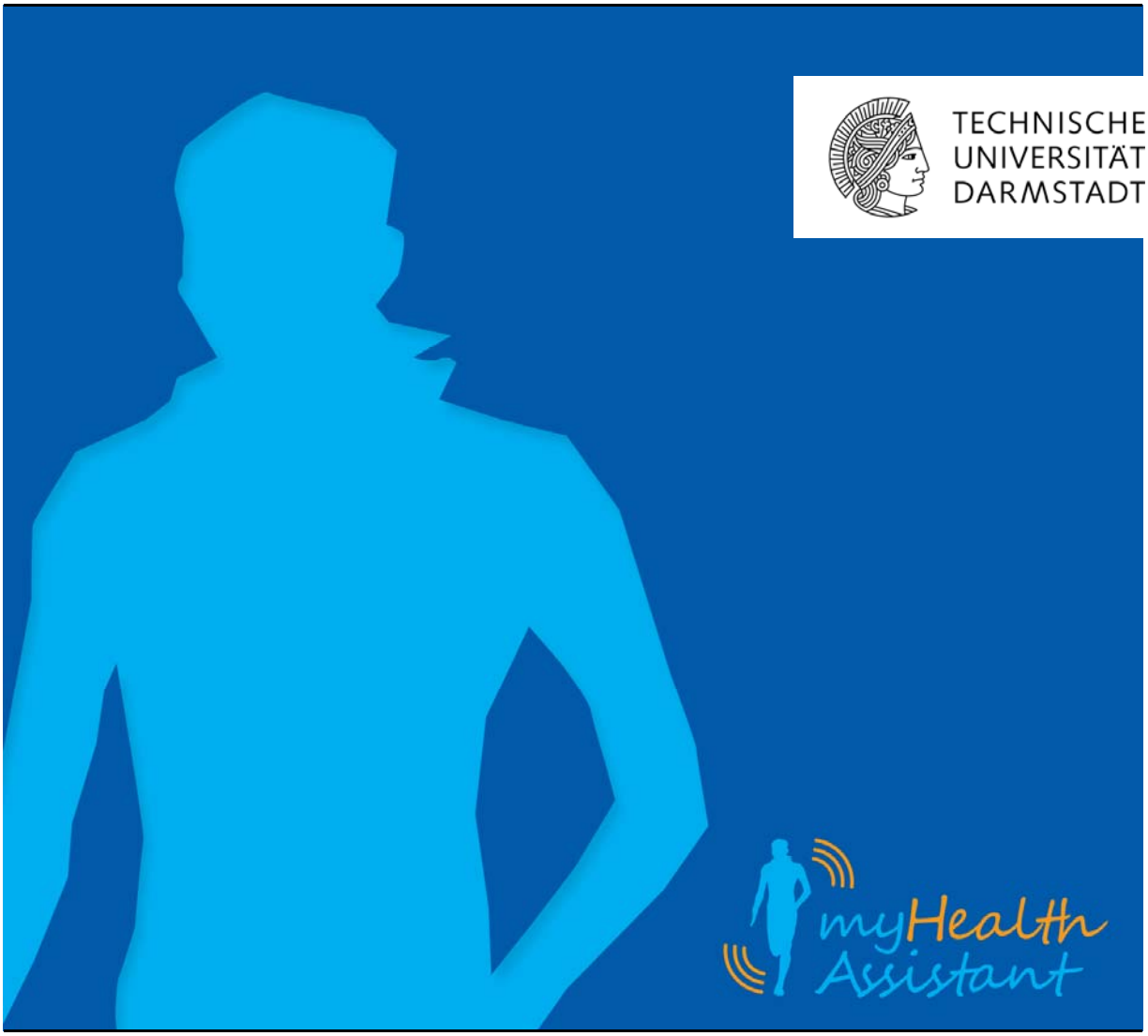
Event-driven Middleware for Body and Ambient Sensor Applications

Ereignisgesteuerte Middleware für körpernahe und umgebende Sensoranwendungen
Vom Fachbereich Informatik zur Erlangung des akademischen Grades Doktor-Ingenieur (Dr.-Ing.) genehmigte Dissertation von M.Sc. Christian Seeger aus Bad Nauheim
Darmstadt, 2014 – D17



1. Gutachten: Professor Alejandro Buchmann
2. Gutachten: Professor Danny Hughes
3. Gutachten: Dr. Kristof Van Laerhoven

Tag der Einreichung: 06.11.2013
Tag der Prüfung: 21.01.2014

Curriculum Vitae

Christian Seeger

Date of birth: September 19th, 1982

Place of birth: Bad Nauheim, Germany

Education	2010 – 2014	TU Darmstadt, Germany *PhD student in computer science*
	2012	ETH Zurich, Switzerland *Visiting student*
	2009	McGill University, Montreal, Canada *Visiting student*
	2007 – 2009	TU Darmstadt, Germany *Master of science in computer science*
	2003 – 2007	TU Darmstadt, Germany *Bachelor of science in computer science*
	1993 – 2002	Sankt Lioba Gymnasium, Bad Nauheim, Germany

Positions	2010 – 2013	TU Darmstadt, Germany Databases and Distributed Systems group of Prof. Alejandro Buchmann, PhD *Research and teaching assistant*
	2010 – 2012	TU Darmstadt, Germany DFG Research Training Group GRK 1362 *Scholarship holder*
	2010 – 2012	TU Darmstadt, Germany Simulation, Systems Optimization and Robotics group by Prof. Dr. Oskar von Stryk *Student assistant*

Abstract

Continuing development of on-body and ambient sensors has led to a vast increase in sensor-based assistance and monitoring solutions. A growing range of modular sensors, and the necessity of running multiple applications on the sensor information, has led to an equally extensive increase in efforts for system development. In this work, we present an event-driven middleware for on-body and ambient sensor networks allowing multiple applications to define information types of their interest in a publish/subscribe manner. Incoming sensor data is hereby transformed into the required data representation which lifts the burden of adapting the application with respect to the connected sensors off the developer's shoulders. Furthermore, an unsupervised on-the-fly reloading of transformation rules from a remote server allows the system's adaptation to future applications and sensors at run-time as well as reducing the number of connected sensors. Open communication channels distribute sensor information to all interested applications. In addition to that, application-specific event channels are introduced that provide tailor-made information retrieval as well as control over the dissemination of critical information.

The system is evaluated based on an Android implementation with transformation rules implemented as OSGi bundles that are retrieved from a remote web server. Evaluation shows a low impact of running the middleware and the transformation rules on a phone and highlights the reduced energy consumption by having fewer sensors serving multiple applications. It also points out the behavior and limits of the open and application-specific event channels with respect to CPU utilization, delivery ratio, and memory usage.

In addition to the middleware approach, four (preventive) health care applications are presented. They take advantage of the mediation between sensors and applications and highlight the system's capabilities. By connecting body sensors for monitoring physical and physiological parameters as well as ambient sensors for retrieving information about user presence and interactions with the environment, full-fledged health monitoring examples for monitoring a user throughout the day are presented. Vital parameters are gathered from commercially available biosensors and the mediator device running both the middleware and the application is an off-the-shelf smart phone. For gaining information about a user's physical activity, custom-built body and ambient sensors are presented and deployed.

Zusammenfassung

Das rasche Voranschreiten der Sensorentwicklung, ihre zunehmende Modularität sowie der resultierende Zuwachs an Anwendungen, haben zu neuen Herausforderungen im Bereich der Sensor-basierten Systeme geführt. Im Rahmen der vorliegenden Arbeit wurde eine Event-basierte Middleware für Sensornetze bestehend aus Körper- und Umgebungssensoren entwickelt, die Anwendungen die von ihnen angeforderten Informationen mittels eines Publish/Subscribe-Ansatzes zur Verfügung stellt. Durch die Abstraktion von der eigentlichen Sensorik, wird die Anwendungsentwicklung erheblich erleichtert. Außerdem ermöglichen sogenannte Transformationsregeln das Umwandeln von Sensordaten in eine für die Anwendung adäquate Darstellung. Diese Transformationsregeln können zur Laufzeit von einer entfernten Datenbank geladen und installiert werden und ermöglichen somit das automatische Anpassen des Systems an zukünftige Anwendungen und Sensoren und zugleich die Einsparung von Sensorik. Diese Middleware-basierte Umwandlung von Sensorinformationen ermöglicht somit den Verzicht auf die aufwendige Entwicklung anwendungs- beziehungsweise sensorspezifischer Systeme. Die Verteilung der Sensordaten erfolgt entweder über offene oder über anwendungsspezifische Kommunikationskanäle. Letztere ermöglichen ein maßgeschneidertes Ausliefern von Informationen sowie die Kontrolle über die Informationsverarbeitung.

Das System wurde als Android Anwendung implementiert und evaluiert. Die Evaluation zeigt den geringen Einfluss der Middleware und der Transformationsregeln auf die Systemauslastung sowie den durch die Einsparung von Sensorik und dem gleichzeitigen Bedienen mehrere Anwendungen reduzierten Energieverbrauch. Des Weiteren werden das Verhalten und die Grenzen der offenen sowie der anwendungsspezifischen Kommunikationskanäle bezüglich der CPU-Auslastung, der Delivery Ratio und des Speicherverbrauchs analysiert.

Zusätzlich zu dem Middleware-Ansatz werden vier Anwendungen aus dem Bereich (präventiver) Gesundheitsversorgung vorgestellt. Sie machen von den Vorzügen der Middleware Gebrauch und heben die Leistung des Gesamtsystems hervor. Körpernahe und umgebende Sensorik ermöglicht das Ermitteln von physikalischen und physiologischen Parametern und liefert Informationen über die Anwesenheit und Interaktion des Nutzers mit seiner Umgebung. Die vorgestellten Beispielanwendungen erlauben das permanente Monitoring einer Person. Dabei wurden für die medizinische Überwachung kommerziell erhältliche Biosensoren verwendet; als Plattform für die Middleware dient ein herkömmliches Smartphone. Die körpernahe und umgebende Sensorik zur Ermittlung der Benutzeraktivität wurde speziell für unsere Anwendungen entwickelt.

Acknowledgments

First and foremost, I would like to express my special appreciation and thanks to my advisors Alejandro Buchmann and Kristof Van Laerhoven, you have been tremendous mentors for me. Thank you for encouraging and supporting my research and for allowing me to grow as a person as well as a scientist. Without your guidance, persistent help, and motivation this dissertation would not have been possible.

I would also like to thank my committee members, Danny Hughes, Silvia Santini, Matthias Hollick, and Stefan Roth, for letting my defense be an enjoyable moment, and for your brilliant comments and suggestions.

I am indebted to my many colleagues in the DVS and ESS groups who supported me with valuable help and fruitful comments and suggestions. I especially like to thank Daniel Jacobi for his general help at the beginning of my work, Pablo Guerrero for providing me deep insights in the field of sensor networks, Stefan Appel, Tobias Freudenreich, and Sebastian Frischbier for their brilliant help in the area of event-based systems. Furthermore, I like to thank Eugen Berlin and Marko Borazio for their excellent help especially regarding application development. Many things such as the research stays and conference visits would not have been possible without help from Marion Braun, Gabriele Ploch, Ursula Paeckel, and Maria Tiedemann. Thank you.

I consider it an honor that parts of this dissertation arose from a collaboration with Friedemann Mattern's Distributed Systems Group at ETH Zurich. Special thanks to Friedemann Mattern and Gábor Sörös for their hospitality. Moreover, thank you to Benedikt Ostermaier and Simon Mayer for your brilliant ideas and support with ambient sensors.

I am also honored that a second collaboration contributed to this project - thanks to the Telemedical Systems Department of Robert Bosch GmbH, especially Thorsten Sohnke who was my specialist adviser at Bosch, for their support.

Thank you to all the undergraduate students who I had the chance to work with and who contributed to the project: Chi Hieu Ha, Jennifer Conrad, Jens Sauer, Moritz Jansson, Nezir Önkol, Dassi Ponka, Sebastian Niederhöfer, and Thomas Pignede.

A special thanks to my parents. Words cannot express how grateful I am. Without their support, teaching, and unshakeable faith in me, all this would not have been possible to me. I hold my mother in high esteem for teaching me in staying power and my father for his early support in computer science and teaching me in conscientious working. Properties that laid the foundations of working as a scientist.

At long last, I like to express the deepest gratitude to my fiancée. Fabienne supported me in so many different ways: as a doctor she provided the medical background to my work, she always gave me tremendous feedback on non-technical questions, and she motivated me when I was in despair. For what I like to thank her most is her incessant faith in me.

Contents

1 Introduction **1**
- 1.1 Medical Sensor Network Applications . . . 3
  - 1.1.1 Preventive Care . . . 3
  - 1.1.2 Out-patient Care . . . 4
  - 1.1.3 Elderly Care . . . 6
  - 1.1.4 Motivational Example . . . 8
- 1.2 Middleware for Medical Sensor Network Applications . . . 10
- 1.3 Summary of Contributions . . . 12
- 1.4 Outline . . . 14

2 Background **15**
- 2.1 Body and Ambient Sensor Technology . . . 15
  - 2.1.1 Sensor Types . . . 15
  - 2.1.2 Networking Technologies . . . 18
  - 2.1.3 Mediator Devices . . . 20
- 2.2 Related Middleware for Sensor Networks . . . 21
  - 2.2.1 Distributed Middleware . . . 21
  - 2.2.2 Mediator-only Middleware . . . 24
  - 2.2.3 Comparison to this Work . . . 26
- 2.3 Android OS . . . 27
  - 2.3.1 Intents . . . 28
  - 2.3.2 Components . . . 30
  - 2.3.3 Secure Inter-Application Communication . . . 32

3 Event-driven Middleware Architecture **37**
- 3.1 Requirements . . . 37
  - 3.1.1 Multi-Setup Requirements . . . 38
  - 3.1.2 Networking Requirements . . . 39
  - 3.1.3 Mediator Requirements . . . 40
- 3.2 Generic Architecture . . . 41
  - 3.2.1 Message Handler . . . 43
  - 3.2.2 Sensor Module Manager . . . 44
  - 3.2.3 Security Manager . . . 44
  - 3.2.4 System Monitor . . . 44
  - 3.2.5 User Repository . . . 45
  - 3.2.6 Event Composer . . . 45
  - 3.2.7 Transformation Manager . . . 45
- 3.3 Open Communication Channels . . . 45
  - 3.3.1 Subscription . . . 46
  - 3.3.2 Notification . . . 47
- 3.4 Application-specific Communication Channels . . . 47
  - 3.4.1 Advertisement . . . 48
  - 3.4.2 Subscription . . . 49

3.4.3 Notification . . . 49
3.5 Adaptive Event Transformation . . . 49
3.5.1 Transformation Request . . . 51
3.5.2 Transformation Life-cycle . . . 51
3.5.3 Remote Repository . . . 53
3.6 Summary . . . 54

4 Middleware Implementation 57
4.1 Overview . . . 57
4.1.1 Message Handler . . . 60
4.1.2 Sensor Module Manager . . . 60
4.1.3 Security Manager . . . 63
4.1.4 System Monitor . . . 63
4.1.5 User Repository . . . 64
4.1.6 Event Composer . . . 64
4.1.7 Transformation Manager . . . 65
4.2 Open Communication Channels . . . 65
4.2.1 Subscription . . . 66
4.2.2 Notification . . . 67
4.3 Application-specific Communication Channels . . . 67
4.3.1 Advertisement . . . 69
4.3.2 Subscription . . . 70
4.3.3 Notification . . . 70
4.4 Adaptive Event Transformation . . . 71
4.4.1 Transformation Manager . . . 72
4.4.2 Event Transformations as OSGi Bundles . . . 73
4.4.3 Remote Repository . . . 74
4.5 Integration in Existing Environments . . . 74
4.5.1 RedPin . . . 74
4.5.2 InfraWOT . . . 75
4.6 Summary . . . 76

5 Applications using the Middleware 77
5.1 Fitness Diary . . . 78
5.1.1 System Overview . . . 79
5.1.2 Daily Activities . . . 81
5.1.3 Gym Exercises . . . 83
5.1.4 Evaluation . . . 88
5.2 Monitoring of Rehabilitation Patients . . . 89
5.2.1 Interval Training . . . 91
5.2.2 Training Instruction Module . . . 92
5.2.3 Connection to a Telemedical Platform . . . 93
5.2.4 Evaluation . . . 93
5.3 Fitness Trail . . . 95
5.3.1 Exercise Recognition . . . 97
5.3.2 Exercise Counting . . . 98
5.3.3 User Motivation Module . . . 98
5.4 Monitoring Activities of Daily Living . . . 99
5.4.1 System Setup . . . 99
5.4.2 Activity Monitoring . . . 103

5.4.3 System Evaluation 104

6 System Evaluation **109**
6.1 Energy Consumption 110
6.2 Messaging Performance 112
6.2.1 Changing Event Producers and Message Sizes 113
6.2.2 Changing Number of Event Producers and Subscribers 114
6.2.3 Impact of the Hardware Performance 116
6.3 Event Transformation 118
6.4 User Experience 119
6.5 Summary 121

7 Conclusions and Outlook **123**
7.1 Outlook 124
7.2 Epilogue 126

A. For Developers to Use the Middleware **127**

Index **131**

List of Abbreviations **133**

Bibliography **135**

1 Introduction

An ageing population and low birth rates are leading to a demographic change in the West that significantly challenges its health care systems. The *Demography Report 2010* [44] from the European Commission states that change in age structure of the EU-27 countries is of more concern than the change in population size. The report forecasts an increasing share of population that is aged 65 or older from 17.4% in 2010 to 30.0% in 2060. Furthermore, the proportion of people aged 80 or older is expected to almost triple by 2060. The total age dependency ratio[1] is projected to rise from 63.2% in 2010 to 95.5% in 2060. This implies that for almost every working person there is one child or elderly person to take care of. Also the U.S. Congressional Research Service reports a profound demographic change with an increasing proportion of persons aged 65 and older for the United States of America [131].

In addition to the demographic change, the World Health Organization (WHO) predicts that chronic diseases will become the most expensive problem faced by current health care systems [157, 158]. A WHO fact file [160] summarizes that chronic diseases are responsible for 60% of all deaths worldwide. Unhealthy diet, physical inactivity and tobacco use form the major risk factors for chronic diseases. In 2005, one billion adults were overweight. If no action is taken, the WHO predicts 1.5 billion over-weighted adults in 2015. By eliminating the major risk factors for chronic disease, at least 80% of heart disease, strokes and type 2 diabetes would be prevented.

The Federal Government of Germany moved the demographic change to the top of its political agenda [22]. In 2011, the German Federal Ministry of Education and Research (BMBF) published a research agenda for demographic change: *The New Future of Old Age* [21]. In this agenda, maintaining independent and self-determined living of elderly in their own apartments is highlighted as an important step in order to face the demographic change. The elderly require a safe and stimulating residential environment that allows them to look after their affairs for as long as possible without the practical support of others. For this, a sensor-controlled home emergency call and living support system is required. By combining sensors and intelligent data analysis, the inhabitant's state of health is monitored, everyday activities are assisted, and a greater quality of life is achieved. In addition, mobile diagnostics and telemedical support systems help the fight against chronic diseases and support rehabilitation at home. The support of simplified care measures, automatic emergency detection, and improved care documentation, allow an increasing proportion of out-patient care over in-patient care and therefore a reduction of health care costs. Furthermore, the WHO sees the integration of prevention into health care as the main solution in order to fight chronic diseases [157]. A paradigm shift towards integrated, preventive health care as well as equipping patients with information, motivation, and skills in prevention and self-management are described as essential elements for solving this problem. Studies have shown

[1] The age dependency ratio is calculated as the ratio of children and young people aged under 19 and people aged 65 or older to the population aged 20-64.

that prevention and health promotion boost self-responsibility and helps maintaining the physical and cognitive ability for people at any age [21].

The previous visions for facing chronic diseases and the demographic change rely on systems allowing for the gathering and interpretation of information about a patient outside the hospital. Solutions for collecting physiological and physical parameters as well as information about a person's environment and interactions with the environment are important for future health care applications. As body sensor network (BSN) systems are capable of continuously monitoring a person's physiological and physical state, they already form a promising tool. Combined with the additional information from ambient sensory devices, full-fledged health monitoring solutions can be built and the required context information on patients can be provided. In [20], the Federal Ministry of Education and Research present the *Information and Communication Technologies (ICT) Agenda 2020* setting 24/7 telemonitoring solutions and the development of flexible software platforms as the basis for new services in the health sector. The Continua Health Alliance[2] is a non-profit, open industry organization of health care and technology companies. Their vision [33] is in unison with the requirements demanded by the World Health Organization, the Federal Ministry of Education and Research, as well as the vision of our work:

- **Empower individuals and patients to better manage their health** by providing them with information regarding their fitness and health through personal medical devices and services.
- Allow loved ones and professional care givers to more accurately monitor and **coach chronic disease patients and elderly individuals living independently**.
- Enable medical and fitness device manufacturers to **rapidly develop interoperable devices and services**... .
- Enable health care providers, to offer better quality care through **personalized health solutions assembled from** [...] **interoperable health care devices and services**.

In order to empower people to better manage their health, diseases and independent living, we need personalized health solutions which can be swiftly assembled by combining the best fitting software and hardware solutions. In order to provide thus interoperability, we propose a middleware that seamlessly handles sensor communication and provides tailor-made sensor information to applications. We believe that this is an important step towards new health care solutions that cope with both demographic change as well as chronic diseases.

For the remainder of this chapter, we present medical sensor network applications in order to give an idea of the target applications of our middleware. In addition, a motivational example application is presented and used throughout this thesis. Based on this example, the general requirements for a middleware for future medical sensor network applications are depicted. Before we present the outline of this thesis, a summary of contributions is given.

[2] http://www.continuaalliance.org/

1.1 Medical Sensor Network Applications

This section discusses a number of existing medical sensor network applications in order to help in understanding the context and requirements on a middleware supporting those applications. In addition, we will present an example of a future application for out-patient care which tackles all demands made by the presented applications as well as the requirements demanded by the BMBF, Continua Health Alliance and the WHO among others. This motivational example is used throughout this thesis.

There are several surveys [2, 109, 111, 75] on wireless sensor networks (WSN) for health care applications targeting different aspects such as sensor network challenges and hardware design. The survey from H. Alemdar and C. Ersoy [2] mainly focuses on the specific applications and serves as a basis for this section. It deals with 31 pervasive health care monitoring applications classified into five categories: (1) activities of daily living (ADL) monitoring, (2) fall and movement detection, (3) location tracking, (4) medication intake monitoring, (5) medical status monitoring. Since most future health care solutions consist of a combination of those categories, we will distinguish between *preventive care* (1,5), *out-patient care* (1,2,3,5), and *elderly care* (1-5).

1.1.1 Preventive Care

Preventive care or preventive medicine focuses on prevention of disease and the promotion and preservation of health in the individual [99]. The World Health Organization states that "four of the most prominent chronic diseases - cardiovascular diseases (CVD), cancer, chronic obstructive pulmonary disease and type 2 diabetes - are linked by common and preventable biological risk factors, notably high blood pressure, high blood cholesterol and being overweight, and by related major behavioral risk factors: unhealthy diet, physical inactivity and tobacco use" [157]. With respect to those risk factors, several (body) sensor network projects promoting healthy diet, physical activity and the avoidance of tobacco use are discussed in this section.

Healthy Diet

Unhealthy diet elevates health risks for many chronic diseases such as diabetes mellitus, cardiovascular diseases, and obesity. Automated dietary behavior monitoring helps in identifying an imbalanced diet and promoting a healthier one. Oliver Amft et al. developed a system consisting of an ear microphone, a sensor collar (including EMG and microphone) and four upper body inertial sensors that allows monitoring dietary activities [5, 4]. The system targets recognizing of dietry intake movements, chewing and swallowing, which helps in avoiding risk factors such as dehydration, eating too fast or stress eating. In [120, 121], a similar project which focuses only on chewing and swallowing is presented.

Physical Activity

For promoting a physically active lifestyle, there are already several commercial products available starting from simple pedometers like the Omron HJ-112 [102] up to full-fledged coaching

systems such as Polar Smart Coaching [115]. Runkeeper [46] from FitnessKeeper is a smart phone application that allows users to track their fitness activities and motivates them to performing better and more frequent. It facilitates various sensors (GPS, scale, cadence, blood pressure, pedometer, heart rate) most of them wirelessly connected to a smart phone. In contrast to other systems, Runkeeper provides an open API called HealthGraph [45] to which all sensor readings are transmitted. The HealthGraph stores information about the user's nutrition, body measurements, sleep, everyday activity, and workouts. Based on this information various interesting preventive care applications are possible. Another system that performs workout tracking is Runtastic [119]. Both systems focus on endurance training and require the user to inform the system about the activity performed. The work presented in [54] is an example for a system that automatically identifies the activity a user performs. By using three inertial sensors connected to a phone, it recognizes resting, typing, gesticulating, walking, running, and cycling and does not require the user to inform the system about those activities. Strength training is another interesting application area: NorthPark [52], for instance, provides several very simple smart phone Apps for counting sit ups, squats, and push-ups among others. More comprehensive systems are presented in [27, 28]. The Moove applications [147] from VitaLiberty target corporate health management. In addition to applications for sports activity tracking and weight monitoring, they provide an application that derives the user's stress level based on the heart rate variability (HRV).

Avoiding/Reducing Tobacco Use

A system that tracks smoking episodes in an unobtrusive way is presented in [122, 123]. The gathered information can be used in order to visualize smoking behavior and to convince a smoker to reduce or stop tobacco consumption. Switches integrated in two types of lighters allow the detection of when and how often someone is smoking. The integration of wireless transmission is ongoing work.

Table 1.1.2 summarizes the projects presented in this and the next section with respect to the sensors and communication technologies used as well as the corresponding target application. In Section 5, we will introduce our own preventive health care applications: a fitness diary application that captures a person's daily as well as very specific exercises throughout a day and a fitness trail application that specifically monitors and motivates fitness trail exercises.

1.1.2 Out-patient Care

Out-patient care applies to different application areas. A very important area is the monitoring of patients with chronic diseases. Usually, those patients suffer from their disease for the rest of their life and need to be monitored. Reasons for the monitoring are, among others, the detection of a progressing disease and the adjustment of medication intake. Rehabilitation monitoring and assistance after an incident (e.g. heart attack, stroke) or surgery is another application area. In addition, short-term monitoring of physiological parameters could be applied if a physician needs additional information after a doctor's visit. In sensor network based out-patient care, patients benefit from the independence from stationary in-hospital observations, allowing to freely

move and live their daily life while being monitoring over longer times and under more realistic conditions. In the following, we will summarize solutions for chronic diseases (cardiovascular and neurological) and rehabilitation. Solutions for short-term monitoring that monitor for instance blood pressure and ECG streams are implicitly included in chronic disease applications.

Cardiovascular Diseases

Cardiovascular diseases (CVDs) are the number one cause of death globally [159]. Since more people die from CVDs than from any other cause, there are several projects dealing with this disease and cardiovascular monitoring in general. Some examples [104, 87, 107, 41, 80, 108, 166, 84, 71, 6, 130] are listed in Table 1.1.2. In the HeartToGo project [104] for example, a system for cardiovascular disease prevention and detection is presented. The system consists of an off-the-shelf Bluetooth 2-lead electrocardiogram (ECG) sensor including a 3-axis accelerometer, a Bluetooth GPS sensor, and a smart phone. The sensor readings are transmitted to the smart phone which performs continuous monitoring and recording of the ECG stream in real-time. Upon the detection of an abnormal CVD condition, the system automatically calls for emergency assistance including the patient's position. Machine learning techniques allow adaptation to the patient's physiological conditions and to achieve more accurate CVD classification results. The HeartToGo system requires the patient to wear a dedicated ECG sensor. In order to overcome this problem and the stigma of explicitly wearing sensors, several projects [87, 107, 41, 108] worked on including the sensors in clothes. For instance, the researchers from the myHeart project [87, 107] managed to include ECG and respiration sensors as well as simple activity recognition (walking, stairs, running) in an instrumented shirt. Sensor readings are transmitted to a smart phone using Bluetooth. The LifeGuard [93] is a system that not only focuses on patient monitoring, but also on monitoring people working under harsh conditions such as astronauts and first responders is called. It collects data about the blood oxygen saturation (SpO_2), ECG, blood pressure (BP), respiration, activity, and skin temperature and transmits it to a Bluetooth-enabled tablet PC.

Neurological Diseases

The AWARENESS [67] project proposes a system for monitoring neurological diseases such as chronic pain, motor disorder, and epilepsy. The example application discussed in that paper is designed to detect and react to epilepsy seizures captured by an ECG sensor. By taking the user's activity level into account, the seizure detection is optimized. In case of a seizure, an alarm including the patient's position is sent. The LiveNet [136] system provides several monitoring applications. Within this context, the Parkinson's disease monitoring and the epilepsy seizure detection are of special interest. For monitoring Parkinson's patients, wearable accelerometers are used to predict "when the patient feels off or is about to experience troublesome dyskinesia (slow motion)" [136]. For epilepsy seizure detection, accelerometers are used to detect and log seizures in order to provide physicians information about when and how often they occur. Another project dealing with accelerometer-based epileptic seizure detection as described in [38, 110].

Rehabilitation

In the area of rehabilitation, the HipGuard [64] is a posture detection system for patients recovering from a hip operation. Inertial sensors, forming an ANT[3]-based sensor network, are integrated in close-fitting pants or straps placed around the waist and leg and in shoe inlays. If the operated hip's position or its load approaches set limits, the patient is warned and the measured data is transmitted to a smart phone. Further projects are presented in [50, 49, 68, 3, 9] and listed in Table 1.1.2. The authors of [55] give an overview of wireless sensor networks for rehabilitation applications. In Section 5.2, we will present an out-patient care example application for rehabilitation patients that is built upon our middleware.

1.1.3 Elderly Care

According to a survey performed in the USA, 92% of the people aged between 65 and 74 and 95% of the people aged 74 and older prefer living in their own apartment [23]. The costs for nursing a patient with Alzheimer's disease in a special facility amount to 64,000 US$ per year. In contrast, nursing the patient in his or her own apartment costs only 20,000 US$ per year [23]. Enabling senior citizens to live as long as possible in their homes does not only correspond to their wishes but also saves money. It is an important step towards managing the demographic change. Elderly care tackles this problem by assisting and monitoring people in their apartments. It enables them to live safely and independently. On the other hand, every person has his or her special needs and requirements which makes the implementation of elderly care solutions a challenging task. This section discusses corresponding aspects addressed in the research community.

In elderly care, not only monitoring of physiological and physical parameters as described in the previous section is important but also the assistance presented to older people. Therefore, it consists of a combination of out-patient care, Ambient Assisted Living (AAL) technologies, and safety solutions such as fall detection. Since falls are dangerous especially for elderly, many projects focus on fall detection. It ranges from rather simple phone-based approaches using built-in accelerometers [37] up to grammar-based approaches detecting different types of falls [83]. In [57, 97], surveys on fall detection are presented.

In order to provide helpful and meaningful assistance, knowledge about a person's context is inevitably required. Many AAL solutions consist of instrumented environments equipped with sensors allowing among other things the detection of the usage of objects, medication intake, presence in a specific room and active assistance based on certain actuators such as light, heating or robots [134]. Determining the location of a person is an important indicator for a person's context and daily habits. For outdoor tracking, GPS is the most robust and widely available technique, but it does not properly work indoors [2]. Many indoor systems rely on motion and pressure sensors (e.g. [85]) or on radio-based triangulation. The authors of [88] propose a ZigBee-based approach called ZUPS. Bolliger from the Distributed Systems Group at ETH Zurich developed a Wi-Fi-based approach called RedPin [13]. We integrated this solution into our system (cp. Section 4.5.1).

[3] ANT (ANT+) is a proprietary wireless sensor network technology operating in the 2.4 GHz industrial, scientific, and medical allocation of the RF spectrum.

Project	Sensors*	Wireless Communication	Application
	Preventive Care		
UbiLighter [123, 122]	switch	wired (LED & WebCam)	tobacco use monitoring
Chewing Sazonov [120, 121]	microphone, strain sensor	wired	healthy diet
Chewing Amft [5, 4]	microphone, EMG, acc, gyro, magn	wired	healthy diet
Runkeeper [46]	HR, scale, BP, step counter, GPS	Bluetooth, 3G	sports
Runtastic [119]	HR, GPS, cadence	Bluetooth, 3G	sports
Motion Band Activities [54]	acc, gyro, magnc	Bluetooth	daily activities
Nike+ Fuelband [100]	acc, ambient light	Bluetooth	sports
Moove [147]	scale, GPS, ECG	wired (ANT, Bluetooth)	health management
	Out-patient Care		
AWARENESS [67]	ECG, EMG	Bluetooth	epilepsy, chronic pain, motor disorder
CodeBlue [130]	acc, ECG, SpO_2	ZigBee	physiological status monitoring
HeartToGo [104]	ECG, acc, GPS	Bluetooth, 3G	CVD monitoring
MyHeart Cardiac [87, 107]	ECG, acc, resp	Bluetooth	CVD monitoring
MagIC [41]	acc, ECG, resp, temp	Bluetooth	ECG, Respiration
Personal Health Monitor [80]	acc, ECG, SpO_2, BP, GPS, scale	Bluetooth	CVD monitoring
Smart Vest [108]	ECG, SpO_2, BP, GSR, HR	2.4 GHz ISM	general health monitoring
Pervasive Medical Supervision [166]	BP, SpO_2, HR	Wi-Fi, 3G	general health monitoring
PATHS [84]	ECG, acc, GPS	Wi-Fi, Bluetooth	cardiac monitoring
Mobile ECG [71]	ECG	Bluetooth, 3G	cardiac monitoring
AMON [6]	acc, temp, ECG, BP, SpO_2	3G	high-risk cardiac-respiratory patient monitoring
LifeGuard [93]	acc, ECG, temp, resp, SpO_2	Bluetooth	general health monitoring
LiveNet [136]	acc, HR	2.4 GHz, 3G	Parkinson's & epilepsy & rehabilitation
HipGuard [64]	acc, gyro	ANT, Bluetooth, mobile network	rehabilitation from hip operation
Mercury [38, 110]	acc, EMG	Bluetooth	epileptic seizures detection
MyHeart Rehabilitation [50, 49]	strain sensor	Bluetooth	rehabilitation
ActiS [68]	acc, ECG, EMG	ZigBee	rehabilitation
HAMON [3]	acc, ECG	Bluetooth	rehabilitation
Earpiece Sensor [9]	acc, SpO_2, HR	ZigBee, Wi-Fi, 3G	rehabilitation

**acc: accelerometer, BP: blood pressure sensor, ECG: electrocardiogram, SpO_2: pulse oximetry, HR: heart rate, gyro: gyroscope, EMG: electromyography, GPS: global positioning system, GSR: galvanic skin response, temp: skin temperature, resp: respiration, magn: magnetometer*

Table 1.1: Summary of research projects and commercial products for preventive care and out-patient care applications using sensor network technology. An overview of the utilized sensors and communication technologies as well as the target application are given.

In order to provide help for a very specific task such as making tea, information about a person's current position might be not sufficient. Instead, we need information about explicit object usage. Many health care applications for elderly monitor specific Activities of Daily Living (ADL) in order to assist people and to detect deviations in their daily habits [146, 164]. In [85], a location-aware activity recognition system for AAL environments is proposed. It derives interleaved activities based on multimodal and unobtrusive wireless sensor nodes such as current sensors, pressure mats, motion and vibration sensors, RFID readers and tags, accelerometers and reed switches. Based on that system, the activity of making tea is recognized with an accuracy of 91%. The work presented in [114] goes one step further. It detects individual steps of cooking like slicing onions. For further examples of systems and algorithms targeting ADL recognitions we refer to [85, 15, 139, 59, 60, 142].

AlarmNet [155] is a well-known system for assisted living and residential monitoring that combines both a body sensor network based health monitoring as well as the detection of daily activities based on sensors integrated in the environment. It records physiological parameters such as blood oxygen saturation, ECG / heart rate and blood pressure and uses accelerometers for detection postures and falls. A circadian activity rhythms analysis monitors behavioral trends in home with 24-hour cycles and aids context-aware power management and dynamic privacy policies. The dynamic privacy policies, for instance, allow for alerting a nurse and granting access to critical data in case of an emergency even if the access is not granted under usual circumstances.

For further information we refer to a review of smart homes [26] from 2008 as well as a more recent list of smart home projects provided in [111].

1.1.4 Motivational Example

Figure 1.1 sketches a motivational monitoring and rehabilitation application that is used throughout this thesis, assuming a sixty-year-old male patient (140kg, 1.80m, smoker) who suffered a heart attack[4]. During the first two weeks after inpatient treatment in a hospital, the patient receives ambulatory **extensive post-attack monitoring**. In parallel, a tailored one-year-long **patient-specific risk factor monitoring** program starts and monitors the heart, the medication intake, as well as patient-specific risk factors. Our patient suffers from high cholesterol, hypertension and diabetes due to his overweight. Therefore, weight, blood pressure, blood glucose level, and the daily activity level are monitored. As a counter example, for a patient with healthy weight but a cardiac arrhythmia the monitoring would focus on a thorough ECG event detection instead of weight, blood pressure, and blood glucose level. In addition to the clinical monitoring, a tailor-made heart **rehabilitation & preservation program** starts which promotes physical activity in order to strengthen the heart, lose weight and minimize risk factors. Furthermore, the patient agrees to participate in a **clinical study**. The following describes the four individual applications in detail. Many of the described monitoring aspects are based on the advices from the Mayo Foundation for Medical Education and Research [91].

[4] A heart attack happens when the flow of oxygen-rich blood to a section of heart muscle suddenly becomes blocked and the heart is not supplied with oxygen anymore.

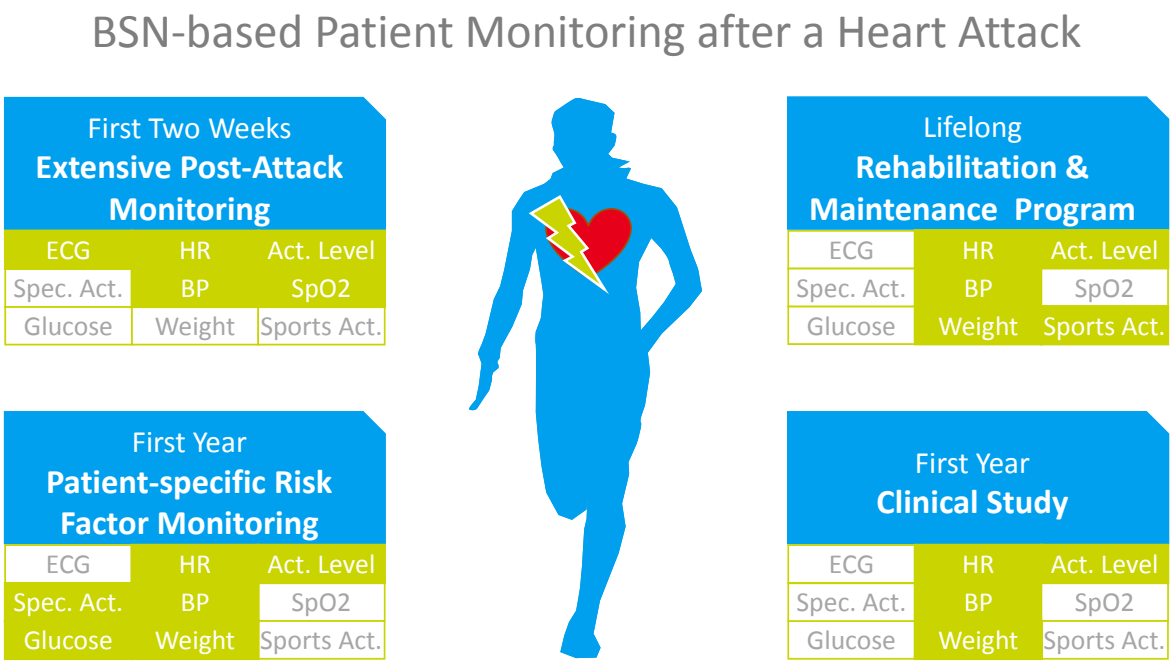


Figure 1.1: Patient monitoring after a heart attack. The patient's physical and physiological state as well as the rehabilitation progress are monitored by combining different monitoring modules. Parameters which are required for each application are highlighted in green.

