AAL applications, which means that depending on the used ARM board the process power is sufficient for the required operations.

- Reducing costs in AAL and home automation is possible through the use of low cost ARM single board computers.

- ARM boards can cope with a variety of communication standards, if necessary through extension modules, which supports the flexibility of a project.

The presented information underlines the use of ARM boards in the scope of this thesis, to develop a cost reducing, flexible and capable platform on the AP side for the Assistance Request scenario.
Continuing with the excerption of information from the related work referenced for

Android in the field of AAL and home automation, is presented in the following listing.

- Due to a high popularity of smartphones, existing resources can be used for AAL applications.
- The installation or update of applications provided for the end user is uncomplicated. This is realized, through the integrated application store, which underlines the flexibility of the platform for the changing needs of the AP.
- Intuitive handling of the Graphical User Interface, since the input is executed through touching.
- Variety and mobility of hardware platforms, which use android benefits the changing requirements of the AP.

Again the excerption shows that Android is capable of fulfilling all required features needed for the Assistance Request scenario proposed in this thesis.
The next chapter will specify the used tools necessary for the development of the Assistance Request scenario in detail and give more insight on the development process.

4. Materials and Methods

The upcoming chapter defines all required steps and tools for the design and development of the Assistance Request scenario. In detail, the technical concept is explained. The specification and selection of possible ARM boards is described, as well as a step by step instruction for the required software tools needed to execute the HOMER platform on the selected ARMs. Furthermore, the HOMER platform and the necessary tools from the OSGi framework are explained. The development process for the integration of the Near Field Communication (NFC) technology with ARM boards in conjunction with Java is described. Finally, the tools for the development of the Android application on the IC side are presented.

4.1 Technical Concept of Scenario

An easier approach to explain the scenario is to split it depending on the location where the tasks are performed. One side is declared as the AP side, where the data input is completed and the second side is the IC side, where the data is received and displayed.

4.1.1 Assisted Person side

The goal is to develop a concept that enables the AP to send a request to the IC, without the need of a call or to write a text message. To fully explain the concept on the AP side it is crucial to mention that, the RelaxedCare project uses HOMER, an open OSGi-based software platform for home automation. A first step of the thesis was to find hardware, powerful enough to run HOMER and at the same time to feature a small size. The small size benefits the design process, since technological

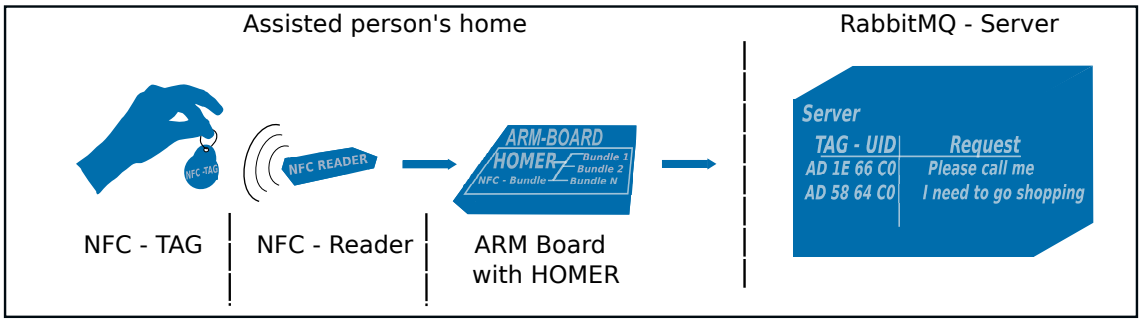


Figure 4.1: The figure shows a simplified view of the concept on the AP's side. The continuing text offers an in depth explanation on this topic.

affinity differs from person to person. Especially the design and development of IT, when focusing on the elderly end users, must support this user group to help them to overcome their fears and enable them to accept the technological aids (Holzinger et al., 2008). A higher acceptance of technology can be achieved through the use of every day objects, such as a vase or decoration, as the designers can integrate the hardware into the mentioned objects. The idea was to use ARM-architecture boards for this purpose under the formerly explained condition, in addition to a wide range of connection possibilities and sufficient memory.

HOMER provides, embedded on the ARM board, the home automation functionality required for the AAL environment. With that in mind the focus can be directed back on the concept.

A visualisation of the basic idea is displayed in Figure 4.1. Since the AP should be able to make a request without calling or writing a text message, the idea was to use a NFC tag, which represents a unique request. For the first prototype two requests are provided, a request in case the AP wants to be called and a request that the AP needs to go shopping. If the AP is in need of a request, a NFC-tag can be put on a NFC reader which is connected to the ARM board.

Through a, for this purpose developed OSGi bundle, which runs beneath the bundles required for HOMER, the NFC reader is directed to read the NFC tag's unique identifier (UID). The UID consists of a 4 byte serial number. The bundle converts the UID into a JSON encoded string and triggers an internal event with a specific topic and the JSON string. A further bundle, which is responsible for the communication with a RabbitMQ server, listens to this specific event. The JSON encoded string is then pushed to the server by the bundle where it is mapped with the actual request.

This design was chosen to ensure the possibility of adding new tags or reusing old tags for new requests. To add a new request for a new tag no active action from the AP is needed. A new request can be conveniently added on the server and assigned to the new tag UID. Similarly an old tag can be assigned to a new request. This design benefits the flexibility, which is needed to fulfil the different and changing needs of elderly people.
The request, likewise JSON encoded, is then pushed in a queue, where it waits till a client subscribes to the queue.

4.1.2 Informal Caregiver side

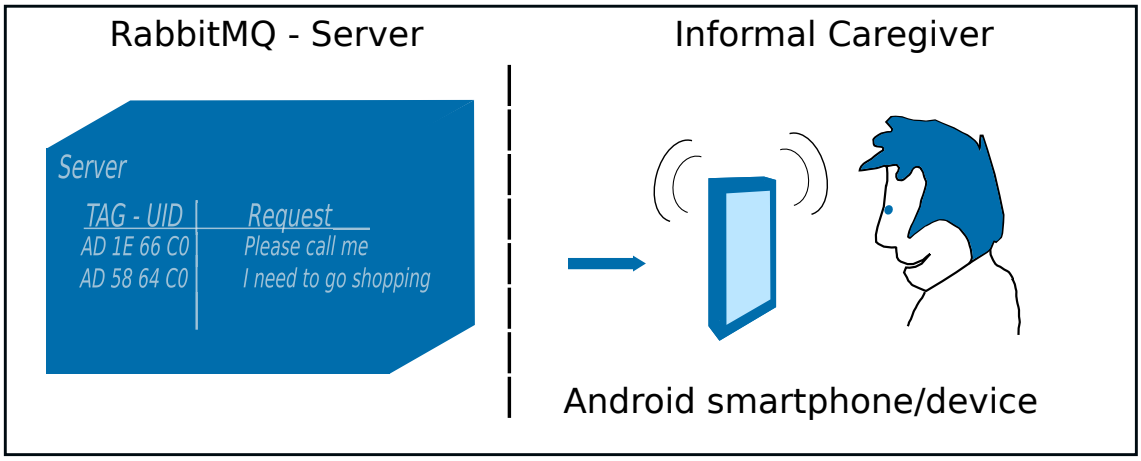


Figure 4.2: The figure provides an elementary view of the concept on the caregivers side. More information is provided in the continuing text.

On the one had the IC should be able to receive requests from the AP at any given location or time and on the other hand the IC should be in position to decide to remain undisturbed.
As solution for the required mobility an Android device, as receiving hardware, was proposed. An illustration of the basic concept on the caregivers side is shown in Figure 4.2. An application was designed, which fits the needs of the IC. The application was engineered to work as a RabbitMQ client. A connection to the RabbitMQ server, where the AP's requests are located, is established. Afterwards the application opens a channel and subscribes to the queue, which holds the AP's requests.

All requests waiting in the queue will be delivered to the application once the subscription is successful. The received requests are JSON encoded and therefore the actual requests have to be extracted from the JSON string.
If more than one request is received the application displays the latest request separately from the previous requests. All requests are stacked in a log with a time stamp, to give the IC a possibility to keep track. The caregiver has the option to run the application in the background, in which case a received request will trigger a pop up notification similar to receiving an email or text message.

4.2 ARM - Boards

The specification for the used ARM boards had to fit the demands, required by the Assistance Request scenario. Carefully considered, the following features where selected to find appropriate candidates for the ARM boards.

- Java version 1.7 should be runnable on the boards operating system.
- The HOMER platform should be executable on the system, therefore the ARM board should provide enough process power.
- Variety of connection standards, to uphold the flexibility of the system to fit changing needs, like USB for the NFC device and Wifi connection or at least an Ethernet access.
- Enough storage, to hold the HOMER framework.
- Small energy consumption.
- Extensibility, in case unexpected modification is needed, for instance a Zigbee module for wireless sensor connection.

4.2.1 Reviewed ARM - boards

In the process of search a variety of ARM boards could be found on the market, over the period of two months reaching from January 2014 to February 2014. The listing

provided in table 4.1 shows ARM boards, unfortunately without an Ethernet or Wifi connection, which would be, as the previously mentioned features are showing, beneficial for the intended purpose.
For the sake of completeness, they are still worth mentioning, since they can be acquired for a very low price, which makes them more attractive. In addition to the fact, that it is possible to assemble the ARM boards with extensional modules for additional connection possibilities. As a choice, the extension modules would successively raise the overall costs, which in turn has to be considered. As can be seen, no internal storage is provided, the intended solution for storage is an additional SD card or micro SD card.

Name	SOC	Core Type	Cores	Freq. [GHz]	RAM [MB]	USB Ports	Storage on-board [MB]/ SATA
Raspberry Pi Model A	Broadcom BCM2835	ARM 1176JZF-S	1	0.7	256	1	0/No
UDOO Dual Basic	Freescale i.MX6 DualLite	ARM Cortex-A9	2+1	1	1024	2	0/No
	Atmel SAM3X8E	ARM Cortex-M3					
OLinuXino	Allwinner A13	ARM Cortex-A8	1	1	512	3+1	0/No

Table 4.1: The table shows the specification for reviewed ARMs without an Ethernet or Wifi connection. The provided data was taken from the respective ARM board's homepage or Datasheet.
The second column refers to the used System on Chip (SOC). The core frequency for the UDOO Dual Basic board is related to the main cores for the ARM Cortex-A9. The listed boards do not offer an on-board storage or a Serial Advanced Technology Attachment (SATA) bus interface for mass storage extension. Storage is implemented by additional SD cards or micro SD cards.

Table 4.2 shows ARM boards reviewed, which exhibit an Ethernet but no Wifi connection. The listed ARM boards mostly include single core processors, with frequencies up to 1 GHz, which puts them in the middle range in regards of process power. Some of the ARM boards likewise include an internal storage, up to 4GB, which makes the purchase of an additional SD card, depending on the purpose,

Name	SOC	Core Type	Cores	Freq. [GHz]	RAM [MB]	USB Ports	Storage on-board [MB]/ SATA
Wandboard Solo	Freescale i.MX6 DualLite	ARM Cortex-A10	1	1	512	1	0/No
Raspberry Pi Model B	Broadcom BCM2835	ARM 1176JZF-S	1	0.7	512	2	0/No
BeagleBone	TI Sitara AM335x	ARM Cortex-A8	1	0.72	256	1	4096/No
BeagleBone Black	TI Sitara AM335x	ARM Cortex-A8	1	1	512	1	4096/No
BeagleBoard-xM	TI Sitara AM37x	ARM Cortex-A8	1	1	512	4	0/No
pcDuino Lite	Allwinner A10	ARM Cortex-A8	1	1	512	2	0/No
OLinuXino A20 LIME	Allwinner A20	ARM Cortex-A7	2	1	512	2	0/Yes
Utilite Value	Freescale i.MX6 Solo	ARM Cortex-A9	1	1	512	4	4096/No

Table 4.2: The illustrated table lists inspected ARM boards, which include an Ethernet but no Wifi connection. The specification data was taken from the respective ARM board's Datasheet or homepage. A comprehensive explanation is provided in the underlying text.

dispensable. The lack of a Wifi connection can be tolerated to some extend for the Assistance Request scenario, although admitting that from the angle of end user design, it is more feasible to chose an ARM board with Wifi connection. The ARM boards listed in table 4.3, include an Ethernet as well as a Wifi connection. Most of the ARM boards shown in the list comprise a dual or even quad core processor, with frequencies from 1GHz to 1.2 GHz and up to 2GB RAM, which puts them in concerns of process power in the upper field. Furthermore, some boards exhibit internal storage from 4GB up to 32GB and more then a half of the listed ARM boards include a SATA bus interface for mass storage. For the Assistance Request scenario these ARM boards would clearly be the first choice, for an end user product. However, considering that this thesis should perform as an experimental platform, the ARM boards selected from table 4.1, table 4.2 and table 4.3 are the Cubietruck,

the UDOO quad and the Raspberry Pi Model B ARM board. Conveniently the chosen ARM boards are listed in table 4.4.

Name	SOC	Core Type	Cores	Freq. [GHz]	RAM [MB]	USB Ports	Storage on-board [MB]/ SATA
PandaBoard	TI OMAP4460	ARM Cortex-A9	2	1.2	1024	2	0/No
Wandboard Dual	Freescale i.MX6 DualLite	ARM Cortex-A11	2	1	1024	1	0/No
Wandboard Quad	Freescale i.MX6 Quad	ARM Cortex-A12	4	1	2048	1	0/Yes
Cubietruck	Allwinner A20	ARM Cortex-A7	2	1	2048	3	8192/Yes
UDOO Dual	Freescale i.MX6 DualLite	ARM Cortex-A9	2+1	1	1024	2	0/No
	Atmel SAM3X8E	ARM Cortex-M3					
UDOO Quad	Freescale i.MX 6 Quad	ARM Cortex-A9	4+1	1	1024	2	0/Yes
	Atmel SAM3X8E	ARM Cortex-M3					
pcDuino v2	Allwinner A10	ARM Cortex-A8	1	1	1024	1	4096/No
pcDuino3	Allwinner A20	ARM Cortex-A7	2	1	1024	1	4096/Yes
Utilite Standard	Freescale i.MX6 Dual	ARM Cortex-A9	2	1	2048	4	8192/Yes
Utilite Pro	Freescale i.MX6 Quad	ARM Cortex-A10	4+1	1.2	2048	4	32768/Yes

Table 4.3: The table lists ARM boards, which include an Ethernet and Wifi connection. The shown ARM boards display high process power and a diversity of storage options. The specification data was taken from the respective ARM board's Datasheet or homepage. More explanation is provided in the continuing text.

4.2.2 Selected ARM boards

Name	SOC	Core Type	Cores	Freq. [GHz]	RAM [MB]	USB Ports	Storage on-board [MB]/ SATA
Cubietruck	Allwinner A20	ARM Cortex-A7	2	1	2048	3	8192/Yes
UDOO Quad	Freescale i.MX 6 Quad	ARM Cortex-A9	4+1	1	1024	2	0/Yes
	Atmel SAM3X8E	ARM Cortex-M3					
Raspberry Pi Model B	Broadcom BCM2835	ARM 1176JZF-S	1	0.7	512	2	0/No

Table 4.4: The table lists the selected ARM boards for the Assisted Request scenario. An in depth explanation is provided in the underlying text.

In comparison with the UDOO quad and the Cubietruck the Raspberry Pi Model B displays a lower process power and in addition does not exhibit a Wifi connection. However, the Raspberry Pi Model B was selected to test the HOMER platform on a device with lower process power and to serve as a reference point for the other selected ARM boards. The upcoming part focuses on the selected boards and provides a more in depth overview, starting with the Raspberry Pi Model B.

Raspberry Pi Model B

The Raspberry Pi Model B, shown in Figure 4.3, is a versatile and capable ARM board, although it is the ARM board with the least process power from the selected candidates. Considering the required features provided in Section 4.2 the Raspberry Pi is able to achieve all the features, however some of them only to a certain degree. The upcoming listing provides more insight on this topic.

- The Raspberry is able to handle Java 7 JDK (Java string version 1.7), which means this feature is fully supported. In fact the Raspbian OS already includes

Figure 4.3: The figure shows the Raspberry Pi Model B, with the used SD card

Java 7 JDK. If for some reason the Java 7 JDK is not included, it can be downloaded directly from the Raspberry repository.

- The HOMER platform is runnable on the Raspbian OS, although some drawback have to be accepted. More information on this topic is provided in Chapter 5.1.

- The Raspberry Pi involves a variety of connection standards, although admitting that a Wifi connection is missing. However, the scope of two USB, an Ethernet, an Inter-Integrated Circuit (I^2C) bus and a Serial Peripheral Interface (SPI) as well as a universal asynchronous receiver/transmitter (UART) are sufficient to cope with the needs of a first prototype for the Assistance request scenario. In addition, the Raspberry Pi can resort to a range of extension modules to upgrade connection standards.

- For this thesis a 4GB SD card is used with the Raspberry Pi. Considering that roughly 1GB are used by the Raspbian OS. The resulting storage space amounts 3GB, which is sufficient for the approximately 1GB large HOMER software. Theoretically it is possible to build the HOMER software on a different Hardware and to only move the framework, which is significantly

smaller, with only about 150MB. In the scope of this prototype this is enough storage, considering the flexibility aim of the RelaxedCare project this may be at some point not sufficient. However, the Raspbian OS supports SD Card sizes up to 32GB, which should meet the requirements.

- The Raspberry Pi is powered by a 5V micro USB supply and consumes depending on the connected peripherals 700 - 1000mA. The low energy consumption makes the Raspberry specially attractive for AAL applications.
- Due to the popularity of the Raspberry Pi a considerable range of extension modules are available, which makes the Raspberry Pi highly flexible.

The Raspberry Pi Model B, as a candidate for the Assistance Request scenario copes, despite the drawback of a lesser process power, very well with the desired features. However the community support behind the Raspberry Pi is outstanding and provides solutions to a range of applications. In addition to a low price of 35€ this ARM board poses an adequate solution for the Assistance Request scenario.