Extensive Post-Attack Monitoring

During the first two weeks after discharge from hospital, the patient is still monitored accurately in order to quickly detect and react to complications or other incidents (e.g. by calling an ambulance). Therefore, an event recorder for ECG, RR-intervals, and heart pulse would monitor the heart very thoroughly. Furthermore, the combination of heart rate and activity level monitoring allows analyzing whether the heart adapts correctly to physical load. In addition to the heart, blood pressure and blood oxygen saturation are regularly monitored.

Patient-specific Risk Factor Monitoring

In order to avoid another heart attack or other heart diseases, patient-specific risk factors need to be lowered and monitored. Therefore, one-year-long monitoring of heart functionality is applied to every patient who suffered from a heart attack. This is done by using a heart rate sensor and sensors for measuring the patient's activity level (e.g. inertial sensors). With this setup the heart's behavior to physical load is monitored and critical heart rates are detected and alarmed. Obesity contributes to high cholesterol, high blood pressure and diabetes which in turn rises the risk of heart diseases. Since our patient is overweight, blood pressure, blood glucose level, and weight are monitored additionally. Certain activities such as smoking, coffee consume and physical activity influence the result of a blood pressure measurement. In order to help interpreting those readings, a history of recent activities is stored and presented to the medical staff. This way, a high blood pressure due to high physical activity shortly before a measurement can be detected and ignored.

Furthermore, as soon as the system detects that the patient is taking a measurement right after strenuous physical activity (e.g. climbing stairs), it can advise to re-perform the measurement after the activity's effect vanished (e.g. after sitting for 10 minutes). Thus, the system helps gathering measurements of high quality without the need of a physician's intervention.

Rehabilitation & Preservation Program

Rehabilitation programs are important for recovering and strengthening the heart after a heart attack. Cardiac exercises train the heart. Therefore, our patient is asked to perform light fitness exercises such as walking and cycling. The combination of a heart rate sensor and sports activity recognition supports the patient to train in the optimal heart rate zone and to avoid overloading. For recognizing specific sports activities, additional sensors are needed. Those sensors can be embedded in sports accessories. In addition to monitoring the training, the system guides the patient through the rehabilitation and its following preservation phase by giving advices, organizing the training load and motivating in performing better and losing weight. By connecting the system to a social media platform for heart attack patients, our patient exchanges information with other people suffering the same problem. By posting successfully completed trainings, the patients stay connected and motivated. In addition, the rehabilitation and preservation program promotes a healthy diet and is applied for the rest of the patient's life in order to minimize the risk of another heart attack.

Clinical Study

Continuous multimodal patient monitoring over a long period of time provides many advantages for clinical studies. By applying machine learning algorithms to data gathered from such studies, new causal relationships can be found and new treatments developed. Our patient decides to participate in such a clinical study by providing information about the combination of heart rate and activity level, weight, blood pressure and sports exercises. The anonymous information is regularly transmitted to a web server for the period of one year after the incident.

1.2 Middleware for Medical Sensor Network Applications

Future applications benefit from integrated and unobtrusive sensors that lower the burden from being monitored (cp. Section 2.1.1). As sensors become invisible to other people the stigma of being monitored vanishes and, thus, patients are more willing to have a more thorough monitoring of their health. However, not every patient requires the same set of sensors and monitored parameters. Furthermore, "care researchers are challenged to find answers for open questions about the consequences of an increasing (multi-) morbidity among care patients" [21]. A single application that monitors every disease is very unlikely. Instead, there will be a modular system with multiple very specific applications monitoring a certain disease or set of health parameters. The presented monitoring solutions tackled rather static scenarios in which only a specific disease or a small range of parameters are monitored. This is a very important step in order to gain

experiences in this new field of health monitoring. As the monitoring technology evolves, there will be the demand of having a more flexible system that allows for extensive, modular and user-specific health monitoring which, in turn, raises new requirements on the underlying or intermediate system. The motivational post heart attack example presented in the previous subsection aims to be such a future health monitoring solution. Based on this scenario we will develop a set of requirements and challenges on a mediator for future health care solutions.

Multiple Application Support

An increasing multi-morbidity among care patients [21] and need for tailor-made solutions raise the demand for a platform that relays sensor information to multiple applications. In our post heart attack scenario, we have multiple applications using the heart rate sensor which is due to the fact that the heart is a critical factor that needs to be strengthened and monitored. Many current sensors provide only exclusive access to the sensor data which, thus, requires to read the heart rate for each application individually. In order to be less obtrusive, most sensors transmit their data wireless which requires in our case the fourfold load on costly wireless transmissions. Thus, a system that collects sensor readings and relays them to the corresponding applications is required. Only when multiple applications are allowed to run on the same system and access the same sensor data, a combination of very specific applications and thus a tailor-made solution becomes possible.

Support of Different Requirements on Data Representations

Not every application has the same requirements on the data representation. Assuming that our heart attack patient is treated in Germany using a scale that measures kilogram and the clinical study is done in the USA using pounds, the weight values need to be transformed before being sent to the clinical study. Otherwise the patient would need to take two measurements. A second example can be observed even within a single application. The patient-specific risk factor monitoring for instance operates on two types of activity readings. Specific activities are used in order to get detailed information about the user's activities before a blood pressure measurement. Activity level information is used for enhancing the heart rate monitoring. We can assume that specific activities already include information about the activity level (e.g. running has a high activity level whereas sitting has a low activity level). Therefore, the activity level can easily be calculated and an additional sensor and costly sensor communication can be saved by transforming the data.

Application-specific Security Requirements

Applications have different confidentiality. For instance, the rehabilitation and maintenance application which is connected to a social platform should not gain access to sensitive information from the patient-specific risk factor monitoring. Mechanisms need to be provided that allow control over the data dissemination.

Changing Application and Sensor Setups

Diseases progress and applications as well as sensor technology evolve. This raises the need for a system that adapts to changing requirements. In order to encourage fast adaptations, a system is required that limits the expense for changing its configuration. Adding a new heart rate sensor, for instance, should not require modification to all applications working on heart rate information and adding, replacing or removing an application should not influence other unrelated applications.

Handling of Changing Sensor Constellations

Sensor constellations change at run-time. Some sensors such as a scale or a stationary blood pressure sensor are only connected or within communication range while the patient is in its proximity. Some sensors like the sensor for detecting sports activities are only worn for a specific purpose. They join the network while the patient performs sports exercises and then leave it again. These changing body and ambient sensor constellations need to be seamlessly handled in order to provide a maximum comfort to the patient and the application developers.

We argue for a middleware that mediates between sensors and applications in particular, because it decouples the applications from the underlying sensing and communication tasks. By having the middleware communicate with the sensors, it can relay the sensor information to multiple applications, transfer the information into the required data representation, and follow the security requirements. For adding, removing or replacing sensory devices, only the corresponding part of the middleware needs to be changed, but not the applications. Changes in the sensor constellations are handled by the middleware. This abstraction from the sensor network reduces the complexity for developers, which accelerates the application development and, thus, the deployment of the system. Furthermore, a modular middleware architecture increases the system's flexibility and, hence, adaptability to new sensors and circumstances.

The Continua Health Alliance which has similar targets as our work "focuses on the interoperability aspect and not prescribes specific functionality" [153]. They target to support interoperability between different sensors of the same type and the application hosting device by defining a common Personal Area Network (PAN) interface. This allows, for instance, to replace a blood pressure sensor from one manufacturer by one from another. Furthermore, Continua focuses on an Electronic/Personal Health Record Network interface that allows transferring health information to back-end services. This functionality can be added as presented in one of our example applications in Section 5.

1.3 Summary of Contributions

The main goal of this thesis is the design and implementation of a middleware for body and ambient sensor networks that allows multiple applications to define and retrieve information types of their interest. The middleware as a layer between the applications and the sensory devices provides an abstraction from the individual sensors. It seamlessly handles interchanging sets of sensors and distributes the sensor information in an event-driven fashion. Upon receiving new sensor information, a corresponding event is generated and sent to the interested applications. The

middleware is implemented as an Android[5] application running on smart phones and tablet PCs. Since most of the current smart phones possess hardware for Bluetooth and Wi-Fi communication, we added various body sensors and ambient sensors using those technologies to our system, although our system is not limited to any particular set of sensors. Our system design evolved over three consecutive stages which represent the three main contributions of this thesis.

First, an event-driven layered middleware architecture that enables modular system design and seamless switching between sets of embedded and ubiquitous sensors is proposed. Sensor modules handle the communication with individual sensors and generate sensor reading events which are then injected into the system. Applications running on the same device subscribe to open communication channels with respect to the event types of their interest. This simplifies deployment. New sensors are introduced by simply adding a corresponding sensor module to the system which means that hardware and protocols can be changed without touching the applications. The system supports multiple applications running simultaneously and operating on the same sensors. Furthermore, applications are allowed to inject events into the system and therefore contribute to the overall monitoring solution. This allows for instance to have only one application instead of multiple applications performing costly activity recognition. In addition, services such as monitoring the system's liveliness and detecting critical health parameters are provided by the middleware.

Second, in order to provide tailor-made information retrieval and control over the data dissemination, we introduce application-specific communication channels. Instead of having open communication channels, each application specifically subscribes to the event types of its interest. Benefits are tailored information retrieval, energy savings, and the provision of required information for security mechanisms as well as adaptive event transformations. Based on event subscriptions, the middleware forwards only the requested sensor readings to the individual applications. Furthermore, only sensors that provide the requested information will be connected to the system. This avoids unnecessary but costly communication with sensors that are not required and saves resources (e.g. energy). A security manager provides control over the data dissemination by checking whether an application is allowed to access certain information and then granting or refusing subscriptions according to it. Sensor modules and applications advertise the event types they provide. By having lists of advertised and requested event types, the system provides sufficient information for event transformations.

Third, adaptive event transformation delivers sensor readings in the desired data representation. Incoming sensor data is hereby transformed into the desired representation which lifts the burden of adapting the application with respect to the connected sensors off the developer's shoulders. The middleware automatically mediates between old applications and new sensors and vice versa. Even more complex transformations such as transforming an ECG stream to simple heart rate information are supported. Furthermore, an unsupervised on-the-fly reloading of transformation rules from a remote server allows the system's adaptation to future applications and sensors at run-time.

[5] Android is an operating system for smart phones (`http://www.android.com/`).

In order to evaluate our middleware, we developed several applications in the areas of preventive and out-patient health care as well as elderly care facilitating different sensor configurations. The evaluation proves that the system runs interconnected devices and supports day-long monitoring. A performance analysis shows good results regarding energy consumption, message delivery ratio as well as CPU and memory usage.

1.4 Outline

The remainder of this thesis is organized as follows: The next chapter provides information about body and ambient sensor technology and the Android operating system. Furthermore, middleware approaches for sensor networks related to health monitoring applications are presented. In Chapter 3, we discuss the requirements on a middleware for ambient and body sensor networks for health care applications and present our middleware architecture and design decisions according to it. The actual implementation is discussed in Chapter 4. Both chapters specifically discuss the three contributions: open and application-specific communication channels, and adaptive event transformations. In Chapter 5, four applications using our middleware are presented: Two focusing on preventive health care, one on out-patient care, and one on elderly care. Based on those applications as well as generic sensor reading generators and applications, the performance of our middleware and the overall system's behavior are discussed in Chapter 6. Chapter 7 concludes this thesis and gives an outlook on future research directions.

2 Background

Body and ambient sensor networks consist of several hardware and software components. In order to give a better overview of the research field as well as related work, this chapter presents and discusses different aspects of body and ambient sensor networks for medical applications and related middleware approaches. Following a bottom-up approach, we will start with body and ambient sensor technology including current and future sensors, networking technologies, and mediator devices. As a next step, we will discuss related middleware architectures before we present the Android operating system which is used for our middleware implementation. Medical applications using body and ambient sensor technology were already presented in the introduction.

2.1 Body and Ambient Sensor Technology

Section 1.1 illustrated the current research state of monitoring applications for preventive and ambulatory healthcare as well as elderly care. Many systems [9, 38, 110, 67, 104, 41, 80, 108, 84, 71, 136, 64, 68, 46, 119, 147, 156, 155] consist of body sensors for sensing physical and physiological parameters and a mediator device that collects the sensor readings and processes or forwards them to a back-end system. Furthermore, the mediator can also be connected to ambient sensors either directly or via a local area network and, thus, can provide environmental sensor readings to the BSN application and vice versa. We will present different types of body and ambient sensors, commonly used network protocols for communication among BSNs, between BSNs and the environment, and between the personal health care system and the back-end system. We will argue for smart phones as a matching mediator device in a final subsection.

2.1.1 Sensor Types

In this section, we distinguish between ambient sensors integrated in the environment and body sensors which are worn by the user/patient. Ambient sensors provide information about environmental parameters (e.g. air quality) as well as activities of the user with the environment. They allow the observation of user motion within an apartment and interaction with instrumented objects. Based on those observations, activities of daily living (ADL) are recognized and deviations are detected. In contrast, body sensors are usually integrated in clothes and accessories or explicitly worn by the user. They provide physical and physiological information which is used for vital parameter monitoring, activity recognition and the detection of abnormal events. In the following paragraphs, both types of sensors and their data rates are presented and discussed. Furthermore, prospects for future body sensors are given.

Sensor	Measurement
Active Infrared	Motion/Identification
Ball-In-Tube Switches	Object Movements
Camera	Activity
CO_2	Air Quality
Humidity	Air Quality
Magnetic Switches	Door/Cabinet Opening/Closing
Passive Infrared (PIR)	Motion
Pressure	Pressure on Mat, Chair, etc.
RFID	Object Information
Smart Tiles	Pressure on Floor
Temperature	Air Quality
Ultrasonic	Motion
Microphone	Activity

Table 2.1: List of ambient sensors used in smart homes, based on a survey on ambient assisted living by Parisa Rashidi and Alex Mihailidis [117].

Ambient Sensors

Table 2.1 lists different types of sensors used for smart homes by integrating them into a person's apartment. They are of special interest for assisted living applications such as elderly care, because they provide information about the inhabitant's activities and for detecting security issues (e.g. a person leaves the apartment with the backyard door still being open). A summary on smart home projects and technologies is given in [117]. Most sensors for activity recognition are binary sensors indicating the user's presence (e.g. pressure sensor) or interactions with objects (e.g. magnetic switches, RFID). Therefore, the data rate of most ambient sensors is rather low. Nonetheless, microphones and cameras provide very rich information for activity recognition with very high data rates, but their deployment is very rare because of privacy concerns.

Usually, ambient sensors are connected to a back-end system for performing activity recognition and surveillance monitoring tasks. This connection is either wired or wireless. Wired systems are difficult to deploy in an already existing home and not feasible for some objects such as a vacuum cleaner. On the other hand, wireless systems have also their drawbacks like link losses, dead batteries, and dead gateway nodes. In [58], experiences and guidelines for smart home deployments are given. In very few systems such as the AlarmNet project [155], the user's BSN is part of the ambient sensor network. In the other cases, the BSN uses the local area network for connecting to ambient sensors and the back-end system.

Sensor	Measurement
Accelerometer	Acceleration
CO_2 Gas	Respiration
Electrocardiogram (ECG)	Cardiac Activity
Electroencephalogram (EEG)	Brain Activity
Electromyogram (EMG)	Muscle Activity
Electrooculogram (EOG)	Eye Movement
Galvanic Skin Response (GSR)	Perspiration
Global Positioning System (GPS)	Positioning
Glucose Meter	Blood Glucose
Gyroscope	Orientation
Heart Rate (HR) Sensor	Heart Rate
Light Sensor	Ambient Light
Magnetometer	Orientation
Microphone	Activity
Piezoresisitive Sensor	Respiration Rate
Pressure	Blood Pressure (BP)
Pulse Oximeter	Blood Oxygen Saturation (SpO_2)
Scale	Body Weight
Strain Sensor	Motor Exercises
Thermal	Body Temperature

Table 2.2: List of (bio-) sensors used in the applications presented in Section 1.1 and listed in several survey papers [29, 117, 109]. Most sensors are either explicitly worn or integrated in clothes and accessories.

Body Sensors

Table 2.2 summarizes the body sensors used in the example applications of our introduction (cp. Section 1.1) as well as in surveys on body area networks [29, 117, 109]. Except for the GPS and the ambient light sensor, all sensors focus on monitoring either the wearer's physiological or physical parameters. In order to avoid cumbersome cables, most sensors use wireless transmission to a mediator or a gateway node. Wireless communication is very costly in terms of energy consumption. Since many sensors sample at a high data rate, transmitting the raw sensor data would drastically reduce a sensor's operation time.

In order to reduce the high transmission workload, sensor data is usually aggregated before transmission or only sent upon an event. A common heart rate sensor, for instance, is based on a 1-lead ECG sensor and transmits its sensor readings with 1 Hz frequency, but internally it samples at a higher frequency that allows for the detection of multiple QRS complexes within one second.

The CorScience CorBelt[1] ECG sensor that we use for streaming ECG data does not support ECG streams per default. Instead, it performs an on-sensor cardiac arrhythmia detection and possesses a button for manually triggering data transmissions. Only upon an event, the ECG sensor sends a chunk of two seconds of raw 200 Hz ECG data around that event to a back-end system. Otherwise, it does not send any data. This allows to still gather important events but saves battery-critical wireless transmissions. In Section 5, we present applications operating on accelerometers. Instead of transmitting the raw 100 Hz acceleration data, only features of the acceleration data such as peaks, mean and variance values are transmitted. This is still enough for performing activity detection and repetition counting, but saves costly transmissions: only 12 Bytes of mean, variance and peak information instead of 100 Bytes of raw sensor data is transmitted every second. Since on-sensor processing is very often more energy efficient than wireless transmissions, body sensors increasingly perform on-sensor aggregations and feature extractions before transmission which drastically reduces the data rates. Therefore, the sensor's data rates do not reflect the actual wireless transmission rates.

Evolving Body Sensors

Some of the current body sensors are still inconvenient for day-long monitoring and it might be difficult to assume that someone is carrying multiple of those sensors throughout the day. Nevertheless, sensor technology is evolving towards small, integrated and unobtrusive sensory devices. Examples are a heart rate and blood oxygen sensor [7] which is worn like a ring and a low-power heart rate sensor [112] that is worn at the ear. Further examples are a respiration sensor [34] that uses a microphone placed on the neck and a cuff-less photoplethysmography-based blood pressure sensor [129]. It is smaller than usual cuff-based blood pressure sensors and works without putting pressure on the arm which makes it essentially unnoticeable to the patient. In [82], an oral sensor is presented that detects how much a person is drinking, chewing, speaking, coughing and smoking. So far, it uses a cable for data transmission. For future work, the authors propose a wireless Bluetooth connection to a smart phone. An EU-funded BIOTEX project resulted in a waist-band sensor for sweat analysis [35]. A textile pump collects sweat and analyzes its pH, sodium, conductivity, and sweat rate parameters. In [111], further technologies and future perspectives on wearable sensors are presented.

In addition to new sensor types and sensing technologies, the combination of sensor readings helps to improve the recognition of diseases (e.g. Parkinson's disease [10]) and, furthermore, it allows providing new reading types such as the estimated continuous cardiac output based on ECG, blood pressure and pulse oximetry readings [39].

2.1.2 Networking Technologies

A health monitoring application that uses both body and ambient sensor technology usually facilitates three different tiers as depicted in Figure 2.1. In a recent survey paper on ambient-assisted

[1] Data sheet of CorScience ECG systems: `http://www.corscience.de/fileadmin/Datenblaetter/BT3_6_12_en.pdf`

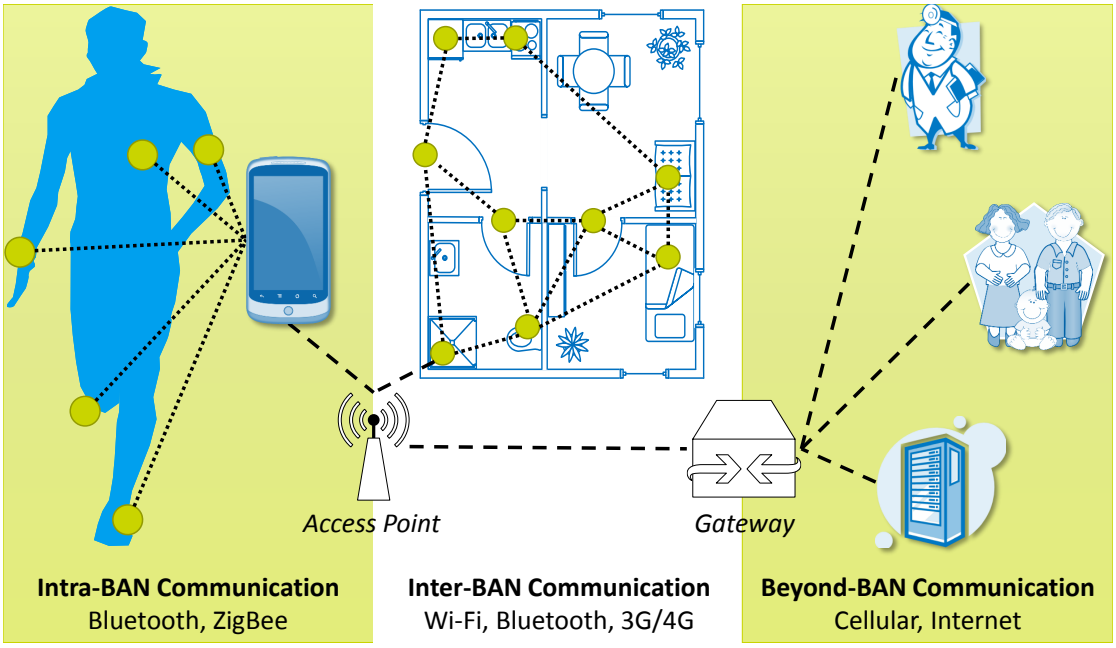


Figure 2.1: Communication in a 3-tier Body Area Network (BAN): body sensors communicate via *intra-BAN communication* among each other and with a mediator device, *inter-BAN communication* among BSNs and the environment including ambient sensors, *beyond-BAN communication* with care-givers and remote repositories.

living tools [117], popular networking standards are summarized: For **intra-BAN**[2] **communication** (cp. Figure 2.1), various IEEE 802.15 standards are used. ZigBee (IEEE 802.15.4) and Bluetooth (IEEE 802.15.1) are the most common ones. ZigBee provides data rates up to 250 kbit/s with a communication range of up to 20 meters. In contrast to Bluetooth, it also provides creating mesh networks with multi-hop communication that allows forming a large sensor network. The classical Bluetooth protocol provides data rates up to 1 Mbit/s with maximum 10 meters distance between two devices. In 2011, the Bluetooth Special Interest Group announced Bluetooth SMART[3], also known as Bluetooth Low Energy (Bluetooth LE/BLE). Bluetooth LE was designed for fitness and health care applications among others. Together with the Continua Health Alliance, different health care and fitness profiles such as profiles for heart rate (HRP), blood pressure (BLP), and running speed (RSCP) were defined. It aims to use less energy than the classical Bluetooth. Unfortunately, it is not backward compatible. Compared to classic Bluetooth, it has a reduced communication range and an application throughput of up to 270 kbit/s. Another communication technology mentioned in [117] is WiMedia (IEEE 802.15.3) with data rates up to 480 Mbit/s and a communication range of up to 10 meters.

Typical technologies for **inter-BAN communication** are Wi-Fi, Bluetooth, ZigBee, cellular, and 3G/4G networks [117]. If no mediator device is used, ZigBee and Bluetooth LE are the most promising technologies for connecting a BAN to the third tier. From a BAN perspective, ambient sensors are part of the inter-BAN network. They form a sensor network that forwards sensor

[2] Body Area Network (BAN)
[3] `http://www.bluetooth.com/Pages/Low-Energy.aspx`

readings to a gateway which connects it to the BAN. Actuators such as air conditioning devices can also be part of the second tier.

The **beyond-BAN communication** connects the system to back-end services such as a remote patient database and to emergency services. Furthermore, it provides information to care-givers such as doctors and relatives. The Internet is the most commonly used networking technology for beyond-BAN communication.

2.1.3 Mediator Devices

Wearable health monitoring applications benefit from unobtrusive and durable sensors that monitor health parameters with a minimum of disturbance. In order to provide small and lightweight sensory devices, the burden of data storage, complex processing, connecting to remote instances, and user interaction is put on a more powerful mediator device. The smart phone shown in Figure 2.1 depicts such a device which is carried by the user and bridges the body sensor network with inter-BAN and beyond-BAN networks. It manages the BSN and forwards sensor readings to a remote database as well as to care providers. Very often the mediator's additional storage capabilities and processing power are used for data storage, aggregation and deriving of (critical) events. Eighteen[4] of the 29 related middleware architectures presented in the next section make use of a mediator device. Most of the other projects assume gateway nodes in the environment to which the BSN connects.

A widespread hardware platform for mediator devices are smart phones. They became very popular and are part of many people's everyday life. In March 2013, comScore published a report on *Future in Focus - Digitales Deutschland* [32] showing trends for the digital future: In December 2012, 57% of the mobile phones used in the EU5 region[5] were smart phones and the share of smart phones is expected to increase. For instance, 77% of the phones sold in Germany in December 2012 were smart devices. As a result, the number of smart phone users in Germany rose by 45.5% from 21.3 million to 31 million users within one year. Furthermore, people tend to use multiple mobile devices: the number of users having a smart phone as well as a tablet increased by 87% from December 2011 to December 2012. Therefore, smart devices will be a ubiquitous platform carried by a huge percentage of users in the near future and they provide great features for serving as a mediator device: In addition to the high processing power and up to 64 GBytes memory, they provide rich networking capabilities (i.e. 3G/4G, Wi-Fi, Bluetooth, NFC, cellular, text messaging) and a well-known user interface.

As technologies for smart devices are evolving, the mediator might be integrated in clothing or in daily worn objects such as a watch or glasses. The Google Glass and the A.I Watch[6] are first examples of future mediator devices.

[4] Middleware architectures using a mediator device: [92, 69, 106, 140, 141, 76, 66, 25, 136, 137, 40, 94, 96, 95, 17, 53, 36, 1, 16, 77, 150, 152, 30, 149, 19, 18, 14, 42, 148, 163, 156, 155]

[5] EU5: Great Britain, France, Germany, Italy, and Spain

[6] Indiegogo campaign: I.A Watch Smartwatch (`http://igg.me/at/aiwatchsmartwatch`).

2.2 Related Middleware for Sensor Networks

This section discusses middleware approaches for body and ambient sensor networks that are related to our work. We distinguish between two underlying approaches: *distributed middleware* that runs on multiple sensor nodes which implement at least parts of the middleware and *mediator-only middleware* that runs on a single mediator device. Most projects that target (also) ambient sensor networks such as CodeBlue [130], LooCI [61, 62], and AlarmNet [156, 155] typically establish a multi-hop network consisting of sensor nodes. For those systems, having a mediator-only middleware that runs on a single device would not provide the required functionality and, thus, they are mostly distributed approaches. In terms of system adaption, distributed middleware is the more powerful approach because it allows deep interventions in the sensor's behavior. Many distributed middleware approaches provide for instance modifications in the sampling and reporting rate as well as inter-sensor communication. On the other hand, for providing those interventions, every sensor needs to be modified in order to implement the middleware functionality. This makes the sensors usually incompatible to other systems. In contrast, mediator-only approaches do not rely on specific hardware which, in principle, increases the amount of compatible sensors with the drawback of less control over the sensor nodes.

An important requirement for future health care systems demanded by the BMBF [21] and the Continua Health Alliance is the property of providing "personalized health solutions assembled from [...] interoperable health care devices and services" [33]. Therefore, we believe that a corresponding middleware approach should not rely on specific hardware. Instead, it should run on a personal device and operate on off-the-shelf sensors implementing standard protocols such as the profiles defined for Bluetooth Low Energy. Therefore, we opted in favor of a mediator-only approach. In Section 6.1, we show that off-the-shelf hardware is powerful enough for continuous patient monitoring throughout a day without sensor modifications. Nevertheless, distributed middleware is important for special applications with very strict requirements and for exploring future directions for new standards. Furthermore, each node in a distributed approach could act as a mediator-only system and provide information to the distributed system.

The following sections list distributed and mediator-only projects and describe two representatives of each middleware type in detail. For each type, we will present a commonly known approach as well as an approach close to our work. This section closes with a comparison of all listed middleware approaches with our myHealthAssistant middleware. For this, we classified the projects with respect to two contributions of our work: application-specific communication channels and adaptive event transformations.

2.2.1 Distributed Middleware

Distributed middleware requires sensor nodes to implement parts of the middleware or the whole middleware. It allows full control over the sensor nodes and, thus, provides an optimized sensor behavior that provides, for instance, energy savings or quality of service requirements. Most middleware approaches for ambient sensor networks are distributed. Examples of distributed

approaches are Milenkovic et al. [92, 69, 106], Codeblue [130], Mires [132, 133], Bacon et al. [17], LooCI [61, 62], SPINE [53], Self-Managed Cell [138, 74], MiLAN [56], CareNet [65], Lifeware [118], Dabiri et al. [36], mSens [77], DexterNet [79], SIXTH [24, 101], MobiSense [150, 152, 30, 149, 151], Kamal et al. [72], VITRUVIUS [19, 18, 14], and AlarmNet [156, 155].

MobiSense

The MobiSense project [150, 152, 30, 149, 151] is a popular representative of a distributed middleware for body area networks. It was developed at the A*STAR Institute for Infocomm Research (I^2R) and focuses on ambulatory health monitoring. In particular, an example application towards monitoring patients with cardiovascular disease was developed and tested in a real-world deployment. The system consists of three accelerometers worn on a belt and both legs, an ECG sensor, and a PDA device. The raw sensor data is transmitted to the PDA which performs heart rate monitoring as well as activity recognition. The authors introduce resource management and node reconfiguration that allow controlling and managing sensor nodes in order to adapt the system to changing application requirements. It allows for instance to toggle sensor nodes in sleep mode or to adjust the sensor's sampling rate.

In order to provide node reconfiguration functionality, the middleware resides on both the sensor nodes and the mediator (PDA). A so-called *lower middleware* running on the sensor nodes processes sensor reconfiguration commands and forwards sensor readings. This part of the middleware is written in nesC [48] using the TinyOS[7] platform. The *upper middleware* is written in C#, runs on a PDA, and consists of a gateway layer and an application layer. The gateway layer performs several tasks such as handling the sensor node communication, forwarding sensor readings to the application, toggling sensors on/off, monitoring the sensor's battery level, and adjusting the sampling rate. The application layer hosts the actual monitoring and activity detection application. Communication within the BSN is done via the IEEE 802.15.4 (ZigBee) protocol and it can be encrypted using a lightweight encryption algorithm (cp. [151]).

In [152], different aspects of a MobiSense prototype are presented and evaluated. Screenshots show a PDA application that displays acceleration and ECG waveforms and demonstrates that the system is capable of receiving and displaying the raw sensor data. A second set of screenshots shows a heart rate detection based on the ECG values as well as an accelerometer-based activity recognition. The activity recognition distinguishes between sitting, lying down, standing, and walking. Its classification is based on data from three accelerometers sampling with 25 Hz and it achieves an overall accuracy of 100%. A feature of having parts of the middleware running on the sensor nodes is the ability to switch sensors on and off. Therefore, an evaluation on the node's battery power consumption with and without toggling sensors on/off is presented. This feature allows battery power consumption to be decreased from approximately 50.5 mW to approximately 50.2 mW in best case which is almost negligible. Another feature for saving battery power is the *critical self-wake* which allows to set a threshold on which an inactive sensor re-activates and starts transmitting. An evaluation on the power savings of this feature is not given.

[7] `http://www.tinyos.net/`

On the whole, the system has proven to be capable of streaming raw data from up to four sensor nodes to a mediator as well as performing heart rate and activity detection on the mediator. It was also shown that by having parts of the middleware running on the sensor nodes, the overall energy consumption can be reduced. Adapting the sampling rate with respect to the application's requirements could also save energy, but this was not evaluated in [150, 152, 30, 149, 151].

VITRUVIUS

The VITRUVIUS project [19, 18, 14] was founded at the Departement of Mathematics and Computer Science of Eindhoven University of Technology in the Netherlands. Its goal is the exploration of "underlying key consequences for the architecture of Body Sensor Networks (BSN) and the handling of information about the individual's body"[8]. Two major concerns of this project are data privacy and security and the support of multiple concurrent applications. Target applications are medical solutions connected to a back-end system, sports coaching, and gaming. Those applications are supported by the VITRUVIUS system which runs on a smart phone and shares data and partial results among the applications. The system relies on off-the-shelf hardware but requires the sensor nodes to run the Open Service Architecture for Sensors [14] developed in this group. Thus, programmable Shimmer nodes are used as accelerometers and EMG sensors. Those sensors are connected to an Android tablet PC via a Bluetooth connection. The Android device serves as a *body hub* which provides several services to the applications and it connects the BSN to a back-end system.

A *Sensor Abstraction Layer (SAL)* running on the body hub handles the sensor node communication and configuration. Based on application's requirements, sensor nodes are configured, for example, adapting the sampling rate to an application's needs. Data is stored and shared by applications using a database in which each data type is represented by a table. In order to avoid a continuous polling of the database, applications subscribe to data types of their interest. Upon a new database entry, corresponding applications are notified. Applications are also allowed to contribute to the database which enables sharing information among applications. Users can download new applications at run-time allowing to adapt the system to the user's needs. A *trust & ownership manager* maintains the trustworthy and privacy properties [18]. If a new application provides new data types, corresponding database tables are created. Furthermore, the connected sensors are configured with respect to the application's requirements. A conflict resolution approach regarding opposing requirements was not mentioned so far.

In a case study [19], the capability of dynamically handling multiple applications was shown. First, an *application for posture detection* is installed. As a result, two new database tables are created: `acceleration` and `posture`. In addition, the accelerometer is configured to stream acceleration readings to the database. Based on this information, the posture detection recognizes whether a person is laying, sitting, or standing and stores it in the database. Upon installing an *activity monitoring application*, two new tables, `emg` and `activity`, are created and the EMG sensor is configured. Since the activity detection algorithm relies on a person's posture, the

[8] `http://vitruvius-project.com/`

second application subscribes to posture readings and, thus, benefits from the findings of the first application. A graphical user interface shows both the raw sensor data as well as the derived events.

VITRUVIUS is a BSN solution that works with programmable off-the-shelf sensors that allow modifications and provides an App-store like adaptation of functionality. Sensors are configured based on application's needs and an inter-application information exchange is enabled by offering a shared database. Although having multiple applications running on the system, application isolation, data privacy, and platform trustworthiness are still provided.

2.2.2 Mediator-only Middleware

Mediator-only middleware resides only on the mediator (e.g. smart phone, PDA) and does not require modified sensor nodes. It handles the sensor communication and forwards sensor readings to the BSN application or simply forwards the readings to a remote instance. Since mediator-only approaches do not require specific protocols on the sensor-side, integrating new off-the-shelf sensors should not take much effort. Distributed approaches can also provides this feature if they do not require off-the-shelf sensors to implement their specific protocols. In this case, the provided services are similar to those of a mediator-only approach. Examples for mediator-only middleware are: Triantafyllidis et al. [140, 141], MobiHealth [76, 66], MobiCare [25], LiveNet [136, 137, 40], Morón et al. [94, 96, 95], WASP [8, 135, 86], Abousharkh et al. [1], Boulmalf et al. [16], HatterHealthConnect [42], Wagner et al. [148], and Yi et al. [163].

LiveNet

LiveNet [136, 137] is a flexible distributed platform for proactive healthcare applications. It builds upon the MIThril [40] hardware platform. Both systems are developed at the Media Laboratory at the Massachusetts Institute of Technology, USA. The LiveNet system continuously monitors a wearer's health state, performs real-time processing, and gives feedback about the health state. Furthermore, health information is transmitted to care-givers and members of the person's social network. The system consists of a PDA hosting the middleware and a sensor hub to which several sensors (EKG, EMG, GSR, skin temperature, accelerometer, IR tag reader) are connected. The sensor hub either stores the sensor readings on a compact flash card or interfaces the PDA with the sensor network.

Sensor readings are collected by the sensor hub and then posted to the *Enchantment Whiteboard* system [40] running on the PDA. The Enchantment system is based on a client/server model that allows clients (e.g. sensors, applications) to post and read structured information. Furthermore, it allows clients to subscribe to specific topics of the whiteboard. Upon occurring changes, the corresponding clients are informed. This system is used for inter-process communication and it is intended to act as streaming database. It captures the state of a system or a person and supports many simultaneous clients. By connecting multiple Enchantment systems, it provides communication within groups. An additional Enchantment Signal System provides point-to-point

communication between clients which allows the transmission of high bandwidth signals such as real-time audio streams.

The authors of [136, 137] provide several potential applications of the MIThril LiveNet system. For a health and clinical classification scenario, a real-time monitoring system was developed. It performs an accelerometer-based activity recognition which distinguishes between activities such as standing, walking, running, cycling, and climbing stairs. Another system characterizes a conversation by using GSR-based stress detectors, detecting head-nodding/shaking and speaking states. Those findings can be streamed to a remote server for further processing. Another LiveNet scenario is the long-term health/behavioral trending that takes activities of daily living as well as health information into account. By enabling real-time audio stream to relatives and care-givers, the system builds a basis for an extensive elderly monitoring solution.

In addition to the middleware, the authors propose an architecture for the development and implementation of real-time context classifiers [40]. A *MIThril Real-Time Context Engine* processes real-world phenomena in four stages (the first three using the Enchantment whiteboard): 1) sensing, 2) feature extraction, 3) modeling, and 4) inference. Overall, the system handles incoming sensor readings and provides them to multiple applications using a subscription-based whiteboard approach. In addition to this, communication among multiple BSNs over the Internet is supported. The system was introduced in 2004 and operates mostly on wired sensors.

Morón et al.

Morón et al. from Departamento de Tecnología Electrónica at Universidad de Málaga in Spain propose a smart phone-based telecare system using a body area network [94, 96, 95]. The system consists of commercial-off-the-shelf Bluetooth vital parameter sensors and a conventional smart phone which allows using the system without any hardware or software modifications. A target application of that work is the efficient monitoring and management of chronic diseases. Therefore, the authors present a system that connects the patient's BSN to a remote server which allows care-givers to monitor patients and to control the system's behavior. A pulse-oximeter provides information about the heart rate, the blood oxygen saturation, and the perfusion level. In addition to the vital parameters, a GPS receiver provides location information to the system.

The software architecture is divided into three subsystems. The smart phone hosts a BAN control application called the *Node of Control (NC)*. This application reports sensor readings to a *Central Control Server (CCS)* using Internet connectivity. For controlling and monitoring the BAN, a web application is provided to care-givers. This application is called the remote *Control and Monitoring Unit (CMU)*. The NC component is implemented as a Python and as a Java ME application and provides functionality for managing the local BAN, visualizing health parameters, forwarding sensor data, managing and executing configuration commands from the CMU, and detecting critical situations based on vital parameter thresholds. Before sending a sensor reading to the CCS via HTTP POST, it is encoded as a JSON object. Configuration commands are retrieved by periodic HTTP GET requests sent to the CCS. The application logic resides on the remote services, except for the detection of critical events which is performed on the smart phone.

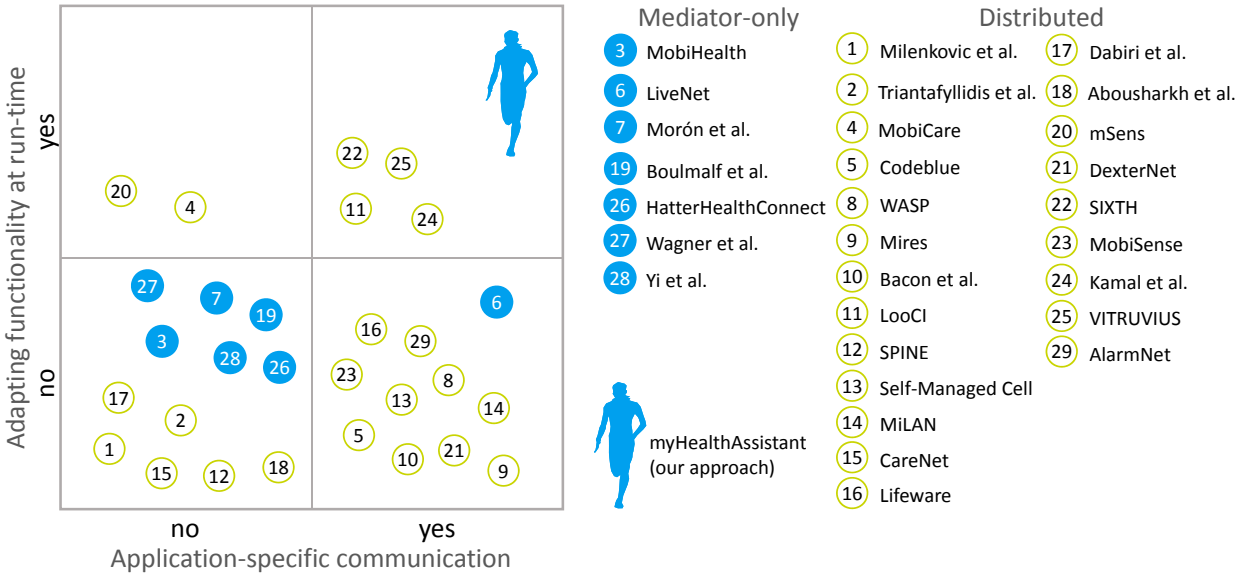


Figure 2.2: Comparison of our work (myHealthAssistant) with related work by focusing on two contributions of our work: *application-specific communication* and *adapting functionality at run-time*. A distinction is made between *distributed* approaches that run on multiple sensor nodes (green circles) and *mediator-only* approaches running on a single device (blue points). Our approach is the only mediator-only system operating on off-the-shelf sensors that provides both features.

Communication among the system's entities is based on web technologies. Therefore, the performance of different technologies for sending data from the patient's NC to the central control server was analyzed. Except for raw TCP connections, HTTP pipelining was the most promising one. Since care-givers connect to the Control and Monitoring Unit via a web browser, different web applications and simultaneous persistent connections were evaluated. A comparison of the energy efficiency between the Python and the Java solution running on the smart phone showed that the Java ME encoded client obtained a greater autonomy of almost 12 hours local health monitoring. For the future, the authors plan to extend their prototype to a Java implementation running on Android phones and a surveillance mode that would report only severe events to the server.

2.2.3 Comparison to this Work

Figure 2.2 depicts a comparison of our work with 29 middleware architectures for sensor networks; most of them focusing on medical applications. The comparison is based on two of the main contributions of our work, namely *application-specific communication* channels and ability of *adapting functionality at run-time* which corresponds to a broader scope of our adaptive event transformations. We assume that if a feature is not mentioned in the corresponding paper(s), it is not supported by the system. Furthermore, a distinction is made between *distributed* middleware (green circles) that requires modifications of the sensor's software in order to provide all services and *mediator-only* middleware (blue points) which does not rely on specialized sensors.

Apart from LiveNet, all mediator-only approaches do neither support application-specific communication nor an adaptation of functionality at run-time. Most of the distributed approaches support tailor-made communication channels and four of them also support on-the-fly adaptations: the projects (22) SIXTH and (24) Kamal et al. provide autonomous self-adaptation of the systems whereas the adaptations of (11) LooCI and (25) VITRUVIUS require user interaction. We propose a system that supports on one hand autonomous self-adaptation of its functionality and, furthermore, the adaptation of sensor data to application's needs. This transformation of sensor data is also mentioned in (10) Bacon et al., but it is not adapted at run-time.

In order to provide a flexible middleware for health care which supports multiple applications as well as a wide range of sensors, the system should not rely on specialized hardware. Although a major benefit of a distributed middleware is the saving of energy by tuning sensors and wireless communication, the lack of available hardware limits the system's interoperability. Therefore, we opted for a middleware approach that runs on a personal device and operates on off-the-shelf hardware while still providing enough resources for monitoring a person throughout a whole day (cp. Section 6.1). Nevertheless, our system is not limited to off-the-shelf sensors and also provides the configuration of sensors as shown in Section 5.4.

Regarding related work, Figure 2.2 shows that our work (*myHealthAssistant*) is the only system operating on off-the-shelf medical sensors that provides both application-specific communication as well as on-the-fly adaptation. The myHealthAssistant project already provides a "flexible software platform(s) as basis for new services in health sector" [20] as demanded by the German Federal Ministry of Education and Research and, therefore, takes an important step towards coping with the demographic change and chronic diseases.

2.3 Android OS

Android[9] is a Linux-based operating system designed for mobile devices such as smart phones and tablets. When we started implementing our middleware in 2010, there existed two promising operating systems for smart phones: Android OS from Google and iOS from Apple. We decided in favor of Android OS since it supported a higher amount of medical sensors and it provided establishing serial ports over Bluetooth using SPP which allows connecting any Bluetooth device/sensor to the system. In addition, the variety of hardware platforms and their cheaper prices were further benefits of Android OS. Nowadays, Android dominates the smart phone market. In December 2012, 53.4% of the devices used in Germany ran Android OS, 21.8% iOS, and 22.6% ran operating systems from Symbian, Microsoft, and RIM [32]. On the global smart phone market Android's share reached 65% in March 2013 [63]. An advantage of Android is its openness regarding hardware platforms. This has led to a great diversity of compatible devices which allow us to choose the best fitting device or to create a tailored device. In July 2013, OpenSignal identified 11,868 distinct Android devices [103].

In the following, different aspects of the Android system are described. After giving a rough overview, we focus on the four Android components as well as Android Intents since they are

[9] http://www.android.com/

important for the middleware implementation. Furthermore, a discussion about secure inter-application communication in Android is made because our system builds upon this functionality.

Android serves as an operating system for mobile devices. It is open source and released under the Apache License. Android uses a Linux kernel (version 2.6 / 3.x) that provides the necessary device drivers, power management libraries, as well as middleware libraries and APIs. In our work, we are using the SQLite database library for persistent data storage. Android applications are written in Java and executed in individual Dalvik virtual machines. "In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more" [51]. This basically creates a secure environment that grants only the required system access to applications. Nevertheless, there are still some security issues, especially in terms of inter-application communication which are discussed in Section 2.3.3.

Before deploying an application on an Android device, Java byte code needs to be translated to Dalvik executable code (`dex`-code). This is usually done by the Android development kit. In case of the event transformations described in Section 4.4, this needs to be done manually using the Android `dx` platform tool. For installation, every application has to provide an `AndroidManifest.xml` file which contains essential information about the application such as the application's name, the package name, its capabilities (cp. Intent Filters in Section 2.3.1), permissions, and the minimum required Android API level among others. Applications are usually downloaded and installed from the Google Play[10] app store. For more detailed information about the Android system, we refer to the Android development website[11].

2.3.1 Intents

Android Intents are used for asynchronous communication among components of a single application and among multiple applications. They either describe an operation that needs to be performed, such as starting an application, or they are used for information exchange, such as informing that the GSM signal strength has changed. A distinction is made between *Explicit Intents* and *Implicit Intents*: an Explicit Intent specifies to which *Android Component* it has to be delivered whereas an Implicit Intent requests the delivery to any Component that supports the desired operation. The Android system always tries to find an appropriate Android Component to respond to a given Intent, creating an instance if necessary or contact the user if multiple Compontents with the same priority match the Intent. Intents are always delivered to their corresponding Android Component type e.g. Broadcast Intents are always delivered to Broadcast Receivers and never to Activities or Services (the individual components are described in Section 2.3.2). Table 2.3 list all attributes an Intent possesses. The attributes category and extras are important for our work.

[10] `https://play.google.com/store`
[11] `http://developer.android.com/index.html`

Attribute	Description
Action	A String that defines the action to be performed or the information type that is exchanged, for example: `ACTION_CALL` targets an Activity to initiate a phone call, `ACTION_BATTERY_LOW` targets a Broadcast Receiver and warns that the battery is low.
Component name	Contains a `ComponentName` field that specifies the target Component for handling an Intent.
Category	A String that specifies the kind of Component that should handle the Intent.
Data	The URI and the MIME type of the data to operate on.
Extras	Key-value pairs containing additional information. The values can consist of primitive data types as well as of type `PARCELABLE`. The latter allows to exchange any information as long as it can be marshaled and unmarshaled.
Flags	Various types of flags. Some for instructing Android how to launch an application.

Table 2.3: Attributes of an Android Intent.

Explicit Intent

Explicit Intents are delivered to a specific Android Component by addressing its unique `ComponentName`. Since the component name is usually unknown to foreign application developers, Explicit Intents are typically used for intra-application communication. An Explicit Intent guarantees delivery to the intended recipient [31]. In order to make an Intent explicit, one of the following methods `setComponent()`, `setClass()` or `setClassName()` needs to be evoked and, thus, the Intent's `ComponentName` field specified. This field consists of two Strings: the recipient's unique package name and its class name. Referring to a BMI[12] (body mass index) Calculator example, the code for explicit addressing is:

```
intent.setComponent(new ComponentName("com.example.bmicalculator",
                                      "com.example.bmicalculator.Activity"));
```

Another interesting method is `setPackage()` which combines explicit addressing with an Implicit Intent resolution. By defining the recipient's package name, it is ensured that the Intent is only sent to that specific application without limiting the Intent to a Component within the application. In the BMI Calculator example, the code would be:

```
intent.setPackage("com.example.bmicalculator");
```

The Implicit Intent resolution starts within the BMI Calculator application and only resolves to Components that are declared in this specific application. Taking a newspaper delivery as an example, an Explicit Intent would deliver the newspaper to a specific person in an apartment. By

[12] The body mass index (BMI) is a measure for human body shape based on the person's mass and height.

using the `setPackage()` method, the newspaper is delivered to any person in the apartment that declared interest in reading newspapers.

Implicit Intent

Implicit Intents do not address a specific recipient. Instead, they describe the capabilities the target Component has to possess. This is done by using the Intent attributes *action, category,* and *data* described in Table 2.3. Coming back to the BMI calculation example, an Implicit Intent does not specify the application, but it describes that it needs an application that, for instance, calculates a person's body mass index. Then, the Intent resolution may lead to three Activities (e.g. BMI Calculator, Android-BMI, Idealgewicht (BMI)) that support the desired functionality and lets the user choose one. In case the user sets a default Activity for that task, this Activity is always chosen. Implicit Intents encourage inter-application collaboration and, thus, code re-use, but they also have some drawbacks when not used carefully. Some resulting security risks are discussed in Section 2.3.3.

For enabling the Android system to perform the Intent resolution, applications need to specify the Intent types they handle. This is done in the application's `AndroidManifest.xml` by using Intent filters as described in the following paragraph.

Intent Filters

An Intent Filter is a list of operations an Android Component supports. This list consists of the *action, category,* and *data* attributes described in Table 2.3. Android Activities, Services, and Broadcast Receivers can list one or more Intent Filters in the AndroidManifest.xml in order to filter out unwanted Intents. This applies only for Implicit Intents. Explicit Intents are delivery regardless of specified filters. Therefore, if a component is made available to other applications, it has to be able to process or at least filter out unwanted Explicit Intents, otherwise unintended behavior as well as security issues may occur (cp. Section 2.3.3). The following example sketches a body mass calculator that tells the Android system that it only processes text data containing height and body mass values:

```
<intent-filter . . . >
    <data android:mimeType="text/height" android:scheme="http" . . . />
    <data android:mimeType="text/bodymass" android:scheme="http" . . . />
</intent-filter>
```

2.3.2 Components

Components are the essential building blocks of Android applications [51]. There are four different types of components, each has its own specific role: An `Activity` is used for the graphic user interface. The purpose of a `Service` running in the background is to perform long running operations without blocking the user interface. `Broadcast Receivers` are used for

information exchange within an application and among applications and the Android system. `Content Providers` are used for a persistent internal data storage and for sharing information among applications. Each component has a distinct lifecycle that defines how it is created and destroyed.

Activity

Activities are used for starting applications and for user interaction. Each Activity represents a single user interface which is described either directly in the Activity itself or via an XML layout file. Since Activities inherit from the Android `Context`, they can access all granted System functions and libraries. An Activity is started by an Android `Intent` which can be evoked by the application itself, a foreign application, or the Android system. The evocation of an Activity Intent within an application could be the switching from one screen to another, such as from a list of blood pressure readings to a visualization of an ECG stream. If an application needs to take a picture but does not implement camera functionality, it can evoke a foreign application to take that picture. Upon the photo is taken, the application control together with the picture are handed over to the initiating Activity. Activities are only active as long as they are visible for the user. As soon as they become invisible to the user, they enter an `onPause` state and cannot perform tasks anymore.

Service

Services are used for long running background tasks such as downloading a file from the Internet or playing music. They do not possess a user interface, but they also inherit from the Android Context which grants them the same functionality as for Activities. By binding to a Service, one or more components can start a Service and interact with it. In order to stop a service, components have to unbind from the Service. If the last component unbinds, the Service is stopped. A Service is not bound to an application's border. By introducing a Service as a `Remote Service`, other applications and the Android system can bind to this Service and interact with it. Apart from binding a service, it can be simply started by an Android Intent, but in this case the invoking component has no control over the Service's life-cycle. Applications that perform continuous tasks running throughout multiple screens or in background typically run their tasks as Android Services.

Broadcast Receiver

Broadcast Receivers are used for system-wide asynchronous communication using Intents. Intent Filters describe the type of Intents a receiver expects. Based on the Intent description (cp. Table 2.3) and the Intent Filters from available Broadcast Receivers, the Android system delivers Intents to matching receivers. Upon receiving an Intent, it is processed and typically relayed to an Activity or Service.

There are three types of Broadcast Receiver Intents: *normal, ordered,* and *sticky*. Normal broadcasts are sent to all registered receivers at once and disappear after delivery. An ordered

broadcast is based on a priority rating. The receiver with highest priority receives the Intent first. It can modify or stop the propagation of the broadcast before it is delivered to the receiver with the second highest priority and so on. Sticky broadcasts remain in the system after they have been delivered. This allows future receivers to retrieve information that was sent before their initialization. Information about the phone's energy level, for instance, is sent as a sticky broadcast which allows starting applications to retrieve the last update on the device's battery level.

Broadcast Receivers can be either *static*, *dynamic*, or *local*. Static receivers are declared in the AndroidManifest.xml using Intent Filters. Their objects are instantiated by Android when a corresponding Intent occurs. As soon as the Intent is processed (the *onReceive()* method returns), the object is abandoned. For a new matching Intent, a new object of the receiver is instantiated. Static Broadcast Receivers can be addressed by Implicit and Explicit Intents. By marking them as *private*, they are only available within an application. Marking them as *exported* makes them available to the whole system. In contrast, dynamic Broadcast Receivers are always exported and can only receive Implicit Intents. They are declared at run-time and their object is not abandoned after an Intent was processed. Local Broadcast Receivers behave like a dynamic receiver without being exported to the system. Since a system-wide Intent resolution is not needed, they are more efficient and they are recommended for intra-application communication.

Content Provider

Content Providers manage the access to structured data across components of an application as well as across applications. Components can query and modify data which is stored in any persistent storage of an application such as the file system or a SQLite database. Thus, Content Providers provide structured information exchange among applications as long as the requesting application has the proper permissions. An example of a Content Provider is the Android's contact list. Applications with the permission to access this list can query and modify contacts. In contrast to the other Android components, Content Providers cannot receive Android Intents. Instead, communication is handled via a common API.

2.3.3 Secure Inter-Application Communication

In this section, we discuss several aspects of secure intra- and inter-application communication based on analyses [73, 31] done at the University of California, Berkeley, USA. The results of this discussion are applied to the application-specific communication channels introduced in Section 4.3. The authors of [73] write: "To provide isolation between applications, Android runs them under separate UIDs, effectively sandboxing the applications [...]. However, problems arise when the Android API pokes holes in the sandbox by allowing applications to communicate with one another". The price for providing inter-application interactions that enables code re-use, swift application development, and an increased user experience is the introduction of an additional attack surface. Intents can be eavesdropped, modified/replaced, or stolen which can lead to unexpected behavior as well as compromised user privacy. If developers expose a Component or a message (unintentionally) to third-party applications, they automatically open a door for

attacks. The *SMS Message Spy Pro* application, for instance, behaves as a tip calculator, but at the same time, it forwards all sent and received SMS text messages to a third party [31]. Therefore, it is important to make a strict distinction between intra- and inter-application communication in order to avoid unnecessary exposure of Components and information. In the following, we will focus on how to secure the communication within the middleware and the communication among applications and the middleware. It is assumed that the operating system is not compromised.

Protection by Component Declaration

In order to receive Intents from foreign applications, Components have to declare themselves in the AndroidManifest. This is done by setting an `EXPORTED` flag or by defining at least one Intent Filter. Intents that match an exported Component are accordingly delivered. Unfortunately, defining Intent Filters does not provide any security mechanisms. A sender can easily send an Intent that matches the filter without containing the expected content. In addition to this, by sending an Explicit Intent to an exported Component, the filter system is entirely bypassed [31]. Therefore, Google introduced permissions to the Android system which allow applications to specify certain permissions a caller must possess in order to evoke a Component. Furthermore, a broadcast Intent sender can also limit the set of Broadcast Receivers to those possessing the right permissions. Application developers can use existing permissions or introduce their own ones. Each permission has a protection level that defines how difficult it is to acquire [31]:

- `Normal` is the default permission which is automatically granted by the system.
- `Dangerous` permissions are required if the requesting application gets access to private data or can negatively impact the user. It needs to be granted by the user during installation. If not granted, the application is not installed.
- `Signature` permissions are only granted if the installing application is signed with the same certificate as the application that declared the permission. They are useful in order to restrict component access within a small set of applications (e.g. applications from one developer).
- `SignatureOrSystem` permissions are granted if the Signature permission is granted or if the application is installed in the system applications folder. The system application folder is restricted to Google applications and those from the device manufacturer.

A component protected with a normal permission is essentially unprotected because any application can easily obtain the permission [31]. Using dangerous permissions and letting the user choose may be a risk in some cases because users might not be aware of the consequences of granting permissions. Furthermore, users are not able to decline individual permissions which usually leads to granting all permissions in order to install an application. For providing strong protection, developers can require that callers hold Signature or SignatureOrSystem permissions. As long as the developers signing key is not compromised, the Components requiring those permissions can be seen as private or unexported [73]. On the other hand, requiring at least Signature permissions limits the interoperability to applications sharing the same signing key.

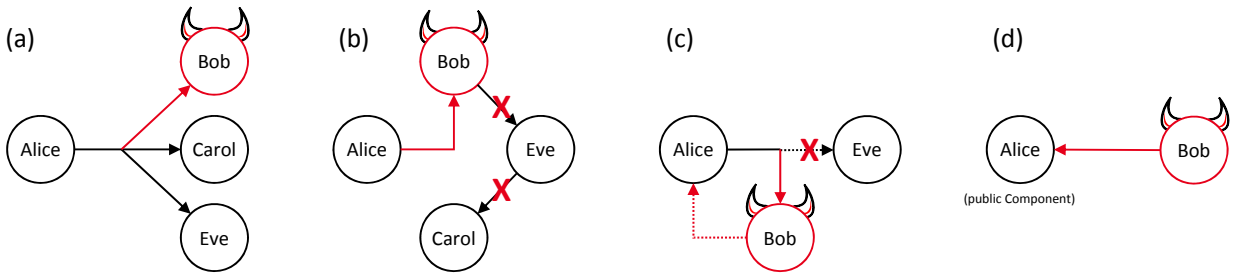


Figure 2.3: Intent-based attacks from Bob: **(a) broadcast eavesdropping** by declaring a matching Intent filter, **(b) denial of service attack** against ordered broadcasts by having the highest priority and not forwarding the Intent, **(c) Activity/Service hijacking** by declaring a matching Intent filter and acting as the desired Component, **(d) Intent spoofing** by sending an Intent to an exported Component. Source: [31].

Intent-based Attacks

Intents are the data structure for inter-application communication as well as for evoking Activities and Services in Android. Therefore, they form an attack surface that needs to be discussed. To unexported Components, the following attacks are not applicable since the Android system isolates those Components in the virtual machine. Components requiring Signature or SignatureOrSystem permissions are also considered as safe and not being subject to Intent-based attacks [31]. Since normal permissions are automatically granted and dangerous permissions are often granted by the user in order to install an application, exported Components requiring those permissions might be targets for Intent-based attacks. Basically, there are two types of attacks: unauthorized intent receipts and intent spoofing.

Unauthorized Intent Receipt

By sending an Implicit Intent, there is no guarantee that this Intent is delivered to the intended recipient. An intruder could declare an Intent Filter that matches the Intent and, thus, intercept the message. As a result, confidential information could leak or an Activity/Service could be hijacked. Figure 2.3 (a) depicts an eavesdropping attack to an implicit broadcast message that is intended only to Carol and Eve. Bob defines a Broadcast Receiver with a matching IntentFilter and, thus, also receives the broadcast message. Besides normal broadcasts, sticky broadcasts are particularly at risk for eavesdropping because they persist in the system [31]. Ordered broadcasts could be used for denial of service or data injection attacks. In Figure 2.3 (b), Bob steals the ordered broadcast message and causes a denial of service by not forwarding it. Activity and Service hijacking entails the risk that an application establishes a connection to a malicious Component. The malicious application can then steal and modify data which could lead to information leaks and unexpected behavior. In Figure 2.3 (c), Bob hijacks the Activity/Service connection to Eve and returns malicious results to Alice.

Intent Spoofing

Figure 2.3 (d) illustrates an Intent spoofing attack in which Bob sends an Intent to an exported Component of Alice. If Alice does not expect malicious Intents and does not check the originator of a message, she might take dangerous actions. Sometimes developers are not aware of having exported Components and, thus, they simply do not expect Intents from foreign (malicious) applications. This makes the situation even more severe. If a Broadcast Receiver blindly trusts incoming Intents, it could operate on malicious information and perform unintended actions such as storing or deleting data, or launching an Activity or Service.

Guidelines for Secure Inter-Application Communication

The authors of [31, 73] drew guidelines for secure communication in Android. In order to provide a flexible software platform for health care solutions, communication between several applications and the middleware is inevitable and security risks need to be avoided. Therefore, the following summarizes aspects of the guidelines that are used for the implementation of our work (cp. Chapter 4):

- For **intra-application communication**, Intents should be sent explicitly in order to avoid unauthorized Intent receptions. If this is not possible, the Intent should be protected by specifying strong permissions. If the receiving Component does not expect Intents from other applications, it should be private and, thus, not exported. This avoids the reception of external Intents.

- In case of **inter-application communication**, private data should be sent using Explicit Intents. If the recipient's Component is not known, Intents can be sent explicitly to an application and then being resolved implicitly within the target application by using the `setPackage()` method. (In [73], Kantola et al. noticed that due "to a bug in the Android source code, setPackage() does not limit the recipient to specific applications for broadcast Intents". Referring to the Android documentation[13], this bug is resolved since Android 4.0). As Intent filters are not security measures, exported Intent receivers should always check the caller's identity before performing any operations in order to avoid Intent spoofing. A second possibility of securing inter-application communication is requiring Signature permissions among applications. In this case, only applications with the same signing key can exchange Intents which secures the system but also limits the interoperability.

In Section 4.3, we will introduce application-specific communication channels that make use of inter-application communication and the guidelines listed above. For communication within the middleware, we are using local receivers that are unexported.

[13] `http://developer.android.com/reference/android/content/BroadcastReceiver.html`

3 Event-driven Middleware Architecture

In this chapter, our event-driven middleware architecture for body and ambient sensor networks (BASNs) is presented. The system targets a software platform for multiple medical applications running on a mediator device such as a smart phone. Sensor readings from on-body and ambient sensors are automatically collected by the middleware and delivered to applications. In Section 3.1, we discuss the software requirements on such a system and corresponding demands on the mediator device. Based on those requirements, a generic event-driven middleware architecture is presented in Section 3.2. Our first middleware approach provides open communication channels that relay sensor readings to applications as described in Section 3.3. Since not every application should be able to access all types of sensor readings, especially in health care scenarios, we introduce application-specific communication channels (cp. Section 3.4) that allow granting or declining sensor reading access for individual applications. Applications may require different data representations of the same information (e.g. lbs versus kg). In order to avoid additional sensors providing the same information or adaptations of the individual applications, an adaptive event transformation that supports providing information in different data representations is presented in Section 3.5. This chapter concludes with a summary and comparison of the requirements on a BASN middleware to the middleware presented in this chapter. The requirement analysis and the descriptions of the architecture are based on [127, 128].

3.1 Requirements

The motivational example presented in Section 1.1.4 highlights different aspects of an extensive health monitoring application for post-heart-attack patients. It consists of a tailor-made set of four specific monitoring solutions: extensive post-heart-attack monitoring, patient-specific risk factor monitoring, a rehabilitation & maintenance program, and a clinical study. We use this example in order to emphasize the requirements on a flexible platform for future health monitoring applications. It already shows that the demands on a body and ambient sensor system change over time (here, after two weeks) and that the system should accordingly be easy to adapt to new sensor configurations. In a comprehensive health monitoring ecosystem, those changes can occur over a long time period (e.g. patients getting additional chronic diseases, improved sensors becoming available, sensors breaking) or within a short period of time (e.g. ad-hoc connection to a scale, sports sensor setup). Since health solutions benefit from the advances in new sensory devices and the resulting services provided to a user or patient, they should be easy to adapt to new circumstances (e.g. sensors, applications, requirements). Therefore, adaptability, extensibility, and seamless handling of sensor configuration changes are significant requirements for a BASN architecture.

We distinguish between two roles of body and ambient sensor network devices: sensor and actuator devices for measurements and physical reactions on one hand, and a central mediator device for collecting sensor data and decision making on the other hand. Sensors provide events in the form of measurement information whereas actuators and the mediator consumes and processes these events. Processing often means to aggregate, store, and forward events, but also means to derive new events. A derived event could be the combination of multiple (often heterogeneous) sensor readings or the reaction to combinations of events (e.g. an emergency call because of a detected heart attack). In order to provide a history of events, the mediator should provide sufficient storage capabilities as well as the capability to communicate with other instances (e.g. a health care provider).

The following will enumerate the requirements on BSN middleware that we proposed in [127]. Compared to that list, the area of application is expanded to ambient sensing and multiple application support as sketched in our introduction (cp. Section 1.2). As a result, requirements for a generic middleware architecture for body and ambient sensor networks that supports future health care solutions are presented.

3.1.1 Multi-Setup Requirements

Multiple-Application Support

With growing multimorbidity, i.e. co-occurrence of two or more chronic medical conditions in one person, a single application that monitors all facets and possible combinations of diseases becomes very unlikely. It is more likely that future health care solutions consist of multiple specialized applications targeting a specific disease or parts of a disease. In the introduction we sketched such a solution in which a patient, after suffering from a heart attack, is monitored by multiple applications, each application focusing on a particular aspect. While doing so, the combination of multiple applications forms a tailor-made health care solution. This solution can be adapted to changing requirements such as a progressing disease by changing the combination of specific applications. In order to provide a flexible health care solution, a platform is required that supports the combination of multiple applications running in parallel.

Inter-Sensor Compatibility

Over time, sensors can break and old sensors are replaced by new and better ones (e.g. less obtrusive, more precise). In order to avoid a continuous adaptation of applications to new sensors and sensor types, the middleware should provide an abstraction from the individual sensors. Furthermore, a sensor might also be replaced by another sensor type which can be used for the same application but provides the information in another manner (e.g. in a coarser/finer granularity). The extensive post-heart attack monitoring application in Section 1.1.4, for instance, requires detailed ECG stream information in addition to the heart rate readings required by all other applications. Since an ECG curve already provides heart rate information, it makes an additional heart rate sensor superfluous. On the other hand, most heart rate applications rely on simple heart rate readings and will not work with ECG curves. Therefore, simple mechanisms

to convert sensor information to a system-wide format should be provided, even if the available information has a higher granularity than the desired one.

Swift Adaptations

Monitoring solutions need to be adapted to a progressing or new disease. An obese patient whose weight, blood glucose level and blood pressure are monitored might develop a sleep apnea. In this case, an additional monitoring application using a respiration sensor might be added to the system. Ideally, this is done in a plug & play manner in which the physician simply adds the new sensor to the system and triggers the installation of the additional monitoring application. Those changes should be easy to perform and cause minimal impact on the overall system. Furthermore, they should not influence unrelated applications. Nevertheless, if an application gains from a new setup (e.g. optional sensor readings becoming available), information about those changes should be passed to the corresponding application.

Security, Privacy

Since body and ambient sensor networks are sensing very personal information, security and privacy are important to keep in mind, even more so for health care applications. Having intruders stealing information about a patient's severity of a disease would obstruct the acceptance of the whole system. With respect to a middleware running on a mediator device, mechanisms need to be provided for securing the communication among sensors, the mediator and remote instances. If there are multiple applications using the BASN, additional mechanisms for controlling data dissemination within the BASN are required. The clinical study in our motivational example should for instance not be able to gain access to information such as the patient's blood glucose level or specific activities. In addition to this, system reliability is critical for the acceptance of health care applications.

3.1.2 Networking Requirements

Protocol Independence

Since network protocols are important for the lifetime, security, and performance of sensor networks and might be adapted to a special application, a generic BASN architecture should not rely on a specific protocol. With the introduction of Bluetooth LE, for instance, new and power-efficient health monitoring devices appeared that are not compatible with the classical Bluetooth protocol. Therefore, future bio-sensors will preferably use Bluetooth LE. The system should enable a swift adaptation to this protocol and hardware. The same applies for ambient sensor networks: for instance, if the smart home monitoring system switches its sensor reading representation from JSON objects to an XML encoding. Instead of relying on a fix set of network protocols, it is important to support different protocols and to allow a swift integration of new protocols. Furthermore, more than one protocol in parallel should be supported. Most body sensors are connected via Bluetooth or ZigBee whereas ambient sensor readings are mostly retrieved from a smart home server over Wi-Fi. With a protocol-independent architecture, the number and

type of sensors that the system can handle is automatically increased and, hence, the supported application area widened.

Ad-hoc Connections

A second networking requirement is the ability of seamless switching between sensor configurations. This increases not only the usability of the system but also its opportunities. By enabling a BASN application to utilize sensors as they become available without any additional interactions, the application's management overhead is reduced and the best possible service provided. In the post heart attack rehabilitation & maintenance application, for instance, additional sensors for detecting specific sports activities are embedded in sports accessories such as running shoes. As soon as the patient wears the shoes, the system should automatically detect the sensors and provide their readings to the corresponding application. Further examples are stationary body sensors like a scale or a blood pressure sensor. Those sensors are not always connected to the BASN, but the connection should be established as soon as they become available.

Bridging/Bundling Networks

In case of an emergency, it might be necessary to stream sensor data such as the heart rate from a Bluetooth body sensor network to beyond-BAN instances such as a physician's PC connected via a TCP/IP network. In this case, sensor information needs to be bridged from one network to another. The same applies for the clinical study in our motivation example. Furthermore, even within one tier sensors might use different network protocols. This should be transparent to applications in order to minimize development effort and to maximize the interoperability of sensors. Therefore, it is important that sensor information is bundled, independent from the originating network protocol.

3.1.3 Mediator Requirements

Central Reasoning

Having the applications running on the mediator device which collects and bundles all sensor readings simplifies the development and maintenance of BASN applications: sensor fusion and reasoning is done on one device which reduces the system's complexity. For changing the system's behavior such as adding a new vital sign monitoring, applications are simply installed, uninstalled or modified. Adaptations to new sensor environments are done by changing parts of the mediator without touching individual sensors. In contrast to a distributed approach, battery-powered sensor nodes can focus on the sensing process, which simplifies their development and helps saving energy.

Modular Architecture

The mediator's software platform has to be adaptive and extensible in order to support changing applications and sensing technologies. The stationary blood pressure sensor used in all four example applications might be replaced by a new sensor (type) such as the cuff-less and less noticeable sensor proposed in [129]. As a result, parts of the architecture might need to be

modified. Another example is the derivation of vital parameters from a combination of sensor readings as a result of new research findings. In order to provide this new feature, it is important to have a modular architecture running on the mediator that allows adding or removing functionality in an easy manner and with minimal impact on the whole BASN system.

Self-configuration

As already mentioned in the networking requirements, sensor units might appear and disappear at run-time. The mediator should be capable of adding or removing sensor-related software functionality as sensor constellations change. This property of self-configuration ensures best available service based on the current sensor configuration. The rehabilitation & maintenance program, for instance, requires a special set of activity sensors. Those sensors also provide the activity level which is required from the other applications. As soon as the sports sensors are connected, the mediator could enable a service that transforms sports activities to activity levels and stop the regular sensors. Upon the fitness sensors disconnect after a fitness workout, the transformation service is stopped and the regular sensors are started again.

Sufficient Local Resources

In order to not being the system's bottleneck, the mediator has to provide enough processing power for performing tasks such as sensor fusion and reasoning. Many cardiac monitoring applications combine activity information with heart rate information. In case the heart rate does not match the activity or history of activities, an alarm is raised. For providing this service, the system needs to perform real-time activity recognition. Real-time classification of steaming sensor data, coming from multiple sources, requires processing power. Furthermore, sensor readings and derived events need to be logged for documentation and further analysis. Due to a lacking Internet connection or for saving wireless communication, the mediator should also provide enough storage capabilities (e.g. storing long-term ECG readings).

Connectivity

Health care applications often require connectivity to remote instances. In our post heart attack example, there is communication among the patient's sensor network and health care providers for sending health reports (i.e., for the clinical study) or raising an alarm in case of an emergency (i.e., both monitoring applications). Especially for the latter case, a feedback channel to the patient is desirable in order to be able to react immediately to a serious event (e.g. calming down a patient or verifying that an emergency is taking place). The rehabilitation & maintenance program makes use of social platforms for which the system typically needs an Internet connection. Besides the connectivity to remote instances, the connectivity to body and ambient sensory devices remains a requirement.

3.2 Generic Architecture

In this section, we describe the generic architecture of our middleware for body sensor and ambient sensor networks. The middleware, running on a smart phone, handles communication with its

surrounding sensors. Upon receiving a sensor reading, it delivers the sensor information to all interested applications in a uniform manner. In a first version, the delivery is done via open communication channels. In Subsection 3.4, we introduce application-specific channels that allow securing the communication between applications and the middleware. Subsection 3.5 describes how the adaptive event transformation expands the application area of sensors and applications.

Event-driven Middleware

Key properties of a middleware for medical applications are the support of an arbitrary wide range of different sensors and support for multiple applications. In addition to this, a BASN middleware should be able to seamlessly switch between sensor constellations in order to always provide the greatest possible information base to applications. We believe that event-driven systems fit the nature of body and ambient sensor networks because:

a) Sensor constellations and running applications change over time and, thus, the set of event producers and consumers changes.

b) Most body and ambient sensors send their readings in an event-driven manner, e.g. when a threshold is exceeded or as soon as an event is detected.

c) Sensors do not know which applications are interested in their readings.

Those characteristics fit the loosely-coupled communication paradigm of event-based systems very well. In addition to this, event hierarchies allow for the establishment of a common and extensible data abstraction on which various applications can be built. Converting sensor information from a higher granularity to a lower one is achieved by an event hierarchy and enables the system to handle future sensors without changing major parts of the system. Furthermore, by introducing event transformations (e.g. ACTrESS [47]) or ontologies (e.g. CONNECT [12, 11]) to the system, a comprehensive interoperability and integration in other event-based systems are possible which increases the system's opportunities.

Events in our system consist of the following fields: an `eventType` that follows a system-wide event structure (cp. Section 3.3), a unique `eventID`, a `timestamp`, and an event `producerID` that identifies the actual producer (e.g. a specific sensor or application). For some medical applications, we believe that it is important to know the event producer. It allows applications to adapt their behavior to specific sensors and, thus, provides a better service to the users/patients. In addition to the event descriptors, every event contains one or more event type specific data fields. For instance, sensor reading events additionally contain a `sensorType`, a `timeOfMeasurement` and a `payload` field. The time of measurement differs from the event's timestamp if a sensor reading could not immediately be transmitted to the mediator.

Smart Phone as the Mediator Device

In today's society, smart phones are very popular and accepted end-user devices (cp. Section 2.1.3). They provide large touchscreens, high processing power, large storage and networking capabilities. The performance of smart phones is constantly evolving. Since connectivity is an important factor for smart phones, most phones support a wide range of the latest networking

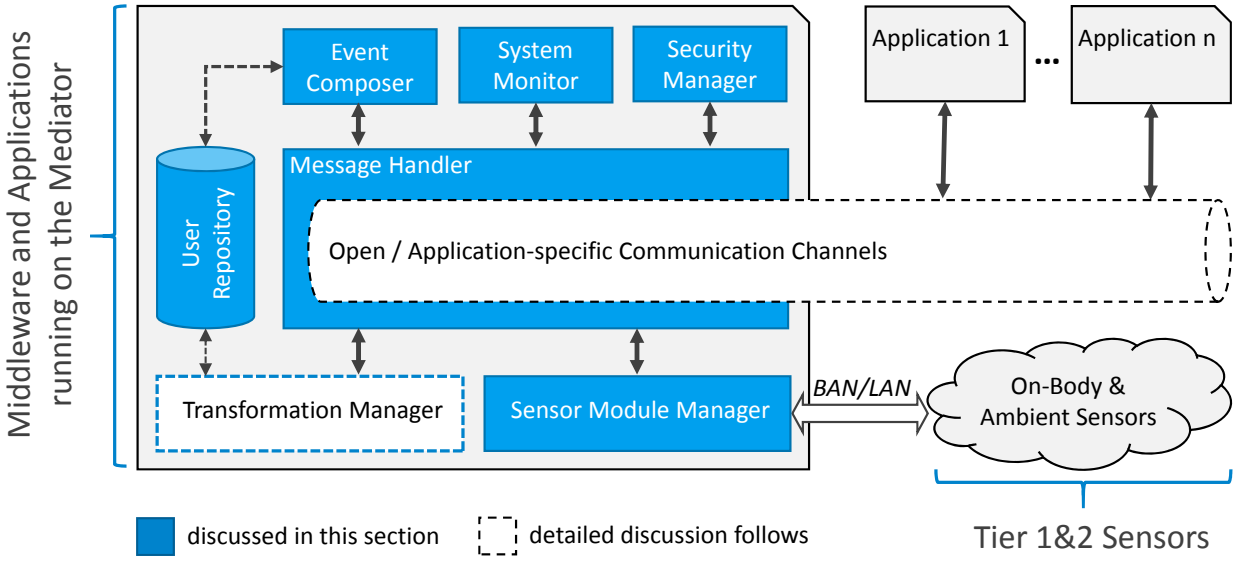


Figure 3.1: Event-driven middleware architecture consisting of an `Event Composer` for reasoning such as deriving sensor fidelity information, a `System Monitor` for keeping track of the overall system status, a `Security Manager` for encryption, verification, and managing permissions, a `User Repository` storing user specific data, a `Message Handler` for message passing, a `Transformation Manager` for converting sensor data to the desired representation, and a `Sensor Module Manager` that administers sensor connections.

technologies. In addition to this, the supported traditional phone calls are very helpful for health care and elderly applications: For example, instead of immediately sending an ambulance, a simple phone call could show that the person is fine and a sensor malfunction raised the alarm. Since smart phones are powerful, well-fitted for mediating among networks, and already carried by many users, we chose this technology for the mediator device.

Architecture Overview

Figure 3.1 depicts the mediator's architecture consisting of several modules, communication channels, applications, and the communication with on-body and ambient sensors. Both the middleware and the applications are running on the mediator which connects to on-body and ambient sensors using BAN and LAN connections. Connections to remote services remaining on the third tier need to be established by the individual applications. The middleware does not provide a gateway to remote services because we believe that this strongly depends on the individual application and is not part of a BASN middleware. The blue modules shown in Figure 3.1 are described in detail within this section. For detailed descriptions of the Message Handler's individual communication channels and the Transformation Manager, we refer to the Sections 3.3, 3.4 and 3.5 respectively.

3.2.1 Message Handler

The **Message Handler** is the central communication component that acts as a broker. It consumes events arriving from modules and applications and publishes them in the corresponding event

channels. We distinguish between two operation modes: *open communication channels* which are based on event types and *application-specific channels* that establish private communication channels to applications that are connected to the middleware. For simple scenarios without any security and privacy requirements, the open channels described in Section 3.3 are an easy and fast to develop solution. For health care applications which usually require security and privacy features, the application-specific communication described in Section 3.4 should be chosen.