Preparing the Raspberry Pi Model B for HOMER

As operating system the Raspbian OS was selected. The installation of the Raspbian OS was performed as instructed in Follmann (2014) and includes the following steps. All steps for preparation were executed on a Linux system.

- The Raspian OS was downloaded from the Raspberry Pi homepage `http://www.raspberrypi.org/downloads/`
- The SD card was formatted with FAT, to prepare the SD card for the image.
- The SD card was unmounted `unmount /dev/sdd` whereat sdd stands for the partition of your SD card.
- The Raspbian image was written to the SD card, with the shell command `dd bs=1M if=imageDirectory.img of=/dev/sdd` again sdd is the partition of your SD card.
- The SD card was plugged into the Raspberry Pi. The predefined user name and password are `username:pi password:raspberry`

- Optional the shell command `startx` can be used to start the graphical interface.

The HOMER framework demands Java 7 JDK and upward versions. This version of Java should be installed on the Raspbian OS. As mentioned Raspbian OS already ships with Java 7 JDK, if fore some reason this version of Java should not be present on the system, it can be added from the Raspbian repository `sudo apt-get install oracle-java7-jdk`.
Finally the HOMER framework can be moved on the Raspberry Pi. It is not recommended to build HOMER on the Raspberry Pi, since on the one hand, it would require to install Maven version 3.x.x, which is not an obstacle. On the other hand, the building process takes, due to the lesser process power, a very long time.

UDOO quad

The UDOO quad, a 11 cm x 8.5 cm sized ARM board, shown in Figure 4.4, is designed to provide, beneath a high process power, an embedded Arduino-compatibility, without the use of further middleware. The features, required by the Assistance Re-

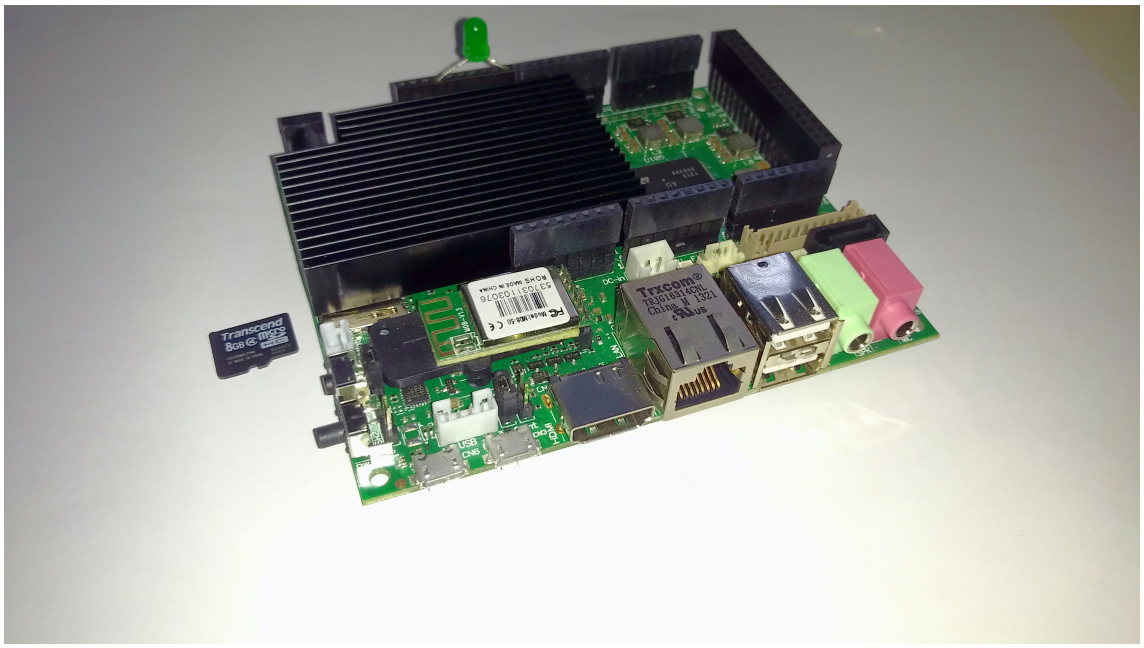


Figure 4.4: The figure illustrates the UDOO quad ARM board and the used micro SD card.

quest scenario are fully covered, by the ARM board. The upcoming listing gives a more in depth description on this topic.

- For the operating system the UDOObutu OS, a lubuntu 12.04 LTS armHF based OS was chosen, which fully supports Java 7 JKD. The benefit of the UDOObuntu OS is the use of the Ubuntu repository for ARM, which provides a range of software tools.

- The HOMER framework is as well runnable on the UDOO quad, more insight on this matter is discussed in Section 5.1.

- The scope of connection standards, the UDOO quad comprises, involves an Ethernet and Wifi connection, as well as 76 fully available general-purpose input/output (GPIO) Arduino-compatible pinouts , two USB, two Micro USB, a SATA connector with power header, an I^2C bus and a SPI. The UDOO quad therefore provides sufficient connection standards for the Assistance Request scenario. Furthermore, applications for extension modules are fully supported especially Arduino based solutions.

- The storage used in this thesis, for the UDOO quad, is a 8GB micro SD card. However, the UDOO quad ARM board supports a SATA connection which would allow the use of mass storage hard disks. Therefore storage concerns can be discarded, with the UDOO quad.

- The included power supply of the UDOO quad provides 12V at 1A, which is above the values needed for the Raspberry Pi.

- The UDOO quad exhibits an excellent extensibility, since it is designed to fully support Arduino based platforms.

The requirements deployed by the Assistance Request scenario are entirely fulfilled by the UDOO quad. In addition, to providing the possibility to mount Arduino hardware, every pin of the 76 GPIO can be accessed directly. Therefore, the UDOO quad provides in the `/sys/class/gpio` directory for every pin a folder, which comprises the pin direction and the pin state text file. The pin direction, can be changed by writing `"in"` or `"out"` into the direction file. In the same way the pin output can be changed to `"1"` logical high or `"0"` logical low in the pin state text file.

Preparing the UDOO quad for HOMER

As mentioned a lubuntu 12.04 LTS armHF based UDOObuntu OS was selected for the operating system. The following steps were performed before the HOMER software was installed. All steps for preparation were executed on a Linux system.

- The UDOObuntu OS was downloaded from the UDOO home page `http://www.udoo.org/downloads/`
- A prepared micro SD card was formated with FAT32.
- The micro SD card was unmounted from the system `unmount /dev/sdd`.
- The UDOObutu OS was written on the micro SD card by executing the shell command `dd bs=1M if=imageDirectory.img of=/dev/sdd`.
- As a last step the micro SD card was plugged into the UDOO quad and the device was turned on. The UDOObutu OS switches automatically into the graphical interface. The username and password are predefined as ubuntu `username:ubuntu password:ubuntu`.

The UDOObutu OS uses the Ubuntu ARM repositories, hence the Java 7 JDK can be installed by executing `sudo apt-get install oracle-java7-jdk` from a terminal. The system is ready to run the HOMER framework, although the UDOO quad would be powerful enough to build the HOMER platform on the ARM board itself. In this case Maven 3.x.x should be installed on the system.

Cubietruck

The Figure 4.5 shows the Cubietruck ARM board and the used micro SD card. The 11cm x 8cm sized board is designed for process power consuming applications. As seen in table 4.4 the Cubietruck exhibits 2GB of RAM and a dualcore processor, which favours the Cubietruck for the use with HOMER and the Assistance Request scenario. How well the Cubietruck copes with the required features, is provided in the upcoming listing.

- The Cubietruck ARM board was shipped with a pre installed Android OS. However, the HOMER platform was not build for an Android OS, thus the

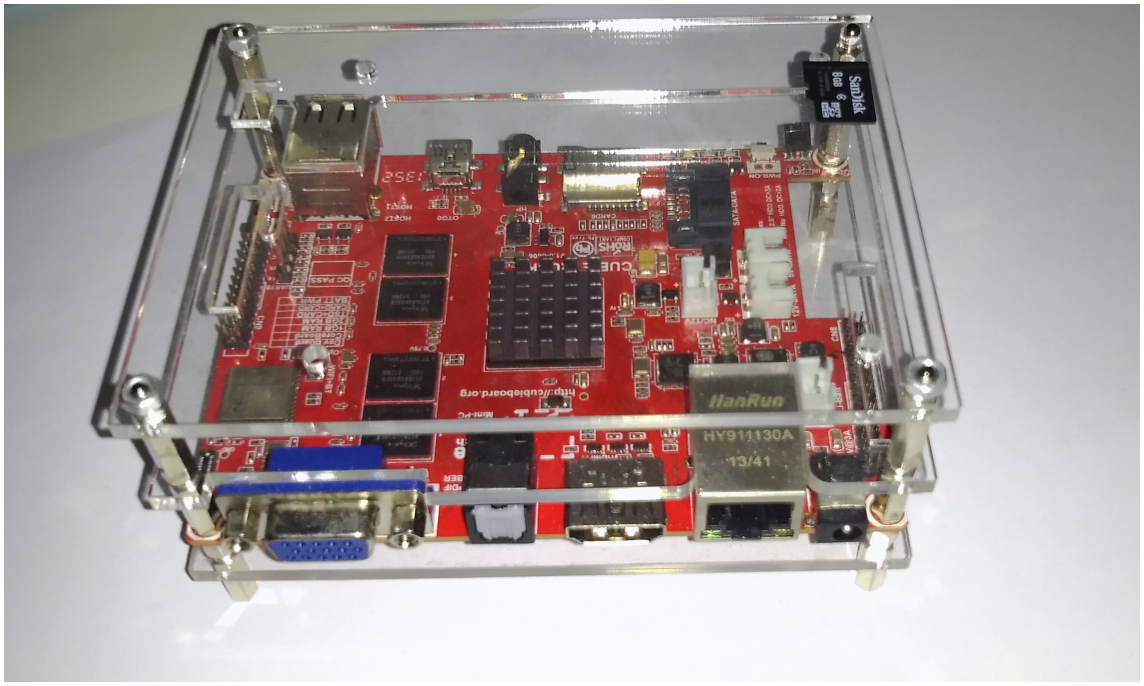


Figure 4.5: The figure illustrates the Cubietruck ARM board and the used micro SD card.

selected operating system was the debian ARM version for the Cubietruck called Cubian, which fully supports Java 7 JDK.

- The Cubietruck ARM board is capable of handling the HOMER platform. Chapter 5.1 gives more insight on this mater.

- The connection standards provided by the Cubietruck ARM board include the upcoming connections. Beneath an Ethernet and Wifi connection, the Cubietruck board even supports Bluetooth 2.1. In addition, a SATA 2.0 interface, which support 2.5 HDD, 2 x USB connections and 54 extended pins, which include I^2C, a SPI, a UART connection and an Infrared Data Association connection (IrDA) (CubieBoard, 2014).

- The internal 8GB sized storage in addition to the possibility to use a micro SD card enables the Cubietruck to provide sufficient storage space fort the HOMER platform and the Assistance Request scenario. In addition, to the mentioned SATA 2.0 interface for mass storage, for future applications. However for the Assistance Request scenario a 8GB mirco SD card was used.

- In regards of expansion possibilities, the Cubietruck offers additional expansion

boards which enable the Cubietruck to add further modules for Arduino or connections standards like Zigbee and other different applications.

The HOMER framework and the required features for Assistance Request scenario are fully supported by the Cubietruck ARM board.

Preparing the Cubietruck for HOMER

The following steps were performed to prepare the Cubian OS for the HOMER platform. All executed steps were performed on a Linux system. The performed steps correlate with the preparation for the UDOO board.

- The Cubian OS was downloaded from the Cubietruck home page `http://http://cubieboard.org/download/`
- A prepared micro SD card was formated with FAT32.
- The micro SD card was unmounted from the system `unmount /dev/sdd`.
- The Cubian OS was written on the micro SD card by executing the shell command `dd bs=1M if=imageDirectory.img of=/dev/sdd`.
- The micro SD card was plugged into the Cubietruck ARM board. By default the graphical interface is started. The username and password are predefined as `username:cubie password:cubie`.

The installation of the Java 7 JDK as well correlates with the UDOO quad, by executing the shell command `sudo apt-get install oracle-java7-jdk`. The provided process power and memory would enable the Cubietruck to build the HOMER platform on the system itself. Again the Maven 3.x.x software would be required for this purpose.

4.3 Software Applied for this Thesis

The development techniques and tools, applied in this thesis, comprise alongside Apache Karaf OSGi, Apache Maven and Blueprint. The HOMER software needs the mentioned software components to be fully operational. The upcoming text provides more insight on this topic.

4.3.1 OSGi

The OSGi framework enables the modularity of the HOMER platform. In fact the simplest explanation for OSGi, is the definition as a modularity layer for Java (Hall et al., 2011). The modularity of OSGi is accomplished through the concept of individual bundles. These bundles or packages can be remotely installed and uninstalled as well as updated, or started and stopped, without the need of rebooting the running system. Interactions and dependencies between the bundles are handled by the framework itself. The framework manages the search and binding of services, which can be seen as exposed functionalities within the OSGi bundles (Kropf et al., 2012). As mentioned it is possible to dynamically install, resolve, start, update, stop, and uninstall the individual bundles of the OSGi framework. The upcoming part provides more insight on the life cycle of a OSGi bundle.

Life cycle

The OSGi framework strictly adheres the transition between states. The transition of states is shown in Figure 4.6. It is not possible to install a bundle and to jump afterwards directly to a active state, without first passing through the resolved and starting state (Goodyear and Edstrom, 2013).

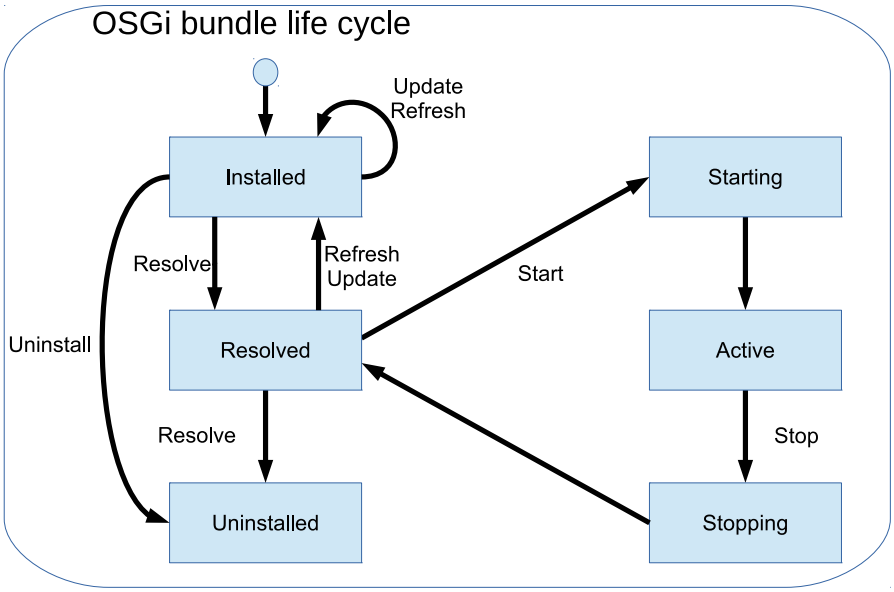


Figure 4.6: The figure illustrates the transition of states in a life cycle of a OSGi bundle.

Transition states

The existence of a bundle, in the OSGi framework, starts with the installed state. It is not possible for a bundle to be immediately started from this state. This also means that an installed bundle is not active, after it is installed. The installed state allows three transitions, which a bundle can undergo. This transitions are resolve, unistall and refresh. Considering that, the installation of a bundle does not imply that the bundle is ready for use, the next step for the activation would be to resolve the bundles dependencies in the framework.
As the bundle enters the resolved state, all dependencies have been set by the framework. At this point, the transition to the starting state is possible. A further option is the transition back to the installed state, through refreshing. A third option for the resolved state, poses the transition to the uninstalled state. The resolved state implies that a bundle is not active, however it ready to be activated.
If a resolved bundle is pushed to the starting state, the framework initializes the resolved bundle into a active state, as the starting state is transitory. The transition to the starting state always implies the transition to the active state. In the active state a bundle is fully resolved and provides or consumes services in the OSGi framework. Further transitions, would first require the bundle to be stopped.
In case a bundle update takes places, the framework will re-evaluate the dependencies of this bundle. During this process the wiring to and from the specific bundle is broken. Considering that, depending on the amount of interconnected bundles in a given framework, an update can cause a domino effect, by updating a heavily entangled bundle other bundles are updated as well and so on.
Stopping a bundle in the active state transitions it to the resolved state, from where it is possible to restart the bundle. Transitioning a bundle from the installed or resolved state to the uninstalled state, does not mean that a bundle is removed from the environment. The bundle is in fact, no longer available for use, but references to the bundle can still exist, which can be used for introspection.
The simplest way to activate a bundle, is through the use of a BundleActivator. A class may be declared as the Activator class implementing the `org.osgi.framework.BundleActivator` interface. This allows the specification of actions while the bundle is starting or stopping to be performed. However, as a first approach the bundle

in this thesis was in fact developed to use a bundle activator, but in the course of development the bundle activation was established through Blueprint.