3.2.2 Sensor Module Manager

The **Sensor Module Manager** administers the connection to on-body and ambient sensors. It contains a repository of **Sensor Modules** that are started and stopped based on application requirements and it does the mapping between sensors and Sensor Modules. Sensor Modules implement generic or sensor-specific protocols for communicating with individual on-body and ambient sensors. Upon receiving a sensor reading, a corresponding event is injected to the system by sending it to the Message Handler. Sensor Modules abstract from individual sensors and the underlying network technology which allows replacing sensors and network technologies without touching the rest of the system. For a generic module which implements for instance a Bluetooth LE heart rate profile, sensors can be replaced without changing the Sensor Module. Secure communication among sensors and the mediator is handled by the Sensor Modules which implement the sensor-specific protocols.

In case of application-specific communication channels, Sensor Modules can provide a start/stop functionality in order to stop the sensor if there is no subscription to the corresponding event type(s) and vice versa. This reduces sensor communication and, thus, allocated resources.

3.2.3 Security Manager

The **Security Manager** provides information about permissions of applications and modules. It is used to define which application is allowed to subscribe to which event type and to validate subscribers. By providing encryption keys, the Security Manager allows establishing secure communication channels between applications and the middleware. Since the middleware is designed for smart phones, we can assume that applications and the Security Manager can easily connect to a public key infrastructure and download required encryption keys at run-time.

3.2.4 System Monitor

The **System Monitor** is responsible for monitoring the overall system. In case of a critical situation such as a low running battery, the System Monitor injects an alarm event and starts appropriate reactions. This reaction is very system-dependent. In our Android implementation, for instance, the reactions to low battery power are a notification to the user as well as an SMS text message to a predefined phone number. In order to detect a crashed System Monitor or a lacking connection to the network carrier which would hinder the detection and dissemination of further

problems, heartbeat messages containing status information are periodically sent to a server. This allows remote monitoring of the system's liveliness. Further monitored aspects of the system are: signal strength to the network carrier, memory and CPU usage, changing sensor and application constellations, crashing applications, and the middleware's workload among others.

3.2.5 User Repository

The **User Repository** stores user-specific data such as the age, emergency numbers, size, weight, and thresholds for vital signs. The values, especially the thresholds for vital signs such as blood pressure and heart rate, are entered by a physician or transmitted from a physician's management application. User specific data is important for some calculations performed by the Event Composer or within the Transformation Manager. Those calculations could be an estimate of the calories burnt based on heart rate, size, and weight information.

3.2.6 Event Composer

The **Event Composer** consumes sensor and application related events and provides extra functionality to the user and application developer. It identifies general situations on which the system has to react (e.g. critical vital signs) and creates a corresponding derived event. In order to detect critical vital signs, the Event Composer connects to the User Repository and reads the user-specific thresholds given by a physician. Furthermore, it detects inaccurate or invalid sensor data and emits events enriched with fidelity information as presented in [126] and in Section 4.1.6.

3.2.7 Transformation Manager

The **Transformation Manager** administers event transformations that convert events from one event type to another or derives new events based on (multiple) incoming events. The transformation from an ECG curve to heart rate events could be such an event transformation. Section 3.5 presents detailed information about the Transformation Manager and event transformations.

3.3 Open Communication Channels

Open communication channels are one of the two message passing approaches we developed for the Message Handler. Events are published in public event channels. Every application running on the system can register listeners to those channels and immediately receives upcoming events. Application developers benefit from the middleware's sensor management and the automated event delivery. In order to use body and ambient sensors, applications only need to start the middleware and register their listeners. The burden of implementing the sensor protocols and sensor management is removed the developer. Furthermore, some systems do not provide multiple applications to use the same network stack which results in single application systems due to exclusive sensor connections. This problem is addressed by having only the middleware access the

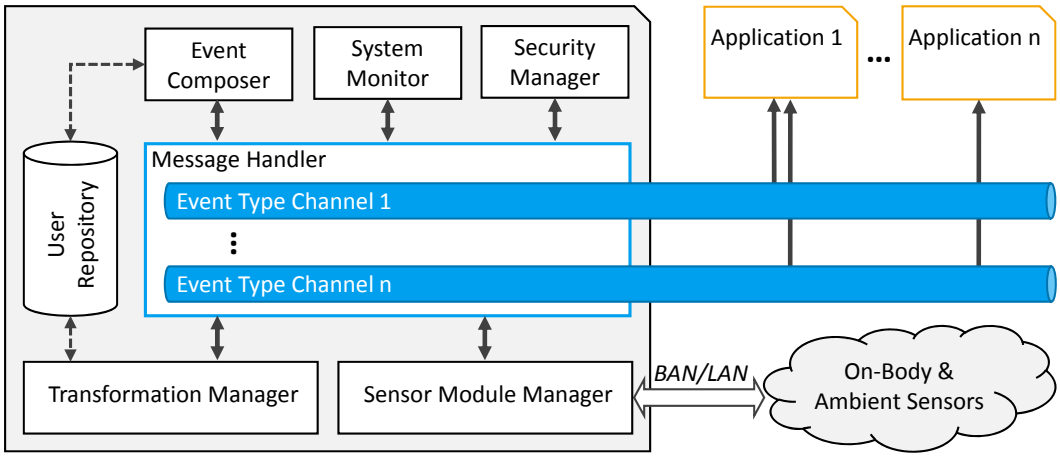


Figure 3.2: Middleware architecture with open event channels (highlighted in blue) implemented in the Message Handler. Events are distributed in multiple channels with respect to their event type. By subscribing to one or more open event channels, applications receive event notifications as shown by the arrows from the event channels to the individual applications.

sensors. By not requiring explicit subscriptions, open communication channels are straightforward to use.

3.3.1 Subscription

Figure 3.2 depicts the middleware's architecture with open communication channels. Each event type has its own communication channel to which applications subscribe. The actual subscription is done by registering an event listener to the desired channels. Explicit subscription messages are not required which simplifies the handling for application developers, but results in a lack of information about the subscribers on the middleware side. In order to provide extendability and a basis for event transformations, event types are structured in a tree. Figure 3.3 sketches the structure for a weight reading measured in kilograms. A new event type such as a weight reading

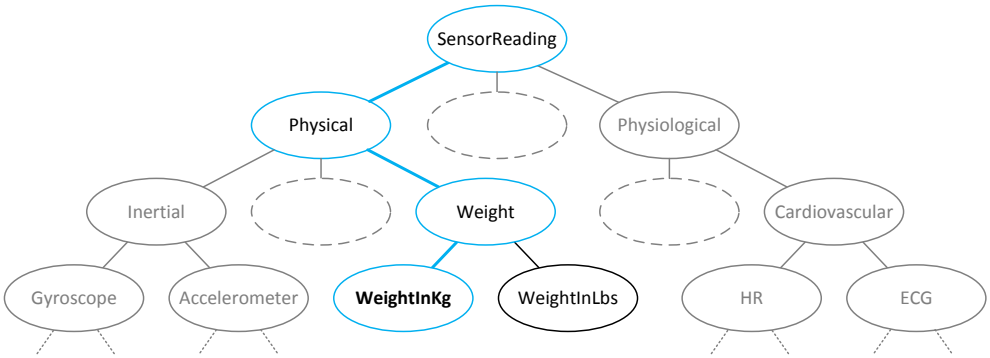


Figure 3.3: Event type tree illustrating an example of an event hierarchy. The hierarchy allows to introduce new event types and to subscribe to groups of event types.

in grams is introduced by adding it as a successor of `Weight` next to its siblings `WeightInKg` and `WeightInLbs`. In order to not require applications subscribing to each individual leaf of the event

type tree, events are published in individual event channels along that tree. The weight reading, for instance, is published in the following channels:

```
SensorReading.Physical.Weight.WeightInKg
SensorReading.Physical.Weight
SensorReading.Physical
SensorReading
```

Publishing events along that tree provides a more flexible way of subscribing to event types and groups of event types to the applications.

3.3.2 Notification

Upon a new sensor reading that was received by a Sensor Module, a corresponding sensor reading event is created and injected into the system. The event injection is done by publishing it to a single receiver channel for all event types. The distribution on basis of the event structure is then processed by the Message Handler. This allows to change the distribution method without changing individual components of the system or applications using the middleware. Ideally, events are distributed with respect to the event type tree as presented above. Applications that have listeners registered to a corresponding event type channel are then notified.

3.4 Application-specific Communication Channels

In the introduction, we highlighted the rapid development towards sensor networks that consist of on-body and ambient sensors which commenced a new paradigm in telemedicine and elderly care. By using the same sensor network for multiple applications, solutions can be adapted to individual requirements. On the other hand, having a variety of sensors and applications within one sensor network results in two major challenges that are not solved with the previously presented open communication channels:

- How to collect and deliver only the desired and permitted sensor readings to an application in order to manage information dissemination and save resources?
- How to deal with the diversity of sensors and data representations without requiring application developers to adapt to new sensors and changing sensor constellations?

The second question targets the problem of having for instance a heart rate monitoring application and an ECG sensor connected to the system: Although the ECG stream provides heart rate information, the heart rate application cannot use the readings unless it is extended to operating on ECG streams. In Section 3.5 we present an adaptive event transformation that makes this extension superfluous. The information basis for choosing a matching event transformation is provided by the Message Handler and will be discussed within this section.

First, we focus on the first question: how to collect and deliver only the desired and permitted information. Figure 3.4 depicts the architecture with application-specific communication channels.

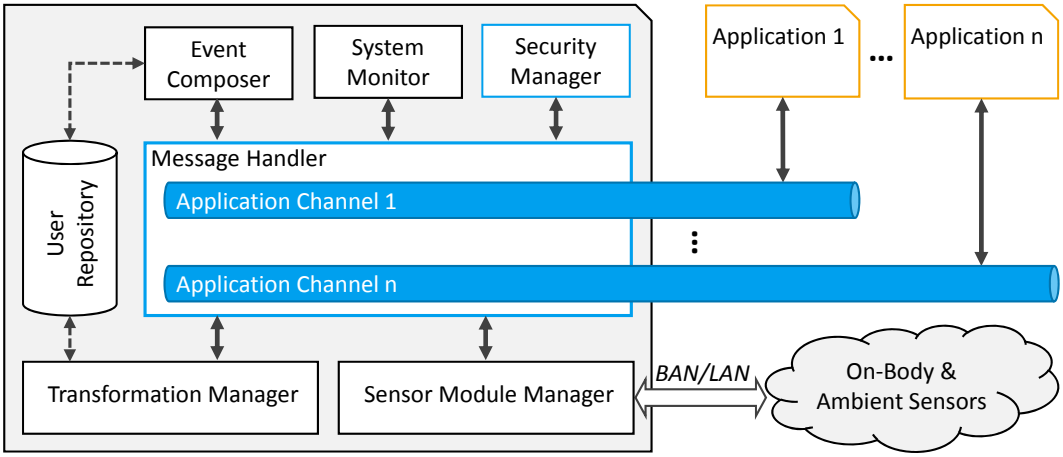


Figure 3.4: Middleware architecture with application-specific communication channels that allow secure communication and a tailor-made event delivery. Each application sends subscription and advertise messages to the Message Handler which, in turn, establishes the tailored communication channels.

Each application connected to the Message Handler communicates over its own channel. In order to establish a private channel, the Message Handler uses the Security Manager for checking the authenticity of an application and to handle a key exchange protocol if necessary. Having a secured communication channel between an application and the middleware prevents other applications from gathering classified information such as sensor readings with a higher confidence level and information about the connected sensors. In contrast to the open communication channels, this approach administers a lists of event producers and event consumers. Every part of the middleware, as well as, the applications can hold both roles of an event producer and event consumer. Subscription messages are sent to the Message Handler in order to express interest in a specific event type and advertisement messages are used for informing the Message Handler about produced event types. In the following, we will discuss advertisements, subscriptions, and event notifications in detail.

3.4.1 Advertisement

In order to announce future events to the system, applications and (sensor) modules send advertisement messages to the Message Handler. An advertisement consists of an `advertisement ID`, `event producer ID`, the `event type` that is advertised, start/stop capabilities of the event producer, and additional properties regarding the event producer (e.g. sampling rate). This information is stored and used for event subscriptions. If an event producer provides start/stop capabilities (e.g. an on-body sensor that can be controlled by the Sensor Module), the Message Handler decides whether to start or stop the sensor based on the event subscriptions. Stopping unnecessary sensors saves valuable energy. Event producers can revert their advertisements, in case they do not provide further events to the system, for instance, due to a disconnected sensor. If, as a result, this event type becomes unavailable to the system, subscribers are informed.

3.4.2 Subscription

In order to retrieve information from sensors as well as other applications and modules connected to the system, an application has to subscribe to event types. This is done by sending a subscription message to the Message Handler. Upon receiving a subscription, the Message Handler forwards the subscription's credentials to the Security Manager in order to verify that the application has permission to subscribe to the requested event type. If the request is granted, the Message Handler checks whether the requested event type was advertised. If not, a transformation request consisting of the requested event type as well as all advertised event types is sent to the Transformation Manager which in turn searches for a corresponding transformation (cp. Section 3.5). If a subscription succeeds, an acknowledgment including the event producer's availability is sent to the subscriber and corresponding events are disseminated over the application-specific channel. Event subscriptions are processed asynchronously, because requesting the Transformation Manager might take some time depending on the network connection to the Remote Transformation Repository. Granted but unsuccessful subscriptions (i.e., when no producer is available) are stored. As soon as an event producer for the requested event type becomes available, the subscribed application is informed and the event forwarding installed. Applications can unsubscribe from event types by sending a corresponding unsubscribe message. If there are no further consumers interested in an event type and the event producer supports start/stop functionality, a stop message is sent.

3.4.3 Notification

As soon as an event (e.g. a sensor reading) becomes available, the Message Handler disseminates it to subscribed consumers. Since events are disseminated over individual channels, they have to be sent multiple times. Depending on the number of subscribed applications and the event hierarchy, this could take more effort than sending an event over several open event channels. On the other hand, the advantages of application-specific communication channels are a reduced effort for filtering events on the application-side and security features which are indispensable for health care applications.

In case the mediator's operating system does not provide secure communication among applications, the Message Handler has to encrypt the events before delivery. For this, it requests the applications public key from the Security Manager and generates the cipher text. On the application's side, the message is decrypted by the application's private key. The same applies for the management messages such as advertisements of provided event types and subscriptions to required event types by using the middleware's public key.

3.5 Adaptive Event Transformation

Different applications have different requirements on data and its representations. The patient-specific risk factor monitoring in our motivational example operates on simple heart rate readings, whereas the extensive post-attack monitoring requires ECG information. By transforming the

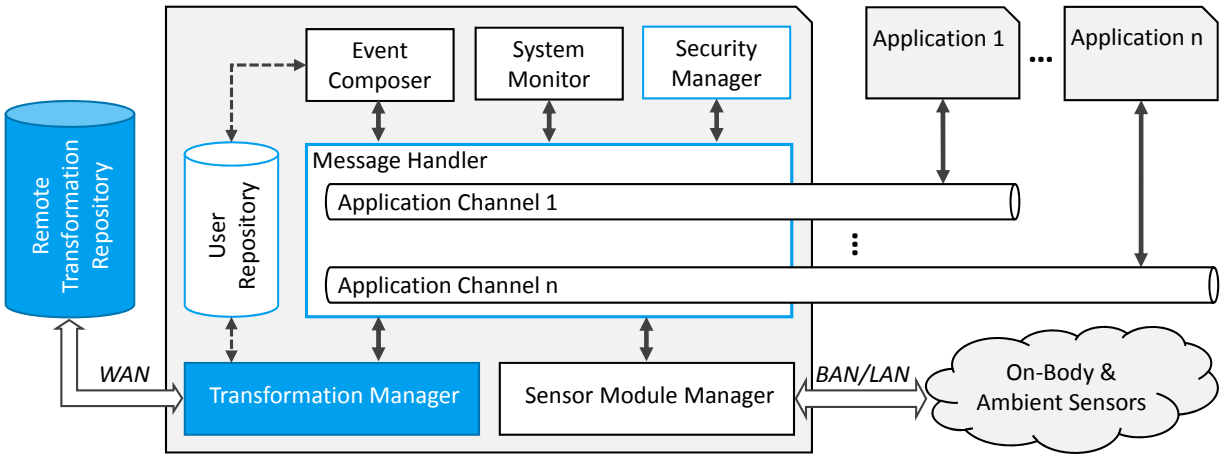


Figure 3.5: The middleware's Transformation Manager (blue) receives transformation requests from the Message Handler. Upon an incoming request, it checks whether it has a matching transformation locally installed. If this is not the case, the request is forwarded to the Remote Transformation Repository (blue) and a matching transformation downloaded at run-time.

ECG data into heart rate information instead of having an ECG and a heart rate sensor connected to the system, the patient has to carry fewer sensors. Furthermore, removing one sensor and, thus, reducing the amount of wireless transmissions results in a more energy efficient system (cp. Section 6.1). In addition to those benefits, replaced or future sensors might provide the same sensor information in another representation (e.g. data format, unit, granularity). As a result, replacing a sensor would require application developers to adapt their applications to the new sensor. By introducing a layer between sensors and applications that transforms the available information into the desired format, the amount of potential applications and sensors is increased and future applications and sensors are supported.

In this section, we present the Transformation Manager, an adaptive event transformation approach that transforms data into the desired data representation and adapts its own behavior with respect to the actual system configuration at run-time. Figure 3.5 highlights the corresponding middleware components: transformations are administered by the **Transformation Manager**. In case the local transformation repository does not provide a matching transformation, a **Remote Transformation Repository** is requested. We assume that the remote repository is a trusted entity that does not require additional security checks. In case of an untrusted repository, the Security Manager could be used for verification. Some transformations might rely on user-specific data such as a maximum heart rate for health alarms or the body weight and size for estimating the calories burnt. In order to avoid the creation of user-specific transformations, event transformations gain access to the user repository. It allows generic transformations to adapt their behavior to the user-specific needs and thresholds.

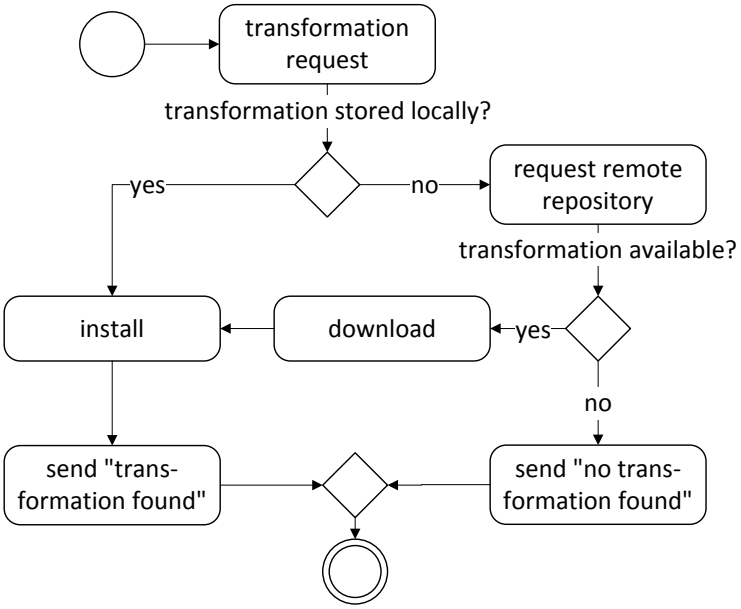


Figure 3.6: Processing an event transformation request: If there is no transformation from the advertised event type(s) to the required event type stored locally, the request is forwarded to a Remote Transformation Repository allowing the system to adapt to new requirements.

3.5.1 Transformation Request

In case the Message Handler receives an event subscription to an event type which is not advertised, it sends an event transformation request to the Transformation Manager. Such a request consists of a *request ID*, the *requested event type*, and a list of *advertised event types* which contains all event types that are currently advertised. Based on this information, the Transformation Manager checks whether there is a transformation from one or more advertised event types to the requested event type already stored or whether it needs to forward this request to a remote repository. Figure 3.6 depicts this process in detail. In case of a successful local or remote request, the corresponding transformation is started and a notification message is sent. Downloading and initialization of new event transformations and, thus, of new system functionality is done at run-time.

3.5.2 Transformation Life-cycle

Non-required transformations are stopped and do not require any CPU cycles and main memory. As soon as a transformation is required, the Transformation Manager triggers the start of this transformation. When starting, a transformation subscribes to the event type(s) it requires for its transformation. In case of an untrusted remote repository, the subscriptions can be treated as application subscriptions which requires a security check. After a successful subscription, the new event type is advertised including the obligatory start/stop functionality and the transformation starts working triggered by incoming events. If a transformation is stopped, it unsubscribes and reverts its advertisement.

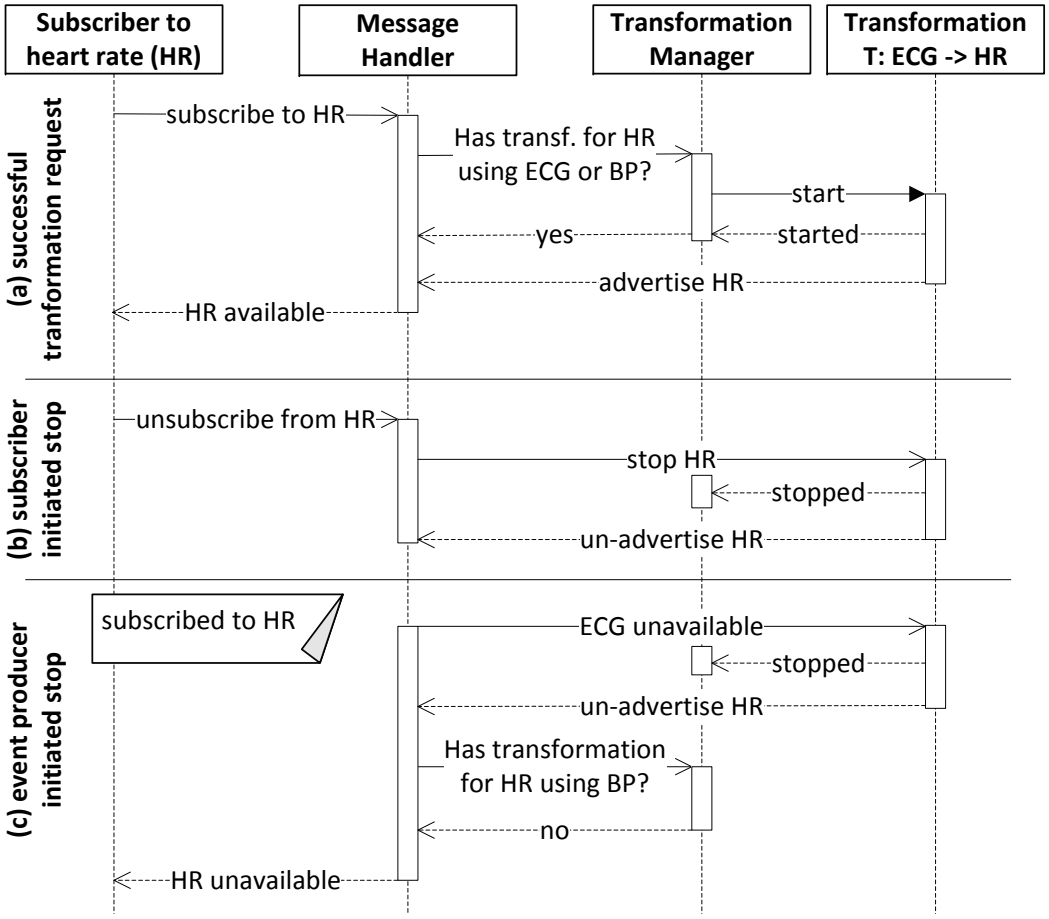


Figure 3.7: Sequence diagram illustrating the communication among software components for starting and stopping an event transformation: (a) installing an event transformation for transforming events of type ECG to type HR, (b) the event transformation is stopped because the last subscriber to HR unsubscribed, (c) the event transformation is stopped because the event producer for events of type ECG is no longer available.

Figure 3.7 depicts the communication for starting and stopping a transformation in detail. Communication with the remote repository is omitted for clarity. Case (a) shows the communication for starting a transformation: an application subscribes to the event type heart rate (HR). Since there are only event producers for ECG and blood pressure (BP) readings available, the Message Handler sends a transformation request to the Transformation Manager. Because the Transformation Manager has a transformation $T_{ECG \rightarrow HR}$ stored, it starts the transformation and sends an acknowledge message to the Message Handler which in turn informs the application. Furthermore, $T_{ECG \rightarrow HR}$ advertises events of type HR. In case (b), the application unsubscribes from HR events. Since it was the only application subscribed to HR events, the Message Handler sends a stop command to the transformation which has to support the start/stop functionality. The transformation stops, informs the Transformation Manager, unsubscribes, and reverts its advertisement of HR events. Case (c) depicts how the system reacts on a required event producer becoming unavailable. In this example, the ECG sensor becomes unavailable which causes a stop of the transformation $T_{ECG \rightarrow HR}$ and, hence, HR events become unavailable. Since there are still subscriptions to HR events, the Message Handler invokes a new transformation request containing only the advertised BP events.

Because there are no matching transformations available, the request remains unsuccessful and, thus, the Message Handler informs the applications about the unavailable HR events.

3.5.3 Remote Repository

Every new event producer and event consumer potentially leads to the need for a new event transformation. With an increasing amount of applications and sensor devices available, the number of possible transformations increases rapidly. Having all potential transformations installed on a mediator for on-body and ambient sensor networks would overload most devices. Therefore, we decided for a **Remote Transformation Repository** that enables the mediator to keep only transformations which are required and to download new transformations in case the requirements change. This way, resources are saved and an adaptive system is provided. If the device runs out of memory, transformations that were not activated for a while, are deleted. Having transformations running within the sensor network instead of a remote entity brings two main advantages: a) the system can operate without an Internet (or similar) connection and b) sensitive user information remains within the sensor network.

For requesting a new transformation, the Transformation Manager connects to a remote repository. We assume that this is done via a secured network connection and after a successful authentication process in order to prevent the installation of malicious transformations. After the connection is established, the Message Handler's transformation request is encoded and sent to the Remote Transformation Repository. If the repository finds a match, it provides the corresponding transformation. If there are several potential transformations available, a cost function determines the best transformation for the requesting mediator. After the transformation is downloaded, the Transformation Manager has to provide an environment for executing loaded code for starting the transformation. In our implementation (cp. Section 4.4), we decided for an OSGi framework.

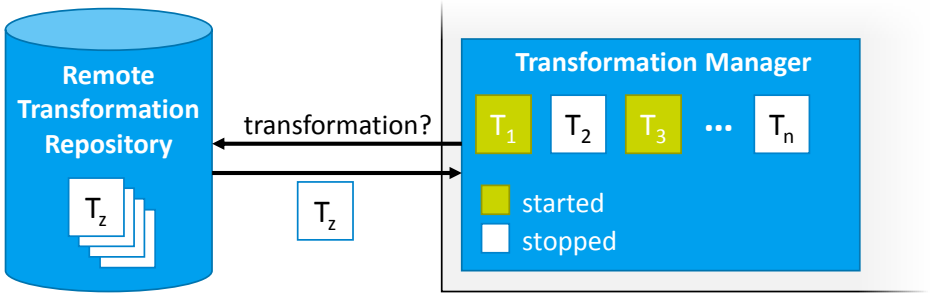


Figure 3.8: Transformation Manager with started and stopped transformations. A priori unknown transformations are downloaded from a Remote Transformation Repository.

Figure 3.8 depicts the local Transformation Manager consisting of started and stopped transformations as well as a remote repository providing all potential transformations. Since transformations are allowed to consume multiple event types, they are not restricted to simple event type conversions. In our implementation we developed for instance a transformation that detects a critical heart rate based on heart rate and activity events. In case an application requires such an alarm,

Requirement	Open Channels	Application-specific
Multi-Application Support		
Multiple-Application Support		
Inter-Sensor Compatibility		
Swift-Adaptations		
Security & Privacy		
Networking Requirements		
Protocol Independence		
Ad-hoc Connections		
Bridging/Bundling Networks		
Mediator Requirements		
Central Reasoning		
Modular Architecture		
Self-Configuration		
Sufficient Local Resources		
Connectivity		

Table 3.1: Comparison of the requirements on a middleware for medical body and ambient sensor networks to the properties supported by our middleware architecture. A distinction is made between open communication channels and application-specific channels in combination with event transformations.

this functionality can be added without user interaction or interrupting the system. The capability of reloading missing functionality enables an adaptive system behavior.

3.6 Summary

In this chapter, we presented an event-driven middleware architecture for body and ambient sensor networks running on a smart phone as a mediator device. After an analysis of requirements on BASN middleware for medical applications, the generic middleware architecture was presented. The system's functionality is implemented in modules that mainly communicate via a Message Handler. We presented two approaches for this Message Handler: open communication channels and application-specific channels. Table 3.1 compares both approaches to the requirements. By introducing advertisements and subscriptions for the application-specific channels, the system

keeps track of all event producers and consumers. In case of a subscription without a fitting advertisement, the Transformation Manager which we introduced in the last section is queried in order to find a matching transformation.

Multiple applications are supported by both systems since the sensor readings are forwarded to all interested (and granted) applications. Using events as an abstract data representation provides already a good inter-sensor compatibility. Nevertheless, by introducing event transformations that convert readings from one type to another, the application-specific approach provides a broader inter-sensor compatibility. The same applies for the swift adaptations. Changing the system at run-time is supported by both approaches, but having transformations can minimize the use of sensors and calculations. With open communication channels confidential information can be accessed by any application and, thus, security and privacy within the mediator is not provided. The integration of a Security Manager and encrypted communication channels in the second approach allows control over the data access and a secure communication between applications and the middleware.

The networking requirements are met by both approaches. Sensor modules that abstract from the individual sensors and protocols provide protocol independence. The event-driven system design inherently supports ad-hoc connections and both Message Handlers bridge and bundle different networks implemented by different modules.

By providing sensor information from on-body and ambient sensors at one point, the Message Handler, i.e., the mediator requirement of central reasoning is provided by both approaches. Changing the middleware's functionality is done by changing its modules. Both approaches support self-configuration by adapting their behavior with respect to the connected sensors. In addition to connected sensors, adaptive event transformations allow the adaptation based on subscriptions and advertisements and downloading functionality at run-time. Therefore, the second approach provides a higher degree of self-configuration. By using a smart phone as the mediator device, both sufficient local resources as well as a wide range of networking capabilities are provided.

4 Middleware Implementation

This chapter presents the Android implementation of our middleware approach. It serves for evaluating the overall system performance which is presented in Chapter 6. The system's capabilities and feasibility as a mediator are emphasized by the different body and ambient sensor network applications that are built upon our middleware implementation. Chapter 5 describes and evaluates the corresponding applications. The description of the middleware implementation in this chapter follows the structure of the design description in Chapter 3 including the implementation of the generic architecture, the open and application-specific communication channels and the adaptive event transformation. In addition to this, features such as calculating the fidelity of sensor readings, event transformations of different complexity, indoor localization, and collecting sensor readings from unknown environments are presented in this chapter. We describe the implementation of Sensor Modules for different Bluetooth and Wi-Fi sensors. Parts of this chapter, especially the generic architecture, the communication channels, and the event transformation, are based on work we published in [126, 127, 128]. The integration of Wi-Fi sensors was done in collaboration with Benedikt Ostermaier from the Distributed Systems Group[1] at the ETH Zurich, led by Professor Friedemann Mattern. For the indoor localization and the collection of sensor readings from unknown environments, we collaborated with Simon Mayer who works in the Distributed Systems Group as well.

4.1 Overview

The implementation of our middleware for body and ambient sensor networks, called **myHealthHub**, is built upon Android OS. We decided in favor of Android because of its good Bluetooth support. Most commercially available bio-sensors (e.g. for heart rate, ECG, and blood pressure) as well as most smart phones support the Bluetooth protocol. With the goal of focusing on off-the-shelf hardware, we decided for a BSN consisting of Android devices and Bluetooth bio-sensors. The middleware itself operates on Android 2.1 or higher which means it is supported by 99.9% of the Android devices, with respect to a Google analysis[2] from August 1, 2013. Nevertheless, for security reasons it is recommended to use an Android device running at least Ice Cream Sandwich (4.0.x). Android versions before Ice Cream Sandwich do not implement the `setPackage()` method correctly which allows eavesdropping on explicit Intents (cp. the Android security analysis in Section 2.3.3). The middleware was tested on several Android devices such as: the Motorola Milestone as one of the first Android devices, the Motorola RAZR i using an Intel processor instead of the mostly used ARM processors, HTC Desire C and HTC One V as inexpensive devices, Google Nexus 4 and HTC One X+ as high-end devices, the Samsung Galaxy 10.1 N as a tablet device, and

[1] Website of the ETH Distributed Systems Group: `http://www.vs.inf.ethz.ch/`

[2] `http://developer.android.com/about/dashboards/index.html` (accessed 30-08-2013)

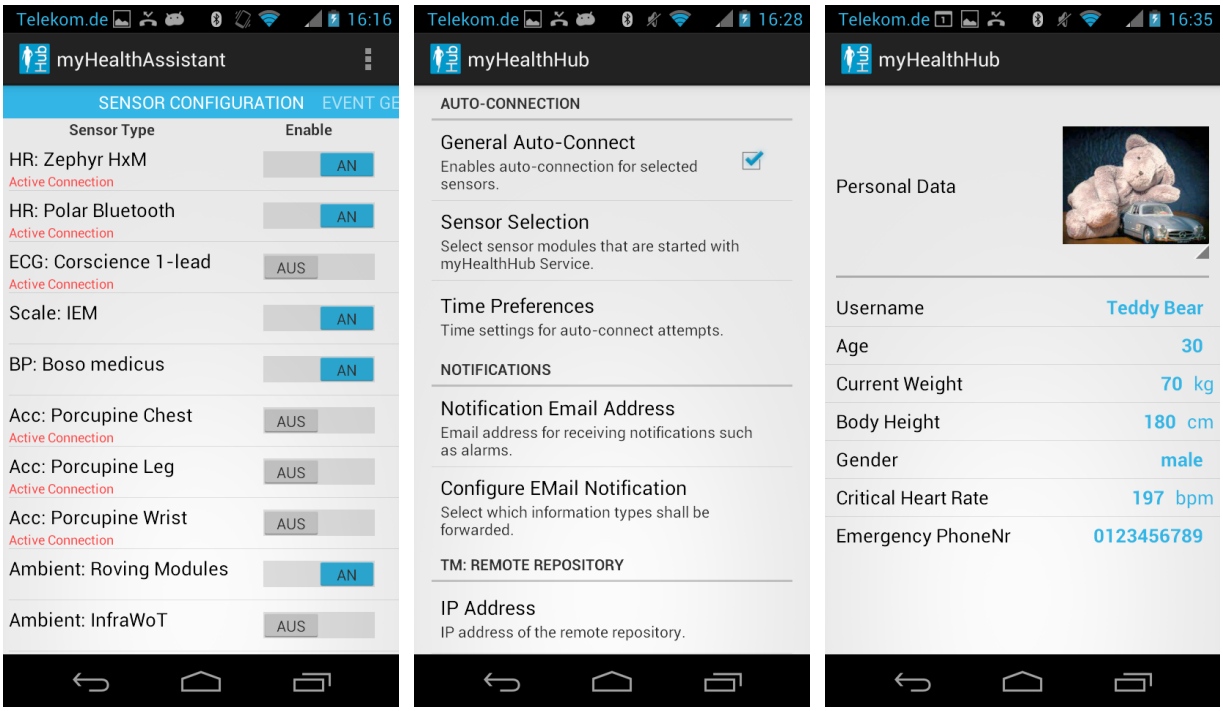


Figure 4.1: Middleware configuration: (a) enable/disable Sensor Modules, (b) configure general settings, and (c) configure user-specific data

the Google Glass as a new device type. All those devices run myHealthHub out-of-the-box and do not require any modifications in the middleware.

The middleware is implemented as an Android Remote Service providing an Android Interface Definition Language (AIDL) interface. For clients, launching and binding a remote service is similar to a local application service. Clients need to implement an instance of the `ServiceConnection` class providing the methods: `onServiceConnected()` and `onServiceDisconnected()`. The methods help to determine whether myHealthHub is started and for detecting the termination of the middleware. As for local services, the Android system manages the life-cycle of myHealthHub avoiding that applications start multiple instances or that an application stops the services although another application is still connected to it. For starting myHealthHub as a remote service, a corresponding Intent needs to be created and the service binding triggered. In the following example, `myHealthHubConnection` implements an instance of the required `ServiceConnection`:

Listing 4.1: Starting and binding myHealthHub as an Android remote service.

```
1 Intent myHealthHubIntent = new Intent(IMyHealthHubRemoteService.class.getName());
2 bindService(myHealthHubIntent, myHealthHubConnection, Context.BIND_AUTO_CREATE);
```

For unbinding, the client needs to invoke the method `unbindService(myHealthHubConnection)`. In addition to the remote service, myHealthHub implements several user interfaces for configuring and testing the middleware. Examples are shown in Figure 4.1: In the left screenshot, Sensor Modules can be enabled and disabled and connections to the sensors manually triggered. It helps to test new configurations and to provide a swift adaptation to new requirements. The list is

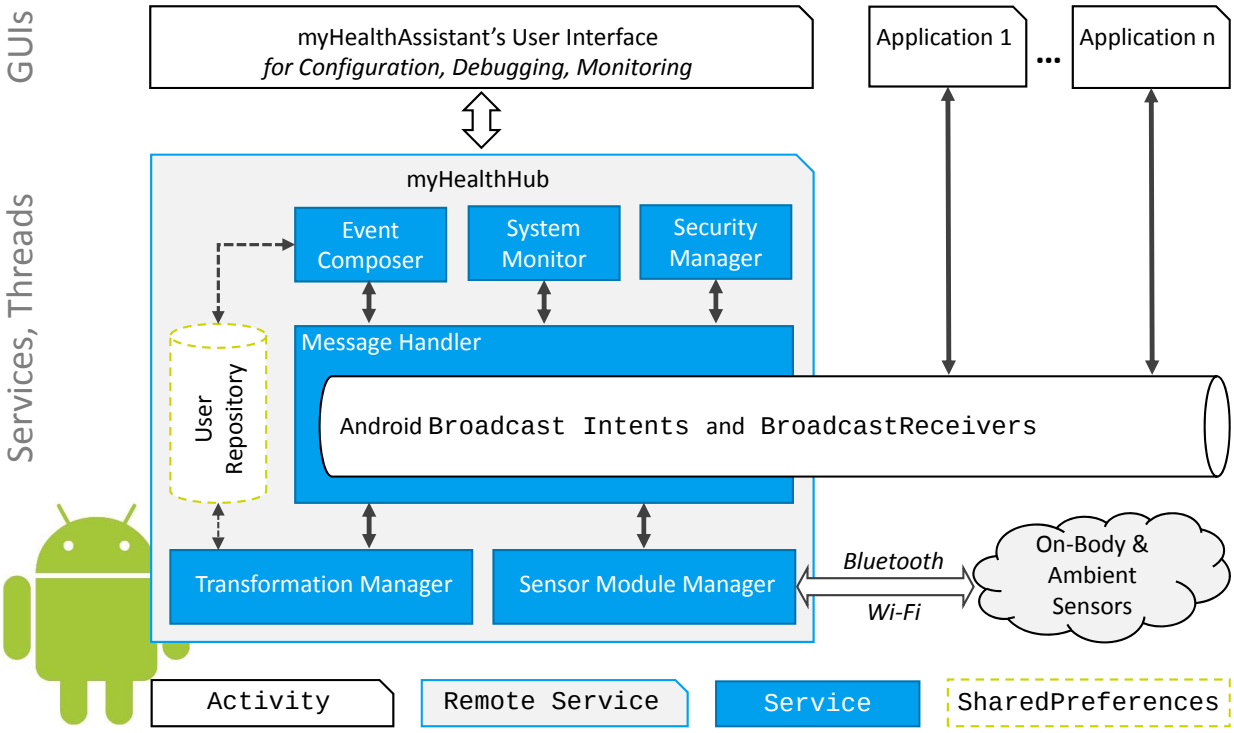


Figure 4.2: Implementation of the event-driven middleware for body and ambient sensor networks as Android Services. The implemented modules correspond to the modules presented in Chapter 3. Bluetooth and Wi-Fi is used for communicating with on-body and ambient sensors and Android Activities provide a user interface for configuring the middleware. Android `Broadcast Intents` and `Broadcast Receivers` are used for inter-application communication.

dynamically generated based on the available Sensor Modules. The screenshot in the middle depicts the general configuration screen. It allows, for instance, to select automatically activated Sensor Modules, to set time preferences for re-connections, to configure the e-mail notifications (e.g. for alarms and health reports), and to configure the IP address and port number of the Remote Transformation Repository. On the right, a screenshot of a mask for changing patient-specific data in the User Repository is shown.