4.3.2 Blueprint

Blueprint is a dependency injection framework based on Spring Dynamics Module, which is able to handle the dynamic environment of the OSGi framework. The term dependency injection describes a design where dependencies between components are decoupled from the actual program code (de Castro Alves, 2011).
Simplified, a Blueprint bundle can be described as a bundle, which contains XML documents specified using the Blueprint schema, which describes how objects, services or references to services are created and assembled. The `<bean>` tag defined in a Blueprint XML documents initiates an objects using the Java class specified in the `<class>` attribute, as shown in the listing 4.1. The `init` and `destroy` attributes are callback methods, which are invoked by the Blueprint container to initialize or destroy beans in case they are enabled or disabled. The `<reference>` tag is meant to take a single service from the service registry for a component based on the service interface. The `<service>` tag allows components to provide OSGi services. This section only provides a very small fraction of OSGi and Blueprint, since the field of this topic would go beyond the scope of this thesis. However, if more information is required, please refer to the OSGi Alliance (2013).

```
1 <bean id="IDbean" class="the.used.JavaClass" init-method ="
    INIT" destroy-method ="Destroy" />
2 <reference id="IDreference" interface="InterfaceReference"
    activation="lazy" />
3 <service id="IDservice" interface="InterfaceService"
    activation="lazy" />
```

Listing 4.1: Blueprint XML document

4.3.3 Apache Karaf

Apache Karaf is a runtime environment based on OSGi, which provides a container for the deployment of components and applications (The Apache Foundation, 2014). The Apache Karaf runtime supports many features, some are provided in the following list.

- Hot Deployment: A deployment directory is monitored for changes. In case a new bundle is dropped in the specific folder it will be automatically installed inside the runtime.

- Extensible Shell console: A shell console is provided, which can manage services and install new bundles, libraries in addition to handling manually the state of the bundles.Furthermore, new commands can be dynamically deployed.

- Logging System: Apache Karaf provides a centralized logging back end.

- Dynamic Configuration: The `ConfigurationsAdmin` OSGi service commonly configures services provided in the framework. It is possible to define the configuration in Karaf by using property files inside a specific directory. Changes of properties are monitored and forwarded to the related services.

For more insight, on all feature provided by the Apache Karaf runtime environment, please refer to the previously stated Apache Karaf homepage .

4.3.4 Apache Maven

Apache Maven can be described as a build-automation tool. Development guidelines, in terms of how to structure your project life cycles and how to execute your build, are defined by the Maven framework (Turatti and Pillitu, 2013). Maven can fetch libraries and other plugins from remote sources to compile Maven builds. These so called Maven repositories, can be reused for other projects. Fetches, will by default access the Maven Central Repository, but it is as well possible to use alternatively repositories from other software vendors.
In other words, a Maven project exhibits dependencies, which are provided in remote web-based folders. These dependencies are needed to compile, build, test or run a Maven project (Lalou, 2013). Basically, projects depend on other projects, which are in turn dependant on other projects as well. However, a project only need to know on which projects it depends on. It is not necessary to inform the projects it depends on, of something. Dependencies are stored in the `pom.xml` file. The Project Object Model (POM) can be described as the core unit of the Maven framework.

The XML file of the POM, implies the needed dependencies as well as configuration details, like the build and source directory (Apache Maven, 2014). The minimum requirements for a POM, are described in listing 4.2.

```
4 <project>
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>com.mycompany.myApp</groupId>
7   <artifactId>myAppId</artifactId>
8   <version>x</version>
9 </project>
```

Listing 4.2: POM minimum requirements

The `<modelVersion>` should be set, as recomended on the Apache Maven homepage, to `4.0.0`. The `<groupId>` tag describes a macro group or family of projects, archives a project belongs to. A unique identifier, defined by the `<artifactId>`, assures the identification of projects sharing the same `<groupId>`. The `<version>` tag defines the numerical version of a specific release for a project. Maven offers an inheritance mechanism for POMs. It is possible to define a parent POM, inside a specific POM file, through the `<parent>` tag. The child POM inherits all configurations defined by the parent POM. An example for inheritance, where a parent POM is defined in a child POM is shown in listing 4.3.

```
10 <project>
11   <parent>
12     <groupId>com.mycompany.myApp</groupId>
13     <artifactId>myAppId</artifactId>
14     <version>x</version>
15   </parent>
16   <modelVersion>4.0.0</modelVersion>
17   <groupId>com.mycompany.myApp</groupId>
18   <artifactId>myModuleId</artifactId>
19   <version>x</version>
20 </project>
```

Listing 4.3: Example for POM inheritance

However, the shown example only applies if the parent POM is one directory above the child POM directory. If, for instance, the directories would be on the same level, the `<relativePath>` tag provides a solution, through which the relative path to the parent POM is defined.

```
21 <project>
22   <modelVersion>4.0.0</modelVersion>
23   <groupId>com.mycompany.myApp</groupId>
24   <artifactId>myAppId</artifactId>
25   <version>1</version>
26   <packaging>pom</packaging>
27
28   <modules>
29     <module>myModule</module>
30   </modules>
31 </project>
```

Listing 4.4: Example for POM aggregation

The explained inheritance mechanism defines a parent POM in a child POM file, but Maven provides a possibility to go the other way, as well. This mechanism is called project aggregation. In more detail, the child POM (modules) are defined in a parent POM. This possibility is defined by the `<modules>` tag, shown in listing 4.4. The explanation, on Apache Maven, covers only a small part of the possibilities Maven provides. For more information please refer to the Maven homepage or related literature.

4.3.5 What is HOMER?

The so far described software tools, are necessary for the software development and execution of the Home Event Recognition System. The upcoming part gives more insight on the HOMER software itself and the features HOMER provides.
The HOMe Event Recognition System is an open source platform for home automation and AAL. (Fuxreiter et al., 2010). HOMER aims to provide a flexible and extensible platform. Communication standards for home automation and AAL are components of the framework, which provide modularity, security and interoperability. The programming language used for HOMER is Java, therefore the Java Runtime Environment is used for execution of the HOMER framework.
HOMER provides all tools necessary for home automation. A graphical interface shown in Figure 4.8 illustrates the graphically implemented finite state machine, as well as a tool to design a flat for home automation or AAL. The red dots indicate sensors or actuators, carefully distributed in the users home. Since HOMER

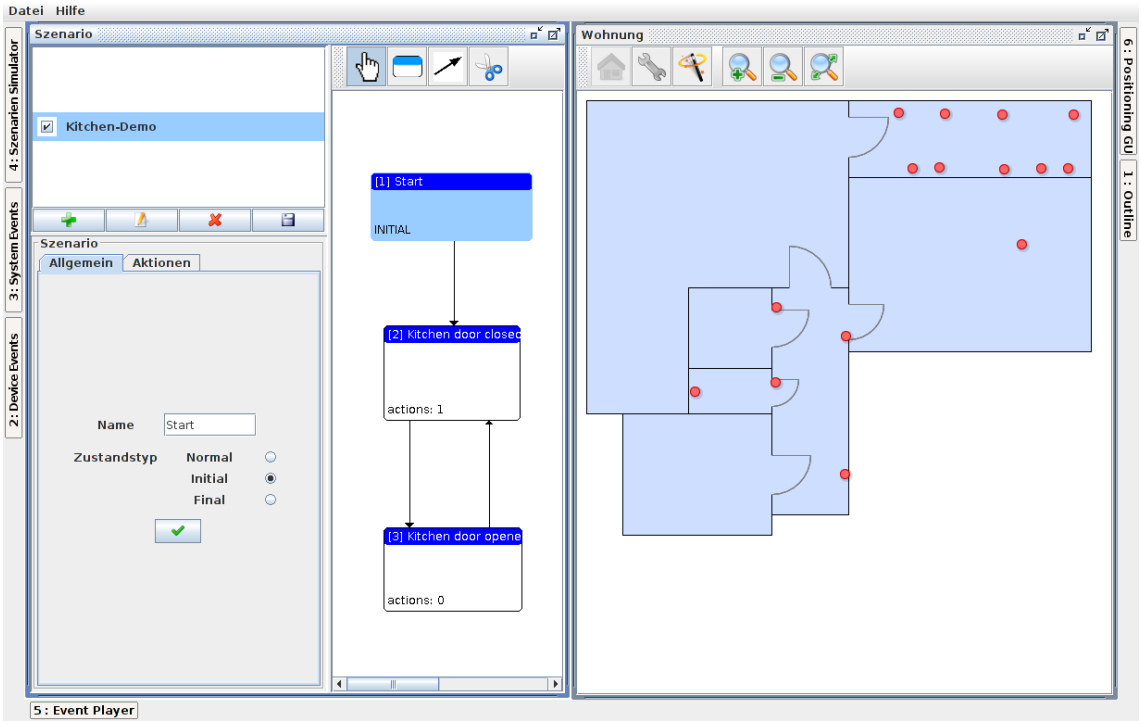


Figure 4.7: The given figure shows the HOMER graphical user interface. On the right hand side of the window a flat, with actuators and sensors is displayed. The left hand side shows the finite state machine sub-window, which provides an environment to design a scenario for home automation.

is able to handle different sensor technologies, it is essential to map the incoming information to one standardized data model (Fuxreiter et al., 2011). The mentioned finite state machine presents the core component for the rule base reasoning of the HOMER platform. The state machine offers as basic elements transition, event, action and state. The design of a state machine is facilitated graphically through the HOMER graphical user interface. It is possible to design several state machines, which run in parallel on one system. Therefore, a system state in one state machine, is able to trigger an event in another state machine. Triggers are usually messages from activated sensors. Every sensor can be selected in the flat plan tool, together with a message type. As mentioned several state machines can be executed in parallel, accordingly it is possible that one sensor starts transitions in different state machines.

In the scope of this thesis, it was necessary to implement the HOMER framework on the selected ARM boards. In addition an OSGi Bundle, developed for the communication with the NFC reader, was implemented beneath the bundles required for

the HOMER framework. For more detailed information, on the HOMER software itself, please refer to the referenced literature.

4.4 NFC under Java on ARM

The upcoming section gives a basic explanation on NFC technology and provides insight on the software development on NFC under Java. Furthermore, NFC technology in conjunction with ARM hardware is described. It has to be highlighted, that this section, is not meant to provide a fundamental explanation on NFC technology and it's scope of application.

4.4.1 Near Field Communication (NFC)

The NFC technology facilitates a short range communication in the high frequency domain (Ahson and Ilyas, 2011). NFC emerged from contactless technologies such as Radio Frequency Identification (RFID) and wireless communications standards such as Bluetooth and Wifi. The initialization for data transmission is established through a simple touch or tapping of two NFC devices to another. The communication distance amounts only a few centimetres. It is possible to apply NFC technology to simple tags, comprising integrated circuits and an antenna. An NFC reader can be used to communicate with these tags, by modulation an applied RF field. The tag receives requests and transmits data back to the reader. NFC technology works at a frequency of 13,56 MHz, with data transmission rates up to 848 kbit/s (Langer and Roland, 2010). The NFC devices used in this thesis was the SCL3711 manufactured by SCM Microsystems.

SCL3711

The SCL3711 is a USB based NFC device, with NXP's PN532 chip. It is a multi-protocol 13.56MHz contactless reader, designed to cope with the ISO14443 type A and B as well with the MIFARE, -Classic, -DESFire, -UL, -UL-C and -PLUS standard. Furthermore, it is able to handle the FeliCa™ standard and is compliant with the Personal Computer/Smart Card (PC/SC) version 2.0 specification. According

Figure 4.8: The figure shows the SCL3711 NFC devices.

to the user manual the SCL3711 works at 5V and and a maximum current of 100mA. The SCL3711 was selected due to a high compatibility with various operating systems, the small compact size and the convenience of a USB connections.

4.4.2 NFC development under Java

As the NFC device is able to use the PC/SC specification, it is possible to adapt the `javax.smartcardio` package, for the Java Smart Card I/O API defined in the Java Specification Request (Oracle, 2006). The API allows Java applications to interact with applications running on a Smart Card. Using the API, it is possible to retrieve data from a NFC tag, through a NFC device, which uses the PC/SC specification. The following list shows the classes used for the interaction with the NFC device, provided in the `javax.smartcardio` package.

- TerminalFactory: The TerminalFactory class represents a factory for CardTerminal objects.
- CardTerminal: The CardTerminal class defines an object for a Smart Card terminal. The Smart Card terminal can be refered as Smart Card reader.
- Card: This class defines an object for a Smart Card, with a established connection.
- CardChannel: The CardChannel class provides a logical channel connection to a Smart Card. Furthermore, it is used to exchange Application Protocol Data Units (APDU) with Smart Cards.
- CommandAPDU: This class defines a command APDU. The structure is defined in the ISO/IEC 7816-4, comprising a four byte header and a conditional body, with variable length.

- ResponseAPDU: This class defines an object for a response APDU. The structure defined by the ISO/IEC 7816-4 comprises a conditional body and a two byte trailer.

The upcoming part shows a first implementation for the NFC communication under Java and the SCL3711 NFC device. In a first attempt, the `NFCread` class, shown in Figure 4.9, was created. The class comprises a timer, which executes a specific task at a given time rate in a background thread. The executed task, shown in

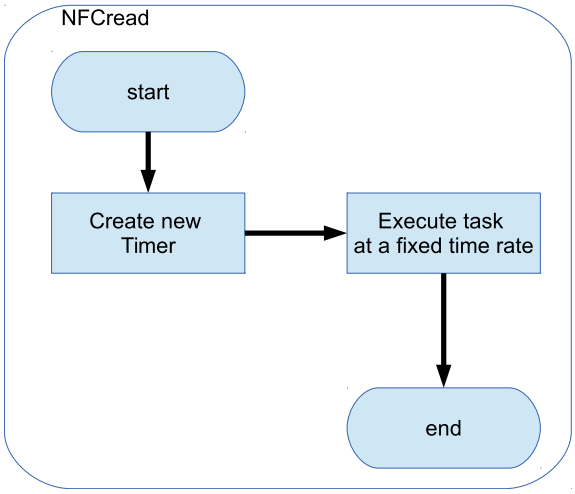


Figure 4.9: The figure shows the NFCread class.

Figure 4.10, attempts to access all available NFC terminals and saves them in a list of type `CardTerminal`. Subsequently, the created list is checked for available NFC terminals. In case no NFC terminal could be found, the task displays a `no NFC terminal` message. If NFC terminals are allocatable, the first NFC terminal from the list is selected. After the selection a second check is executed in case the NFC terminal is lost or the NFC device is removed from the PC. A removed NFC device would again induce the task to display the `no NFC terminal` message. A successful check leads to the inquiry for a NFC tag. An applied NFC tag would induce the task to establish a connection with the NFC tag, apart from that a `no NFC tag` message is displayed. An established connection, causes the task to open a channel to the NFC tag and a specific APDU is transmitted, which induces the tag to send data back. The received data is subsequently displayed and the task is closed. A new instance of the task is executed after the given time period. The developed application works well on a PC with Ubuntu 14.04, used for testing. However, during

a first approach, an issue on the selected ARM boards appeared. As mentioned the `javax.smartcardio` package uses the PS/SC specification to communicate with the SCL3711 NFC device. It seems that the interaction between `javax.smartcardio` and PS/SC, on the selected ARM boards, is in some way defective. After repeated testing this approach was discarded and a new approach, explained in the upcoming section, was applied.

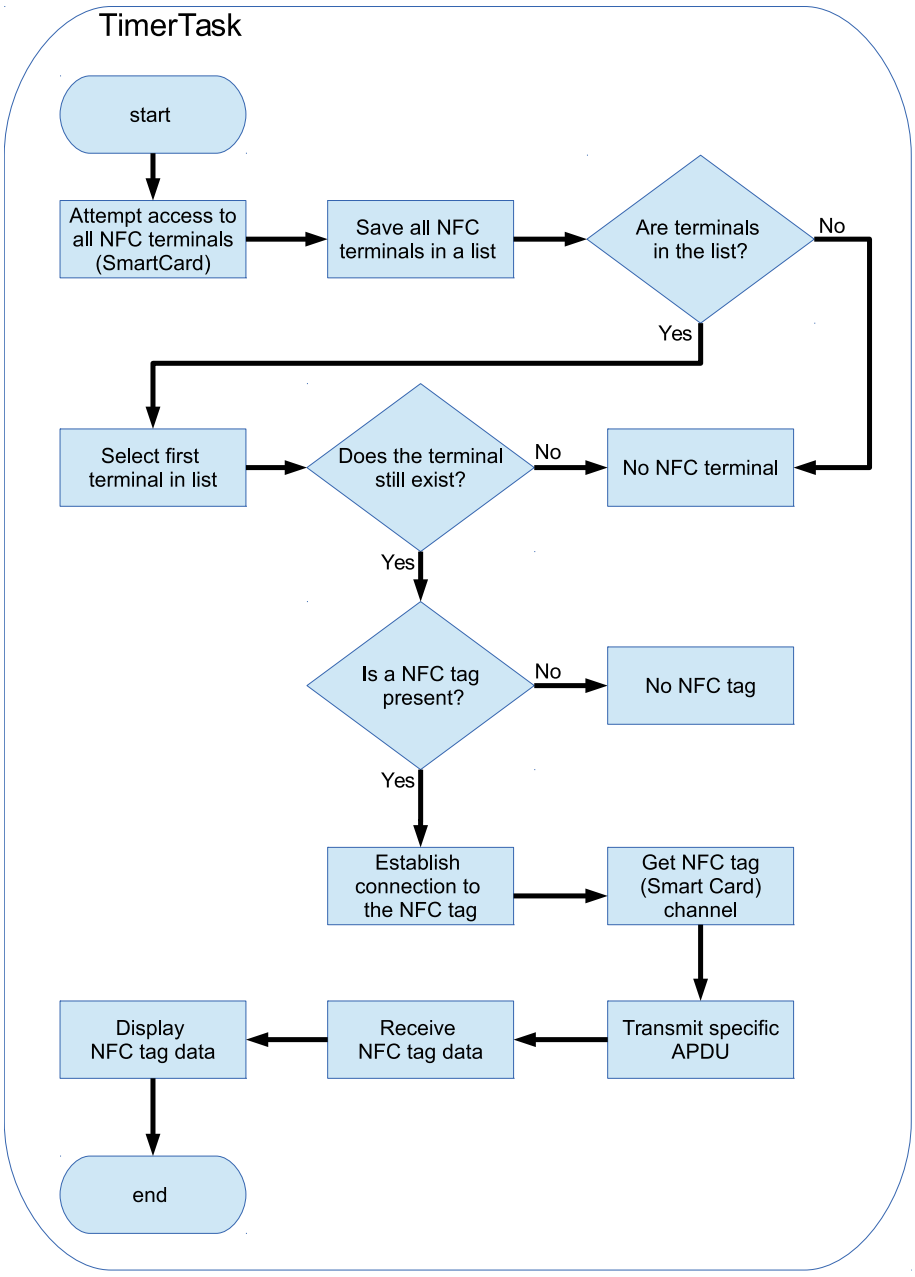


Figure 4.10: The figure shows the TimerTask methos which extends the Timer method.