Figure 4.2 depicts an overview of the Android implementation. It reflects the structure of the middleware presented in the previous chapter. Furthermore, it highlights the different Android components used for implementing each individual part of the architecture. In the upper part, myHealthAssistant's user interface for configuring and debugging myHealthHub together with Android Activities representing applications which are connected to the middleware are shown. MyHealthHub itself is wrapped into an Android Remote Service which allows foreign applications to start and stop it. Most parts of the middleware are implemented as Android Services except for the User Repository which makes use of Android's `SharedPreferences`. In addition to the services, there a several threads invoked within the modules for performing long-running tasks such as downloading a new Event Transformation. Details on the individual modules are given in the following sections.

4.1.1 Message Handler

The Message Handler is either implemented as an Android Service providing open communication channels or as an Android Service providing application-specific channels. In both cases, it is started by the myHealthHub Remote Service and provides the same channel for incoming events. Details on the actual implementations for both message passing mechanisms are given in Section 4.2 and Section 4.3 respectively.

4.1.2 Sensor Module Manager

The Sensor Module Manager is implemented as an Android Service and administers all installed Sensor Modules. It automatically enables modules that are configured for auto-start (cp. Figure 4.1(b)) and reacts on user input received from the Sensor Module configuration screen (cp. Figure 4.1(a)). Communication between the user interface and the Sensor Module Manager is done via a `ServiceConnection` and an `IBinder`. We refer to the Android documentation for details on inter-component communication.

In addition to being enabled and disabled, Sensor Modules need to be started and stopped based on subscriptions (cp. Section 3.4). In case a required sensor does not automatically connect to the mediator, the sensor's availability needs to be checked and a connection established. Therefore, we distinguish between *Active Sensor Modules* and *Passive Sensor Modules*: active modules explicitly need to connect to a sensor whereas passive modules open the counter-part (e.g. a Bluetooth socket) of an active sensor that automatically connects to the middleware upon an event (e.g. a measurement taken). The Zephyr HxM heart rate sensor is a passive sensor to which our system actively needs to connect to and, thus, we developed an active Sensor Module for this sensor. The CorScience Boso BT Prestige blood pressure sensor however is an example of an active sensor that requires a passive module with an open Bluetooth SPP[3] socket.

The state diagram in Figure 4.3 shows the states that a Sensor Module can take within the Sensor Module Manager as well as operations the manager performs based on internal and external events: As soon as a Sensor Module is initialized, it is added to the list of started modules. The initialization and termination of modules is exclusively triggered by the middleware and not by applications. Upon a corresponding *startProducer* event, the Sensor Module Manger queries whether a module is active or passive. For passive modules a `start()` method is invoked and the module starts waiting for incoming readings. Active modules are added to a list of active modules. For every active module in this list the `start()` method is periodically triggered in order to probe for an available sensor. Upon availability, the sensor connection is established and readings are received. In case of an incoming *stopProducer* event, the `stop()` method of both module types is invoked. In case of an active module, it is additionally removed from the list of activated active modules. Unless the `stopSensorModule()` command is triggered, both types of Sensor Modules remain in the list of started modules which means that they are ready for being re-activated again.

[3] SPP: Serial Port Profile for Bluetooth. It emulates a serial connection.

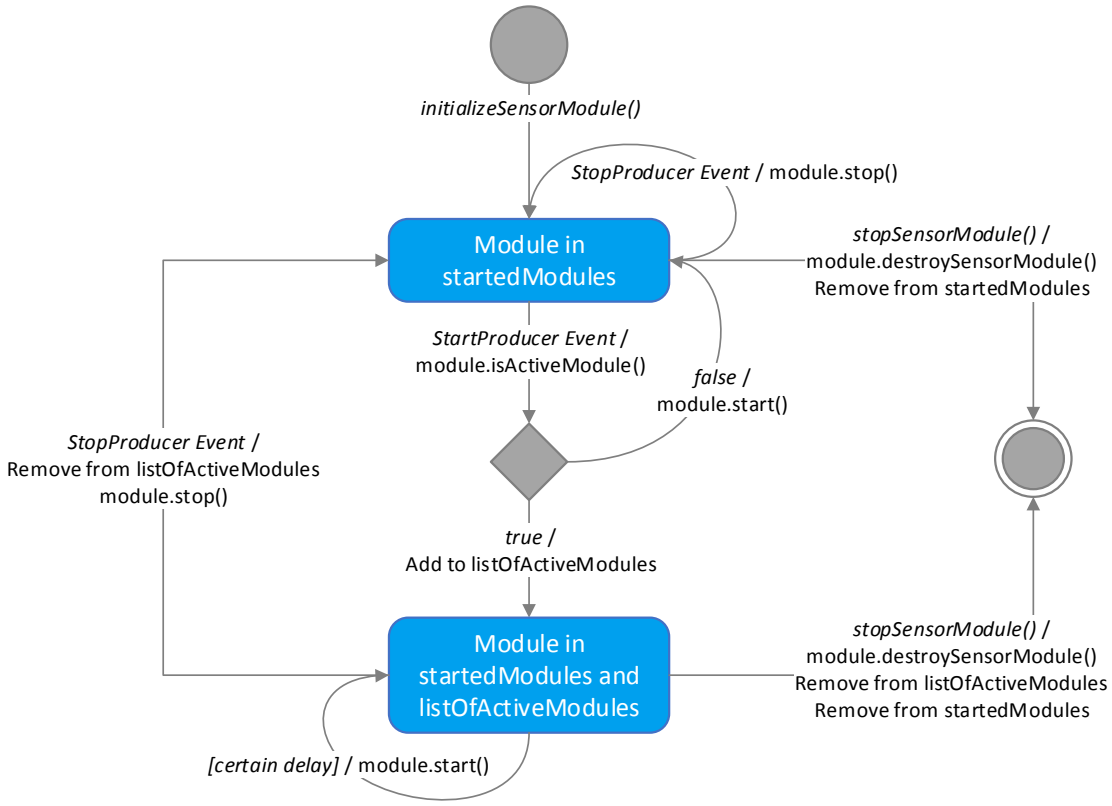


Figure 4.3: State diagram showing the different states a Sensor Module can have within the Sensor Module Manager. Furthermore, state change conditions as well as methods invoked at the module are depicted.

For clarification, all features regarding starting and stopping modules with respect to event subscriptions and (un-)advertisements of event producers only apply to the application-specific communication approach. The open communication approach only distinguishes between initialized and destroyed, and active and passive modules.

Figure 4.4 depicts the states a passive Sensor Module can take and the reaction to method invocations triggered from the Sensor Module Manager. After initialization, the produced event type is advertised and a *passive state* is taken. As a result, the Message Handler is informed about the new event producer which allows it to decide whether to start or stop the producer based on event subscriptions. If the sensor reading type is required, the Message Handler sends a *startProducer* event. Upon this event, the `start()` method is invoked and the module's network port is opened (*waiting for sensor connection* state). If a corresponding active sensor starts a connection to the module, this active connection is announced and the *waiting for readings* state taken. Incoming readings are processed and accordingly published as sensor reading events. Triggered by a sensor disconnection, the module returns to the waiting state and announces the disconnection. If the provided reading type is not required anymore, the `stop()` method is invoked and the *passive state* re-taken. Upon an `destroySensorModule()` invocation, the advertisement is revoked and the module destroyed. We have developed two types of Sensor Modules, Bluetooth modules and Wi-Fi modules, which are described in the following paragraphs.

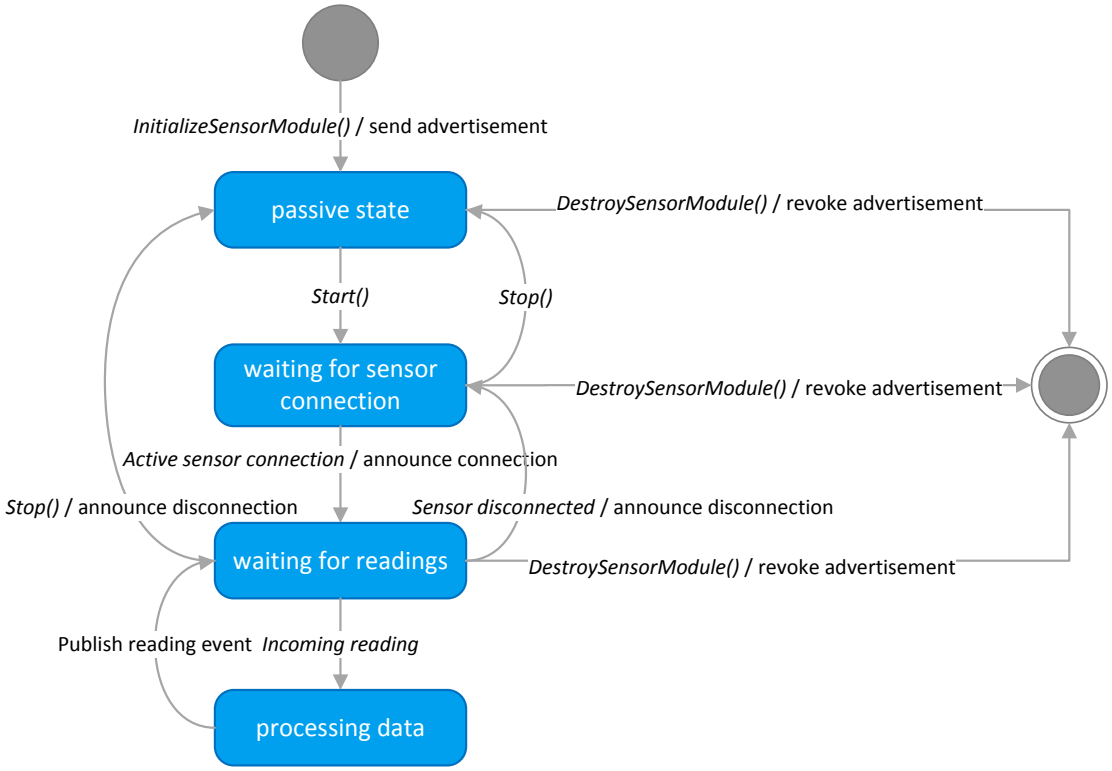


Figure 4.4: State diagram for a passive Sensor Module. As soon as the `start` method is invoked, a network port for incoming sensor readings is opened and corresponding readings injected into the system.

Bluetooth Sensor Module

Our Bluetooth module implementations are used for both on-body and ambient sensors. We provide two abstract classes for secure and insecure Bluetooth communication that allow a swift implementation of Bluetooth Sensor Modules. By extending one of the two classes, only the sensor description, module ID, and the method `deliverPacket(byte[] packet, int length)` need to be provided. New sensor readings are delivered as a byte-Array. The actual implementation only needs to create a sensor reading event based on the byte-Array and invoke the method `sendSensorReading(Event readingEvent)`. The new sensor reading is then sent to the Message Handler. We have implemented modules for the following sensors: CorScience Boso BT Prestige stationary blood pressure sensor, CorScience CorBELT 1-lead ECG sensor, HedgeHog accelerometer from ESS group at TU Darmstadt, IEM Stabil-O-Graph scale, Omron Bluetooth stationary blood pressure sensor, Polar Bluetooth and Polar H7 heart rate sensors, Porcupine accelerometer from ESS group, and the Zephyr HxM heart rate sensor. The HedgeHog sensors are used for both on-body as well as ambient sensing. The scale and both blood pressure sensors are active sensors that require passive Sensor Modules. For swift module development, a simple flag in the abstract Bluetooth module's constructor allows to switch between active and passive module behavior. This feature is very helpful in order to program devices, because for programming an active connection is usually required. In addition to this, the CorBELT ECG sensor works only with an insecure Bluetooth connection. As a result, two abstract classes for both secure and insecure Bluetooth communication are provided that support both an active and a passive behavior.

Wi-Fi Sensor Module

In addition to the Bluetooth modules, we added active and passive Wi-Fi modules for ambient sensing to the system. The active module queries a central server for environmental sensors such as temperature and light sensors. This module allows the integration in an existing ambient sensing environment as described in Section 4.5. Details on the module are given in that section.

In collaboration with the Distributed Systems Group at ETH Zurich, we developed a passive module that gathers readings from active Wi-Fi sensors. The work is based on modified Wi-Fi modules that run a small web server and allow attaching binary sensors. Detailed information about the Wi-Fi modules is given in [105]. We attached three types of binary sensors to the modules: passive infrared (PIR) sensors, ball in tube sensors, and reed switches. Upon a sensor event, the Wi-Fi module generates a JSON object which is sent to the passive Sensor Module using HTTP. For receiving sensor readings, myHealthHub's passive Wi-Fi module opens an HTTP stack with a predefined port number. Incoming readings are converted into reading events and injected into the system. In order to provide a flexible deployment of the Wi-Fi sensors, the Sensor Module parses an XML document including the sensor type and location per MAC address which avoids costly re-programming of sensor nodes if their positions change.

4.1.3 Security Manager

For enabling the Message Handler to decide whether an application is allowed to subscribe to a specific event type, the Security Manager provides a `hasPermission(String packageName, String eventType)` method. It returns `true` if the application has the permission, otherwise it returns `false`. Since events are specifically sent to Android package names, spoofing another application's package name does not enable an intruder to gather confidential information. Events and the response to a spoofed request are sent to the package name transmitted with the request. Although attackers are not able to gather confidential information, they still can run a Denial-of-Service (DoS) attack against the middleware and approved applications by sending a high number of requests to the system. A DoS attack against approved applications can be avoided by checking the sender's identity before processing the request, but it does not avoid DoS attacks against the middleware.

The application-specific communication channels described in Section 4.3 make use of security mechanisms provided by the Android system. Since Android already provides secure inter-application communication as discussed in Section 2.3.3, message encryption is not required and, therefore, the Security Manager in our implementation does not need to provide public and private encryption keys.

4.1.4 System Monitor

The System Monitor keeps track of the smart phone's position and battery level. As soon as the Android system broadcasts an `ACTION_BATTERY_LOW` Intent indicating that the phone is running out of battery, an e-mail as well as an SMS text message including the last GPS coordinates are sent. It allows interventions to a low running battery before the system dies.

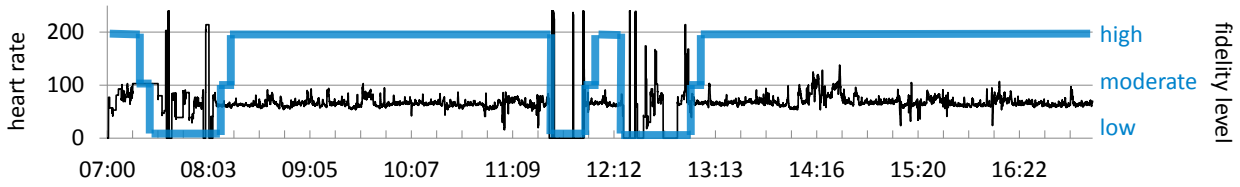


Figure 4.5: Heart rate readings (black line) transmitted by the Zephyr HxM pulse sensor showing some sensing artifacts. By adding fidelity information to the heart rate readings (blue line), applications do not need to filter the readings on their own.

4.1.5 User Repository

The User Repository builds upon the Android `SharedPreferences`. It allows each Android component, especially the Transformation Manager with its Event Transformations and the Event Composer module, to access user-specific data as key value pairs. The user interface for changing user-specific data is shown in Figure 4.1(c) on page 58. In addition to the manual input, an automated approach that sends user information from the physician's PC could be added. Furthermore, the user information could be updated based on incoming sensor reading events such as body weight readings.

4.1.6 Event Composer

The Event Composer provides additional services to patients, physicians, and application developers. We have implemented three services operating on heart rate readings as an illustration of which services the Event Composer can provide:

- **Simple Pulse Monitoring.** Incoming heart rate readings are compared to the thresholds set in the User Repository. If the heart rate is above a maximum or below a minimum heart rate, an alarm is raised. The alarm is sent as an SMS text message or an email or both, depending on the settings form the configuration screen shown in Figure 4.1(b) on page 58.

- **Activity-based Pulse Monitoring.** Instead of checking whether the current heart rate is within a fix band of maximum and minimum heart rate values, the activity-based monitoring takes the actual user activity into account. In Section 5.1, we will present an application that distinguishes between five daily activities: standing, sitting, walking, running, and cycling. Each activity has its own range of heart rate values. If the system detects a heart rate outside the valid range, an alarm is raised.

- **Heart Rate Fidelity Information.** This service enriches heart rate readings with fidelity information allowing applications to decide whether to rely or to ignore an incoming reading. Figure 4.5 depicts heart rate values gathered from the Zephyr HxM heart rate sensor (black line) showing some sensing artifacts. In [126], we distinguish between four types of sensing artifacts due to a too dry skin-electrode contact: 1) extreme outliers, 2) steady heart rate values, 3) a heart rate of zero, and 4) the sensor switching off. Moistening the electrode

restores the desired functionality. Based on those observations, we defined three fidelity levels: high, moderate, and low. High fidelity levels are indicated when no artifacts in the heart rate curve were detected. As soon as a peak is detected or if the heart rate drops to zero within a short period of time (cp. Figure 4.5), the corresponding heart rate event is classified as low since it is usually caused by a too dry skin-electrode contact (blue line at the bottom). If no peaks are detected but the heart rate stays at a constant value for several seconds, the fidelity level decreases from high to moderate (blue line in the middle). Having steady heart rate values does not necessarily mean that the heart rate is not detected correctly, but it might be an indicator since the heart rate usually fluctuates within limited range. If it still remains at a steady value, the fidelity level is finally decreased to low until the heart rate value changes again. If an application developer does not want to filter the heart rate readings she can simply subscribe to the enriched readings.

For a broader overview of services the Event Composer can provide, we refer to our discussion in [126]. A particular focus of that paper is on determining changes in the sensor reading fidelity caused by a) contact-based fluctuations (as shown in Figure 4.5), b) incorrect sensing context (e.g. taking a blood pressure measurement right after high physical activity), and c) wrong mapping of environmental sensors (e.g. someone else is using the scale).

The Event Composer was introduced to the system before we had developed the Transformation Manager. Since Event Transformations are also allowed to access user-specific information, they can provide the same services as the Event Composer. Furthermore, the Transformation Manager provides downloading and installing new services at run-time. Therefore, when using application-specific communication channels and Event Transformations the Event Composer is obsolete. For the open communication channels it is still required.

4.1.7 Transformation Manager

The Transformation Manager is implemented as an Android Service that provides the administration of Event Transformations on one hand and a run-time environment for OSGi bundles on the other hand. Detailed information about the Transformation Manager and the Event Transformations is given in the dedicated Section 4.4 after presenting the communication channels.

4.2 Open Communication Channels

Before presenting details on the implementation, we will argue why we decided for Android Broadcast Receivers as the underlying message passing system (cp. [128]). For providing a user-friendly system, communication among the middleware and applications should not involve decisions from the user. In the Android operating system, there are three base mechanisms for inter-application communication that do not require user interaction [31]:

- *Services* expose their interface to other Android applications by using AIDL (Android Interface Definition Language). This allows applications to invoke methods from other applications and, thus, provides inter-process communication.

- *Content Providers* allow access to a structured set of data. Applications can query this data set by addressing an application-defined URI.
- *Broadcast Receivers* receive Android Intents sent from any Android application. Those Intents can piggyback any data which implements the Android Parcelable object. Before sending an Intent to another application its content needs to be described. If the Android IntentFilter of a Broadcast Receiver matches to an Intent's description, it is received.

Since Android Services are built upon a remote procedure call (RPC) communication paradigm which has some drawbacks for publish/subscribe messaging [43] and Android Content Provider requires a query-based communication paradigm, they both do not fit very well to the event-driven sensor network communication. Android Broadcast Receivers support event-driven asynchronous communication across multiple channels and they allow piggybacking any data as long as it implements Android Parcelable objects. Therefore, we decided for Broadcast Receivers as the underlying communication mechanism.

In the approach of open communication channels, the Message Handler publishes events in channels related to an event type structure. Section 3.3 demonstrated this by using a weight event that is published in channels according to a tree structure. As depicted in that example, the Message Handler implementation publishes an incoming event in channels represented by each node in the event tree from the leaf node towards the root node. Full stops in the event name serve as the tree's edges.

For triggering the Message Handler to distribute an event, it needs to be injected into the system. This is done as shown in the following Listing 4.2 and applies for both the middleware's modules as well as for the applications:

Listing 4.2: Sending an event to myHealthHub. The listing shows a Sensor Module sending a blood pressure reading (i.e., `myBloodPressureEvent`) to myHealthHub.

```
1 Intent i = new Intent();                                          // create Intent
2 i.putExtra(Event.PARCELABLE_EXTRA_EVENT, myBloodPressureEvent);    // add event
3 i.setAction(MyHealthHub.RECEIVER_CHANNEL);                        // set channel
4 sendBroadcast(i);                                                 // send Intent
```

Android Intents are used for intra- and inter-application communication. Before sending an event, it needs to be encapsulated in an Intent (line 2). For this, it is important that the event's Java class extends `android.os.Parcelable` which allows marshalling and unmarshalling across Android applications. In line 3, the receiver channel is set which is the same for all event types. Upon sending an Intent (line 4), the Message Handler, which is listening on the receiver channel, receives it and distributes it according to the event type.

4.2.1 Subscription

For subscribing to an event type or an inner node of an event type, application developers simply need to register an Android `BroadcastReceiver` with an Intent filter according to the event type. In the following example, an event receiver `mReadingEventReceiver` is instantiated (line 1) and registered for events of type blood pressure (lines 2 & 3):

Listing 4.3: Registering an event receiver to blood pressure events.

```
1 mReadingEventReceiver = new ReadingEventReceiver();       // create event receiver
2 registerReceiver(mReadingEventReceiver,                   // register listener for
3     new IntentFilter(SensorReadingEvent.BLOOD_PRESSURE)); // blood pressure readings
```

4.2.2 Notification

Upon an incoming event, the Message Handler distributes it with respect to its structure. The distribution is done similar to the event injection shown in Listing 4.2 with the exception of using the strings of the individual event type nodes instead of the abstract receiver channel in line 3. In order to receive the notifications, applications and modules need to register an event receiver (cp. Listing 4.3) and handle incoming events. Listing 4.4 shows how an event receiver is implemented. It has to extend the Android `BroadcastReceiver` (line 3) and provide an `onReceive()` method (line 5). Upon a new event, this method is invoked and the corresponding Intent handed over. Similar to the encapsulation, the event is extracted from the Intent (cp. line 8) and ready for being processed. Starting with line 10 in Listing 4.4, an example for processing blood pressure readings is given.

Listing 4.4: Android `BroadcastReceiver` that receives events from myHealthHub. Incoming events are proccessed based on their event type (e.g. blood pressure events).

```
1 /** Event receiver implemented as a Android BroadcastReceiver for receiving
2  *  myHealthAssistant sensor reading events. */
3 private class ReadingEventReceiver extends BroadcastReceiver {
4     @Override
5     public void onReceive(Context context, Intent intent) {
6
7         /* Retrieving the event from intent */
8         Event myEvent = intent.getParcelableExtra(Event.PARCELABLE_EXTRA_EVENT);
9
10        /* Check for blood pressure event */
11        if(myEvent.getEventType().equals(SensorReadingEvent.BLOOD_PRESSURE)) {
12            // process blood pressure event
13        }
14    }
15 };
```

4.3 Application-specific Communication Channels

The Message Handler described in Section 3.4 is responsible for establishing individual communication channels to each application. In contrast to the open communication channels which broadcast readings to the whole system, this approach requires secure communication between myHealthHub and the applications. In Section 2.3.3, we presented a security analysis and guidelines on secure inter- and intra-application communication in Android based on the findings from [31, 73]. It

was shown that signed Implicit Broadcasts and Explicit Broadcasts do not allow other applications to eavesdrop on the inter-application communication as long as the operating system is not penetrated. This saves the costs for encrypting the communication between applications and the middleware. A drawback of signed Broadcasts is the need for having the signatures already created beforehand and having them installed with the application which requires a re-installation of the system in case the security parameters change. Therefore, we opted for **Explicit Broadcast Intents using setPackage()** for inter-application communication which saves the need for encrypting the communication and allows changing permissions at run-time. For intra-application communication, we use **Local Broadcast Receivers** that are used like usual Broadcast Receivers but remain unexported to other applications.

In contrast to the architecture proposed in Section 3.4, our Android implementation uses two unidirectional communication channels between the middleware and an application. Since other applications are not able to eavesdrop on Explicit Broadcast Intents, the Message Handler's receiver channel is the same for all applications. This simplifies the application development because the communication channel to the middleware is static and already known beforehand. Within a subscription request, applications transmit the Android Component/Package name to which the Message Handler is expected to send messages. This again simplifies the development: Since the component name inherits the application's name, the Security Manager can easily check whether the request belongs to an installed and admitted application. An explicit authentication check is not implemented so far.

The Message Handler keeps track of all subscribed applications and advertised event types. In case of a subscription without a fitting advertisement, the event subscription together with all advertised event types are forwarded to the Transformation Manager. If a client crashes, the System Monitor could detect this crash by using the Android `ActivityManager` which allows probing whether an application is running. This helps to delete obsolete event subscriptions and advertisements.

In order to manage communication among applications and the middleware, we introduced new event types as shown in Table 4.1 and additional management channels for both myHealthHub and the applications. All events listed in the table extend the abstract `ManagementEvent` from the first row. Besides the typical events for (un-)subscriptions and (un-)advertisements, we introduced three additional event types: `Announcement`, `StartProducer`, and `StopProducer`. Announcement events inform permitted and subscribed applications whether a matching event producer is connected to the system or not. We realized during the development of our demonstrators (cp. Chapter 5) that applications need knowledge about connected sensors in order to provide a better user experience. For instance, a sports application showing the user that the heart rate sensor is still not connected accelerates the setup phase before training. StartProducer and StopProducer events are used for informing event producers whether their events are needed or not. It helps to save the limited resources by, for instance, switching off unused sensors.

Event Type	Constructor
ManagementEvent	`abstract ManagementEvent(String eventType, String eventID, String timestamp, String producerID, String packageName)`
Subscription	`Subscription((ManagementEvent), String desiredEventType)`
Unsubscription	`Unsubscription((ManagementEvent), String desiredEventType)`
Advertisement	`Advertisement((ManagementEvent), String advertisedEventType, String JSONEncodedProperties)`
Unadvertisement	`Unadvertisement((ManagementEvent), String unadvertisedEventType)`
Announcement	`Announcement((ManagementEvent), String affectedEventType, int announcementType)`
StartProducer	`StartProducer((ManagementEvent), String producerID, String affectedEventType)`
StopProducer	`StopProducer((ManagementEvent), String producerID, String affectedEventType)`

Table 4.1: Management events used for subscribing to and unsubscribing from event types, advertising and revoking the advertisement of event types, and for announcing connected event producers.

4.3.1 Advertisement

Before myHealthHub distributes events of a certain type, event producers have to tell the system which event type(s) they produce. This is done with an `Advertisement` event (cp. Table 4.1) sent over the middleware's management channel. The following Listing 4.5 shows how an advertisement event (i.e. `myAdvertisement`) is sent as a secure Explicit Broadcast Intent using `setPackage()`.

Listing 4.5: Sending an advertisement event to myHealthHub using the management channel and explicit addressing with `setPackage()`.

```
1 Intent i = new Intent();                                          // create Intent
2 i.putExtra(Event.PARCELABLE_EXTRA_EVENT, myAdvertisement);        // add event
3 i.setAction(MyHealthHub.MANAGEMENT_CHANNEL);                      // set channel
4 i.setPackage(MyHealthHub.PACKAGE_NAME);                           // set receiver package
5 sendBroadcast(i);                                                 // send Intent
```

As a response to an advertisement, myHealthHub replies either a StartProducer event or a StopProducer event. This event tells an event producer whether its readings are required or not. If the event producer supports a stop functionality such as Sensor Modules and Event Transformations, it stops until a StartProducer event is received.

The Advertisement event constructor in Table 4.1 expects the string of a JSON object. In this object, properties such as the sensors sampling rate, start/stop functionality, accuracy, and operation modes can be encoded and transmitted to the Message Handler. So far, information about start/stop functionality is being processed. This JSON object will be used for adapting sensors to the application's needs in case future sensors support different operation modes.

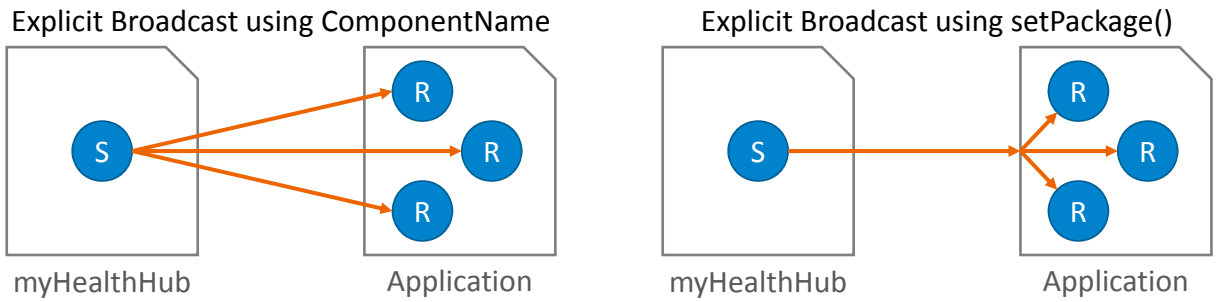


Figure 4.6: Explicit broadcasts using specific component names versus explicit addressing of an application package with implicit addressing within the application.

4.3.2 Subscription

We opted for subject-based event subscriptions. In a subject-based publish/subscribe system, notifications are annotated with subjects expressed as strings and organized in a hierarchy. The event type tree presented in Section 3.3 consists of a hierarchy of strings that represent event types. Applications can subscribe to nodes along the path of this event tree. The subject-based publish/subscribe method is a simple but powerful approach. Limitations regarding the single paths in the event tree are (partly) overcome by the use of Event Transformations that allow bridging two or more paths.

Applications subscribe to event types by sending a subscription event over the system's management channel. This event contains the requested event type and the application's package name (cp. subscription in Table 4.1). As an asynchronous response to the subscription, applications receive announcements telling whether the subscription was successful and whether the event producer is currently available or not. In case the producer's availability changes (e.g. a heart rate sensor (dis-)connects to/from the system), all successfully subscribed applications are informed. An application is successfully subscribed, if the application is permitted to access the desired event type and all fields of a subscription contain valid information. In order to check whether an application is allowed to receive events of a certain type, the Security Manager's `hasPermission()` method can be invoked. Applications can unsubscribe from event types by sending a corresponding Unsubscription event.

4.3.3 Notification

Upon an incoming event, the Message Handler disseminates the event according to the component/package names it has stored from the subscriptions. We have implemented two approaches for secure message delivery as depicted in Figure 4.6. In the left approach, each individual component of an application subscribes to event types. Upon an event, it is forwarded to each component individually. This is the only way of secure message passing for devices running Android OS before version 4.0.x (cp. Section 2.3.3). Drawbacks of this approach are the higher impact on the system (cp. Section 6.2) due to multiple Broadcast Intents sent as well as more complicated security checks. The message delivery on the right hand side provides an explicit addressing of specific

applications using the `setPackage()` method and an implicit delivery within an application. As a result, events are sent to the application only once which is more efficient and makes the security checks less costly and more flexible. Although the left approach provides more fine grained security checks, we believe that if an application is allowed to access an event type, this holds for the whole application (i.e., the package name). As long as the package name remains the same, application developers can change their application without worrying that they could lose the permission to certain event types. In case of an addressing per component name, changing a component such as an Activity would lead to rejected subscriptions. Therefore, we decided in favor of the `setPackage()` approach.

4.4 Adaptive Event Transformation

For evaluating adaptive event transformation, we implemented both the Transformation Manager running as a Service within the middleware and a remote repository providing a priori unknown transformations. Event transformations are implemented as OSGi bundles and the remote repository runs as a Java web server using Eclipse Jetty[4]. Before we describe the individual implementations, a discussion about transformation complexity is given.

Transformation Complexity

Event transformations can have different requirements regarding processing power, memory usage and event subscriptions. We distinguish between three transformation types:

1. **unit transformation**, e.g.: $^{\circ}C \leftrightarrow {}^{\circ}F$, $kg \leftrightarrow lbs$, $mmol/L \leftrightarrow mg/dL$ (for measuring the blood sugar concentration)

2. **type transformation**, e.g.: ECG stream → heart rate (bpm), specific activity (e.g. standing, running, cycling, ...) → activity level (e.g. low, moderate, high)

3. **inferring**, e.g.: activity & heart rate → critical situation, accelerometer data → activity/step counter, history of heart rate values → heart rate values + fidelity information

Unit transformations are usually cheap to perform, whereas type transformations might become more complex. An ECG-stream-to-heart-rate transformation ($T_{ECG \rightarrow HR}$) for instance needs to detect the QRS complexes in an ECG stream and calculate the heart rate based on the RR-intervals of those QRS complexes. Inferring as the third type of event transformations might combine multiple events and use additional domain knowledge in order to derive an event of higher meaning. For instance, deriving a critical situation based on incoming heart rate and activity events ($T_{HR+Activity \rightarrow HRAlarm}$). For our evaluation, we implemented event transformations of each type as listed in Table 4.2. The transformation $T_{HR \rightarrow HRFidelity}$ calculates heart rate fidelity information as already described for the Event Composer module (cp. Section 4.1.6). It demonstrates how the composer's functionality can easily be mapped to an Event Transformation. In addition to this, the $T_{HR \rightarrow HRAlarm}$ transformation, which generates simple heart rate alarms, connects to the User Repository in order to retrieve the

[4] `http://www.eclipse.org/jetty/`

Transformation	Type	Description
$\mathbf{T}_{\mathbf{kg \rightarrow lbs}}$	unit transformation	body weight from kilogram to pound
$\mathbf{T}_{\mathbf{lbs \rightarrow kg}}$	unit transformation	body weight from pound to kilogram
$\mathbf{T}_{^\circ C \rightarrow ^\circ F}$	unit transformation	body temperature from Celsius to Fahrenheit
$\mathbf{T}_{^\circ F \rightarrow ^\circ C}$	unit transformation	body temperature from Fahrenheit to Celsius
$\mathbf{T}_{\mathbf{ECG \rightarrow HR}}$	type transformation	ECG stream to heart rate values
$\mathbf{T}_{\mathbf{Acc \rightarrow Activity}}$	inferring	acceleration readings from knee to activities: sitting, standing, walking, running, cycling
$\mathbf{T}_{\mathbf{HR \rightarrow HRAlarm}}$	inferring	heart rate alarms based on thresholds set in User Repository
$\mathbf{T}_{\mathbf{HR+Activity \rightarrow HRAlarm}}$	inferring	heart rate alarms based on activity and heart rate events
$\mathbf{T}_{\mathbf{HR \rightarrow HRFidelity}}$	inferring	heart rate event including fidelity information based on history of heart rate events

Table 4.2: List of Event Transformations implemented as OSGi bundles that are downloaded, started, and stopped in the Transformation Manager.

threshold for critical heart rate values. It shows that transformations do not have to be static, but that they are able to adapt their behavior with respect to user-specific aspects.

4.4.1 Transformation Manager

The Transformation Manager installs, starts and stops transformations with respect to the current system configuration (cp. the system architecture in Figure 4.2 on page 59) . Upon an incoming transformation request, it first looks up the local transformation repository for a fitting transformation. If a transformation is found, it is started and it subscribes to the event types required for the event transformation. The Message Handler treats transformations in the same way as applications. Upon receiving a StopProducer event or an un-advertise message of a required event type from the Message Handler, the Transformation Manager stops the corresponding transformation. As a result, the transformation emits an un-advertise message, destroys itself, and releases occupied memory.

In case a requested transformation is not stored locally, it has to be downloaded from a remote repository and installed without requiring a reboot or a user interaction. Usual Android installations require a restart of the application as well as user interaction which would hinder a seamless adaptation to new requirements. In order to still provide an automated adaptation to new requirements, we decided to make use of the OSGi (Open Services Gateway initiative)[5] framework: This is a Java-based framework that allows installing, updating, starting, stopping, and uninstalling application components (`bundles`) at run-time. Apache Felix[6] for Android provides OSGi support for Android applications and is used for this implementation. Consequently, transformations are implemented as OSGi bundles and the Transformation Manager makes use of the OSGi life-cycle.