4.4.3 NFC on ARM

Considering, that the first attempt with `javax.smartcardio` and PS/SC on ARM boards failed, a new approach was needed. The applied solution was to use the open source `libnfc` library for the interaction with the NFC device. The `libnfc` library provides a platform independent, low level NFC Software Developer Kit (SDK) and programmer API (libnfc, 2014). Explained in more detail, the `libnfc` library provides low level commands for the PN53x chipset and works on Linux, Mac Os X and Windows machines. The library supports ISO14443-A/B, ISO18092 (p2p) and JIS X 6319-4 modulation and the NXP Mifare Classic and Sony FeliCa protocol. The library allows complete control of the transmitted data frames. Conveniently the `libnfc` library includes applications and tools for the instant interaction with NFC devices. Furthermore the library is written in plain C and released under the open source GNU Lesser General Public Licence

Libnfc on ARM

The library was successfully installed on the selected ARM boards. For installation the following steps were performed.

- Download the `libnfc` library. Version 1.7.0-rc7 was used for this approach.
- For Linux distributions, as it were used for the selected ARM boards, it is required to install the `libusb-dev` and `libpcsclite-dev` packages.
- In the `libnfc` directory and the shell commands `./configure` and `make` were executed to compile the `libnfc` library
- Afterwards it is required to blacklist `pn533` and `nfc` in the `/etc/modprobe.d/blacklist-libnfc.conf` directory using an editor of choice.
- The last step is to restart the ARM.
- Some of the selected ARM boards exhibit the following error: `libnfc.so.4: cannot open shared object file`. The shell commands `echo '/usr/local/lib' | sudo tee -a /etc/ld.so.conf.d/usr-local-lib.conf` and `ldconfig` should fix the problem.

4.5 Android Client for RabbitMQ

This section provides basic information about the Android client application and why this particular mobile operating system was selected. According to Statistica (2014) Android is the leading smartphone operating system, with a global market share of 84.7%. Due to this fact the Android OS was selected as the operating system for the IC client application.
The intention was to develop a RabbitMQ client application for a Android device. At the time of development, a RabbitMQ client library was not provided in the Android repository. However, since Android is base on Java it was possible to use the Java RabbiMQ client library instead. For this reason the `rabbitmq-client.jar`, `commons-io-1.2.jar` and the `commons-cli-1.1.jar` from the Java Advanced Message Queuing Protocol (AMQP) client library were downloaded. The mentioned `jar` files were moved into a folder in the root of the Android client application. The last step before development, was the implementation of the RabbiMQ library files into the build path of the IC client application.

RabbitMQ

RabbitMQ represents an open source message broker which uses the AMQP (Dossot, 2014). The AMQP enables a loosely coupled architecture for communication. RabbitMQ is an Erlang based implementation of the AMQP. RabbitMQ establishes a communication between producers and consumers, where producers create messages and publish them to a broker server (Alvaro Videla and Williams, 2012). The message comprises a payload and a label. Whereat the payload is the specific data, which the producer wants to send. The label describes the payload and is used by RabbiMQ to determine, which consumer should receive a copy of the message.
The consumers attach to the broker server and subscribe to a queue. In case a message arrives at a specific queue, RabbitMQ forwards it to the subscribed consumers. The provided information illustrates the basic concept of a range of applications RabbitMQ offers. An in depth explanation on the Android client for RabbitMQ is provided in Section 5.3. For more information on RabbitMQ please refer to the references literature.

5. Results

The upcoming chapter provides the answer to the question, if the HOMER software is runnable on ARM platforms. Furthermore, the implementation of the NFC bundle for the HOMER framework is described.

5.1 HOMER Platform on ARM Boards

One of the questions, posed in this thesis was, if it is possible to execute the HOMER software on an ARM board. The upcoming section provides the results on this topic. The selected ARM boards were prepared as explained in Chapter 4.2.2. All of the selected ARM boards were able to execute the HOMER software, however there are differences in performance of execution. The HOMER software used on the selected ARM boards was build on a PC running Ubuntu 14.04 and only the framework was moved to the ARMs, for testing. In advance it has to be stated, that the performance on the selected boards strongly, depends on the bundles and the amount of bundles used for the HOMER framework. Considering, that it is possible to neglect bundles such as the GUI bundle by setting up the HOMER framework with a predefined configuration. Therefore the performance values provided in table 5.1 are to be considered as guide values for the comparison of the selected ARM boards executing HOMER. The measured percentage of the total CPU time, as well as the percentage share of the physical memory was recorded, after all bundles reached the active state in the OSGi life cycle. As tool for measurement the `top` shell command was used, conveniently integrated in all linux distributions. Furthermore, the start time, the specific board needed to start the HOMER software, whereat all included bundles reached the active state in the OSGi life cycle, was measured. For the measurement of the start time, the `time` shell command was used.

Name	%cpu	%mem	start time
Raspberry Pi Model B	68,4%	30,9%	13[m] 12[s]
Cubietruck	43,0%	13,3%	8[m] 16[s]
UDOO quad	26,0%	32.5%	2[m] 4[s]

Table 5.1: The table shows the percentage of total CPU time, the percentage of the used physical memory by the HOMER software on the selected ARM boards, as well as the required time to start the HOMER software. More explanation is provided in the related text.

As expected the Raspberry Pi Model B exhibits the lowest performance in comparison with the other boards. The Cubietruck ARM board poses the middle field of the selected boards, although the start time could be viewed as long. Given the performance values the UDOO quad is the ARM board which exhibits the best performance executing the HOMER software.

Stability

The stability of the HOMER software on ARMs was tested in a first attempt for a period of one week. During this period, the Raspberry Pi Model B, experienced a total freeze of the whole operating system after two days, resulting a shutdown of the HOMER software. A second attempt showed a stable operational time of 4 days again the whole operating system shut down. For the Cubietruck the testing period, showed a stable behaviour of the HOMER system, however the Wifi connection was temporally lost, resulting that some bundles did not reconnect. Nonetheless, the HOMER system was executed continuously during this period. For the UDOO quad no issues were recorded, during the test period. The executed HOMER framework included the developed NFC bundle described in the upcoming section.

5.2 How to Design a NFC-Bundle

It would be helpful to review the course of events for the Assistance Request scenario. Roughly described, the AP puts a tile on a NFC reader, which is recognized as a request by the system and in succession the request is sent to a server, which processes the received information for further use.

```
32      <reference id="eventAdmin" interface="org.osgi.service.
           event.EventAdmin" availability="mandatory" />
33
34      <bean id="nfclibnfc-activator" class="at.ac.ait.hbs.
           relaxedcare.homer.NFClibnfc.Activator" init-method="
           start" destroy-method="stop">
35          <property name="eventAdmin" ref="eventAdmin"/>
36      </bean>
37
38      <service ref="nfclibnfc-activator">
39        <interfaces>
40           <value>org.osgi.service.event.EventHandler</value>
41        </interfaces>
42        <service-properties>
43            <entry key="event.topics">
44            <array value-type="java.lang.String">
45                <value>eu/relaxedcare/event/*</value>
46            </array>
47            </entry>
48        </service-properties>
49      </service>
```

Listing 5.1: XML snippet of Blueprint

As mentioned before, HOMER is based on Apache Karaf OSGi, explained in Section 4.3. The structure consists of components, which are surrounded by a container. The components are the separate applications or bundles, which form the whole framework. The bundles can be installed, removed, started and stopped at run time.

An additional layer is needed to wire the individual bundles. This can be done in different ways, in this approach blueprint is used. Blueprint declares a selected class as bean and subsequently an init-method and destroy-method can be specified, as explained in Subsection 4.3.2. The init-method is invoked when a bundle is started and vice versa the destroy-method is invoked when the bundle is stopped. To be able to send events a reference to the Event Admin is needed. The necessary wiring is as well established through blueprint.

If it is desirable not only to send events, but as it is in this case, also receive events from the Event Admin, this service has to be implemented in blueprint as well. As shown in listing 5.1 the Eventhandler has to be implemented as a services with a topic of interest. The value of the topic is a string, in addition a wildcard asterisk can be used as the last token of a topic. This will match any topic with

the same first tokens. The NFC bundle comprises three classes. The `Activator`, `PollNfcData` and the `Feedback` class. The Figure 5.1 provides a simplistic overview of the overall implementation of the NFC bundle, shown on the left hand side and the forwarding to the IC application, shown on the right hand side. The `Activator` class can be viewed as the source from which all further action is started or stopped. The `PollNfcData` class is responsible for the NFC related tasks and transmission of the NFC UID. The `Feedback` class is intended, for response implementation on different hardware, for the AP. The upcoming part offers an in depth description on the developed classes and their methods.

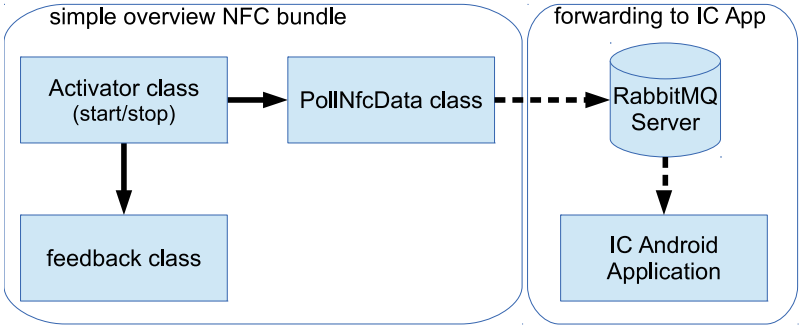


Figure 5.1: The figure shows a simple overview for the NFC bundle and the forwarding to the IC application. The NFC bundle comprises three classes, the `Activator`, `PollNfcData` and the `Feedback` class. The related text provides more information about this topic.

Activator class

The `Activator` class, seen in Figure 5.2, is declared as bean. The `Activator` class contains four methods. The `start` and `stop` method, which are the mentioned init- and destroy- methods, the `setEventAdmin` method and the `handleEvent` method. The `setEventAdmin` method is used to link the Event Admin referenced in the blueprint XML and the Event Admin declared in the field of the `Activator` class. The `setEventAdmin` method is invoked through blueprint, as described in Chapter 4.3.2. The `handleEvent` method is invoked in case a specific topic, declared in the blueprint XML, is triggered. Given the `handleEvent` method is invoked and the topic of the event, which triggered the method equals a predefined topic, as result a method of the `Feedback` class is triggered in turn. The `Feedback` class is described

further below in the text. The `start` method launches a thread of type `PollNfcData`. In case the bundle is stopped or removed from the framework, the `stop` method will cancel the launched `PollNfcData` thread.

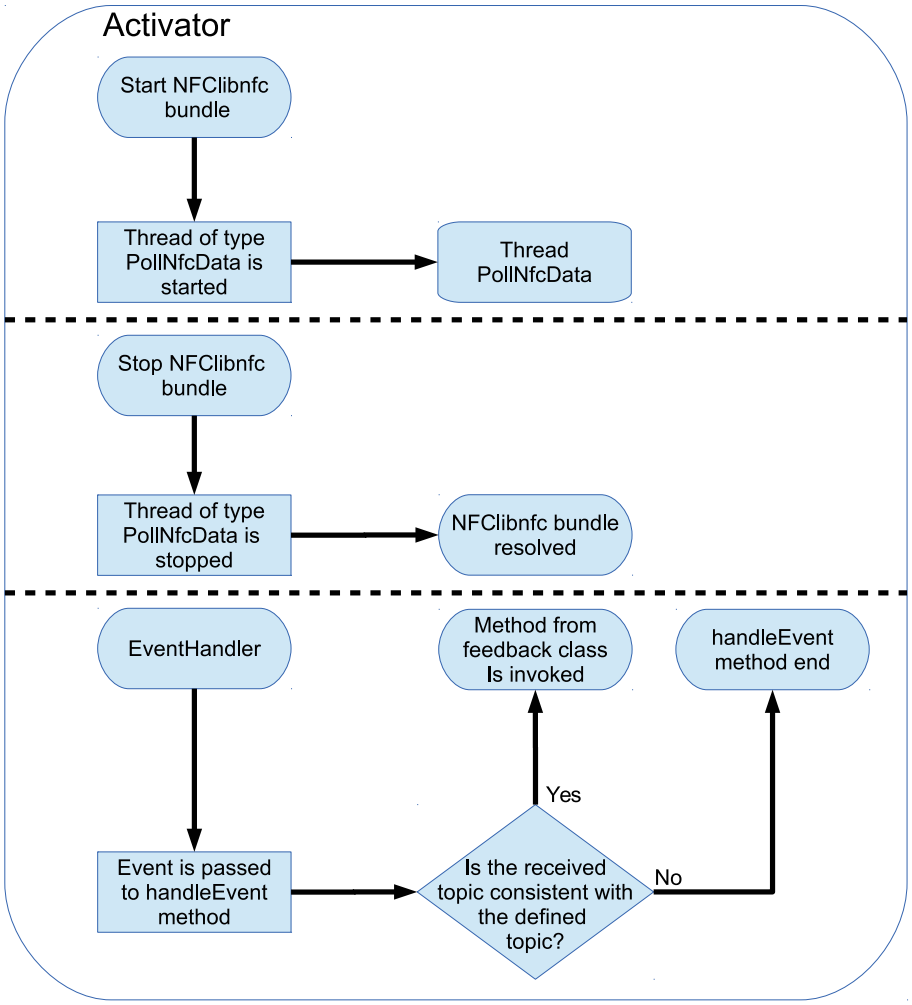


Figure 5.2: The given flowchart illustrates the `Activator` class. When the NFC-bundle is started the start-method of the `Activator` is invoked. In case the bundle is stopped the stop-method of the `Activator` is called and the bundle is resolved. More information can be found in the text above.

PollNfcData class

A flowchart of the mentioned `PollNfcData` thread can be seen in Figure 5.3. The constructor of `PollNfcData` creates a context to the `Activator` class, which will later provide the link to the Event Admin, declared in the field of the `Activator` class. In the run method of the `PollNfcData` thread, the `control` method is invoked and the

current UID and the previous UID are passed to this method. `PollNfcData` initializes the current UID and the previous UID with different values for the first run. The `control` method is describe in more detail further below. After the `control` method finishes the `exePoll` method is called, with a command as argument to poll for a tag UID. The `exePoll` method returns either the current UID or "no tag UID detected". For more information about the `exePoll` method, look further below in

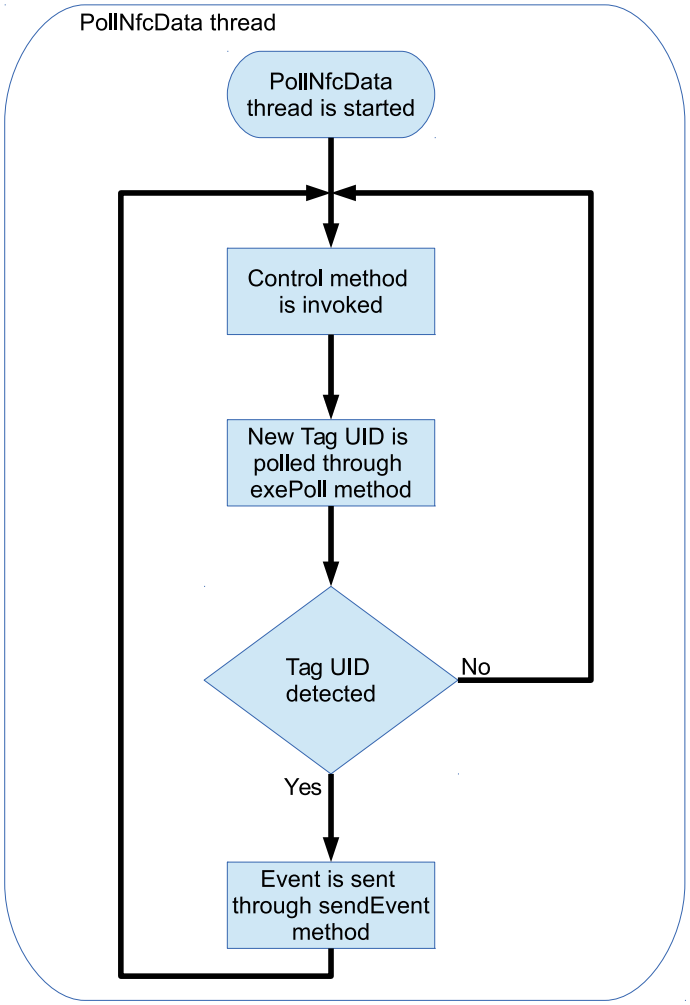


Figure 5.3: The Flowchart shows the `PollNfcData` thread, which was started by the `Activator` class. The `PollNfcData` thread in succession invokes the `control` method. The current UID and the UID which was polled before are passed to the `control` method. At the first run these variables are initialized with different values so the `control` method will not take further steps, for more information look up the text above or Figure 5.4.

the text. Afterwards, the `PollNfcData` thread checks if the returned value is a UID or if "no tag UID detected" was returned by the `exePoll` method. In case a UID was

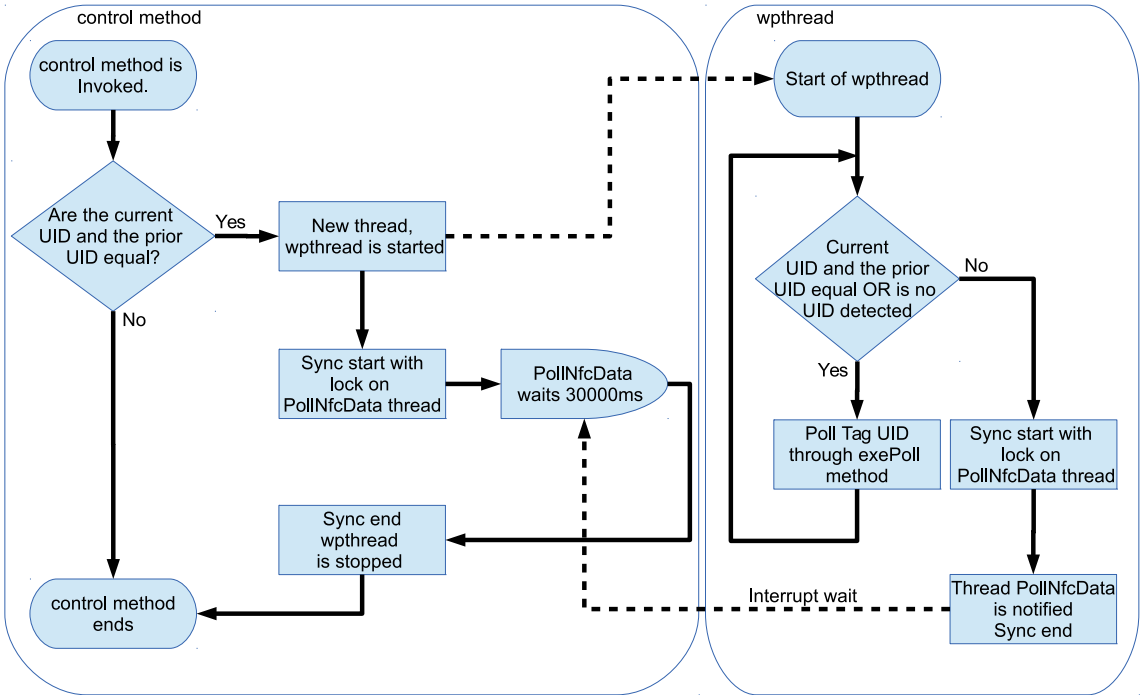


Figure 5.4: The flowchart shows the `control` method and the invoked `wpthread`. When the `control` method is invoked, the current NFC UID and the prior NFC UID are passed on to the method. The `control` method compares the NFC UIDs and sets further action in case the NFC UIDs are equal. The related text provides a more in depth explanation.

returned, the `sendEvent` method is invoked and in succession an event is triggered. Afterwards, the value of the current UID string is shifted to the previous UID string and the loop starts anew.

The mentioned `control` method, seen in Figure 5.4, provides two safety features. The first feature provides a relieve, for the recipient server, from permanent input stress, in case the AP would forget the NFC tag on the NFC-reader. This safety feature ensures that the servers process time wont be wasted on redundant information. The second feature secures, that a change of the applied tag is recognized, while the first safety feature is active. The significance of the second safety feature will be obvious, after the first feature is described in detail. The `control` method receives two parameters. The current UID and the previous UID. These two parameters are compared and in case they are equivalent, a synchronized block with a lock on the `PollNfcData` thread is created. The synchronized block further implies an instruction to freeze the `PollNfcData` thread for 30000ms and prevent the thread from triggering an event during this period. The event would otherwise trigger the

transmission to the server. This structure prevents the server from overload of redundant information and represents the first safety feature. In case the parameters

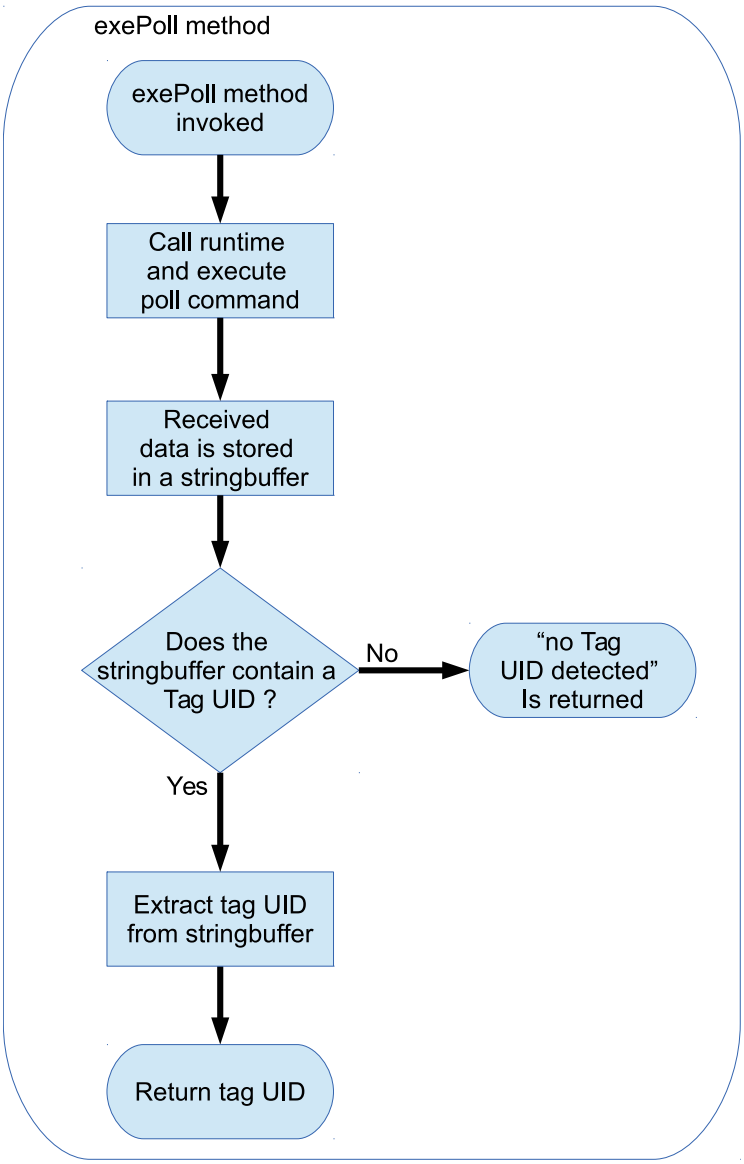


Figure 5.5: The `PollNfcData` thred invokes the `exePoll` method and passes a command which will be executed by the java runtime.

are different, the `control` method will end. As already explained the first safety feature implies a wait period, in case the AP forgets the tag on the NFC-reader. Given the possible situation that the AP would forget the tag on the NFC reader and after a while the AP wants to send a different request. The wait period from the first safety feature, would cause the request either to be sent after 3000ms or if the AP just briefly puts the tag on the NFC reader, it wouldn't be recognized at all. For this possible scenario the second safety feature was designed. Before the mentioned

synchronized block of the `control` method is activated, a second thread is started. This second thread polls for new tag UID's during the `PollNfcData` thread sleeps and while the polled tag UID stays the same. In case the tag UID changes, the new UID is handed to the `PollNfcData` thread and a synchronized block, with the same lock on the `PollNfcData` thread as the synchronized block of the first safety feature, is started. It is important that the lock, of the synchronized block, is on the exact same thread for this process to work. The synchronized block of the second feature implies a notification, which will wake up the `PollNfcData` thread, if the lock is the same. This ensures that a new tag UID is always recognized, even if the `PollNfcData` thread sleeps. This implementation represents the second safety feature.
After the wait period has ended or the `PollNfcData` thread was woken up, the second thread is stopped, in case it is still active and the end of the `control` method is reached. To poll a tag UID, `PollNfcData` needs to invoke the `exePoll` method, seen in Figure 5.5. As parameter a command of type string is passed to the `exePoll` method. The `exePoll` method calls for the Java runtime and passes the command to it. The command executed through the Java runtime will initiate the NFC-reader to poll for a NFC Tag, through the libnfc library. The returned input is allocated in a BufferedReader and afterwards transferred, line by line, into a Stringbuffer. If the Stringbuffer contains a tag UID, the UID is extracted from the Stringbuffer.

```
50 MESSAGE:
51 {
52    "topic": "eu/relaxedcare/event/sensorevent/nfc",
53    "properties": {
54       "messageUId": "<message-uid>",
55       "deviceId": "<device-id>"
56       "sensorevents": [
57          {
58             "sensorid": "<sensorid>", (e.g. the serial
                  number of the used sensor)
59             "sensordata": {
60                "value": "<tagID>"
61             },
62             "timestamp": <sensor event (linux)timestamp
                  nanoseconds> (e.g. 1399365461038825864)
63          }
64       ]
65    }
66 }
```

Listing 5.2: Definition of JSON string

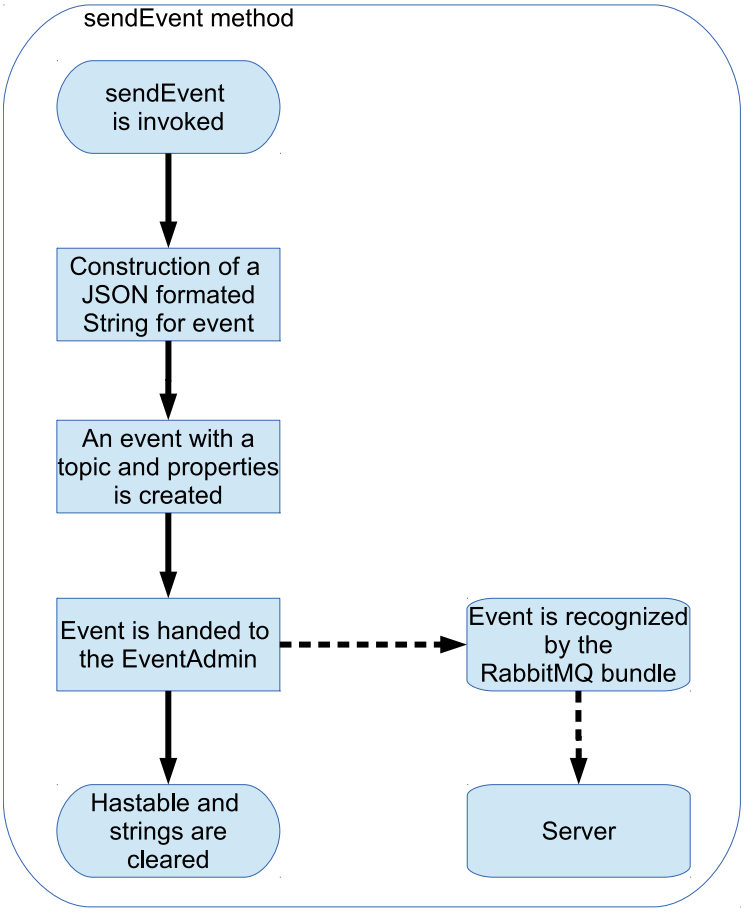


Figure 5.6: The `sendEvent` method creates a hastable of type String,String after it is invoked. The hastable holds a description tag and a value, in our case the description is "TagUID" and the value would be the eight-digit hex code.

Consequently the UID is returned to the thread or method, which called the `exePoll` method. In case the Stringbuffer holds no tag UID, "no tag detected" is returned. To trigger an event the `sendEvent` method, seen in Figure 5.6, is used in this approach. As a parameter the tag UID is passed to the `sendEvent` method. An Object of type Event needs properties and a specific topic. The required properties are constructed from a Hashtable of type <String,Object> and JSON objects. The fully composed JSON encoded string was defined within the project and can be seen in listing 5.2. In this approach the JSON string contains a topic, specific for the nfc event and properties. The properties furthermore, contain a `messageUId`, a `deviceId` and `sensorevents`, which are composed of a `sensorID`, the sensor data and a time-stamp. The `messageUId` ensures the uniqueness of each message, sent within the system. The `deviceId` and `sensorID` are, in our case, the same string. The key

value pair of `sensordata` holds the `tagID`, which will be sent. The last item is a linux time-stamp in nanoseconds. An object of type `Activator` is used as context to provide the Event Admin referenced in Blueprint. The event is passed to the Event Admin and consequently triggered. The RabbitMQ-bundle listens for this specific event and sends the JSON encoded string to a RabbitMQ server. The server maps the JSON encoded tagID with a request e.g. “Please call me“. Before the `sendEvent` method is closed, the created properties and used strings are cleared.

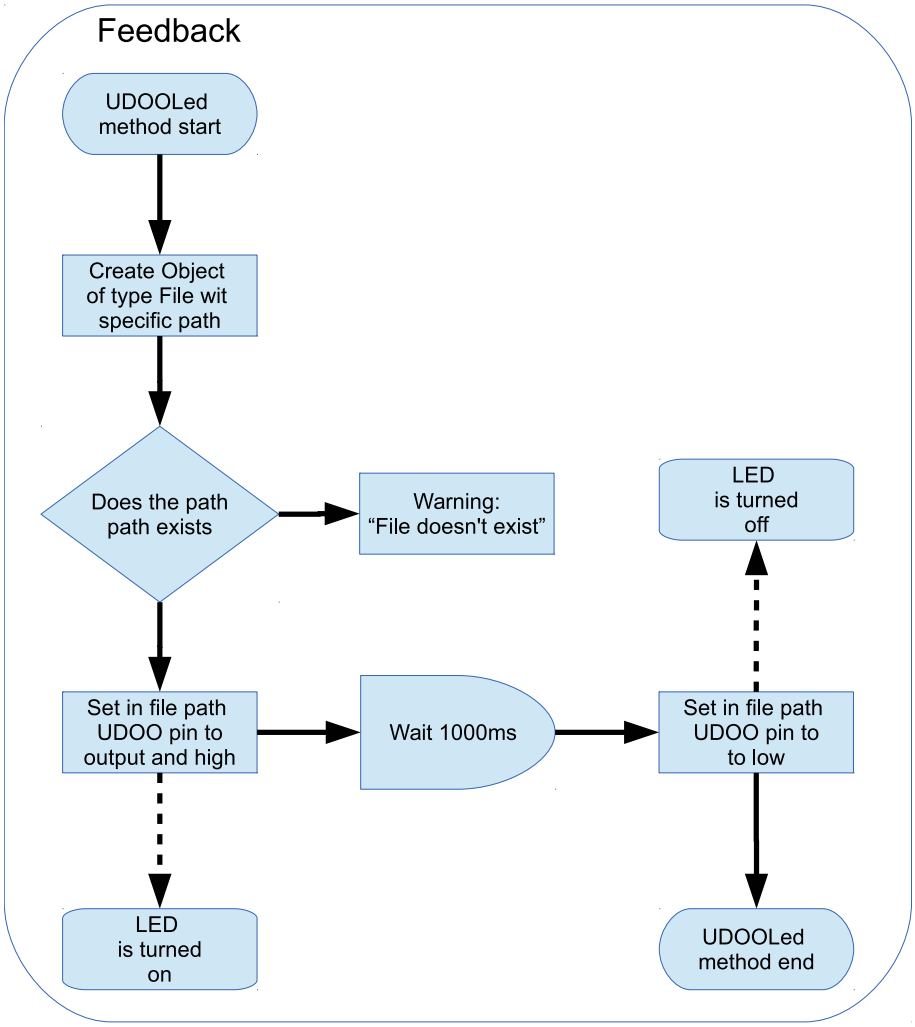


Figure 5.7: The given flowchart illustrates the `Feedback` class, which implements a response for the AP, in case a message is sent to the IC. The related text describes the `Feedback` class in more detail.

Feedback class

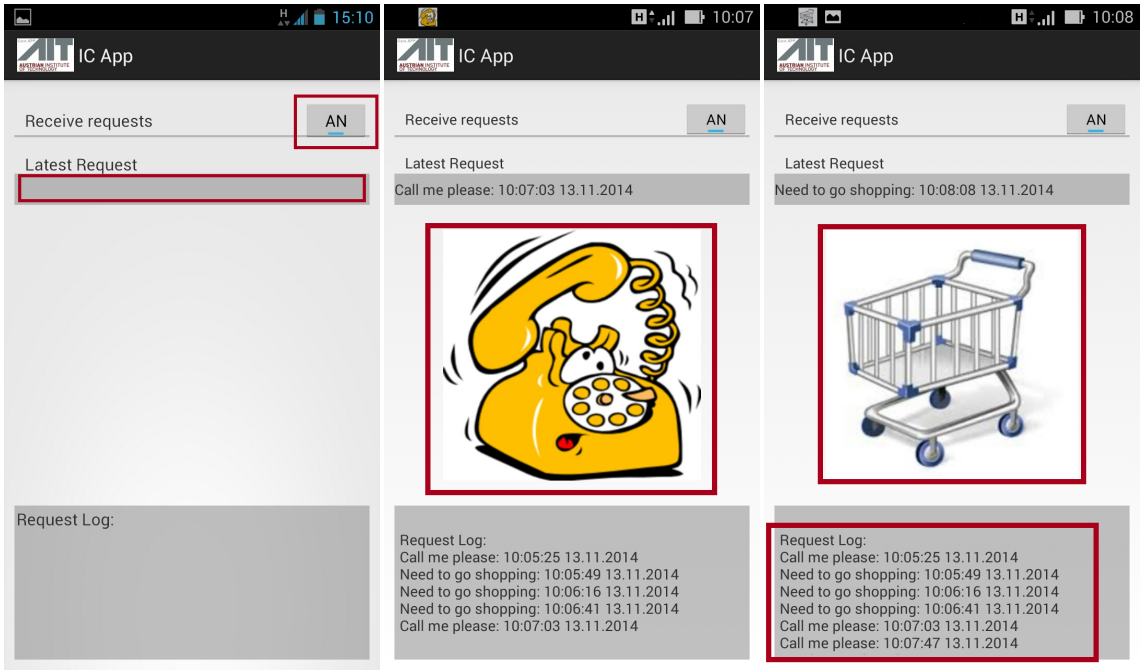
The `Feedback` class, seen in Figure 5.7, is meant to hold methods, which provide some kind of feedback for the AP, after a request was successfully transmitted to the server. So far, a method for feedback, on the UDOO quad board, was implemented. The UDOOLed method creates an object of type File, with a predefined path. If the file path exists, the file in the path is changed according to Chapter 4.2.2 and a pin on the UDOO board is set to logic high. Subsequently a LED is turned on. After a wait period of 1000ms, the file is changed again an the pin is set to logic low, which turns off the LED. More feedback methods will be implemented in the future, for different hardware and various display of feedback for the AP.