[5] `http://www.osgi.org`
[6] `http://felix.apache.org/`

Listing 4.6: Extraction from the *component.xml* file of a transformation from body weight in kilogram to pound. The fields `bind` and `unbind` define the methods invoked when starting or stopping the transformation.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <components xmlns:scr="http://www.osgi.org/xmlns/scr/v1.0.0">
3     <!-- Transformation Component -->
4     <scr:component enabled="true" immediate="true"
5                    name="BodyWeightInKgToBodyWeightInLbs">
6         [...]
7
8         <!-- PreferencesService dependency description -->
9         <reference name="android.content.Context"
10             interface="android.content.Context"
11             cardinality="1..1" policy="dynamic"
12             bind="bindAndroidContext"
13             unbind="unbindAndroidContext" />
14    </scr:component>
15 </components>
```

4.4.2 Event Transformations as OSGi Bundles

Event Transformations are implemented as OSGi bundles which are started and stopped by the Transformation Manager. In contrast to the sequence diagram shown in Figure 3.7 in which the transformations themselves react on StopProducer events, we decided for the Transformation Manager performing this task. This avoids that every Event Transformation needs to implement a management receiver and, thus, reduces the complexity and footprint of Event Transformations. The same applies for unavailable event types. If a transformation is not needed anymore or a required event type was unadvertised or is currently not available, the Transformation Manager stops affected transformations by invoking the `unbindAndroidContext` method and stopping the OSGi bundle afterward. Listing 4.6 shows parts of the OSGi component description of a transformation that converts weight readings from kilogram to pound. By referencing the Android Context (cp. lines 9 & 10), transformations can perform the same operations as other Android Components. This allows them, for instance, to access the User Repository in order to retrieve the threshold for the maximum heart rate value. Lines 12 and 13 name the methods invoked when a transformation is started and stopped respectively. When a transformation is started, it subscribes to the required event types and advertises the produced event type. When stopping, it unsubscribes and revokes the advertisement.

When an OSGi bundle is created, it typically consists of a JAR file that contains Java bytecode, a *MANIFEST.MF* file defining the required packages among others, and a *component.xml* file as shown in Listing 4.6. Since Android uses Dalvik virtual machines that use their own bytecode (.dex), the OSGi bundles need to be converted using the "`dx`" tool provided by the Android SDK before being used as an OSGi bundle for Android. The resulting transformations have a reasonable footprint of about 9 kB for a simple unit transformation (i.e., $T_{kg \rightarrow lbs}$) and about 13 kB for a more complex heart rate extraction from an ECG stream ($T_{ECG \rightarrow HR}$).

4.4.3 Remote Repository

The remote repository is implemented as an Eclipse Jetty web server. Transformation requests are sent as JSON encoded requests containing the requested event type and a list of event types advertised to the requesting system. If a matching transformation is found, it is transmitted to the Transformation Manager, otherwise the system is informed that no transformation was found. In case of multiple matching transformations, the remote repository has to decide which one fits best. For this, a corresponding cost function based on the complexity of a transformation and the energy consumption can help determining the best transformation. By adding priority information to the advertised event types of a request, a requesting Transformation Manager could influence the decision making. Since the current implementation forwards the first transformation found, implementing a cost function remains as future work. The combination of an OSGi framework running within the Transformation Manager and a Remote Repository providing new Event Transformations allows our system to adapt its functionality at run-time.

4.5 Integration in Existing Environments

This section presents how the middleware, myHealthHub, is integrated in an existing sensing environment. In this scenario, our middleware is used to bridge two existing projects resulting in a location-aware environmental sensing system. We use this as an example for showing how easy it is to integrate existing technologies in myHealthHub and, furthermore, with how little effort foreign applications are modified in order to operate as an event producer for myHealthHub. The integration in the InfraWOT environment (cp. Section 4.5.2) demonstrates the first case and the extension of RedPin (cp. Section 4.5.1) the latter one.

4.5.1 RedPin

RedPin[7] [13] is an indoor localization approach using the signal-strength of GSM, Bluetooth, and Wi-Fi access points on a mobile phone. It provides room-based accuracy and runs on Symbian, iOS, and Android phones. For training, users tell the system the name of a new room while the phone is in this room. The gathered localization information is transmitted to a RedPin server. The next time a user enters this room it is recognized. RedPin is a collaborative approach that allows training and detection in parallel.

Our goal is to provide sensor readings from ambient sensors in a room-based granularity. Since there already exists an open source Android implementation of RedPin, we chose this approach. RedPin automatically recognizes when a user enters a new room, but this information remains in the application and is not provided to other applications. In order to use the location information, there were two approaches for extending RedPin: a) provide an AIDL interface to other applications or b) use RedPin as a Sensor Module for indoor localization events. We decided in favor of b) because it reflects the event-driven characteristic of a user entering a room and it does not require

[7] http://redpin.org/

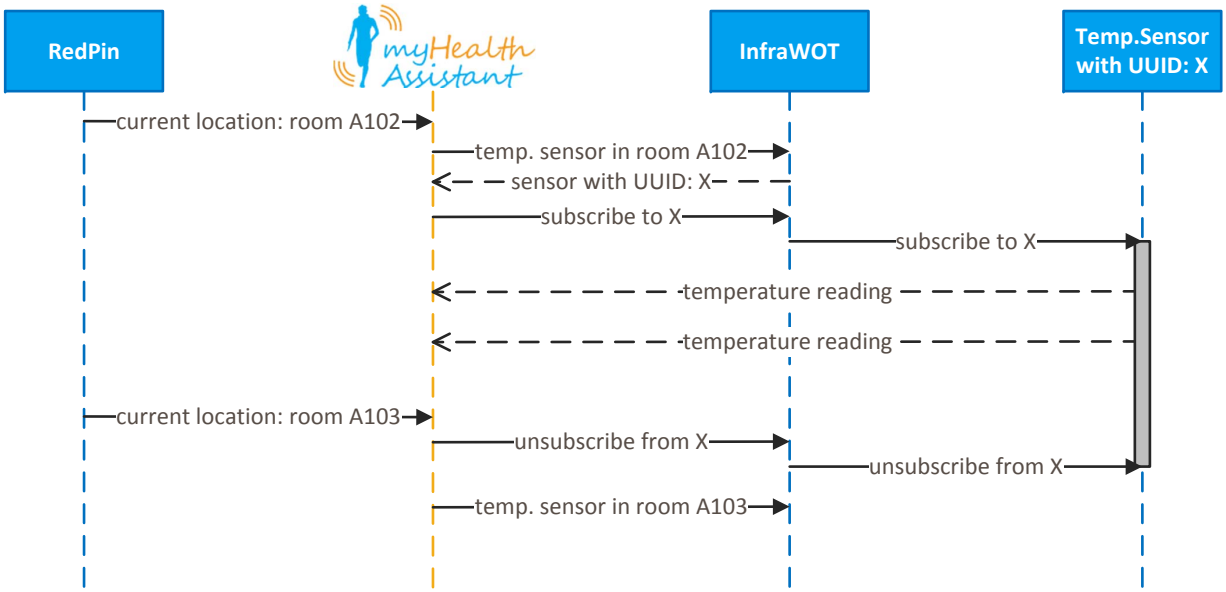


Figure 4.7: Sequence diagram showing the interaction among RedPin, myHealthHub (myHealthAssistant), InfraWOT, and ambient sensors.

myHealthHub to continuously query for new locations. The extension of RedPin was straightforward: we implemented it for the open channel approach which required only to create and send a `RedPinLocationEvent` as soon as a new room is detected. For the application-specific approach, an additional advertisement of the location events is required. With this extension, myHealthHub obtains location information from RedPin, which in turn can be used for querying sensors in a specific room using InfraWOT.

4.5.2 InfraWOT

The InfraWOT [89, 90] project focuses on discovery and look-up of sensor readings in Web-based smart environments. It serves as a discovery service that allows clients to query for resources such as sensors using HTTP GET requests. Upon a request, InfraWOT sends a JSON object including the addresses, UUIDs, and properties of available resources. In our scenario, we query for temperature and light sensors in a specific room. For instance, a query for a temperature sensor in room A102 is sent as follows: `GET http://infraWoTurl?q=temperature+at+A102` (cp. Figure 4.7). If there are corresponding sensors available, the first representative of each sensor type is selected for retrieving sensor readings. The sensor readings retrieval is either query-based or subscription-based. We decided in favor of the subscription-based approach that automatically forwards sensor readings to myHeathHub upon changes in the sensor reading values. Subscriptions are done by sending a HTTP POST including a JSON object to a specific URL with the sensor's UUID concatenated, e.g.: `POST http://infraWoTurl/subscriptions/sensorUUID`. The JSON object contains the callback URL to which the sensors shall send their sensor readings which is typically myHealthHub's IP address including a port number for each sensor type (e.g. temperature, light). Upon a successful subscription, new sensor readings are automatically sent to myHealthHub (cp. Figure 4.7).

For receiving sensor readings from ambient sensors, the corresponding Sensor Module opens HTTP server sockets. Upon an incoming ambient temperature or ambient light reading, a sensor reading event is created and sent to myHealthHub's Message Handler. In case the user changes the room, unsubscribe messages are sent and the sensor discovery and subscription is performed for the new location (cp. Figure 4.7). A benefit gained from the combination of both systems is the ability of gathering sensor reading from unknown environments. The only information needed is the URLs of the RedPin and InfraWOT services. We tested the system in the offices of the Distributed Systems Group at ETH Zurich. Our middleware was bridging between both systems and, thus, provided ambient sensor readings from the offices to a subscribed test application. The integration of the InfraWOT services in myHealthHub was done within two days and showed how easily the system can be extended. Furthermore, this example emphasizes how well the event-driven system design fits into ambient sensing environments.

4.6 Summary

In this section, we presented the Android implementation of our middleware architecture proposed in Chapter 3. We tested the system on several Android devices such as tablet PCs, different types of smart phones, and the Google Glass. For providing application-specific communication channels that do not allow eavesdropping on the communication, we decided in favor of the secure communication mechanisms built-in the Android system. We have presented both active and passive Sensor Modules and their interaction with the Sensor Module Manager. Furthermore, Sensor Module implementations for Bluetooth and Wi-Fi sensors were described. They allow gathering various types of sensor readings from on-body and ambient sensors. In addition to the raw heart rate readings, a system was proposed that adds fidelity information and saves application developers from individually implementing this functionality. We described the implementation for open and application-specific communication channels as well as the Transformation Manager and remote repository in detail. Furthermore, we developed several event transformations providing unit and type transformations, as well as the inferring of new information. Finally, the integration of myHealthHub in an existing sensing environment was presented to highlight the extensibility of our approach. In the appendix starting at page 127, we explain how to use the presented middleware implementation as an application developer.

5 Applications using the Middleware

In this chapter, four applications are described that were developed for demonstrating our system's capabilities and its feasibility for daylong health monitoring: three applications for body sensor networks and one application for body and ambient sensor networks. Figure 5.1 gives an overview of the implemented applications with their event subscriptions and depicts all sensor types connected to our system. Two applications, myFitnessDiary (cp. Section 5.1) and myFitnessTrail (cp. Section 5.3), are sports applications that focus on preventive health care. The first application monitors a wearers activities, heart rate and calorie expenditure throughout a day and, thus, provides important information and one step towards the integrated chronic disease prevention as stated by the WHO [157]. As soon as the user wears his or her gym clothing, the system automatically switches to more detailed fitness monitoring that detects individual cardio and weightlifting exercises and performs repetition counting. The second preventive health application focuses on monitoring and motivating a workout on a fitness trail including different strengthening exercises. It dynamically detects the exercise currently being performed to which basic statistics and a repetition count-down are given. A rehabilitation application, myReha, is described in Section 5.2. It is a representative of the rehabilitation and maintenance application from our motivational example presented in the introduction. This application instructs an outdoor rehabilitation program consisting of sitting, standing, walking, and running periods by giving audio feedback. In addition to this, it monitors the heart rate and warns when the patient's heart rate exceeds a predefined range. By connecting this application to a telemedical platform, care givers can monitor the patient's rehabilitation progress. The 4th application monitors the patients daily activities and vital parameters. For getting more detailed information about specific activities, ambient sensors are attached to objects and to walls in rooms. As soon as they detect the movement of an object or the user's presence, this information is sent to our middleware and the application is informed. The assistance of elderly people could be a target application for that solution, because it allows older people to live independently for a longer time which could help dealing with the impact of the demographic change on our health systems (cp. Section 1). Therefore, this application is called myElderlyCare and focuses on the detection of daily activities such as food intake and personal hygiene. It is presented in Section 5.4. From a middleware perspective, there is no difference between a preventive care application and a health care application that monitors an actual disease. Aspects such as stability, run-time, sensor handling, and messaging performance can be evaluated with both types of applications, but it is much easier to acquire healthy people than patients. Therefore, we mainly focused on preventive health care applications that can be evaluated with healthy participants.

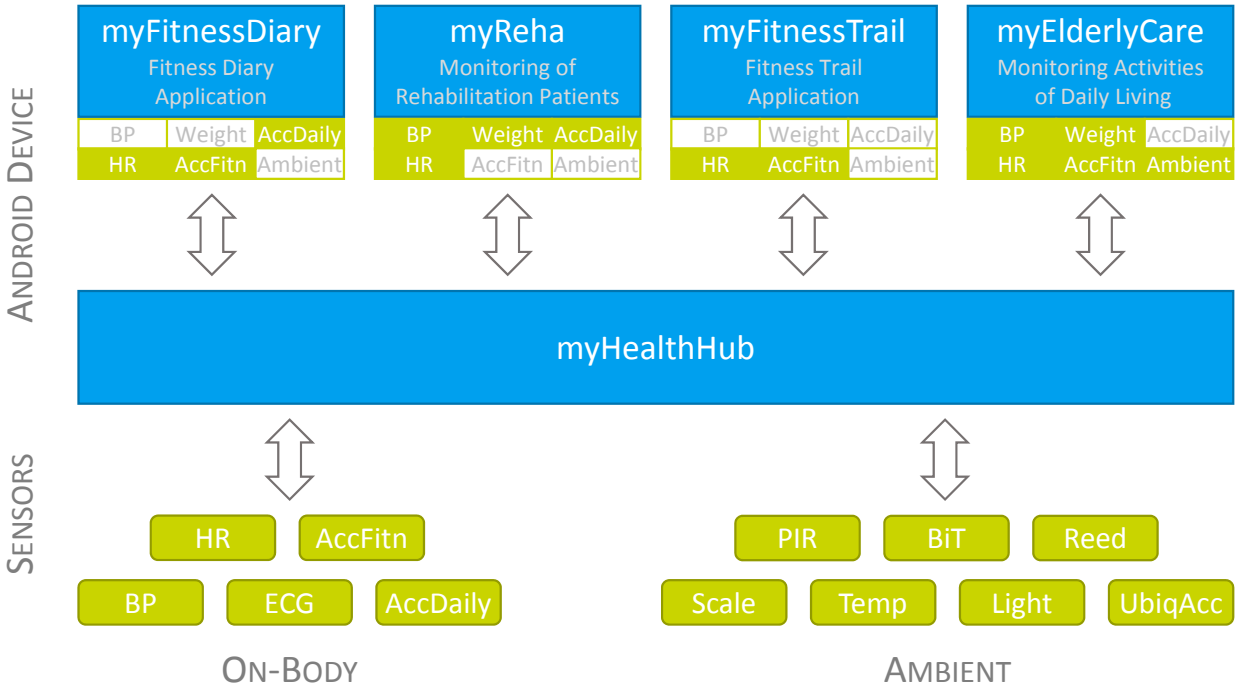
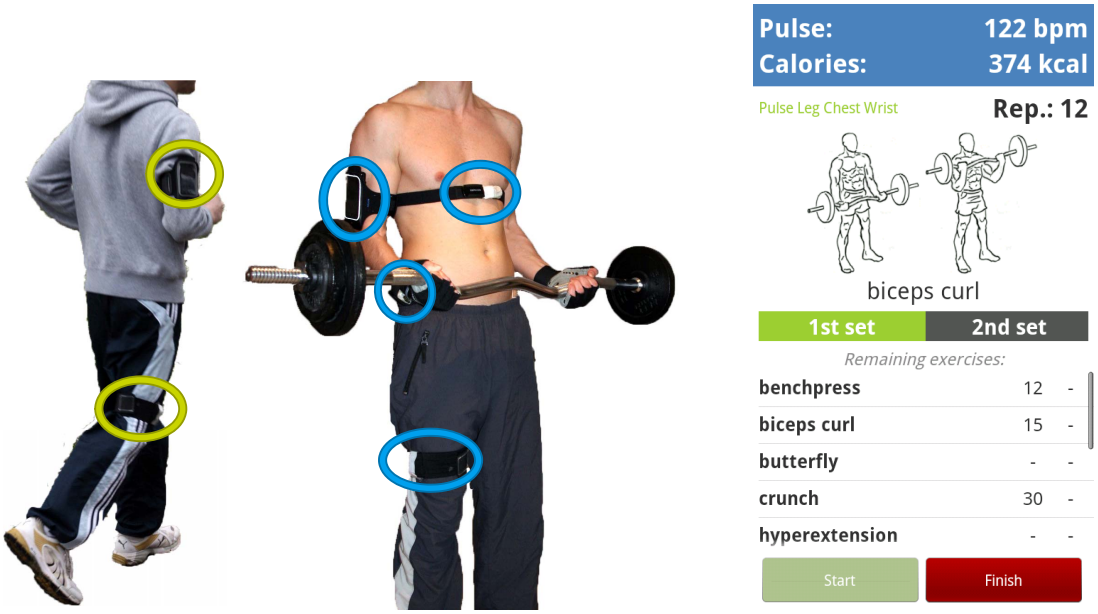


Figure 5.1: Overview of the applications described in this chapter. The highlighted boxes below the applications depict the event types to which each application subscribes. The bottom layer shows all connected sensor types. A distinction is made between an accelerometers used for recognizing daily activities (*AccDaily*), fitness activities (*AccFitn*), and interaction with the environment (*UbiqAcc*). The *BiT* sensor is a ball-in-tube sensor for detecting the movement of an object and the *Reed* sensors are magnetic switches.

5.1 Fitness Diary

The *myFitnessDiary* app [124, 125] performs daylong activity recognition and heart rate monitoring and adapts to given requirements on activity recognition. It contributes to the field of BSN-based preventive health care applications that equips patients/users with information for managing their own health. In a basic setup for daily activity monitoring, a set of a single customized accelerometer, a smart phone, and a heart rate sensor are used to detect five different activities, to monitor the heart rate and to calculate the calorie expenditure. When a person wears additional fitness accessories during a workout, a more detailed activity recognition that gives precise workout information is provided. In this gym workout setup, two more customized accelerometers are added which allow detecting sixteen activities and counting of individual weight lifting exercises. Pre-processing on the sensors saves the system's resources and allows an operating time of more than 12 hours.

This section is structured as follows: First, a system overview including the different hardware configurations, connection to myHealthHub, and the consumed event types is given. The following Sections 5.1.2 and 5.1.3 discuss the monitoring of daily activities and detailed gym exercises including repetition count respectively. The performance of the activity recognition approaches are discussed within the sections. Section 5.1.4 discusses the overall system performance including the system's capabilities for day-long health monitoring.



(a) (b) (c)

Figure 5.2: Body sensor network consisting of a smart phone, a heart rate monitor, and a setup for daily activity recognition (a) and gym exercise detection (b). Figure (c) shows the user interface of the *myFitnessDiary* app displaying the current heart rate, calorie expenditure, repetition count, exercise, and workout details.

5.1.1 System Overview

The application, myFitnessDiary, focuses on automated activity recognition and works on different granularities. For monitoring a person's daily activity, a coarse-grained activity recognition that detects only a few fitness-relevant activities is sufficient and does only require a small sensor network. For detecting all aspects of a gym workout, more precise activity recognition is necessary and additional information like the repetitions of weight lifting exercises is desired. This fine-grained activity detection needs a larger network of body sensors and increases the complexity of the system. The following fitness diary recognizes both the coarse-grained daily activities as well as the fine-grained gym exercises including additional repetition information. It stresses different aspects of a flexible body sensor network platform such as in-network processing, adaptability, seamless switching between sensor configurations, and multi-modal data processing. Figures 5.2(a) and 5.2(b) show the sensor configurations of our case study. The calorie expenditure calculation shown in Figure 5.2(c) is based on a study from [162] using age, gender, weight, and heart rate.

Sensor Setup

Much of the early data processing in the proposed system is done as close as possible to the sensors. For the inertial sensor, a custom platform called Porcupine was implemented that samples the data from a three-dimensional accelerometer (the ADXL330 from Analog Devices) at 100Hz and calculates per axis the mean and variance over a sliding one-second window as well as characteristic peak features to enable exercise counting (as shown later in Section 5.1.3). The

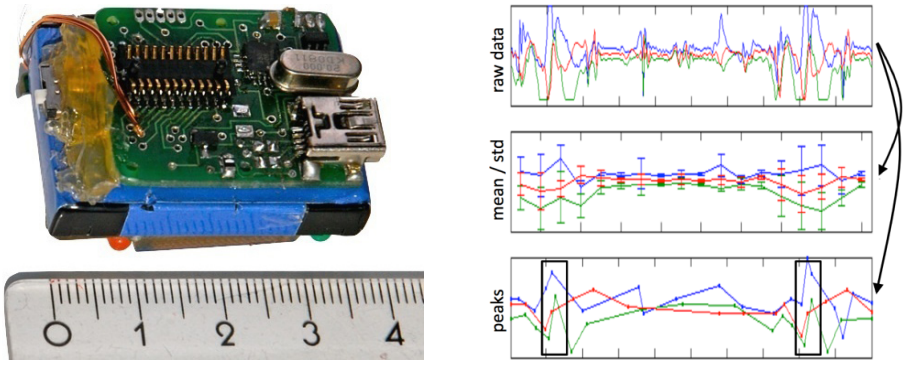


Figure 5.3: The Porcupine inertial units were designed to operate as long as possible on a lightweight battery. A dedicated microcontroller calculates basic statistics and the peak features on-board before transmitting them wirelessly to the smartphone.

micro-controller, a fast PIC 18F4550 from Microchip, operates at 48Mhz when calculating and is put to a low-power idle mode between samples being taken and processed. Figure 5.3 shows the uncased prototype of our wearable inertial sensor, with the battery wedged between the inertial sensing board and a ConnectBlue SPA 311 OEM module. On a small 360mA Li-Ion battery that is fully charged, the sensor can operate under the aforementioned conditions for over 50 hours. Recharging the module can be done over a standard USB port. For more information, we refer to the Porcupine/Hedgehog website[1] from the Embedded Sensing Group.

As a heart rate sensor, we connected the Zephyr HxM[2] to our system. It monitors heart specific parameters including heart rate, calories burned, and RR intervals as well as the wearer's step counts, speed, and distance. The sensor operates for 24 hours with a full charge.

The sensor setup for daily activity detection (cp. Figure 5.2(a)) consists of one accelerometer attached to the user's leg, a heart rate sensor, and a smart phone running both the myHealthHub middleware and the myFitnessDiary application. For a more detailed detection of the weight lifting exercises, including counting, two more accelerometers are needed: one in a weight lifting glove, and the other integrated in a chest-strap (cp. Figure 5.2(b)). A person simply needs to switch the sensors on and wear them; the system automatically connects to the newly available sensors and begins the fine-grained gym exercise detection.

Software Setup

By starting the myFitnessDiary application, myHealthHub's services are started and a seamless sensor handling is provided. The communication among myFitnessDiary and myHealthHub is done via open communication channels that provide the following reading events:

- `HeartRateEvents` provide the user's heart rate, an increasing heart beat counter similar to a message counter, and the sensor's battery level.

[1] `http://www.ess.tu-darmstadt.de/projects/porcupine`
[2] `http://www.zephyranywhere.com/products/hxm-bluetooth-heart-rate-monitor/`

- `AccSensorEvents` provide the mean (`x_mean, y_mean, z_mean`) and variance (`x_var, y_var, z_var`) values from 100 Hz acceleration data per axis based on a one-second sliding window.
- `CountAccSensorEvents` extend the `AccSensorEvent` and provides additional counting information (cp. peak detection in Section 5.1.3).

Once myHealthHub is configured to automatically connect to accelerometers and heart rate sensors, no further interaction with the middleware is required.

5.1.2 Daily Activities

For monitoring a user's daily activity level, a distinction is made between being idle (i.e., sitting, standing), doing moderate movements (i.e., walking, cycling) and doing sports (i.e., running). The detected activity is then correlated with the user's current heart rate. If the heart rate does not fit to the current activity, an alarm is sent which provides a basic patient monitoring functionality. Showing the calorie expenditure already gives feedback about the level of physical activity.

Experiment Setup

For activity recognition, we use the Porcupine, a three-dimensional accelerometer unit, attached above the right knee (cp. Figure 5.2(a)). The sensor samples with a frequency of 100Hz and sends Bluetooth packets including the variances and mean values of the last 100 readings per axis every second. The low sending frequency was chosen in order to save energy. Based on the readings transmitted from the Porcupine sensor, myHealthHub generates `AccSensorEvents` that are sent to the myFitnessDiary application. For training data, we collected for each walking, running, and cycling 120 data samples and for each sitting and standing 18 data samples from a male subject (age: 28 years, height: 1.80m, weight: 67kg). Based on these samples of mean and variance values for each axis, we modeled the six-dimensional Gaussian distribution for each class and used this information for the activity detection directly on the phone. The closest distance to one of these classes of an incoming sample decides on the current activity. Based on this, we achieve sufficiently accurate recognition for the same subject as suggested by previous work [98, 81, 143].

Subject-dependent Evaluation

To evaluate the limitations of the proposed system, it is important to know in which ranges (e.g. speed) of a given activity the system still recognizes the correct activity and at which points it tends to fail. Therefore, we stress-tested our system with different walking and running speeds as well as different pedaling speeds for cycling. The tests were done with the same subject as for the training data. Figure 5.4 depicts the results for walking/running performed on a treadmill. The solid line shows the recognition accuracy for walking. For a very slow walking speed of 1.0 km/h the system does not detect walking properly since the movements are too slow. Only a recognition accuracy

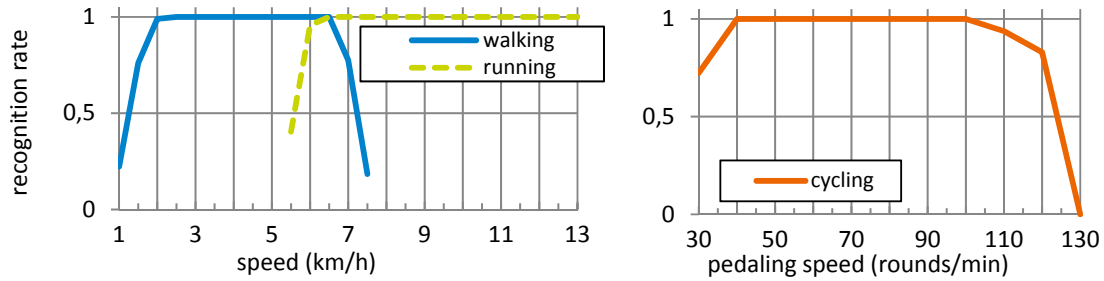


Figure 5.4: Stress-testing the daily activity recognition: Accuracy for different speeds, showing the regions where the activities are reliably detected.

Subject	Gender	Age	Weight	Height
1	female	52	70kg	1.65m
2	female	22	55kg	1.66m
3	female	22	72kg	1.66m
4	male	53	85kg	1.78m
5	male	28	82kg	1.70m
6	male	24	76kg	1.88m

Table 5.1: Details on the group of test subjects used in the evaluation. Extra care was taken to have a wide variety especially in age and fitness.

of 22% is achieved and in most cases the detected activity was "standing". By accelerating the walking to a more realistic speed the accuracy increases very fast (76% for 1.5 km/h and 99% for 2.0 km/h). In a range between 2.5 km/h and 6.5 km/h the activity "walking" was always detected correctly. With a very fast walking speed of 7.0 km/h the accuracy dropped again. The dashed line shows the results for running. In this case, we started with 5.5 km/h and an accuracy of 40%. At a speed of 6.0 km/h the algorithm already achieved 96% of accuracy and reached 100% accuracy with a speed of 6.5 km/h. We stopped the test at 13.0 km/h with still 100% recognition accuracy. The graph on the right hand side of Figure 5.4 shows the results for cycling. The algorithm needs at least a pedaling speed of forty rounds per minute in order to reach a 100% accuracy and drops above a speed of one hundred rounds per minute.

Our tests have shown that the activity detection is robust even if the activities are performed in other speed ranges than they were trained for. In addition to indoor treadmill tests, the subject performed several outdoor runs as well, with similar recognition results: The figures show that the detection is accurate for realistic walking, running, or cycling speeds. These tests were done by the same subject as for the training data. We will test for subject-independence in the following section.

Subject-independent Evaluation

Ideally, an activity recognition system for (preventive) health care or fitness applications is deployable without any additional training procedures. If an application requires activities to be recognized, the patient or user simply adds one or more new sensors and should not be bothered

	Walking	Running	Cycling	Standing	Sitting
Walking	3208	1	1	0	0
Running	0	3094	12	0	0
Cycling	0	0	2938	0	0
Standing	0	0	0	3120	0
Sitting	0	0	1	0	3290
Accuracy	**100%**	**99.9%**	**99.5%**	**100%**	**100%**

Table 5.2: The confusion matrix and accuracies for subject-independent recognition.

by (re-)configuring the system. We therefore tested the performance of our system for subject-independence. The chosen subjects vary in age, height, weight, and gender (cp. Table 5.1) and every subject differs at least in one factor from the person used for the training data (male, 28, 67kg, 1.80m). Every subject was doing the exercises at three different speeds (4 minutes each). They had to cycle with 70, 80, and 90 rounds per minute and to walk with 3.0, 4.0, and 5.0 km/h. For running they could choose their own three speed levels whereas levels between 7.0 and 10.0 km/h were chosen. Sitting and standing were not varied, as they were detected correctly.

Table 5.2 shows the results of our cross test. For all subjects, walking was detected with an accuracy of 100%, running had one outlier and cycling had 14 outliers with still more than 99% of detection accuracy. Those results show that the activity recognition is very reliable and does not need a person-specific training.

5.1.3 Gym Exercises

The previous section has shown that our application can handle coarse-grained activity recognition robustly on daily data, using a dedicated accelerometer and a heart rate monitor connected to a phone as a basic setup. This is sufficient for monitoring the user's overall daily activity, but more activities and specific information is desired for specific workouts: In this section we like to demonstrate that our middleware also copes with more complex BSNs at run-time, and that seamless BSN configuration changes are supported in the detection as well. A second workout mode is introduced for a more fine-grained workout diary. In this mode, a separate activity recognition module distinguishes between 16 activities: 5 cardio exercises and 11 weight lifting exercises shown in Table 5.3. In addition to this, more detailed activity information is given as a counting algorithm detects and counts single repetitions of each weight lifting exercise.

Exercise Recognition

For the detection of 16 gym workout exercises, the same Gaussian-model-based classifier and platform as for the daily activity detection is used, with the only difference being that two more dedicated Porcupine accelerometers are assumed to be present. This makes it easier for the model to distinguish between the exercises: One sensor is attached to a sensor strap around the chest and a second sensor is attached to the right weight lifting glove (cp. Figure 5.2(b)). By embedding

ID	Exercise	Posture	Type
1	Walking	-	Cardio training
2	Running	-	
3	Cycling	-	
4	Rowing	-	
5	Elliptical trainer	-	
6	Wide grip lat pulldown	Sitting	Back strengthening
7	Barbell rear delt row	Standing	
8	Hyperextensions	Standing	
9	Barbell bench press	Lying	Chest strengthening
10	Butterfly	Sitting	
11	Front barbell raise	Standing	Shoulders strengthening
12	Dumbell lateral raise	Sitting	
13	Barbell curl	Standing	Arms strengthening
14	Cable triceps extensions	Standing	
15	Barbell squat	Standing	Legs strengthening
16	Table top crunch	Lying	Abs strengthening

Table 5.3: The set of 16 gym exercises consists of 5 cardio workouts and 11 weight lifting exercises.

the sensors into the weight lifting outfit, the sensors are always located at the same position and no additional straps are needed. As before, every sensor transmits the mean and variance for each acceleration axis per second, expanding the total input data space for the Gaussian models from six to eighteen dimensions for exercise recognition. Applications can distinguish between the different `AccSensorEvents` by their producer IDs.

The exercises shown in Table 5.3 consist of 5 popular cardio workouts and 11 popular weight lifting exercises for training the chest, back, shoulders, arms, abdomen, and legs. The selection of exercises were chosen to provide a realistic and varied full-body workout set that regular gym users would perform. The execution of the exercises, especially their typical speed, is expected to be subject-dependent, so on-line training of the Gaussian models was implemented within the application. All results in this section will therefore be based on person-dependent evaluation.

The evaluation of the gym exercises detection was done in five runs in an actual gym environment. The extended BSN of smart phone, basic accelerometer sensor, and the two gym-specific sensors embedded in glove and chest-strap, were used for both the capturing of the training data, as well as the classification in real-time for the trained setup. Every exercise was performed for about 45 seconds and recognized amidst regular background data.

Exercise Counting

Previous work [27, 28] has identified the detection of weights and the counting of repetitions for weight lifting exercises as an important feature in a gym diary. For the amount of weights, the authors of [27] propose an RFID-weight mapping utilizing RFID tags on the weights and a glove with an RFID reader. The glove can send Bluetooth packets including the recognized tags to the phone which then could use the mapping for the weight calculations. In our work, we are only

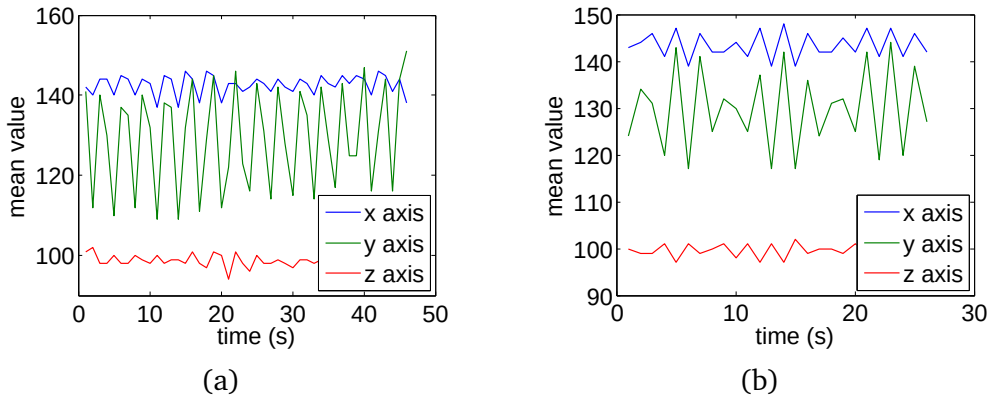


Figure 5.5: The 1Hz mean acceleration values from 15 barbell curls in (a) normal workout speed and (b) very fast workout speed.

focusing on the number of repetitions and leave the weight calculations as a possible expansion of this project.

For the counting of the exercises, a two-layer approach was followed, depending on the subject's speed of performing the exercises. This is a necessity since the 1Hz communication between inertial sensors and the smart phone might become too slow to count repetitions of faster exercises (such as those performed with lighter weights). We cope with this by detecting the workout speed and switch to peak detection on the inertial sensor boards for counting the exercise repetitions. For normal workouts, however, the mean values that the inertial sensors are sending by default can be re-used for the counting. Note that we distinguish between *exercise counting on the phone* and the *on-sensor peak detection*.

Exercise Counting on the Phone

Visual inspection (see Figure 5.5) of the wrist's mean values sent every second by the sensors indicates that straightforward autocorrelation on the variance-dominant axis is sufficient for calculating the number of repetitions for those workouts that were done slowly. Figure 5.5(a) depicts for instance the mean values of the wrist sensor for barbell curls: The number of peaks in the y axis shown in the middle section of the plot matches exactly the 15 repetitions the test subject did. In particular, the exercise state counting module on the smart phone calculates the dominant axis, and measures through autocorrelation the number of repeating states as soon as a new exercise has started, in real-time.

Some exercises, however, can be executed at a higher tempo, which inadvertently leads to missed counts as is illustrated in Figure 5.5(b). This has led to an implementation of a similar peak detection algorithm on the inertial sensors themselves, as explained in the next paragraph.

On-Sensor Peak Detection

The exercises presented so far were done at a usual workout speed, taking on average approximately three seconds per repetition depending on the exercise. In certain conditions, people tend to do their exercises at an accelerated pace, causing the system that was discussed in the previous paragraph to miss counts. To deal with this issue, the Porcupine module is extended to pre-process

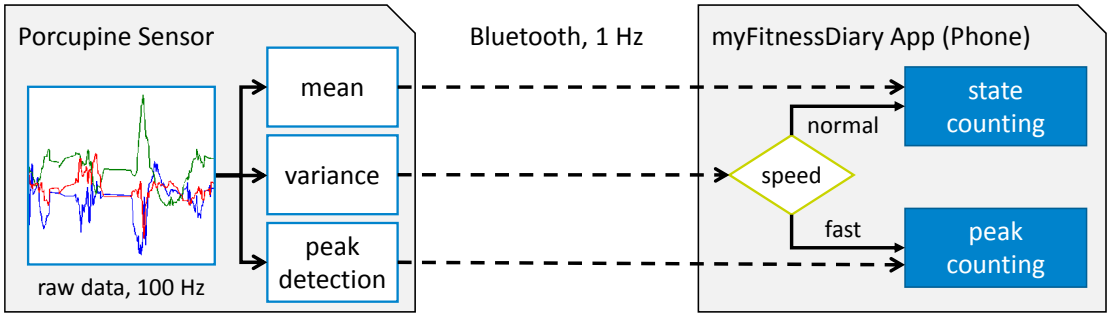


Figure 5.6: Data flow from raw data on the accelerometer to counting exercise repetitions on the phone.

not only the mean and variance per second but also a basic peak detection, as depicted on the left side of Figure 5.6 and related to the technique presented in [27].

An alternative solution would be to increase the frequency at which the inertial sensors report mean and variance to the smart phone. As our data shows that the faster exercises tend to repeat themselves between 1.5 and 3 seconds, this would mean doubling or tripling the 1 Hz sensor messages. This would however influence the power consumption of the whole system to deal with the increased communication speed, as well as cause a significant rise in data processing on the smart phone. In Section 6.1, we show that tripling the transmission rate leads to a doubled overall energy consumption.

The sensor-based peak detection is done per axis and works as follows: A low pass filter of size five is applied on the last one hundred acceleration samples (which equals the one second time window) in order to filter out small variations that would cause tiny peaks to be reported. Then, peak detection is done on this filtered data by finding local maxima and minima over the last second. The two most pronounced peaks found per axis are then piggybacked on the packet carrying the mean and variance values of the wrist sensor. As a result, myHealthHub sends a `CountAccSensorEvent` including this information to interested applications. Thus, instead of being able to detect a repetition lasting at least 2 seconds by using the mean values gathered from `AccSensorEvents`, this process allows to detect recurring patterns over at least 1 second, using solely the most prominent peaks.

On the smart phone side, the peak counting module has to first decide whether one of these six values (two per axis) was significant for a finished repetition. For this, two parameters are important: 1) the dominant peak axis and 2) a peak threshold characterizing a significant peak. In order to find these training parameters at run-time, a routine was designed that works completely unsupervised (i.e., the wearer just needs to provide the exercise at training but without the counts): The dominant peak axis is characterized by the highest absolute sum of peaks outside the band of medians. In order to find the peak threshold, the peak samples from the dominant axis are clustered into two clusters using *kmeans*: From the typical speeds, one resulting cluster will describe the non-characteristic values close to zero, and the other cluster will describe values closer to the characteristic peaks. The value between the median of both clusters' codebook vectors is then defined as the threshold. As soon as both the dominant peak axis and its threshold per axis are found, the peak counting module is able to estimate the faster workout speeds as well. In

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	156															
2		210														
3			211													
4	41			176		42		42						42		
5	3				210				1						8	42
6						168										
7							210									
8								168								
9									206							
10				12						211						
11											210					
12												210				
13													210			
14				21										119	30	
15				1										6	163	
16	9															167

Table 5.4: Confusion matrix for recognition of the 16 gym-specific exercises (cfr Table 5.3). Overall precision and recall are on average 92% respectively 95%.

Section 5.3.1, a second approach for on-sensor peak detection is presented that remedies the effort of finding the threshold characterizing a significant peak.

Subject-dependent Evaluation

Table 5.4 displays the raw classification results in a matrix showing the inter-class confusion for subject-dependent training. Classification accuracy ranges from 71% for the standing cable triceps extensions (labeled *14* cp. Table 5.3), which is frequently confused with activities *4* and *15*, to an almost 100% accuracy for 9 of the activities. For this evaluation, false classifications at the start and end of the exercises were minimized by ignoring the first and last seconds per exercise (i.e., not classifying them as exercise class or background class). This method for discarding borderline data was also implicitly applied in the training process, where the system waits for 3 seconds after the user presses the training button on the smart phone's screen to start the collection of training data.

The evaluation of the sensor-based peak detection was done in MATLAB[3], using the accelerometer sensors with embedded peak detection during a high-speed workout set from the same subject on another day. The result of this unsupervised algorithm is a miscount rate of 12.12%. We believe that this is still sufficient since a proper workout is recommended to be done more slowly.

Using the normal workout speed counting algorithm for a second training set resulted in four miscounts on the whole set of exercises and an overall miscount rate of 2.42%. The second training set was done by the same subject but on another day. Figure 5.7 shows the average results per exercise in case of normal and speed-up performance. The decision on which counting approach should be used for a given exercise is done in real-time based on the current variances.

[3] `http://www.mathworks.de/products/matlab/`

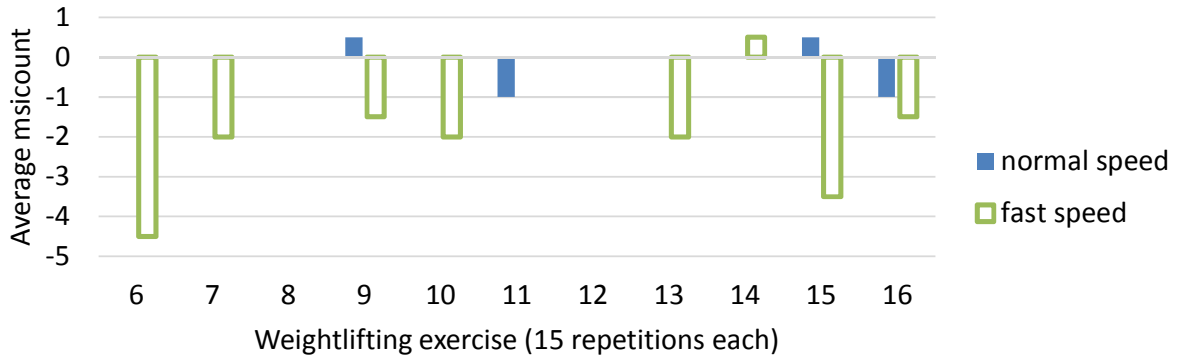


Figure 5.7: Average error (measured in miscounts) for the counting algorithm, for both normal execution speed (blue bars) and fast ones (green boxes).

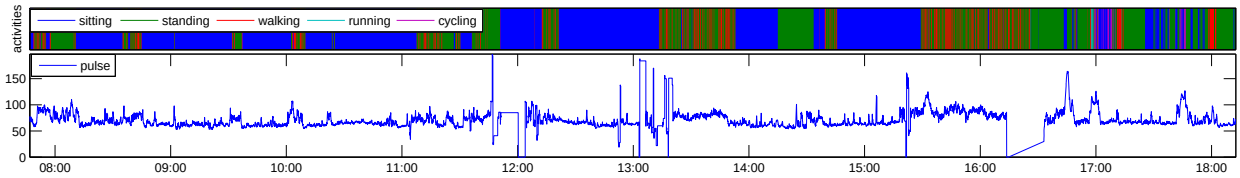


Figure 5.8: Visualization of the data captured by the BSN system on the phone: detected activities and heart rate values during one of the day-long tests.

5.1.4 Evaluation

The myFitnessDiary application serves as an example showing that the combination of Bluetooth sensors and a smart phone running our middleware and a monitoring application is capable of daylong health monitoring. The entire system, with phone and sensors, lasts under realistic conditions (the phone being used frequently, all sensors turned on) for at least 12 hours of daily activity and heart rate monitoring without a recharge. This is enough time for monitoring a person during the day and charging the system at night. However, the run-time of approximately 12 hours for the entire system largely depends on the phone itself. For our tests, we used the Motorola Milestone phone which possesses a 600 MHz processor, 3.7 inches screen size, and a 1400 mAh Li-Ion battery. In Section 6.1, we show that Bluetooth communication has a major impact on the system's energy consumption. With more power-efficient protocols such as Bluetooth LE the system's total run-time can be expected to improve.

Figure 5.8 shows a day-long test of daily activity and heart rate monitoring. It also shows some unusual positive and negative peaks. The negative peaks resulted from a too dry contact between the sensor strap and the skin which can be resolved by moistening the strap. Usually the heart rate returned to the actual value after some seconds. In our application, a *PulseMonitor* compares the heart rate against an overall range of healthy heart rate values. Since the peak values in Figure 5.8 were outside these ranges, the system triggered a "dangerous heart rate" alarm and, hence, informed the user about the sensor malfunction. In order to avoid those messages about malfunctions, the application can subscribe to heart rate events with fidelity information (cp. Event Composer in Section 4.1.6) and discard malicious readings.

For gym exercise recognition, sensors have to be attached to three positions on a subject's body (cp. Figure 5.2(b) on page 79). The heart rate sensor together with one accelerometer is attached to the chest. Since the heart rate sensor's strap is very comfortable and both sensors are small, these sensors are unobtrusive and attaching them is easy to do. The wrist sensor is combined with a weight lifting glove which makes it also easy to attach. Only attaching the leg's accelerometer happened to be slightly difficult for subjects because it is not clear where the sensor has to be. This problem was solved for the following application by labeling the strap with arrows. Once we had shown how to attach the leg's sensors, attaching it was no more an issue. Some subjects had difficulties with the strap slipping during the cardio exercises, especially during a run. For future sensors, a dimpled rubber strap could avoid slipping. Overall, attaching the sensors was fairly easy and readjustments were not necessary.

Despite having to deal with peaks from the heart rate sensor, our system worked quite reliably. Only during the long-term tests, the leg's accelerometer disconnected once or twice. By enabling the auto re-connection in the Sensor Module Manager, no more user interaction is required. Except for the long-term test, there were no Bluetooth disconnections during our tests.

From an end-user's point of view, the system seemed to be easy and intuitive to use. Except for the leg's accelerometer, the sensors are unobtrusive and wearing the system in a gym did not attract attention. Furthermore, changing from a textual user feedback on the phone to illustrations in form of pictures of the detected activity resulted in an improved user experience, especially for elderly people.

This application has proven that the combination of on-body sensors for monitoring physical and physiological parameters and a smart phone for collecting and interpreting those readings is powerful enough for day-long health monitoring. In order to put more workload on the mediator, we implemented a real-time activity recognition system which detects 5 daily respectively 16 gym activities. Despite the additional reasoning, the smart phone running both our middleware and the fitness diary operated reactive and without crashes. It highlights that the system is capable of more than just logging health information. The robust detection of daily activities presented in this section is also used for the following applications.

For future work, two often requested features for the gym exercise detection are the detection of the amount of weight lifted and feedback about the quality of performed exercises. For the amount of weights, we refer to the proposal of [27] using RFID technology as mentioned before. Interesting work addressing the assessment of the quality of weight lifting exercises is presented for instance in [144, 145]. Further possible extensions are real-time audio feedback that does not require the user to look at the screen and the connection to a remote database for further analysis. We have implemented both features in our myReha application presented in the next section.

5.2 Monitoring of Rehabilitation Patients

The fitness diary application presented in the previous section mainly focused on the detection of activities and the alerting of unhealthy heart rates. In addition to that, the *myReha* application described in this section extends the activity detection by real-time audio feedback that guides users through their training. Furthermore, daily health parameters are collected and sent together

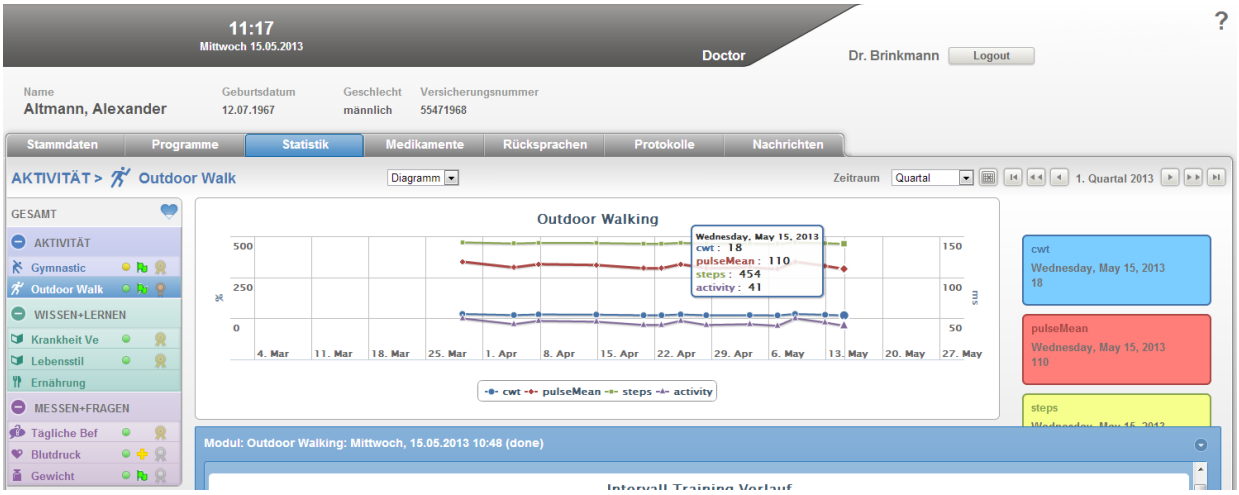


Figure 5.9: Telemedical platform showing the physician's patient monitoring screen. It allows monitoring of a patient's rehabilitation progress and configuring new rehabilitation parameters such as fitness exercises, medication intake, and knowledge quizzes.

with detailed training information to a remote telemedical platform. This platform resides on the third tier of a typical health monitoring solution (cp. Figure 2.1 on page 19).

This demonstrator arose in collaboration with Bosch Telemedical Systems. Its goal is to monitor rehabilitation patients and to safely guide them through a personalized interval training. For analyzing the training and rehabilitation progress, many aspects of an interval training such as instructed and performed exercises, heart rate behavior, and training intensity are transmitted to a telemedical platform. This platform allows physicians to monitor and modify a patient's rehabilitation program. Figure 5.9 depicts a screenshot of the telemedical platform: it shows the physician's statistics screen for a fictitious patient Alexander Altmann. The navigation bar on the left allows physicians to select different aspects of a rehabilitation program such as physical activity, knowledge quizzes, and taking measurements. The screen in the middle illustrates performance values of the patient's outdoor walking exercises after selecting it in the navigation bar. The telemedical platform provides extensive patient care including communication with the patient, modifying training instructions, changing the prescriptions on the medication intake, and adding new knowledge quizzes and training exercises.

The myReha application presented in this section provides two main services to the patient: it collects and depicts the last health measurements and interval training, and it guides the patient through a customized interval training. In addition, it monitors the patient's heart rate for providing an optimal training effect and to avoid overloading the patient. Figure 5.10 depicts screenshots of those two services. The left screenshot shows the application's home screen which informs the user about the last health measurements (i.e., blood pressure and weight) as well as the last interval training. As soon as a new measurement is taken, it is automatically uploaded to the telemedical platform. In case the transmission fails, a corresponding flag is set and a re-transmission initiated. By pressing the green button ("*Neues Training starten*"), the interval training is started. First, the application checks whether all sensors are connected. This is done

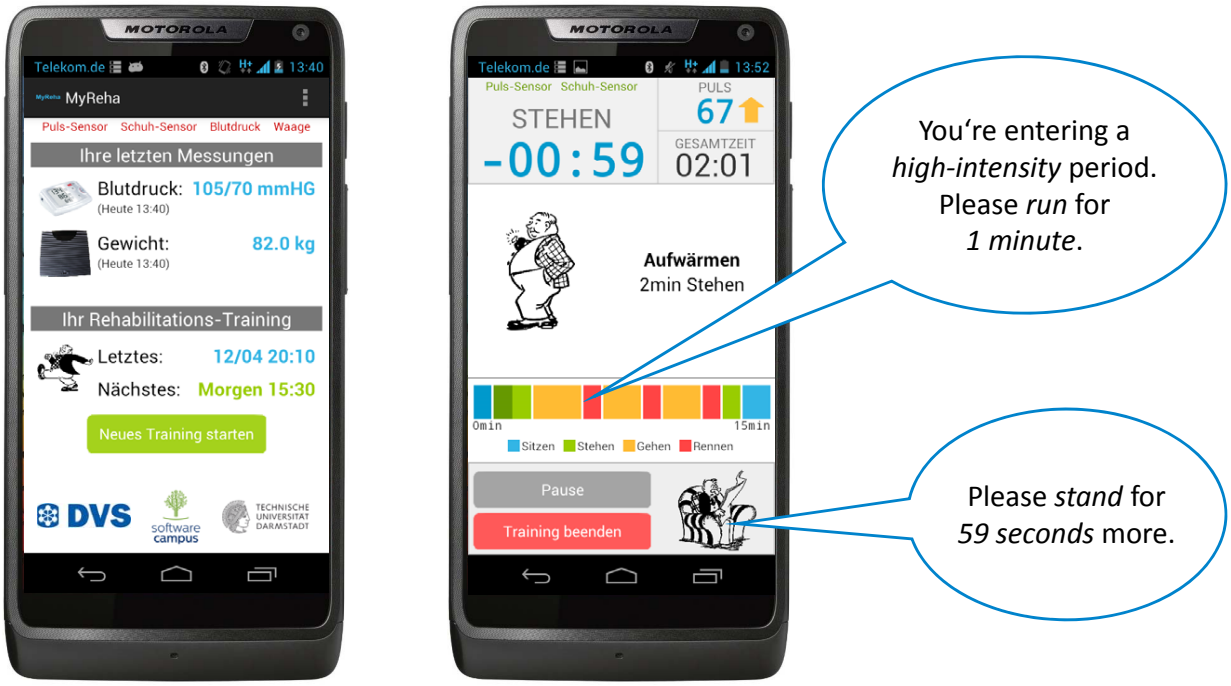


Figure 5.10: Screenshots of the myReha application showing the application's home screen on the left hand side and the interval training screen together with audio feedback on the right hand side.

based on the announcements sent from the Message Handler. If all sensors are connected and an interval training was selected, the training starts and, thus, the right screenshot is shown.

On the right screenshot in Figure 5.10 many aspects of a running interval training are presented to the user. It shows the remaining time of an interval exercise, the total training time, the current heart rate with a range indicator, a picture and textual information about the instructed exercise, a color-coded overview of the whole workout, and the currently detected activity. In addition to the visual feedback, the system performs audio feedback. Before describing the audio feedback in Section 5.2.2, we will focus on the interval training itself. Section 5.2.3 describes the connection to the telemedical platform and in Section 5.2.4, the system is evaluated based on the result of a user performing an interval training.

5.2.1 Interval Training

Interval training is a physical training that usually consists of low- to high-intensity exercises interspersed with rest or relief periods. For rehabilitation programs, the high-intensity is adapted to the patient's capabilities with respect to his or her disease. In many cases this is lower than the actual intensity the patient could handle. The goal of an interval training for rehabilitation is to restore and maintain the functionality of organs that are/were impaired by a disease. It is important not to overload the patient with too high training intensity. Therefore, the user instruction module described in Section 5.2.2 warns the patient and stops the workout if the workload passes given thresholds.

The usual interval training for patients suffering a heart disease consists of periods of resting (i.e., sitting, standing), low intensity (i.e., slow walking), and moderate intensity (i.e., strong walking). The daily activity detection presented in Section 5.1.2 distinguishes between sitting, standing, walking, running, and cycling which covers most of the rehabilitation activities. Thus, we decided to re-use this technology for the rehabilitation system.

In order to provide a good training effect, the interval training needs to be varied and adapted to the training progress. Furthermore, the telemedical platform allows care givers to modify the rehabilitation program. Therefore, we developed a solution that allows to easily change the interval program. It automatically adapts its behavior to a new training plan and visualizes the new training as shown in the right screenshot in Figure 5.10. Training plans are exchanged as XML documents: They contain the workout's name, a critical heart rate at which the system immediately stops, thresholds for the general minimum and maximum training pulse, and a list of exercises. Each exercise represents a period of the interval training and is described by a target activity, the spoken activity's name, a name/descriptor of the period, and a duration. Optionally, an exercise-specific minimum and maximum pulse can be defined. The spoken activity's name can differ from the target activity and it can be adapted to the patient's language. We developed a user interface for creating new training plans or for swift adaptations of existing plans.

5.2.2 Training Instruction Module

A huge benefit of having a system that performs real-time activity and heart rate monitoring is the direct feedback that can be given to the user. Figure 5.10 shows the system's user interface which already summarizes the whole training and gives feedback about the current period and heart rate values. In case the heart rate is outside of the predefined range, it is indicated by an orange arrow pointing up or down depending on the threshold that was crossed. In case the critical heart rate is reached, this is indicated by a red signal. In addition to the heart rate monitoring, the current exercise including its count-down is shown.

A major drawback of using the graphical user interface for instruction is that the user always has to look a the phone's screen which is very cumbersome, especially when performing walking or running activities. Therefore, we decided to give additional audio feedback to the user. By wearing headphones, the user does not need to look at the screen anymore and, thus, can focus on performing the interval training. The training instruction module guides the user through the interval training by giving the following real-time feedback:

- **Training instructions:** Before a new training period begins, the descriptor, the target activity, and the exercise duration are given. The upper speech bubble in Figure 5.10 shows the audio instruction for a period with the descriptor *high-intensity*, a target activity *run*, and a duration of *1 minute*.

- **Training advices:** In case the user performs the wrong activity for more than 10 seconds, the instructor module reminds him or her of the target activity and the remaining time it needs to be performed. An example is depicted in Figure 5.10: the phone reminds the user to *stand* for another *59 seconds* triggered by a detected sitting activity.

- **Heart rate monitoring:** In case the user passes one of the heart rate thresholds of the current exercise or a general threshold in case no specific thresholds are given, the user is informed to perform slower or faster depending on the threshold. In case the critical threshold is reached, the user is asked to quit the training and sit for a while before setting off for home.

5.2.3 Connection to a Telemedical Platform

We have connected our application to a telemedical platform as a representative of a 3rd tier BASN service. Every time a new health reading is available or an interval training is finished, a connection to the remote service is established and un-transmitted readings are forwarded. This is done either via Wi-Fi or via the phone's 3G network. In case a transmission fails due to lack of connectivity or a rejection from the remote service, a corresponding flag in the database marks the readings as not transmitted, ensuring that it will be transmitted on the next connection.

Health readings and interval training results are transmitted as JSON objects. Each object contains at least a patient ID which is known by the telemedical platform and one result. A result is either a blood pressure reading containing a timestamp, the systolic and the diastolic blood pressure, and the patient's pulse or a weight reading containing only the patient's weight or the summary of an interval training. This summary contains the start time and duration of the training, the mean and variance values per axis and second transmitted by a Porcupine accelerometer, the instructed target activity, the detected activity and a list of heart rate values. JSON objects are sent as HTTP POST messages.

5.2.4 Evaluation

The two main contributions of the myReha application are the training instruction module and the connection to a remote telemedical platform. The activity recognition itself is based on the daily activities described in Section 5.1.2 of myFitnessDiary. In contrast to the fitness diary, myReha uses myHealthHub's application-specific communication channels. While testing the application, myHealthHub automatically started the Porcupine accelerometer and the heart rate sensor when the training started and, thus, the application subscribed to the corresponding sensor reading events. The announcements sent from the Message Handler help to give feedback about unconnected sensors or to inform applications and users of a disconnection. (This was of special interest since one of the accelerometers started to break and, thus, operated only sporadically.) Using myHealthHub as an Android Remote Service integrated seamlessly into the myReha system and supported swift development.

For evaluating the training instruction module, the following interval training was created: 1 min. standing; 2 min. walking; 2 min. running; 2 min. walking; 2 min. running; 1 min. sprinting; 2 min. walking; and 1 min. standing. The training was performed by a male subject (30 years, 1.80 m, 68 kg) based on the audio feedback given by the instruction module. Figure 5.11 depicts the training results gathered during the workout. The boxes at the bottom compare the instructed against the detected activities. The detected activities are filtered by a 5 second filter returning the most

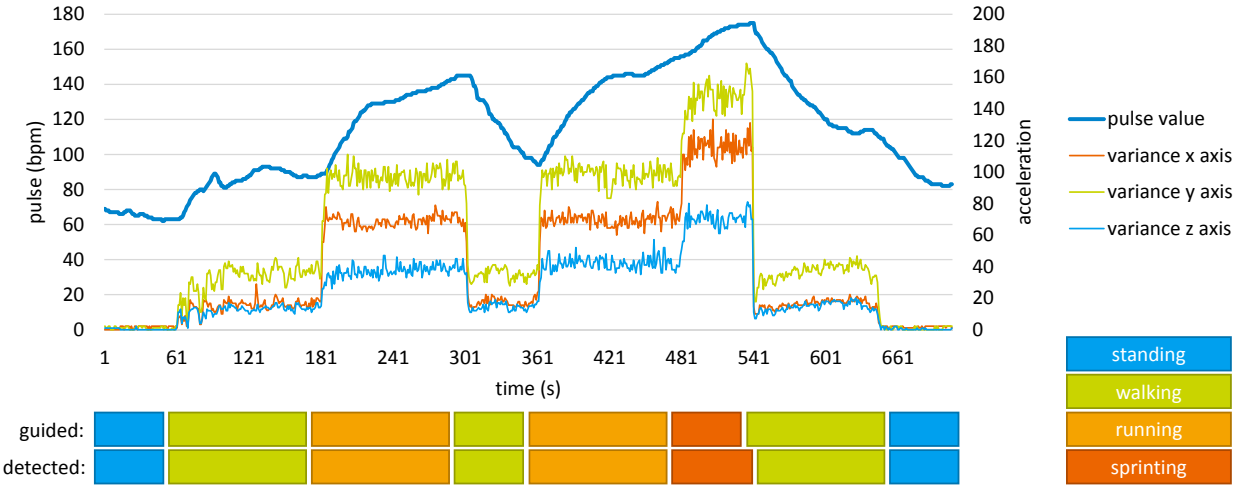


Figure 5.11: Training results of an interval training showing the heart rate adapting to different training intensities as well as boxes indicating the guided target and the detected activities.

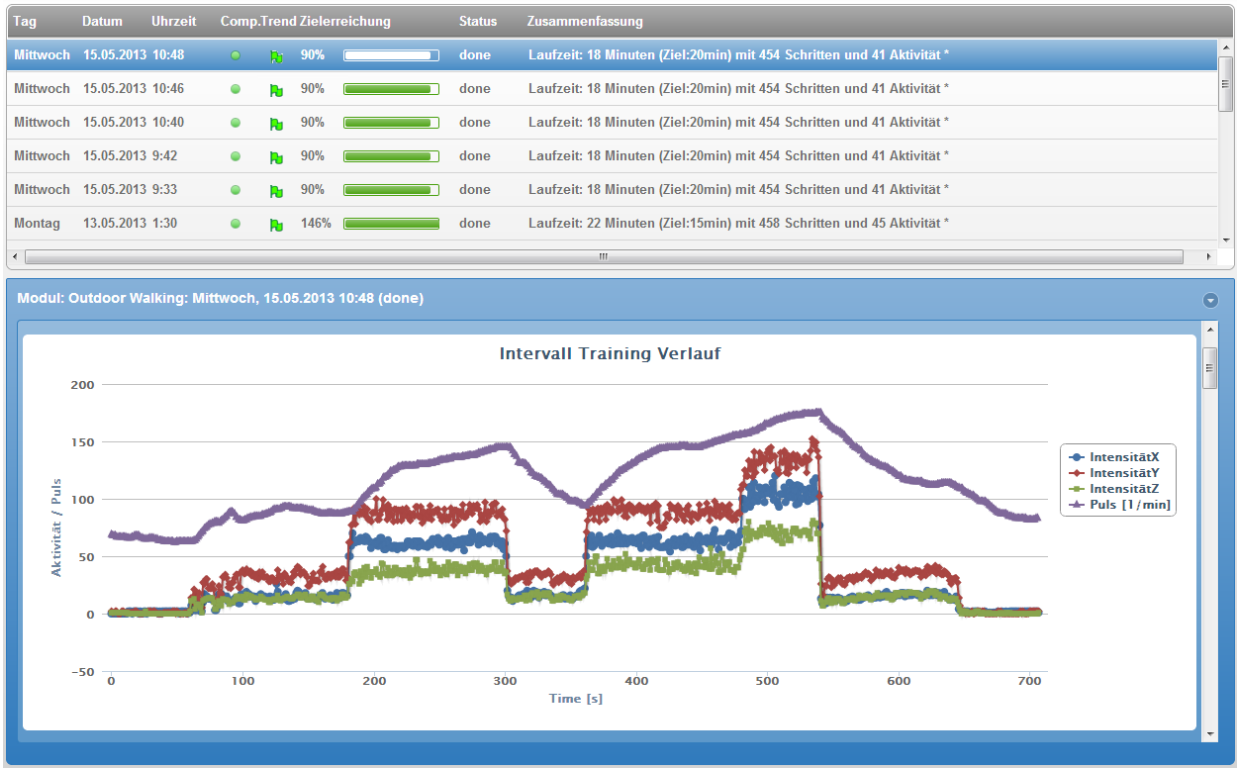


Figure 5.12: Telemedical platform showing the training results of an interval training that was transmitted from the myReha application using a JSON object.

dominant activity. Therefore, isolated outliers are filtered and not illustrated. The figure shows that users can very accurately follow the instructions. The only exception occurred during the transition from sprinting to walking. The subject's run out phase was still detected as sprinting/running. The graph in the upper part of Figure 5.11 shows the heart rate and variance values gathered during the workout. The variances highlight the different intensities of workout periods and allow a more detailed performance analysis. For instance, the difference between running and sprinting is very distinctive. The heart rate values show the adaptations of the cardiovascular system to different exercise intensities. During the walking periods, the heart rate recovers but increases as soon as a new high-intensity period starts. The heart's behavior to different intensities is a good indicator for the overall training progress of multiple interval trainings.

The second contribution of myReha is its connection to a remote database. Figure 5.12 shows a part of a screenshot taken from the physician's patient monitor of a telemedical platform. By comparing the interval training depicted in that screenshot to the training results illustrated in Figure 5.11, it can be observed that the training was successfully transmitted to the back-end system. Target versus detected activities are not shown in the figure but they are also transmitted and can be compared on a textual basis. In addition to this, blood pressure and weight readings are transmitted to the platform and plotted in the telemedical platform.

5.3 Fitness Trail

Fitness trails are paths equipped with obstacles or stations distributed along its length for exercising the human body to promote good health. The trails are designed for physical fitness training in the style attributed to Georges Hebert [154]. Equipment exists to provide specific forms of physiological exercise and can consist of natural features including climbable rocks, trees, and river embankments, or manufactured products (stepping posts, chin-up and climbing bars) designed to provide similar physical challenges. Figure 5.13 depicts example exercises performed at a fitness trail for strengthening the back, legs, chest, and abdomen. The recognition performance of our system for those exercises is given in Figure 5.15 among other exercises.

Important information about how a user performed on a fitness trail is which obstacles or stations were taken and how many repetitions were done. Instead of having the user keeping track of every single exercise and repetition, a body sensor network performing activity recognition and exercise counting can do this automatically and let the user focus on his/her fitness workout. In addition to this, by providing a user motivation that recognizes the current fitness station and dynamically adapts to it, the user can freely choose the station and still receives an optimal motivation program. Our *myFitnessTrail* application described in this section represents such a solution. For user motivation, we implemented two audio feedback features: when starting an exercise, the absolute maximum number of repetitions at that station as well as the number of repetition performed the last time at the same station are given. Towards the end of an exercise, a count-down that motivates in performing one more repetition than last time helps the user in pushing his/her limits. Compared to the myReha application, the user does not have to follow a strict training plan and the system motivates reaching personal limits. From a developer's perspective, an advantage of fitness trails compared to gym workouts is that most exercises are

(a) diagonal crossing (b) squats (c) pull-ups

(d) push-ups

(e) abdominals

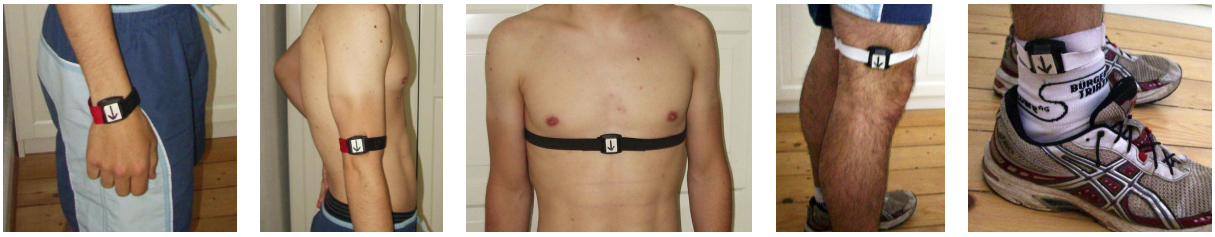
Figure 5.13: Various strengthening exercises performed on a fitness trail.

performed with the own body weight, which does not require to recognize the amount of weight someone is lifting.

For the case study, we extended the *HedgeHog*[4] inertial sensor developed in the Embedded Sensing Group at Technische Universität Darmstadt with a Bluetooth module. Our initial idea was to evaluate the energy savings using the more energy efficient Bluetooth Low Energy profiles. Therefore, we first added a Bluetooth LE module (OLS425I) from ConnectBlue to the sensor which worked fine with iOS devices but did work with Android devices. Although Bluetooth LE is officially supported since Android version 4.3 and the modules were recognized by our Android phones (Google Nexus 4, HTC One X+), the Android's Bluetooth stack did not provide the received packets. Therefore, we decided to use less energy efficient Bluetooth 2.1 modules from ConnectBlue (OBS410I) with the option of upgrading the sensors as soon as Bluetooth LE is sufficiently supported by Android.

In addition to the dynamic user motivation presented in Section 5.3.3, we determined the best sensor positions from a user's and a developer's perspective and improved the on-sensor counting algorithm, both presented in Section 5.3.1. The system's activity recognition was evaluated at

[4] `http://www.ess.tu-darmstadt.de/hedgehog`



(a) wrist (b) upper arm (c) chest (d) upper leg (e) ankle

Figure 5.14: Tested sensor positions. Participants are asked to wear the sensors such that all arrows are pointing towards the bottom when standing still with the arms close to the body.

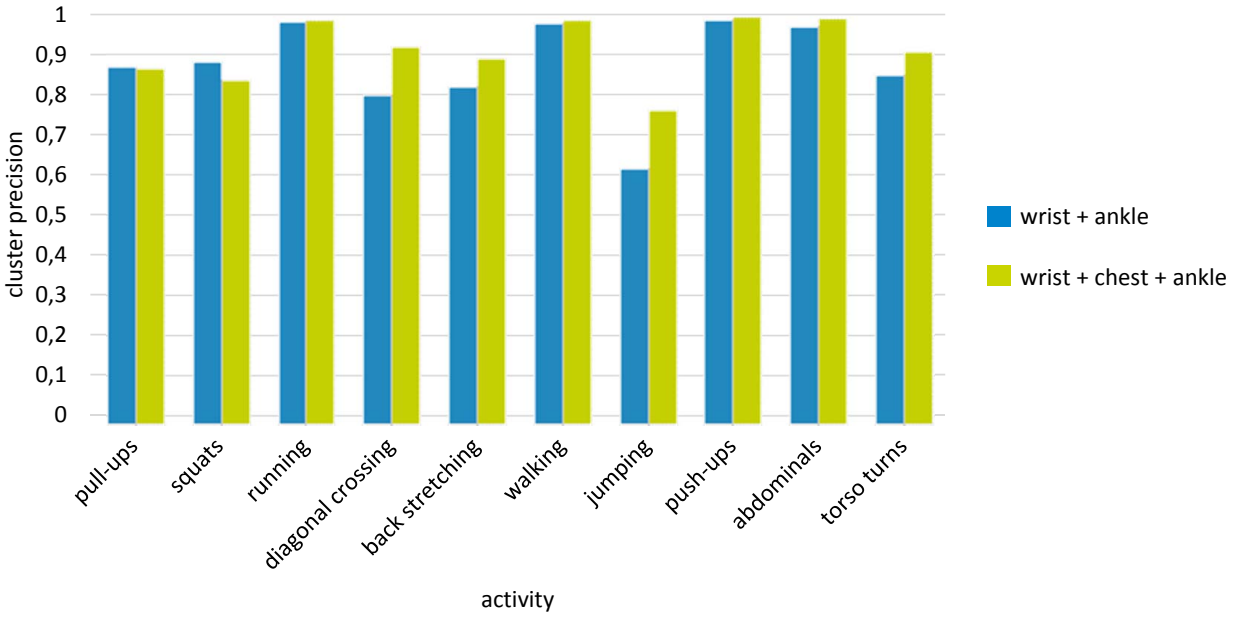


Figure 5.15: Cluster precision for two combinations of sensors: wrist & ankle and wrist, chest & ankle. The combination of three sensors performs slightly better. For details on the individual exercises, we refer to [113].

two fitness trails. In addition to this, a system for adding new exercises and fitness courses was implemented.

5.3.1 Exercise Recognition

For developing an optimized activity recognition and repetition counting for fitness trails, we supervised a Bachelor thesis on that topic [113]. Based on previous work [70, 161, 78, 27, 124], five recommendable sensor positions (cp. Figure 5.14) were analyzed with respect to the activity recognition accuracy and user acceptance. Two combinations of sensors were used for data collection: (wrist, chest, ankle) and (upper arm, chest, upper leg) and eight subjects were asked to perform one of two fitness tracks and to answer a user comfort questionnaire. Regarding user acceptance, all participants preferred the wrist and ankle sensors.

For detecting fitness trail activities, the phone uses the same algorithm as for the fitness diary application (cp. Section 5.1). In contrast to that application, we optimized the selection of features

that are extracted from the 100 Hz acceleration data and transmitted to the phone. In [113], the impact of different window sizes, combinations of sensor positions, and transmitted feature pairs on the cluster precision were analyzed. The window size was varied from 1 second to 10 seconds. For the sensor positions, eleven different combinations were tested and the following feature pairs were taken into account: mean-variance, mean-minimum, mean-maximum, variance-minimum, and variance-maximum. As a result, using the combination of mean and variance values based on a four seconds sliding window taken from either the ankle and wrist sensor or the combination of ankle, wrist, and chest sensors is the recommended setup for activity recognition on fitness trails. Figure 5.15 depicts the cluster precision for both sensor combinations. It results in an overall precision of 80% for a subject-independent activity recognition. For more details on the feature selection process and performance values, we refer to [113].

5.3.2 Exercise Counting

For repetition counting, we implemented a peak detection algorithm[5] by Eli Billauer on the HedgeHog sensors. In order to provide a proper repetition counting, the minimum time difference between two repetitions and the minimum difference in acceleration data between the minimum and the maximum, was determined in [113]. Using this peak detection, the state change detection presented in Section 5.1.3 becomes superfluous. Since every sensor sends a true or false value for every axis indicating whether a repetition was detected or not, the application only needs to select the representative sensor and axis. This is done based on the activity which is currently performed. So far, the dominant axis per activity was selected manually. For future work, this could be done similar to the selection described in Section 5.1.3 or based on the smallest variation of counting values compared to user input given during a system training mode.

Three new sensor reading event types were introduced for the myFitnessTrail application. Instead of distinguishing between sensor readings from different body positions based on the producer ID (cp. myFitnessDiary in Section 5.1), we decided for extending the `AccSensorEvent` by three new children in the event tree: `AccPeakSensorEventAnkle`, `AccPeakSensorEventChest`, and `AccPeakSensorEventWrist`. It is a refinement of the event type that still provides the required information (i.e., producer ID) for applications subscribed for `AccSensorEvents`. All events contain mean and variance values as well as the counting information.

5.3.3 User Motivation Module

The user motivation module provides exercise-specific workout statistics and a repetition countdown. Completed workouts are stored in a database including number of sets performed, repetition count, and the walking/running time between two fitness trail stations. The user motivation operates on activity events and can basically be used for any application using exercise repetitions such as the fitness diary. A sliding window of 3 seconds filters fluctuations in the stream of incoming activities. As soon as the recognized activity changes, the repetition count of the latest as

[5] `http://billauer.co.il/peakdet.html`

well as the overall maximum are presented using audio feedback. It allows the user to assess how many repetitions he or she should perform. In addition to this, the user motivation module keeps track of the repetitions and gives an audio count-down triggered by incoming repetitions. In the current implementation the count-down starts with the third-last repetition compared to the latest repetition count of that exercise. It does not motivate the user to perform one more repetition. If the user stops performing an exercise the motivation module stops the count-down.

For future work, the user motivation module will be evaluated with participants performing one of the trained fitness trails. Furthermore, training plans with different levels of expertise can help finding a good starting point and choosing a more challenging training after a while.

5.4 Monitoring Activities of Daily Living

The BMBF highlights maintaining independent and self-determined living of elderly in their own apartments as an important step for facing the demographic change [21]. In order to support older people in their daily life, an assistance system has to rely on information about the person's current status and context. So far, we focused on gathering health information and physical activity from body sensor networks. In elderly care, not only the physical and physiological states are important, but also activities of daily living (ADL) become very important since changes could reflect changes in the health status [146]. In contrast to the previous health applications, an elderly care system that assists a person in daily activities such as feeding, housekeeping, and personal hygiene needs to know what a person is currently doing and what he/she already did. Many residential sensing systems consist of hundreds of sensors [58] for estimating an inhabitant's position in the apartment and interactions with objects in the apartment assuming that future homes will be equipped with ubiquitous sensors. Most homes built nowadays do not possess those sensors and it is expensive in terms of time and money to deploy them afterward. Therefore, this section focuses on an ADL system that is easy to deploy and still allows monitoring daily activities and reacting to missed activities. The main idea is to attach (small) sensors to objects that are representative for daily activities. Those sensors inform an ADL application, called *myElderlyCare*, about user interaction which allows us to derive certain daily activities.

This section is structured as follows: First, the system setup consisting of body and ambient sensors is presented. We will describe how the system is installed and configured for a new apartment using myHealthHub. Furthermore, methods for system monitoring are presented. In Section 5.4.2, our approach for a rule-based activity recognition is presented before we conclude this section with a system evaluation in Section 5.4.3.

5.4.1 System Setup

Figure 5.16 depicts the system overview: body as well as ambient sensors report their readings to a smart phone running myHealthHub and the myElderlyCare application. A scale, an accelerometer, a heart rate, and a blood pressure sensor provide physiological information to the system and Wi-Fi modules developed at the ETH Zurich (cp. Section 4.1.2) report the user's presence using PIR sensors and user interaction with the environment using Reed-switches and ball-in-tube

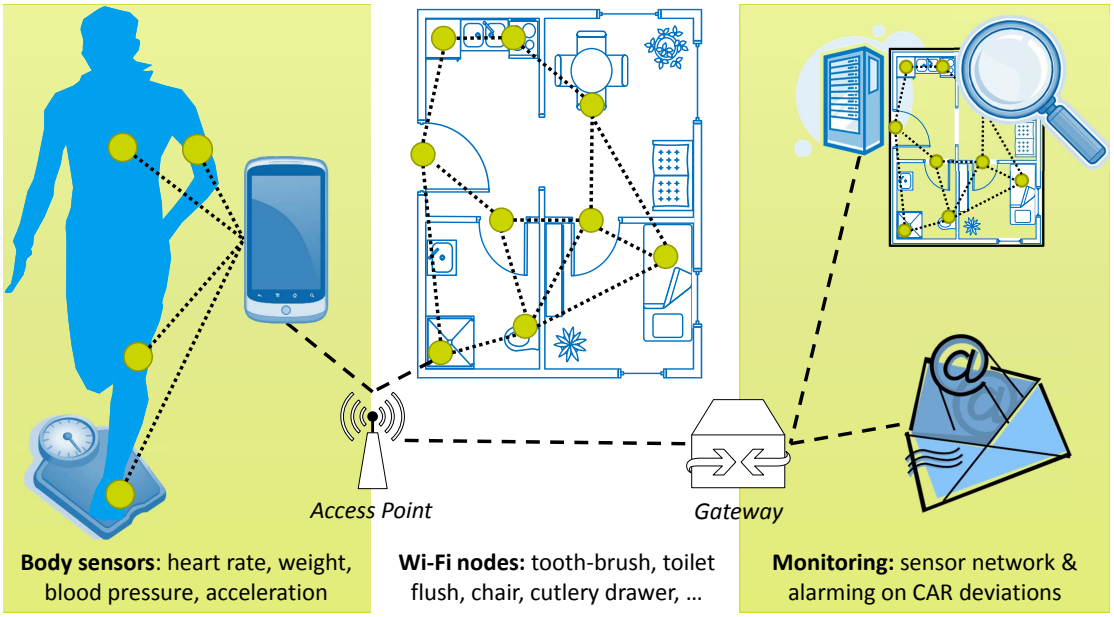


Figure 5.16: System setup of the *myElderlyCare* application consisting of Bluetooth on-body sensors and Wi-Fi environmental sensors. The environmental sensors are used for detecting deviations in the circadian activity rhythm (CAR) and alarming via e-mail. Heartbeat messages sent from the BASN allows monitoring the system's liveliness.

(a) Reed-switch at door (b) boxed PIR sensor (c) boxed ball-in-tube sensor

Figure 5.17: Wi-Fi modules with sensors attached. The first picture shows a Reed-switch attached to a door. The second picture shows a cased modules with a PIR sensor attached (black device). Figure 5.17(c) depicts how a Wi-Fi module with two AAA batteries and a ball-in-tube sensor are placed in their casing.

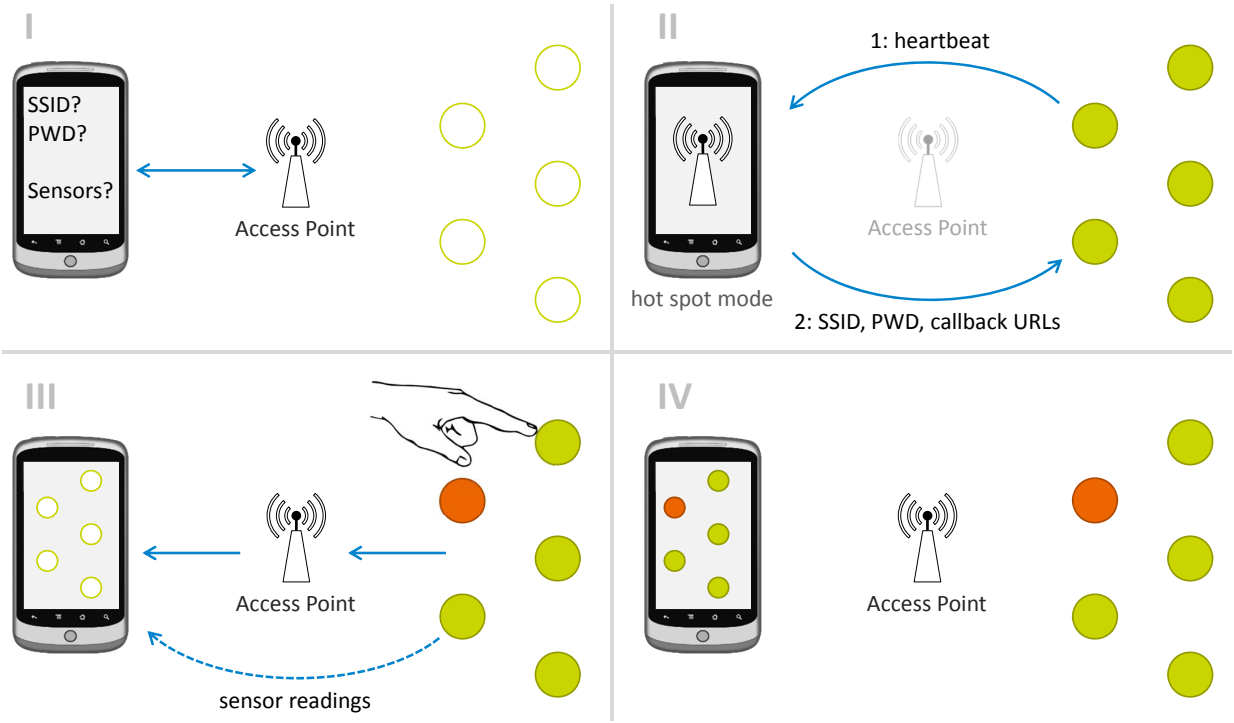


Figure 5.18: 4 steps configuration of the ambient Wi-Fi sensors: I) The SSID and password of the apartment's Wi-Fi are queried and tested by the Android application. II) When switching on the sensors, they automatically connect to a predefined AP that is provided by the phone. In response, the phone configures the nodes for the apartment's AP. III) The user is asked to trigger the sensors which send their readings to the phone. IV) The phone visualizes which sensors are working properly.

sensors. PIR sensors detect whether someone is moving within their visual range, reed-switches use the magnetic field of magnets in order to detect whether a door/window was opened or an object removed, and ball-in-tube sensors fire as soon as they are moved. Figure 5.17 depicts representatives of each sensor type. In order to monitor the system's liveliness, the smart phone as well as the ambient sensors send frequent heartbeat messages to a remote server. As soon as the phone's heartbeat message is missing, an alarm is raised. The messages sent from the Wi-Fi modules are used to update a website showing the system status including among others the battery level, last RSSI value, and the number of sensed events from each sensor.

Configuring Sensors

A goal of this project is to achieve an easy and swift system deployment and the integration in an existing environment. Instead of establishing a new network infrastructure, the system targets using the already existing one. For this, the smart phone as well as the Wi-Fi sensor nodes connect to the home's access point (AP). Our Roving Wi-Fi modules do not provide any user interface for entering the AP's SSID and password (cp. pictures of the Wi-Fi node in Figure 5.17). Therefore, we developed a four-step installation procedure as depicted in Figure 5.18. In the first step, myElderlyCare's configuration screen on the phone asks for the AP's SSID and password and checks

for correctness by trying to connect to the network. Furthermore, care givers tell the system which sensors will be installed in the apartment. In step two, the phone disconnects again from the home's AP and enables a hot spot mode with a pre-defined SSID and password. Upon switching on the Wi-Fi modules, they automatically connect to this pre-defined AP provided by the phone and start sending heartbeat messages to it. With every heartbeat message they wait for a response which includes a JSON object for sensor configuration. This response allows among others to configure a new AP and the callback URLs for heart beat messages and sensor readings. This JSON object is used to configure the apartment's AP on all nodes and to set the correct callback URLs. In step three, the configuration is done and the phone re-connects to the AP. In this step, the user is asked to interact with all objects to which sensors are attached. Based on the information about the target configuration given in the first step, the application indicates in step four which sensors are online (green circle) and which sensors need to be reconfigured (orange circle). With help of this configuration procedure, sensors can be swiftly programmed and their working order validated.

Generating Sensor Events

After installation, detected events are sent to the smart phone. In order to receive those readings, myHealthHub opens a server socket listening on port 8080. This server socket is implemented as a passive Sensor Module (cp. Section 4.1.2) within the Sensor Module Manager. Since the Wi-Fi nodes do not know to which object they are attached, they only transmit their MAC address, a timestamp, and a true or false value as an HTTP POST message to myHealthHub's server socket. The mapping to an object and a room is done within the Sensor Module. This allows changing sensors without the need of re-programming them. An XML file provides information about the sensor type (i.e., PIR, Reed, ball-in-tube), the room and the object to which a sensor is attached. Based on this information, the Sensor Module generates `OccupancyEvents` informing that an object was used or that the user is within a PIR sensor's coverage range. By using XML files, the sensor configuration is easily adapted to new requirements.

System Monitoring

For monitoring the Wi-Fi node's liveliness, each sensor node regularly (i.e., every hour) sends a heartbeat message including the network's signal strength, the node's battery level, the number of sensed events and statistics about the node's up-times to a remote server. By responding to that message, the server is able to re-configure the sensor nodes allowing to modify the callback URLs for sensor readings and heart beat messages, the reporting frequency, the AP, and further adjustments. It allows to remotely monitor and configure the system without the need of being on-location. The smart phone's myElderlyCare application also sends frequent heart beat messages to a remote server including information about the last GPS position and the battery status. In case heart beat messages are missing, an alarm is raised.

Listing 5.1: Example rules for detecting whether someone entered the appartment extracted from an XML file of application- and environment-specific rules. Those rules are interpreted by our simple rule engine. If the Reed switch at the door fires within 5 seconds after the PIR sensor outside the apartment measured movement, the action *entering* is taken.