5.3 Informal Caregiver Application

The goal of the Android application is to provide visual and auditory notification for the IC, in the event of a received assistance request, in an unobtrusive manner. To ensure the unobtrusiveness of the application, the IC has the freedom to choose, if requests should be received or not. The application further provides a log of requests the AP sent in the past, so it is convenient for the IC to keep track .
The application can be run in the background without the concern to miss a request, due to a notification pop-up. The notification pop-up, is shown when a request is received, in the same manner as an email or a text message is received, on an android device.

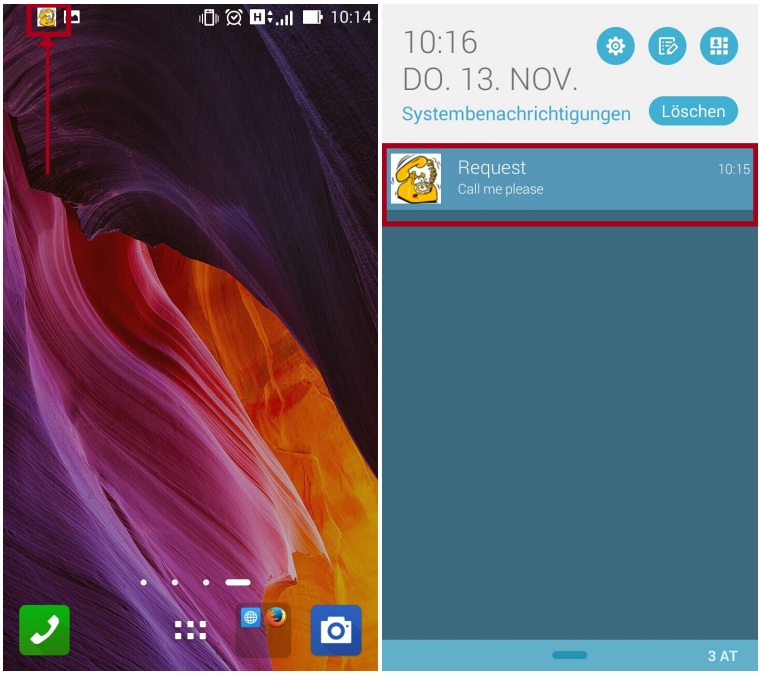
5.3.1 Graphical User Interface

The Graphical User Interface (GUI), shown in Figure 5.8 a, displays at the top right corner a button to start a server connection and thus receive requests from the AP. Further below a textview can be found which shows the latest received request.



(a) **(b)** **(c)**

Figure 5.8: The given figures show the user interface with different information outputs.



(a) **(b)**

Figure 5.9: The left hand side shows a status bar notification pop-up, during the application is run in the background. On the right hand side an illustration of the pulled down status bar is shown, which reveals a more informative outlook of the notification.

As shown in Figure 5.8 b and c, a visual notification for the received request is displayed in centre of the GUI. In the underpart of the GUI a scrollable log is placed, which displays previously received requests. All received requests are displayed with a request string and a time specification.
A notification pop-up, shown in Figure 5.9, safely ensures that a request is noticed by the IC, even if the application is run in the background. The pulled down notification, seen in Figure 5.9 b, instantly takes the user to the main user interface.

5.3.2 Application

The IC Application consists of three classes, which are the `MainActivity` class, the `ConnectToRabbitMQServer` class and the `MessageConsumer` class. For more insight on the big picture of the IC application the following part will provide information, how the main components of the android client are linked together.

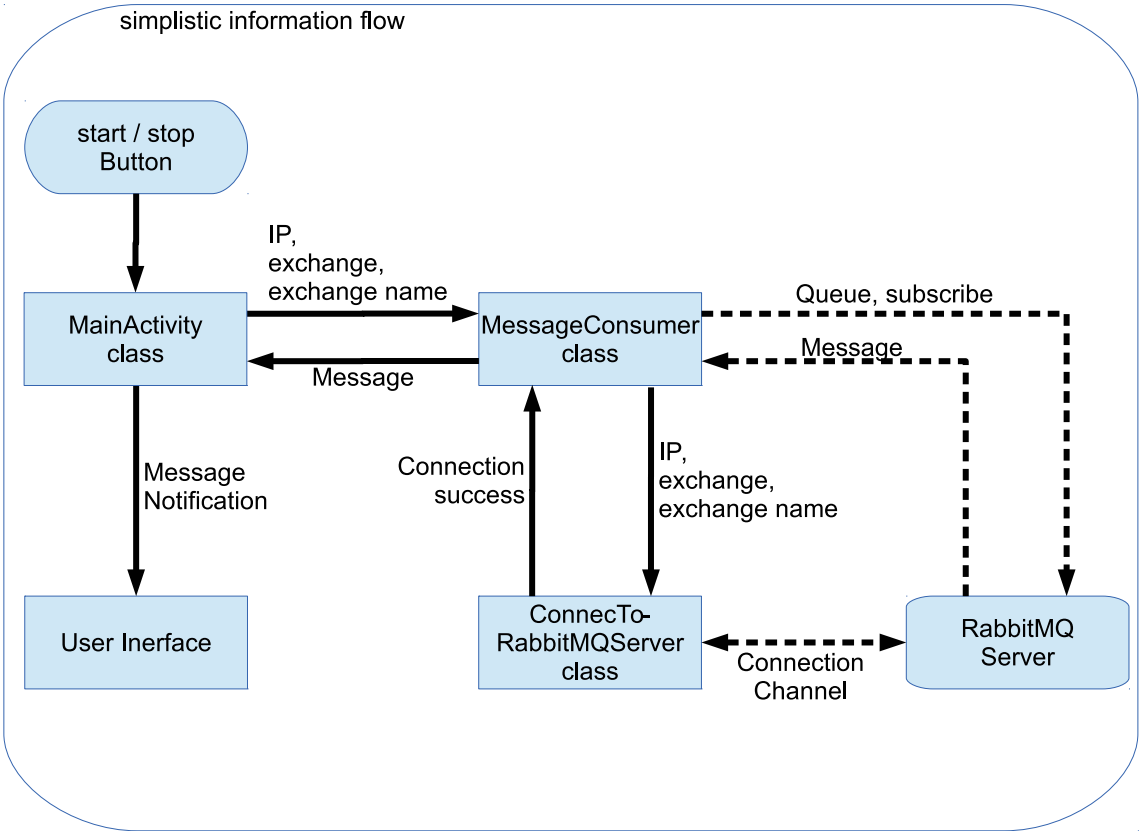


Figure 5.10: The figure shows a simplistic information flow for the android caregiver application. It is meant to provide a better context of the single classes. The continuing text offers a more in depth explanation on this topic.

Information flow overview

As in Figure 5.10 shown, the `MainActivity` class is as anticipated the initial point of operation. Moreover, it provides the IP address, the exchange name and the exchange type to the `MessageConsumer` class. The `MessageConsumer` class is developed as an inheritance of the `ConnectToRabbitMQServer` class. The `MessageConsumer` class passes the former mentioned parameters to the superclass.
The `ConnectToRabbitMQServer` class then uses this parameters to connect to a RabbitMQ server. In case of a successful connection the class permits the `MessageConsumer` class to subscribe to a queue and receive messages from the server.
A received message is then posted back to the `MainActivity` class which displays the information in the user interface. The upcoming text explains the relevant classes and methods in more detail.

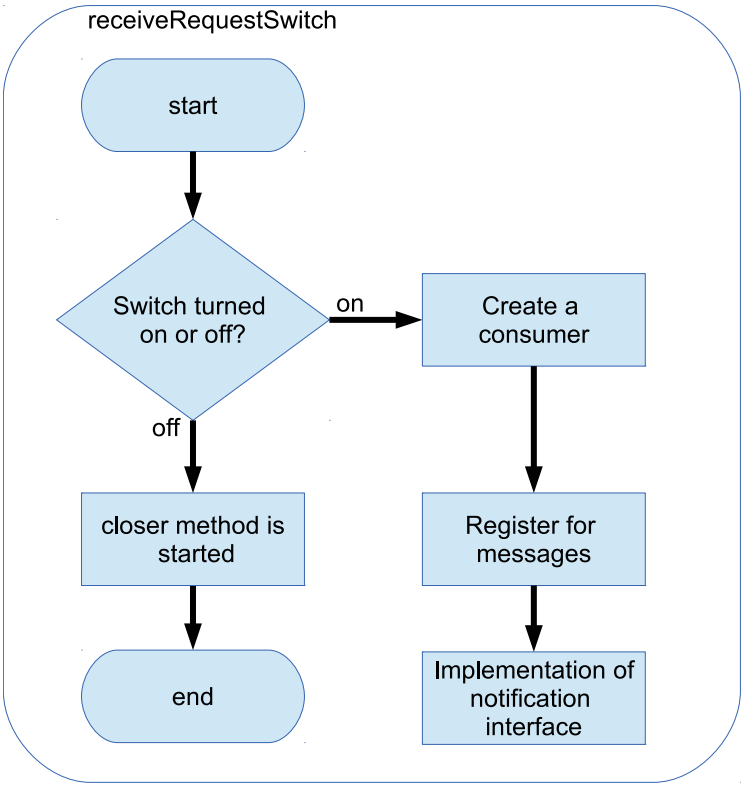


Figure 5.11: Flowchart of the `receiveRequestSwitch` method. The method is invoked by selecting the receive requests button. This method induces the client-server communication and gives the user the freedom of choice to receive or not to receive requests. An in depth description is provided in the continuing text.

MainActivity class

All methods comprised by the `MainActivity` class are thoroughly explained in this section. For information about the application life cycle under android, please refer to the referenced literature in Chapter 4.5. The `receiveRequestSwitch` method of the `MainActivity` class, shown in Figure 5.11, is called when the received request button shown in Figure 5.8 is pushed. The method checks if the button is turned on or off. If the button is on, an object of type `MessageConsumer` is created. The server IP, an exchange string and the exchange type is handed to the object. A basic connection to the message broker is invoked and the registration for messages is set through the method. In addition the `ReceivedRequestSwitch` method contains the implementation of the notification interface for the `MessageConsumer` class, which will be explained in more detail further below. As mentioned the application can be

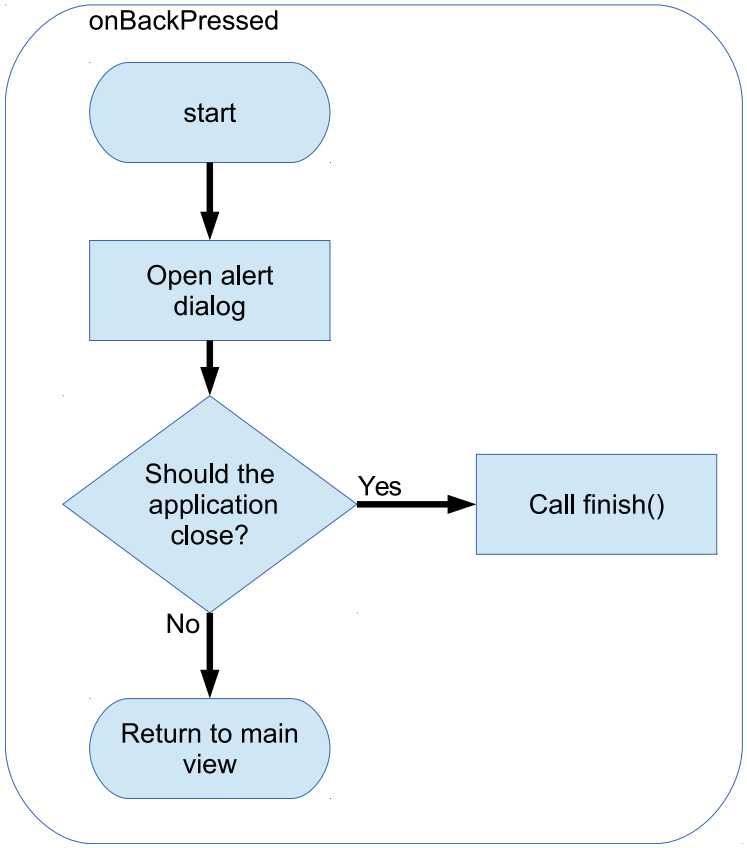


Figure 5.12: The given flowchart displays the `onBackPressed` method. This method is called by selecting the back button on the android device. An alert dialogue is called by the method which gives the user the choice to proceed in which case the finish method is executed and consequently the `onDestroy` method is invoked.

run in background. To achieve this work it is essential, that the server connection is sustained. Subsequently the user should put the application in background by selecting the HOME button and not the back button, which would close the application an in addition the server connection. In case the user selects the back button unintentionally the `onBackPressed` method is invoked. A flowchart of the `onBackPressed` method can be seen in Figure 5.12. The `onBackPressed` method displays an alert message, which reminds the user that the server connection will closed. The application gives the user the possibility to proceed and therefore detach the client-server connection or to chose to stay on the main view. The application likewise reminds the user to use the home button if the user's intention was to run the application in background. A screenshot of the alert message can be seen in Figure 5.13 If the

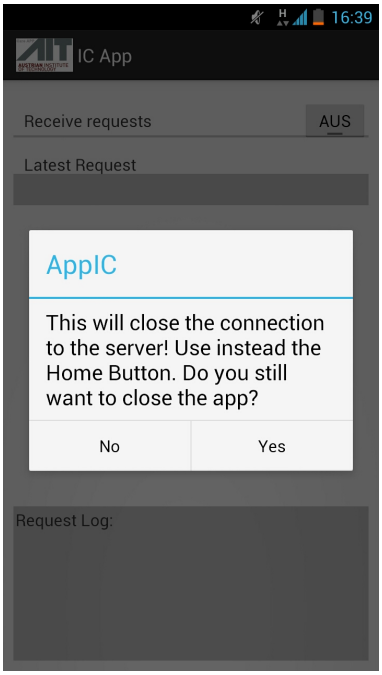


Figure 5.13: The figure shows an alert message in the event the user would select the back button. May it be with intent or unconsciously the application however reminds the user that if he or she proceeds the client-server connection will be severed. The application suggests to use the home button in case the user intents to run the application in background.

user chooses not to close the connection the main view will be displayed. In case the user wants to close the connection the finish method is invoked which will call the `onDestroy` method of the android lifecycle. In Figure 5.14 a the `onDestroy` method is shown. The method calls the `closer` method, before the application in closed.

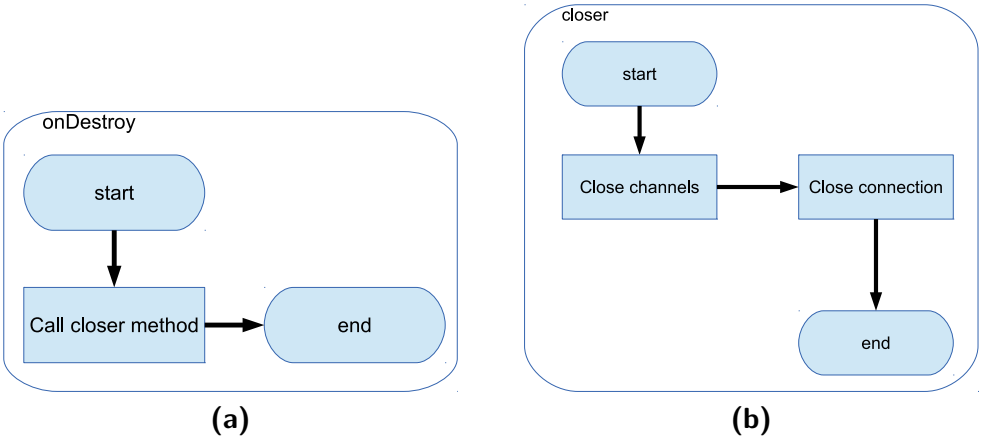


Figure 5.14: On the left hand side the `onDestroy` method is illustrated. This method is executed right before an application is closed and it's only task is to call for the `closer` method which is presented on the right hand side. More information can be found in the correlating text.

The `closer` method can seen in 5.14 b.

The `closer` method creates a thread which first closes the established RabbitMQ channels. After the channels are closed the thread secondly closes the connection between the RabbitMQ server and the android client. The method ends after closing the server connection. The mentioned implementation of the message handler interface located in the `MessageConsumer` class is provided through the `onReceiveNotification` method, seen in Figure 5.15. The byte message received in the MessagConsumer class is converted in a UTF8 standard. Furthermore, the received message is JSON encoded and thus the request has to be extracted. The request is afterwards displayed in the latest request textview, consisting of the request string and a time stamp. The latest request is then added to the request log. A switch case statement is used to distinguish the type of the request. Depending on the request type a visual notification is set visible for the user. Furthermore the same visual notification is selected as icon for the pop-up notification in the status bar.

A default ringtone is set up by the ringtone manager for the notification pop-up. Afterwards a notification builder is used to construct a notification pop-up by handing a context, which is related to the notification, in our case the `MainActivity` class. Further, a notification string (which is the request string), the mentioned icon as well as the default notification sound and a request title are passed to the notifica-

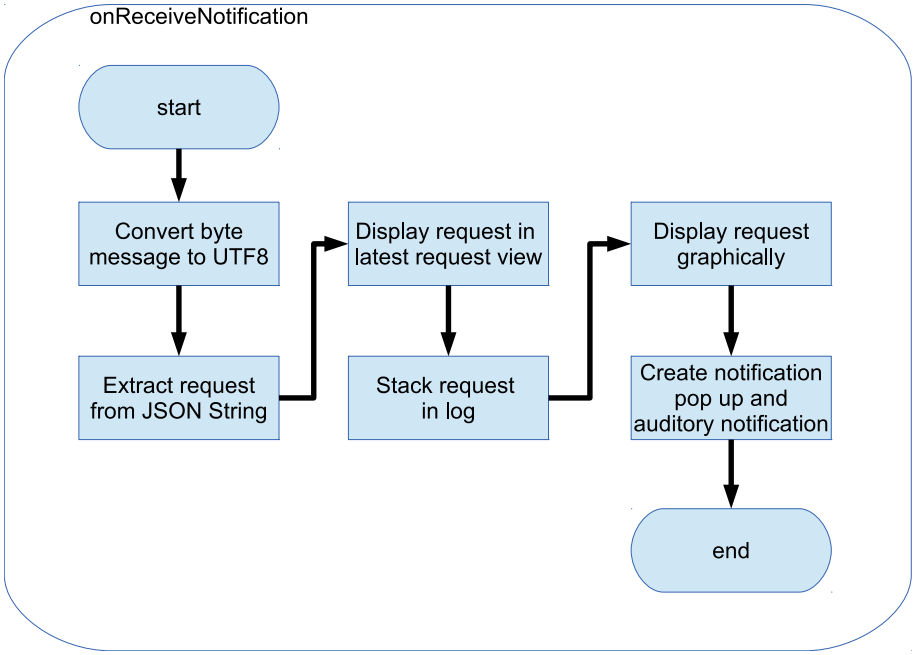


Figure 5.15: The given flowchart represents the implementation of the message handler interface method located in the `MessageConsumer` class.
Conversion of the received byte messages as well as the construction of a notification and displaying the received information are accomplished by this implementation. A comprehensive explanation is shown in the correlating text.

tion builder. The constructed notification is then triggered and visible for the user to interact.

ConnectToRabbitMQServer class

The upcoming part focuses on the `ConnetToRabbitMQServer` class. This class is designed as the base for objects that connect to a RabbitMQ server. It holds a constructor for basic information acquiring and a `connectToRabbitMQ` method which manages the communication with the RabbitMQ server. The parameters passed to the constructor of the class imply the IP address of the RabbitMQ server, a name for the exchange and an exchange type. The former mentioned `connectToRabbitMQ` method is of type boolean and returns true, if the communication with the server is successfully established. An illustration of the method is given in Figure 5.16. The method first checks if a previous channel still exists and if it is open. If this is the case, true is returned. Otherwise a new object of type `connectionFactory` is

created. Through this object's methods the server IP address is set, as well as the username and password. Optional a port can be appointed. If no port is given a standard port is used.

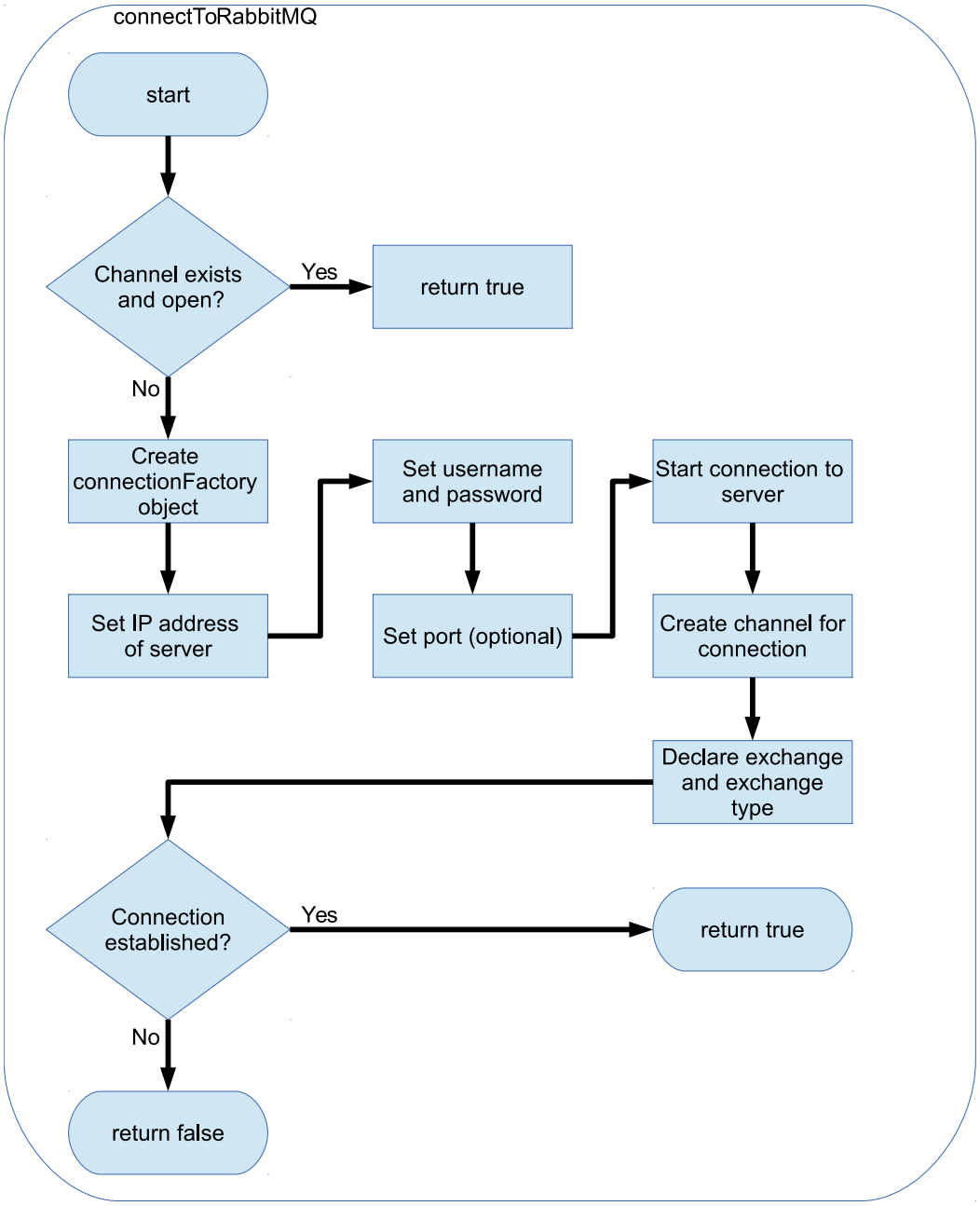


Figure 5.16: The shown flowchart represents the `connectToRabbitMQ` method of the ConnetToRabbitMQServer class. The method is of type boolean and is in charge of the server client connection. In case of a valid communication with the RabbitMQ server the method returns a boolean true .A more sufficient explanation is provided in the associated text above.

A connection to the server is then established with the mentioned parameters and a channel for the connection is created. In the last steps of the method, the exchange is declared using the exchange name and type. Again a boolean true is returned if the connection is valid.

MessageConsumer class

The `MessageConsumer` class extends the `ConnectToRabbitMQServer` class. It is designed to use the basic client-server connection, the `ConnectToRabbitMQServer` class provides, and to add methods for message consumption.
The constructor of the `MessageConsumer` class receives as parameters, exactly as the `ConnectToRabbitMQServer` class, the IP address of the RabbitMQ server, the exchange name and the exchange type. Additionally the constructor passes the parameters to it's superclass. The client application has to subscribe for a queue on the RabbitMQ server. The queue name is defined in the field of the `MessageConsumer` class. Furthermore, this class provides an interface, that has to be implemented by an object that is interested in receiving messages. The implementation of the interface is located in the `MainActivity` class. For better understanding the flowchart in Figure 5.15 may be recapitulated.
Additional elements of the `MessageConsumer` class field are the used handlers. The runnables required for the handlers are developed on the one hand to execute the consumer method. An in depth explanation of the consumer method is provided further below in the text. On the other hand, to pass on a received byte message from the server to the former mentioned interface and hence to post back the message, to the implementation in the MainActivty class. The `MessageConsumer` class extends the `connectToRabbitMQ` method provided in the superclass.
An illustrated flowchart of the extended method is shown in Figure 5.17. This method is executed from the `MainActivity` class. It first checks if the superclass has successfully connected to the RabbitMQ server. A successful connection permits the method to declare a queue with the provided queue name and appropriate parameters. A subscription to this queue on the RabbitMQ server can then be established. Immediately upon the subscription to the queue the runnable, which holds the consumer method, is posted to the handler and therefore executed. The method returns then a boolean true. If the superclass would fail to connect to the

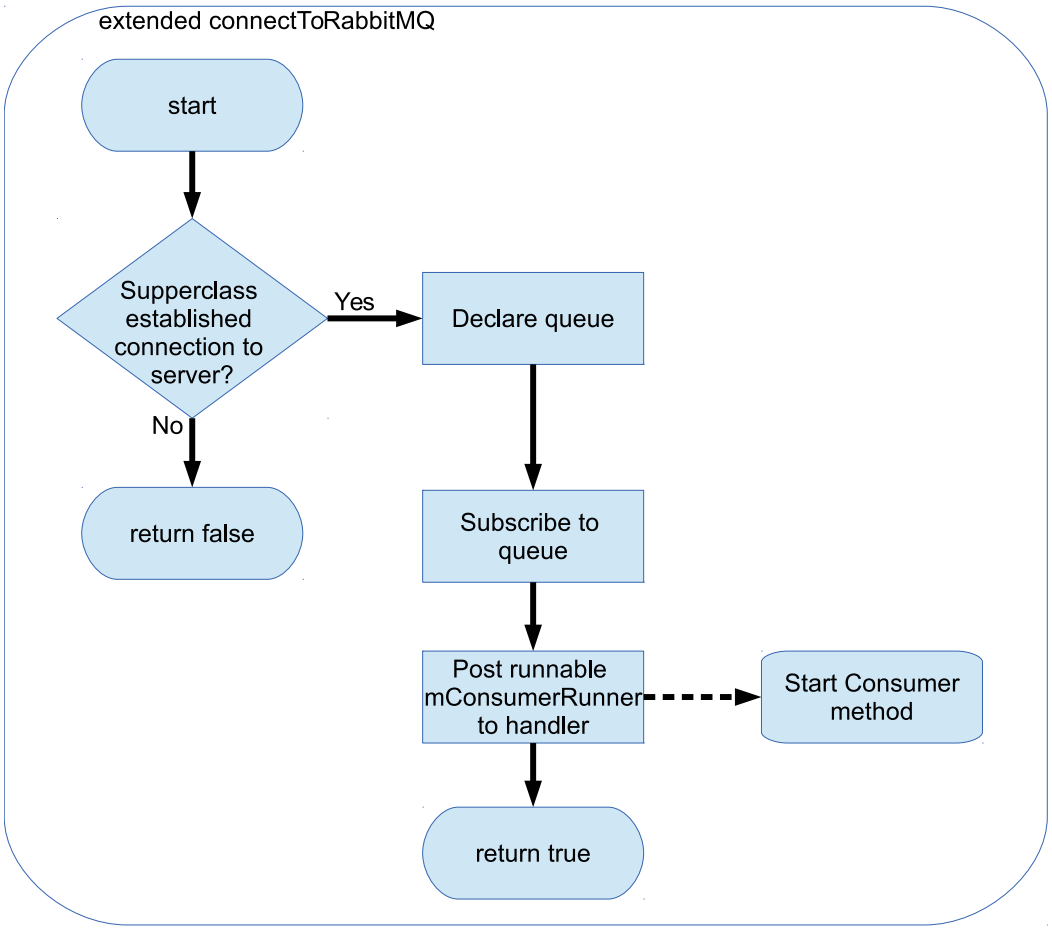


Figure 5.17: The illustrated flowchart shows the extended `connectToRabbitMQ` method located in the `MessageConsumer` class. This extension is essential for the subscription to a queue provided by a RabbitMQ server. For more insight on this method please look up the continuing text.

server, the method returns a boolean false.

The upcoming part focuses on the consumer method, which is shown in Figure 5.18. The `consumer` method starts a thread, which executes a nextDelivery method from the RabbitMQ library to wait for the next message from the server and to return it to a Delivery object, likewise provided through the RabbitMQ library. In the next step the message body is shifted to an object of type byte.

Afterwards the secondly mentioned runnable is passed to a handler. This step posts back the byte object to the implementation of the interface to the `MainActivity` class and the message is presented to the user, as formerly explained. As last action the `consumer` method acknowledges the received message.

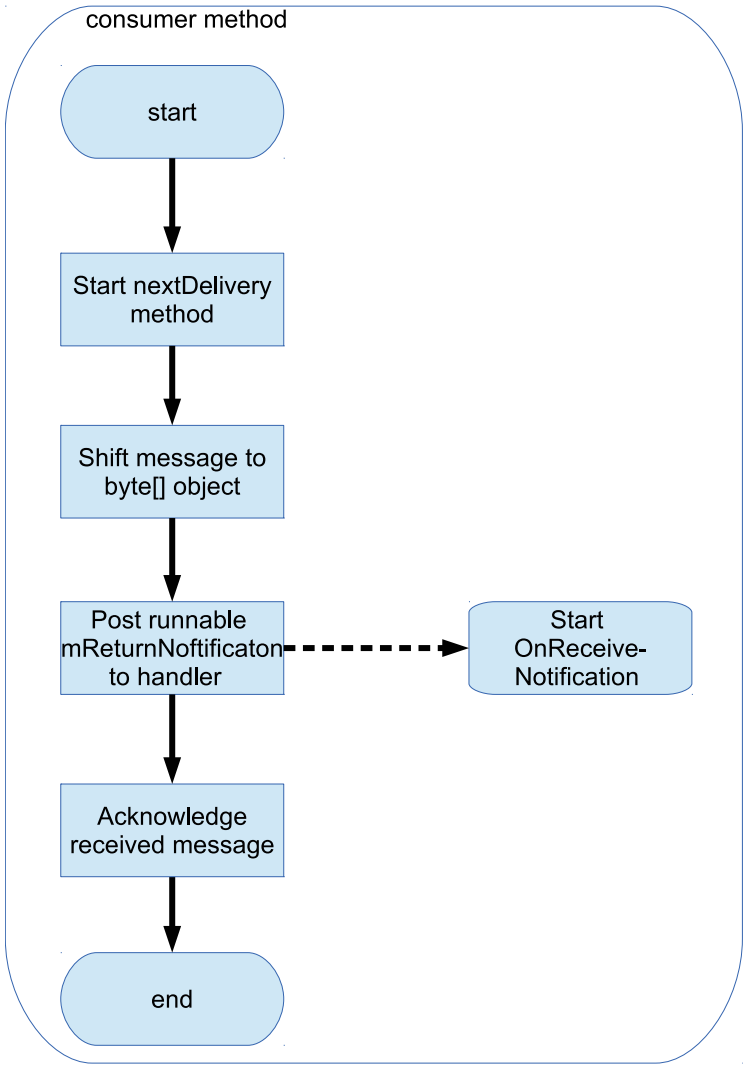


Figure 5.18: The given flowchart shows the `consumer` method located in the `MessageConsumer` class. This method waits for a message to be received. In addition the method posts the `onReceiveNotification` runnable to a handler. This action passes the received message back to the `MainActivity`, where it is shown to the user. More information is provided in the correlation text.

6. Discussion and Lessons Learned

The work done in this thesis showed, that it is definitely possible to use ARM boards for home automation. Technology in the field of ARM development, advanced so far, that fast, multiple core ARM boards with enough memory, are achievable for a reasonable price. The HOMER software, used in this thesis for home automation, was runnable on all the selected ARM boards. However, with differences in the performance, dependent on the specific ARM board.

In regards of stability, further trials would be necessary, since the test period of one week may be to short to make a significant impression. Furthermore, the HOMER software is still in development and as explained the software should be provided as open source, correspondingly developers have the possibility to adjust it to ARM platforms.

The developed OSGi bundle for NFC communication on ARM boards, is successfully operational. Although the use of the `libnfc` library was necessary, further research in regards of PCSC on ARM platforms and the access of the Java framework to PCSC, could improve this issue. During the test period the NFC bundle proved to be stable. However, as applied on the stability of the whole HOMER software, a longer test period would be beneficial to articulate a more accurate statement on this topic.

In concerns of the Android application for the IC, all basic functionalities are provided. Nonetheless, the IC application is far from a final end user product. Despite, some minor issues, which can be related to the fact, that the application is still a prototype, the appearance of the application may not be pleasing for an end user. The visual design of the application should be improved, to be more appealing, for the IC to use.

Summarized, the knowledge gained from this work showed that a home automa-

tions concept with ARMs is reasonable. Drawbacks, such as the PCSC issue on ARMs in conjunction with Java, require further research. In concerns of stability of the HOMER software on ARM boards, the best experience was achieved with the UDOO quad board, which showed no issues during the test period. However, further testing with longer periods is required. In addition to a stability test, a use case trial, would exhibit potential issues with the appliance of the developed technology for the AP, as well as for the IC. This task may be applied in future work. In respect to the IC application, an improvement of the visual design would be beneficial, for an end user product.

The benefit gained from the use of an ARM board for this scenario, is beneath an advantage in regards of design, the financial advantage, due to low prices for ARM boards compared to standard PCs. The small size of ARM boards enable the end user product, to be integrated in various objects, depending on the preferences of the AP. The choice to use Android as the operating system for the IC application facilitates, beside the mobile possibilities Android offers, the possibility to reach a large group of society.

7. Conclusions

Considering the knowledge gained in the course of this thesis, the conclusion to use ARM boards for home automation proved to be possible. Especially, due to the technical progress ARM boards have recently undergone, nowadays powerful and reasonably low cost products, are available in this field. A variety of operating systems are adaptable for use, depending on the intended application, although these mostly Linux-based. Therefore, Java-based applications, such as the HOMER software used in this thesis, can be employed, since the various Linux distributions provide a Java Virtual Machine. However, the development progress of low-cost ARM boards is still ongoing and hopefully future products will provide improved hardware and industrial support, to offer even more feasible products for home automation and AAL.

Regarding the selection of the operating system for the IC, Android seems to be a favourable choice, considering the substantial acceptance this operating system enjoys in the consumer market. However, for the IC, any mobile operating system, which provides a RabbitMQ client implementation, would be acceptable.

With respect to the entire Assistance Request scenario implemented on an ARM platform, the prototype is operational and ready for a first trial with users.

8. Future Work

The work done in this thesis facilitates the possibility, to be used as a basis in the scope of the RelaxedCare project, as an ARM based alternative for the Assistance Request scenario.
Future work, should focus on an out of the box functionally for the HOMER software, in regards to the communications with NFC. In more detail, the interaction of Java and PCSC on ARM should be improved. Furthermore, longer stability tests should be realized, to provide more insight on the long time behaviour, as well as trials with end users to eliminate possible issues with the appliance of the Assistance Request scenario. Object designs, for a final product, in regards of an unobtrusive appearance for the AP should not be neglected in future work as well.
However, the course of future avenues lies within the decision of the RelaxedCare project, which in turn has to clear questions, in regards of the acceptance of the RelaxedCare project with the end users, cost related questions and questions related to the applied scenarios. Especially, the cost related questions would benefit from the use of low cost ARM boards, for AAL applications. The crystallization of these questions determines the course for future work, established on the performance done in this thesis.

List of Figures

List of Tables

References

Ahson, Syed and Mohammad Ilyas [2011]. *Near Field Communications Handbook*. Auerbach Publications, Boca Raton, USA. ISBN 9781420088151, 380 pages.

Alhamoud, Alaa, Felix Ruettiger, Andreas Reinhardt, Frank Englert, Daniel Burgstahler, Doreen Böhnstedt, Christian Gottron, and Ralf Steinmetz [2014]. *SMARTENERGY. KOM: An Intelligent System for Energy Saving in Smart Home*. In Nils Aschenbruck, Salil Kanhere, Kemal Akkaya (Editor), *Proceedings of the 3rd IEEE LCN International Workshop on GlObal Trends in SMART Cities (IEEE goSMART 2014)*, pages 685–692. EDAS Conference Services. ISBN 978-1-4799-3784-4.

Alvaro Videla and Jason J.W. Williams [2012]. *RabbitMQ in Action: Distributed Messaging for Everyone*. Manning Publications. ISBN 1935182978, 312 pages.

Apache Maven [2014]. *The Apache Maven project*. `https://maven.apache.org/`. Last access 11/2014.

Bolin, K., B. Lindgren, and P. Lundborg [2008]. *Informal and formal care among single-living elderly in Europe. Health Economics*, 17(3), pages 393–409. ISSN 10991050. doi:10.1002/hec.1275.

Boulos, Maged N Kamel, Steve Wheeler, Carlos Tavares, and Ray Jones [2011]. *How smartphones are changing the face of mobile and participatory healthcare: an overview, with example from eCAALYX. Biomedical engineering online*, 10(1), page 24. ISSN 1475-925X. doi:10.1186/1475-925X-10-24.

Chan, M., C. Hariton, P. Ringeard, and E. Campo [1995]. *Smart house automation system for the elderly and the disabled*. In *1995 IEEE International Conference on Systems, Man and Cybernetics. Intelligent Systems for the 21st Century*,

volume 2, pages 1586–1589. IEEE, New York, NY, USA. ISBN 0-7803-2559-1. doi:10.1109/ICSMC.1995.537998.

CubieBoard [2014]. *A series of open source hardware.* `http://cubieboard.org/`. Last access 09/2014.

Cubo, Javier, Adrián Nieto, and Ernesto Pimentel [2014]. *A cloud-based Internet of Things platform for ambient assisted living. Sensors*, 14(8), pages 14070–105. ISSN 1424-8220. doi:10.3390/s140814070.

de Castro Alves, Alexandre [2011]. *OSGi in Depth.* 1st Edition. Manning Publications Co., Greenwich, USA. ISBN 193518217X, 9781935182177, 392 pages.

De Luca, Gabriele, Paolo Lillo, Luca Mainetti, Vincenzo Mighali, Luigi Patrono, and Ilaria Sergi [2013]. *The use of NFC and Android technologies to enable a KNX-based smart home.* In Rozic, N; Begusic, D (Editor), *2013 21st International Conference on Software, Telecommunications and Computer Networks - (SoftCOM 2013)*, pages 1–7. IEEE, New York, NY, USA. ISBN 978-953-290-040-8. ISSN 1848-1744. doi:10.1109/SoftCOM.2013.6671904.

Dossot, David [2014]. *RabbitMQ Essentials.* Packt Publishing Ltd., Birmingham, UK. ISBN 9781783983209, 182 pages.

Doumas, Diane [2014]. *Advanced Sensor Technology Enhances Operational Effectiveness in Senior Living.* `www.healthsense.com/images/phocadownload/WhitePapers/wp_operational_effectiveness.pdf`. Last access 09/2014.

Follmann, Rüdiger [2014]. *Das Raspberry Pi Kompendium.* Xpert.press, Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-54910-6, 197–223 pages. doi:10.1007/978-3-642-54911-3.

Fuxreiter, Thomas, Matthias Gira, Lukas Roedl, Johannes Kropf, and Sten Hanke [2011]. *Rule-based indoor localization using non-intrusive domotic sensors and the HOMER platform.* In Kung, Antonio, Francesco Furfari, Mohammad-Reza (Saied) Tazari, Kush Kwadhwa, Reza Razavi, Marius Mikalsen, Barbara Raither, Joe Gorman, and Marco Eichelberg (Editors), *Proceedings of the AAL Forum 2011*, pages 569 – 576. Smart Homes, Endhoven. ISBN 9789081970907.

Fuxreiter, Thomas, Christopher Mayer, Sten Hanke, Matthias Gira, Miroslav Sili, and Johannes Kropf [2010]. *A modular platform for event recognition in smart homes.* In *The 12th IEEE International Conference on e-Health Networking, Applications and Services*, pages 1–6. IEEE. ISBN 978-1-4244-6374-9. doi: 10.1109/HEALTH.2010.5556587.

Gebhardt, Jan, Michael Massoth, Stefan Weber, and Torsten Wiens [2014]. *Ubiquitous Smart Home Control on a Raspberry Pi Embedded System.* In Mauri, Jaime Lloret, Christoph Steup, and Sönke Knoch (Editors), *UBICOMM 2014 : The Eighth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 172–177. IARIA, Wilmington, USA. ISBN 978-1-61208-353-7. ISSN 2308-4278.

Georgieff, Peter [2008]. *Ambient assisted living – Marktpotenziale IT-unterstützter Pflege für ein selbstbestimmtes Altern.* 17. ISSN 18615066. `http://creative-labs.org/fileadmin/_fazit-forschung/downloads/FAZIT-Schriftenreihe_Band_17.pdf`.

Giannakouris, Konstantinos [2008]. *Ageing characterises the demographic perspectives of the European societies. Eurostat: Statistics in Focus*, 72, pages 1–11. ISSN 19770316.

Goodyear, Jamie and Johan Edstrom [2013]. *Instant OSGi Starter.* Packt Publishing, Birmingham, UK. ISBN 1849519927, 9781849519922, 58 pages.

GrandCare Systems [2014]. `http://www.grandcare.com/`. Last access 09/2014.

Gurek, Alper, Caner Gur, Cagri Gurakin, Mustafa Akdeniz, Senem Kumova Metin, and Ilker Korkmaz [2013]. *An Android based home automation system.* In *2013 High Capacity Optical Networks and Emerging/Enabling Technologies*, pages 121–125. IEEE, New York, NY, USA. ISBN 978-1-4799-2569-8. doi: 10.1109/HONET.2013.6729769.

Hall, Richard, Karl Pauls, Stuart McCulloch, and David Savage [2011]. *Osgi in Action: Creating Modular Applications in Java.* 1st Edition. Manning Publications Co., Greenwich, CT, USA. ISBN 9781933988917, 548 pages.

Hanson, Mark A., Adam T. Barth, and Christopher Silverman [2011]. *In Home Assessment and Management of Health and Wellness with BeClose™; Ambient, Artificial Intelligence.* In *Proceedings of the 2Nd Conference on Wireless Health*, pages 25:1–25:2. WH '11, ACM, New York, NY, USA. ISBN 9781450309820. doi:10.1145/2077546.2077574.

Holzinger, Andreas, Kizito Ssamula Mukasa, and Alexander K. Nischelwitzer [2008]. *Introduction to the Special Thematic Session: Human–Computer Interaction and Usability for Elderly (HCI4AGING).* In Miesenberger, Klaus, Joachim Klaus, Wolfgang Zagler, and Arthur Karshmer (Editors), *Computers Helping People with Special Needs*, *Lecture Notes in Computer Science*, volume 5105, pages 18–21. Springer, Berlin Heidelberg. ISBN 9783540705390.

Igarashi, Yuichi, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting [2014]. *Vision: Towards an Extensible App Ecosystem for Home Automation through Cloud-Offloa.* In *Proceedings of the fifth international workshop on Mobile cloud computing & services - MCS '14*, pages 35–39. MCS '14, ACM Press, New York, New York, USA. ISBN 9781450328241. doi:10.1145/2609908.2609949.

Jain, Sarthak, Anant Vaibhav, and Lovely Goyal [2014]. *Raspberry Pi based interactive home automation system through E-mail.* In *2014 International Conference on Reliability Optimization and Information Technology (ICROIT)*, pages 277–280. IEEE, New York, NY, USA. ISBN 978-1-4799-2995-5. doi:10.1109/ICROIT.2014.6798330.

Junqueira Barbosa, Simone Diniz, Phoebe Chen, Alfredo Cuzzocrea, Xiaoyong Du, Joaquim Filipe, Orhun Kara, Igor Kotenko, Krishna M. Sivalingam, Dominik Ślezak, Takashi Washio, Xiaokang Yang, Angelo Costa, Oscar Gama, Paulo Novais, and Ricardo Simoes [2014]. *A Different Approach in an AAL Ecosystem: A Mobile Assistant for the Caregiver.* In Corchado, Juan M., Javier Bajo, Jaroslaw Kozlak, Pawel Pawlewski, Jose M. Molina, Benoit Gaudou, Vicente Julian, Rainer Unland, Fernando Lopes, Kasper Hallenborg, and Pedro Garcia Teodoro (Editors), *Highlights of Practical Applications of Heterogeneous Multi Agent Systems. The PAAMS Collection*, *Communications in Computer and Information Science*, volume 430, pages 101–110. Springer International Pub-

lishing, Cham, Switzerland. ISBN 978-3-319-07766-6, 978-3-319-07767-3. doi: 10.1007/978-3-319-07767-3_10.

Kovacshazy, Tamas, Gabor Wacha, Tamas Daboczi, Csanad Erdos, and Attila Szarvas [2013]. *System architecture for Internet of Things with the extensive use of embedded virtualization.* In *2013 IEEE 4th International Conference on Cognitive Infocommunications (CogInfoCom)*, pages 549–554. IEEE, New York, NY, USA. ISBN 978-1-4799-1546-0. doi:10.1109/CogInfoCom.2013.6719308.

Kropf, J., L. Roedl, and A. Hochgatterer [2012]. *A modular and flexible system for activity recognition and smart home control based on nonobtrusive sensors.* In *6th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2012*, pages 245–251. IEEE, San Diego, CA. ISBN 978-1-4673-1483-1.

Lalou, Jonathan [2013]. *Apache Maven Dependency Management: ProQuest Tech & Business Books - English/German.* Packt Publishing Ltd., Birmingham, UK. ISBN 9781783283019, 158 pages.

Langer, Josef and Michael Roland [2010]. *Anwendungen und Technik von Near Field Communication (NFC).* Springer, Berlin Heidelberg. ISBN 9783642054969, 265 pages.

libnfc [2014]. *libnfc reference manual.* `http://www.libnfc.org`. Last access 11/2014.

Liu, Yuansheng [2006]. *Design of the Smart Home based on embedded system.* In *2006 7th International Conference on Computer-Aided Industrial Design and Conceptual Design*, pages 1–3. IEEE, New York, NY, USA. ISBN 1-4244-0683-8. doi:10.1109/CAIDCD.2006.329442.

Live!y [2014]. *Personal Emergency Response – Reimagined.* `http://www.mylively.com/`. Last access 09/2014.

Mei, Fang, Xuanjing Shen, Haipeng Chen, and Yingda Lv [2011]. *Embedded Remote Video Surveillance System Based on ARM. Journal of Control Engineering and Applied Informatics*, 13(3), pages 51–57. ISSN 1454-8658.

Morandell, Martin, Jonathan Steinhart, Emanuel Sandner, Sandra Dittenberger, Andrea Koscher, and Martin Biallas [2014]. *RelaxedCare: A Quiet Assistant for Informal Caregivers*. In *CSCW 2014 Workshop on Collaboration and Coordination in the Context of Informal Care (CCCiC 2014)*, pages 10–21. ACM, Baltimore, Maryland (USA). ISSN 102173643.

National Alliance for Caregiving [2011]. *eConnected Family Caregiver: Bringing Caregiving into the 21st Century*. `http://www.caregiving.org/data/FINAL_eConnected_Family_Caregiver_Study_Jan%202011.pdf`. Last access 09/2014.

Oracle [2006]. *Java Specification Request 268: Smart Card I/O API*. `https://docs.oracle.com/javase/7/docs/jre/api/security/smartcardio/spec/javax/smartcardio/package-summary.html`. Last access 11/2014.

OSGi Alliance [2013]. *OSGi Alliance: The OSGi Compendium, Release 5. The OSGi Alliance*.

Pers+™[2014]. *Safety and Independence for Active Older Adults Use Case*. `http://www.healthsense.com/phocadownload/usecases/pers_plus_final.pdf`. Last access 09/2014.

Pochobradsky, Elisabeth, Franz Bergmann, Harald Brix-Samoylenko, and Henning Erfkampand Renate Laub [2005]. *SITUATION PFLEGENDER ANGEHÖRIGER. Österreichisches Bundesinstitut für Gesundheitswesen (ÖBIG)*.

Prasanna, G. and N. Ramadass [2014]. *Low Cost Home Automation Using Offline Speech Recognition. International Journal of Signal Processing Systems*, 2(2), pages 96–101. ISSN 2315-4535. doi:10.12720/ijsps.2.2.96-101.

Ramlee, R.A, M.A Othman, M.H. Leong, M.M. Ismail, and S.S.S. Ranjit [2013]. *Smart home system using android application*. In *2013 International Conference of Information and Communication Technology (ICoICT)*, pages 277–280. IEEE, New York, NY, USA. ISBN 978-1-4673-4992-5. doi:10.1109/ICoICT.2013.6574587.

RelaxedCare [2014]. *RelaxedCare, connecting people in care situations*. `http://www.relaxedcare.eu`. Last access 11/2014.

Rest Assured® [2014]. *Rest Assured®, Caring People Remarkable Technology*. http://www.restassuredsystem.com/. Last access 09/2014.

Röcker, C, M Ziefle, and A Holzinger [2011]. *Social inclusion in AAL environments: Home automation and convenience services for elderly users*. In *Proceedings of the International Conference on ...*, pages 55 – 59. CSERA Press, New York, USA. ISBN 1601321856.

Roush, R. E. [2009]. *eNeighbor: a preventive system monitoring residents' behavior for health services*. *Gerontechnology*, 8(2). ISSN 15691101. doi:10.4017/gt.2009.08.02.013.00.

Santos-Perez, Marcos, Eva Gonzalez-Parada, and Jose Manuel Cano-Garcia [2013]. *ECA-based control interface on Android for home automation system*. In *2013 IEEE International Conference on Consumer Electronics (ICCE)*, pages 70–71. IEEE, Las Vegas, NV. ISBN 978-1-4673-1363-6. ISSN 2158-3994. doi:10.1109/ICCE.2013.6486799.

Saracchini, Rafael F. V. and Carlos C. Ortega [2014]. *An Easy to Use Mobile Augmented Reality Platform for Assisted Living Using Pico-projectors*. In Chmielewski, Leszek J., Ryszard Kozera, Bok-Suk Shin, and Konrad Wojciechowski (Editors), *Computer Vision and Graphics*, *Lecture Notes in Computer Science*, volume 8671, pages 552–561. Springer International Publishing, Cham, Switzerland. ISBN 978-3-319-11330-2, 978-3-319-11331-9. doi:10.1007/978-3-319-11331-9_66.

SimplyHome [2014]. *Innovative Solutions for Independent Living*. http://www.simply-home.com/. Last access 09/2014.

Sonamba [2014]. *Medical Alert System*. http://sonamba.com/. Last access 09/2014.

Statistica [2014]. *Smartphone OS global market share by quarter 2011-2014*. http://www.statista.com/statistics/236035/market-share-of-global-smartphone-os-shipments-by-mobile-operating-system-per-quarter/. Last access 11/2014.

The Apache Foundation [2014]. *Apache Karaf*. http://karaf.apache.org. Last access 11/2014.

Turatti, Maurizio and Maurizio Pillitu [2013]. *Instant Apache Maven Starter*. Packt Publishing Ltd, Birmingham, UK. ISBN 9781782167600, 62 pages.

Yi, Wu, Wu Tong, and Liu Pai [2013]. *Smart home system based on ZigBee and ARM*. In *2013 IEEE 11th International Conference on Electronic Measurement & Instruments*, volume 2, pages 754–759. IEEE, New York, NY, USA. ISBN 978-1-4799-0759-5. doi:10.1109/ICEMI.2013.6743209.

Zhu, Annan, Peijie Lin, and Shuying Cheng [2012]. *Design and Realization of Home Appliances Control System Based on the Android Smartphone*. In *2012 International Conference on Control Engineering and Communication Technology*, pages 56–59. IEEE, New York, NY, USA. ISBN 978-1-4673-4499-9. doi:10.1109/ICCECT.2012.36.

Ziefle, Martina, Carsten Rocker, and Andreas Holzinger [2011]. *Medical Technology in Smart Homes: Exploring the User's Perspective on Privacy, Intimacy and Trust*. In *2011 IEEE 35th Annual Computer Software and Applications Conference Workshops*, pages 410–415. IEEE, New York, NY, USA. ISBN 978-1-4577-0980-7. doi:10.1109/COMPSACW.2011.75.