```
<extendedrule>
    <id>5</id>
    <description>Person entering an apartment. First. PIR event outside apartment.</description>
    <event-type>[...].reading.environmental.raw.pir</event-type>
    <specific-room>entrance</specific-room>
    <specific-location></specific-location>
    <left-term>Event.sensor_value</left-term>
    <operator>=</operator>
    <right-term>true</right-term>
    <duration>5</duration>
    <ruleid>6</ruleid>
</extendedrule>
<consecutiverule>
    <id>6</id>
    <description>Person entering an apartment. Door is used.</description>
    <event-type>[...].reading.environmental.raw.reed</event-type>
    <specific-room>entrance</specific-room>
    <specific-location>door</specific-location>
    <left-term>Event.sensor_value</left-term>
    <operator>=</operator>
    <right-term>true</right-term>
    <action>entering</action>
</consecutiverule>
```

5.4.2 Activity Monitoring

So far, we mainly focused on services provided by the myHealthHub middleware. The actual myElderlyCare application provides in addition to basic health monitoring the detection of activities of daily living and an alarm in case an activity is missing. For gathering the required information, it subscribes to blood pressure, accelerometer, heart rate, and weight events as well as to `OccupancyEvents`. Based on the occupancy information, activities of daily living are derived. Since some activities consist of a combination of multiple sensor readings, we decided for implementing a simple rule engine that allows expressing simple Event Condition Action (ECA) rules. Those rules are declared in an XML file which contains simple and extended rules. Simple rules are used for instance for detecting coffee consume: as soon as the sensor attached to the coffee machine in Figure 5.19(b) fires, it is assumed that coffee will be consumed. Extended rules are used if multiple readings are required for deriving an event such as entering versus leaving an apartment. Listing 5.1 shows an example of an extended rule that derives whether someone is entering the apartment: In case the *PIR* sensor in the *entrance* area sends a *true*, the rule with *ID 5* enables rule with *ID 6* for *five* seconds. If within those five seconds the *Reed* switch at the *door* in the *entrance* sends a *true*, the activity *entering* is derived, and a corresponding event generated. We created several rules for entering and leaving an apartment, eating, toileting, tooth brushing, taking a shower, airing, and working. The system behavior is adapted by selecting an XML file containing the location-specific rules. The human-readable format of the XML files allowed us to swiftly modify rules on the phone which eased the deployment process in case something changed.

(a) ball-in-tube at chair

(b) ball-in-tube at coffee machine

(c) ball-in-tube at kettle

Figure 5.19: Wi-Fi Sensor Modules attached to a chair, a coffee machine, and a kettle.

5.4.3 System Evaluation

The system was installed in three different apartments. The first evaluation was done as a prove of concept without the phone-aided configuration phase. Figure 5.17(a) shows an installation on the apartment's front door using a Reed-switch. For this first installation, the sensors were un-boxed and deployed in an old apartment with thin wooden walls in an area with only a few wireless networks that could interfere. The overall system performance was reliable and most of the performed activities were detected. For the next evaluation, the sensors were put a white cardboard boxes as shown in Figure 5.17(c) and Figure 5.19. When testing the phone-aided configuration as described in Section 5.4.1, we realized that the reconfiguration of an access point only works for networks with WEP encryption. For the mostly used and more secure WPA/WPA2 encryption, the Wi-Fi nodes cannot be reprogrammed which means that the access point needs to be flashed directly on the Wi-Fi module. Since this would hinder a swift deployment, an access point was configured to bridge the network of the Wi-Fi modules with the apartment's network. By adding an additional access point to the system, it still gains from the existing network infrastructure without requiring it to use only WEP encryption. With this new setup, the sensors were tested in a larger apartment with thicker stone walls and a higher amount of interfering networks. In this second deployment, we did not manage to connect all sensors to one access point. Especially sensors that were placed in the bathroom had either very poor or no connection to the access point. As a result, object movement was detected by the sensors, but the transmission failed. Since they do not provide any re-transmission, the readings were lost.

In a last setup, we placed the sensors and the access point that way that they were well connected and chose an almost free Wi-Fi channel number. In total, 13 Wi-Fi sensors were deployed: 11 ball-in-tube sensors and 2 PIR sensors. Figure 5.19 shows three ball-in-tube sensors deployed on a chair in the living room, a coffee machine in the kitchen, and a kettle in the kitchen. Figure 5.17(b) depicts the PIR sensor that was deployed in the bedroom. The second PIR sensor was deployed

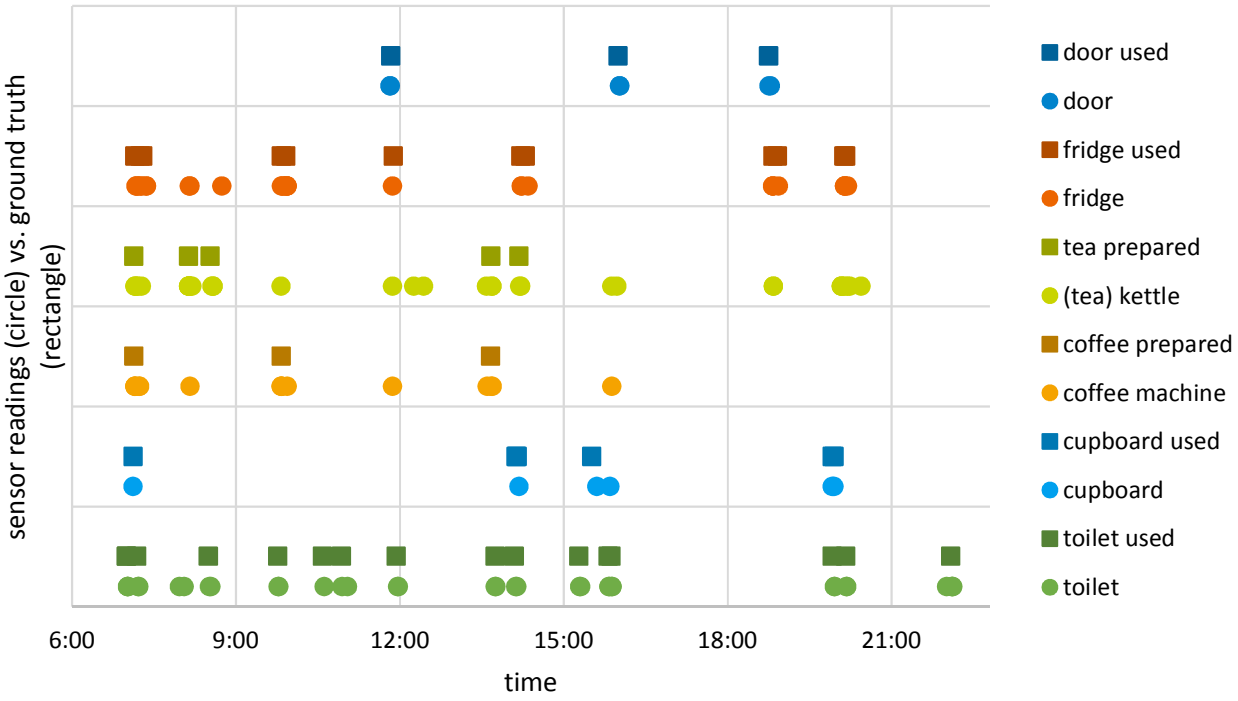


Figure 5.20: Classification of events based on simple rules that map the interaction with an object to an activity. The rectangles are the ground truth data given by the participants and circles are sensor readings. The overall precision is 68.6% and the overall recall 100% for a 10 minutes suppression of re-occurring events.

in the floor for detecting whether someone enters or leaves the apartment (cp. extended rule in Listing 5.1). After the deployment process, the sensors were only monitored using the monitoring website. Both PIR sensors caused problems during the evaluation phase: the PIR sensor in the bedroom was not sensing properly and detected movements even if no one was in the apartment and the other PIR sensor lost the network connection very soon. Therefore, we will only focus on the ball-in-tube sensors. For a better overview, we distinguish between sensors used for simple rules and those used for the extended rules that combines multiple readings.

Figure 5.20 depicts the sensor readings and the performed activities (ground truth) in case of simple activity rules. From top to bottom, the sensors were placed at the entrance door, the fridge's door, the kettle (cp. Figure 5.19(c)), the coffee machine (cp. Figure 5.19(b)), a cupboard in the bathing room, and on the toilet lid. An activity is assumed as soon as the corresponding sensor fires. For instance, as soon as the fridge is used, the activity fridge used is derived. After an activity is derived, the corresponding rule is blocked for ten minutes in order to suppress multiple emissions. Both sensors deployed in the bathroom worked very reliably. The other sensors triggered the generation of an activity event without the activity actually being performed. If an activity was performed, it is was always detected which leads to a recall of 100%. There are several reasons for a sensor reading without a corresponding activity performed: a) the participants could have missed the activity, b) the objects are moved without performing the desired activity[6], c) the sensor mistakenly measured a movement without any user interaction. Since the fridge, the

[6] The coffee machine and the kettle were next to each other. Sometimes the other object was moved without using it (cp. coffee machine reading on Figure 5.20 at 8:00pm).

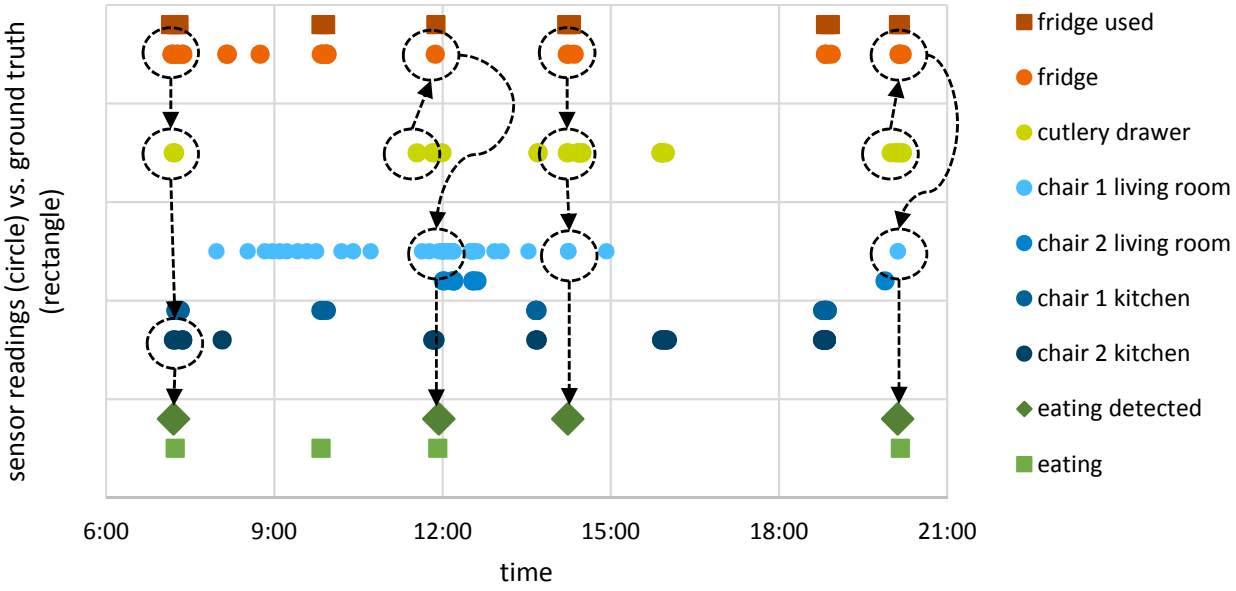


Figure 5.21: Detection of the activity *eating* as an extended rule: the fridge and the cutlery drawer are used up to 45 minutes before one or more of the four chairs are used, the activity *eating* is derived. At 9:50am the participant was eating without using the cutlery drawer before and, thus, the activity was not detected. Chair 1 in the living room was occupied throughout the day.

coffee machine, and the kettle stood next to each other and sensed movements at the same time, we believe that the cases b) and c) occurred often. Object-specific thresholds could help to avoid mistaken detections. In total, an overall precision of 68.8% is achieved.

In Figure 5.21, the results of two extended rules for detecting whether someone is eating are depicted. The extended rules consist of three linked rules: if the fridge is used, a second rule that waits for an incoming cutlery reading is activated for 10 minutes. If this rule fires, a third rule that injects an *eating activity* event upon one of the four chairs is moved is activated for 45 minutes. The second extended rule swaps the first and the second rule allowing to first use the cutlery drawer an then the fridge. The detection of more complex activities such as eating or food preparation require a careful selection of exploited objects and rules. The combination of the fridge, cutlery drawer, and a chair detected the main meals but missed a snack at 9:50am and mistakenly detected a meal in the afternoon. The snack was not detected because no cutlery was used, but its usage is required for detection. The meal in the afternoon was detected because a tea was prepared (cp. Figure 5.20) which required both a spoon from the cutlery drawer and milk from the fridge. In addition to this, chair number one was used throughout the day. The combination of all three events mistakenly resulted in a detected eating event. For a more precise selection of object and rules, further investigations are required.

Due to the fact that WPA and WPA2 encryption are not fully supported as well as the network connectivity causes problems to the Wi-Fi modules, we decided against further investigations with those sensors. From a middleware perspective, this application and especially the passive Sensor Module for Wi-Fi sensors showed how Web-based sensors are integrated in myHealthHub and how easily corresponding events are created. XML files help to swiftly adapt the system to changes in

the environment. A simple rule engine provides activity detection based on simple events or a causal series of events.

The integration of Bluetooth modules that are attached to objects and directly send their readings to the phone is ongoing work. By using Bluetooth technology, readings are sent only within the Personal Area Network (PAN) of a person which saves energy and makes the deployment even faster. In addition to this, Bluetooth modules are usually smaller than Wi-Fi modules, allowing in combination with the lower energy requirements, developing sensors with a smaller form factor. Nevertheless, Wi-Fi modules remain interesting in case environmental status information outside the user's PAN is required. For instance, sensors at doors, the oven, an iron could be used for warning the user that they are open/on when he/she is leaving the apartment.

This application demonstrated how ambient Wi-Fi sensors are installed, connected, and a mapping from the node's ID to actual objects and rooms was established in myHealthHub. Furthermore, we showed how simple ECA rules allow deriving daily activities from augmented objects. By sending heartbeat messages to a server in the third tier, both the smart phone as well as the ambient sensors were monitored and problematic situations detected. Since the ECA rules consume `OccupancyEvents` that are generated by a Sensor Module running in myHealthHub, changing the sensors from Wi-Fi to Bluetooth sensors would not require changing the application. The new Sensor Module would produce the same type of events and, thus, changes in the sensing technology would not require the myElderlyCare application to adapt.

6 System Evaluation

This chapter evaluates four important aspects of our system: I) the system's energy consumption and its feasibility for health monitoring throughout the day, II) the messaging performance of both types of Message Handler implementations (open versus application-specific channels), III) the energy savings introduced by using event transformations, and IV) the overall user experience regarding hardware and applications. A detailed analysis of the individual applications running on top of myHealthHub were already given within the application section. Nevertheless, the details on the user experience are based on the feedback we received from participants using the applications.

During the development of both the middleware and the applications, we installed and tested them on several Android devices. Table 6.1 summaries all devices on which myHealthHub was installed. None of those devices required any adaptations of the middleware in order to install it on the device. Even the Google Glass and the Intel-based Motorola RAZR i phone ran myHealthHub out-of-the-box. The Android versions given in the last column are the currently installed versions. The bold entries (#3, #4, #5, #7, #8) in Table 6.1 were used for the following performance evaluation. Since the software is not available on the Android Play app store, it is either installed by copying and starting the myHealhHub.apk file or by connecting the phone to a PC running the Android developer kit and starting it from there.

This chapter is structured as follows: In the next section, the system's energy consumption is analyzed which shows that the combination of a phone and some body sensors is powerful enough for health monitoring throughout the day. Furthermore, the impact of additional wireless communication is considered within this section. In Section 6.2 the Message Handler's messaging

#	Device	Processor	Memory	Android Version
1	Google Glass	2x1 GHz	1024 MB	4.0.4
2	Google Nexus 4	4x1.5 GHz	2048 MB	4.3
3	**HTC Desire C**	1x0.6 GHz	512 MB	4.0.3
4	**HTC Desire S**	1x1.0 GHz	768 MB	2.3.5
5	**HTC One V**	1x1.0 GHz	512 MB	4.0.3
6	HTC One X+	4x1.7 GHz	1024 MB	4.1.1
7	**Motorola Milestone**	1x0.6 GHz	256 MB	2.2
8	**Motorola RAZR i**	1x2.0 GHz	1024 MB	4.1.2
9	Samsung Galaxy 10N Tablet	2x1.0 GHz	1024 MB	4.0.4
10	Samsung Galaxy S III	4x1.4 GHz	1024 MB	4.1.2

Table 6.1: List of Android devices that are compatible to the myHealthHub middleware. Devices with a bold name are used for the system evaluation.

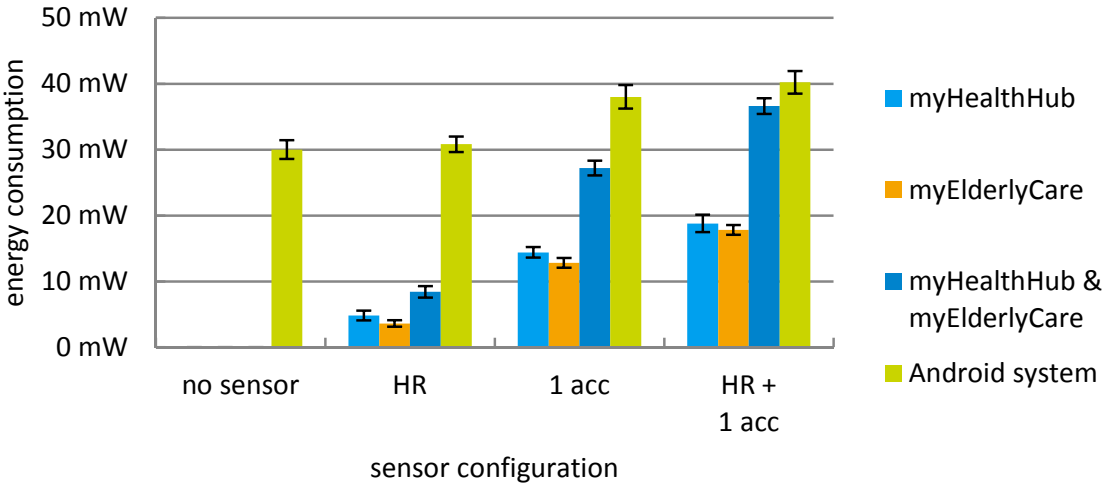


Figure 6.1: Energy consumption of the myElderlyCare application for different sensor configurations. By connecting a heart rate (HR) sensor, the application monitors the wearer's heart rate. Connecting an accelerometer (acc) enables the recognition of daily activities (cp. Section 5.1.2).

performance and the impact of having application-specific communication are discussed. Section 6.3 compares the energy consumption of having an ECG and HR sensor connected to the phone to a solution using only an ECG sensor and an event transformation to HR events. Before we summarize the findings of this chapter, a reflection on user feedback is given.

6.1 Energy Consumption

The operating time of a body and ambient sensor network system is critical for its usability and user acceptance. For health care, many systems such as the examples mentioned in the introduction are worn throughout the day and charged at night. Therefore, a health monitoring system should operate for at least 12 hours. We analyzed the energy consumption of our more power consuming myElderlyCare application (cp. Section 5.4) that monitors heart rate and performs a near real-time activity recognition based on body and ambient sensors. The energy consumption values used in this chapter are taken from the PowerTutor application [165, 116].

Figure 6.1 depicts the energy consumption of four different sensor configurations for the myElderlyCare application running on phone #4 (cp. HTC Desire S in Table 6.1). The heart rate sensor (HR) emits 59 bytes per second including the heart rate, a battery charge indicator, and a list of timestamps of the last heart beats. The Porcupine accelerometer (acc) sends only six bytes per second but triggers a daily activity recognition which explains the higher energy consumption. It can be observed that with an increasing message workload, the energy consumption of both application and the middleware as well as the Android system increases. The energy overhead of the Android system for the additional communication tasks is about 30%.

Since the energy consumption of single processes does not indicate how long a device can operate, we checked remaining battery power of a phone at the end of a day. Again, the more energy consuming myElderlyCare application with a setup of 6 ambient sensors for detecting

Week Day	Start	Stop	Duration	Remaining battery power
Monday	9:45	22:15	12 hours 30 minutes	30%
Tuesday	7:20	21:20	14 hours 00 minutes	20%
Wednesday	9:45	22:00	12 hours 15 minutes	50%
Thursday	6:10	22:20	16 hours 10 minutes	15%
Friday	6:10	21:20	15 hours 10 minutes	20%

Table 6.2: Remaining battery power of the phone after a day of activity and health monitoring. On Wednesday the accelerometer ran out of battery.

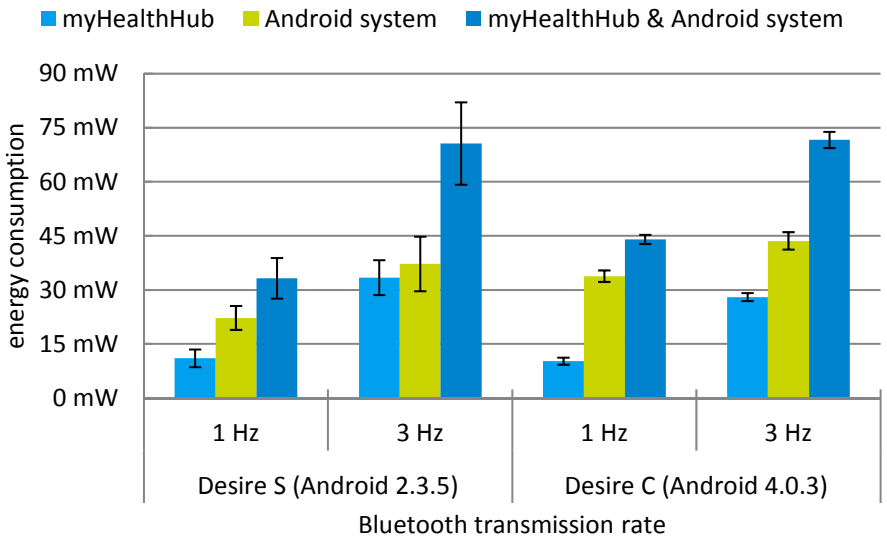
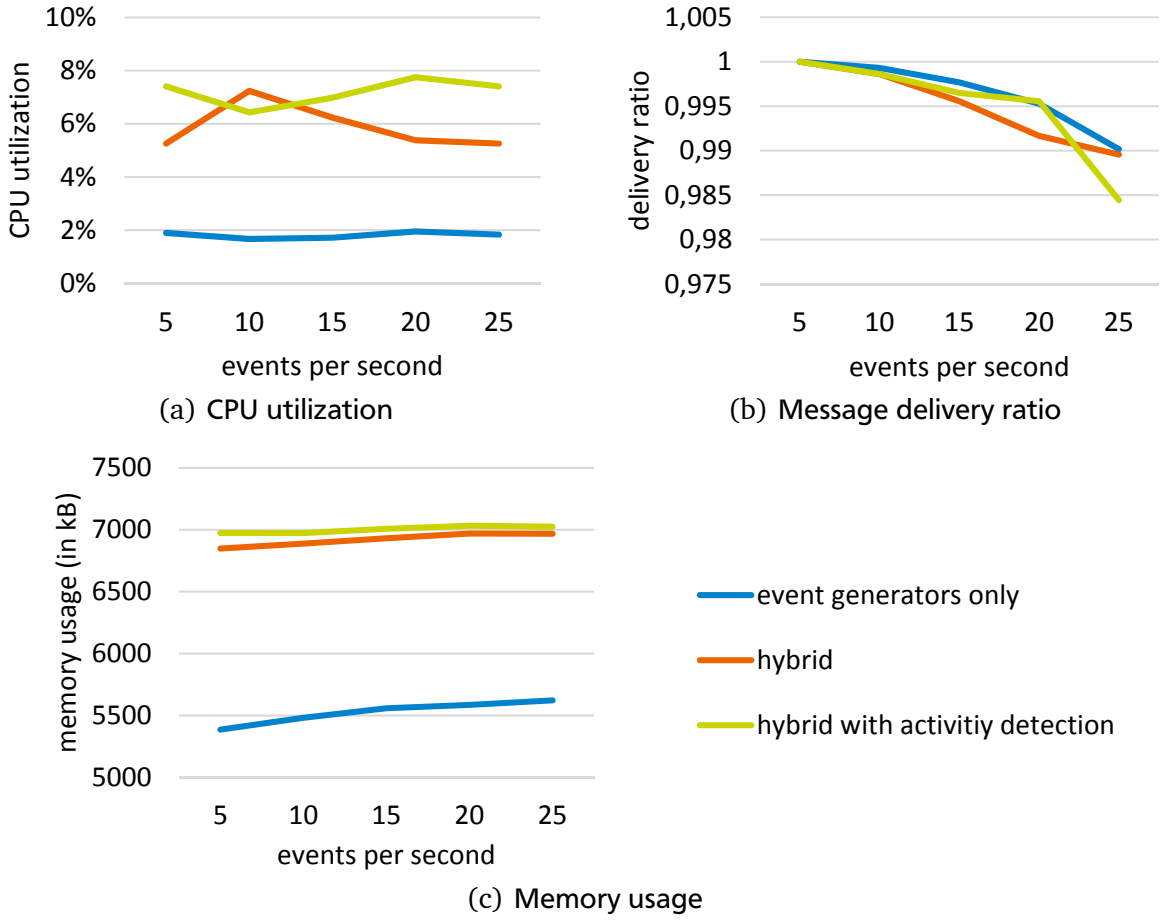


Figure 6.2: Energy consumption for two different Bluetooth transmission rates (1 Hz versus 3 Hz). Increasing the transmission rate drastically increases the overall energy consumption.

interactions with the environment, a heart rate sensor, a scale, a blood pressure sensor, and an accelerometer for continuous activity detection was used. Weight and blood pressure readings were taken in the morning and in the evening and the accelerometer was worn all the time. The heart rate sensor was worn only sporadically. Table 6.2 shows the results after five days of monitoring with phone #3. Even with the additional overhead of performing near real-time activity detection, the system has still 15% remaining battery power after 16 hours of operating. This is sufficient for most monitoring tasks. On day three, the accelerometer ran out of power and, thus, was not connected all the time. The 50% remaining battery power compared to the 30% of day one, both after 12 hours, show the impact of wireless communication on the battery life. In addition to this test, Figure 5.8 on page 88 depicts the result of one day of heart rate and activity recognition performed by the myFitnessDiary application.

The previous table already showed that wireless communication is costly in terms of energy consumption. A benefit of using a layer between the sensors and the applications is the saving of individual and redundant communication of multiple applications with the same sensor. Figure 6.2 shows a comparison of the energy consumption between having a Bluetooth sensor sending a reading only once a second (1 Hz) compared to sending the reading three times a second (3 Hz). This evaluation was done with phones #3 and #4 from Table 6.1. It highlights the impact of



(a) CPU utilization

(b) Message delivery ratio

(c) Memory usage

Figure 6.3: Comparison of the system's performance for: I) event generators only, II) a hybrid setup, consisting of an increasing amount of event generators as well as a Bluetooth heart rate sensor and a Bluetooth accelerometer, and III) the hybrid setup plus activity recognition as described in Section 5.1.2.

wireless transmissions. For the Android 4.0.3 phone, the overall energy consumption was increased by 63%. For the Android 2.3.5 phone, the increase of 115% in power consumption was even more dramatic. This figure emphasizes the strength of using a mediating middleware that distributes sensor readings to multiple applications instead of requiring multiple re-transmissions of sensor readings.

6.2 Messaging Performance

This section evaluates the messaging performance of myHealthHub's Message Handler implementations for both open communication channels and application-specific communication. For the application-specific communication, we distinguish between an *explicit* and a *setPackage* mode which specifies how event receivers are addressed. Before measuring the performance of the individual message passing implementations, the impact of wireless communication and the impact of the message size on the CPU utilization, delivery ratio, and memory usage are analyzed. This is important for classifying the results obtained from our benchmarking approach. For all tests, event producers send with 1 Hz frequency. In order to increase the workload, the number of event

producers is increased. Furthermore, we implemented a benchmarking application that subscribes to the events and counts every incoming event. Those counts are used for calculating the message delivery ratio. All results consist of the average of three runs with 240 injected events per run and per event producer. Except for the hybrid test runs, wireless communication was switched off. For the following tests, the phone #4 HTC Desire C with a 600 MHz CPU and the phone #5 HTC One V with a 1 GHz CPU were used.

6.2.1 Changing Event Producers and Message Sizes

For evaluating the system's performance, the number of event producers and event consumers needs to be varied and especially increased. Android limits the number of applications running in parallel. For the phones used in that chapter, the maximum number of applications was limited to ten applications which limits the maximum number of event consumers to nine since the middleware runs as a separate application. In a real deployment, the maximum number of connected sensors and, thus, the number of event producers is usually also limited, either by the user's comfort level or by the utilized network protocol. Bluetooth, for instance, handles up to 7 active connections in parallel. Most Bluetooth sensors send their readings once or twice a second. In order to provide a higher workload than 14 events per second and a more reproducible performance analysis, we implemented event generators that are implemented as Sensor Modules and emulate sensors. This allows us to produce a higher workload than by using Bluetooth sensors. Nevertheless, a workload of more than 10 wirelessly transmitted sensor readings per second is not applicable for most medical sensor network applications since it would limit the system's run-time.

For estimating the additional overhead of Bluetooth communication, Figure 6.3 depicts test results for an hybrid setup including Bluetooth sensors and event generators compared to a setup with event generators only. For the hybrid setup, we connected a heart rate sensor and a Porcupine accelerometer to phone #4. Furthermore, we tested the impact of an activated activity recognition. Having Bluetooth sensors connected to the system significantly increases both the CPU utilization (cp. Figure 6.3(a)) as well as the memory usage (cp. Figure 6.3(c)) because of additional overhead evoked by the Bluetooth communication. The impact on the delivery ratio is low. An enabled activity recognition leads to a slightly lower delivery ratio because the detected activities are sent to the middleware which results in a higher workload. As a result, the usage of real sensors has a higher impact on the system than the emulated event generators. Therefore, benchmarking based on event generators helps to evaluate the Message Handler performance, although for a real deployment, the messaging performance will be lower. Having the results of the energy consumption for additional sensors in mind, a system with more than 10 sensors would not last for a long time. Thus, the following tests evaluate the middleware's performance, although for a real deployment, the battery power of the sensors and the smart phones are the limiting factor.

The events injected for the previous evaluation were rather small events consisting of only three (i.e., heart rate, battery level, heartbeat counter) respectively six (i.e., mean and variance per axis) Integer values as payload. Some events such as ECG events consist of larger payload. The ECG events generated from the CorScience ECG sensor have a payload of 107 Integers. Therefore, Figure 6.4 depicts the system's behavior for larger event sizes. The payload size of the injected

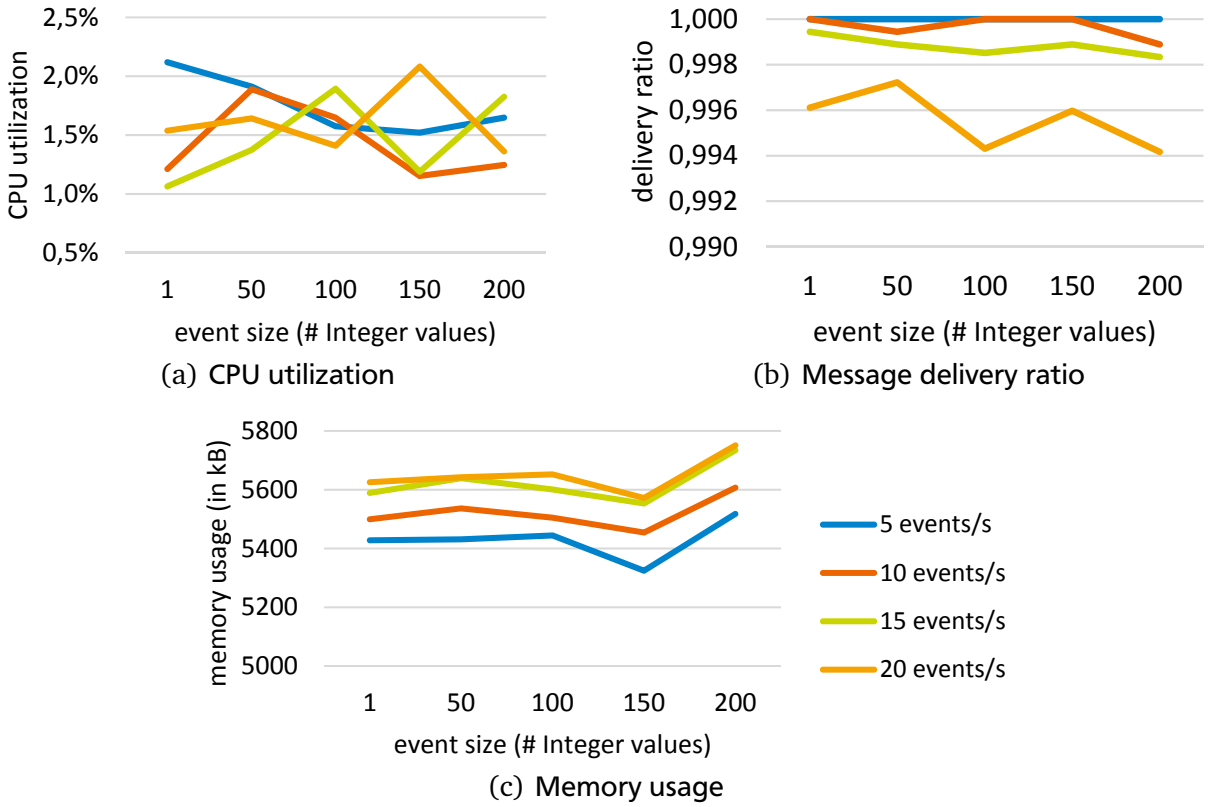


Figure 6.4: Increasing payload size from 1 to 200 Integer values per event and different workload configurations. In addition to the payload, each event consists of an `ID`, `type`, `timestamp`, `producer ID`, `sensor type`, and `time of measurement`.

events was increased from 1 to 200 Integer values and four different workload configurations were tested. For the CPU utilization (cp. Figure 6.4(a)) no effect can be observed. The middleware decides only based on the event type to which channel an event has to be sent; it does not inspect the payload. Therefore, the events payload remains unread. The delivery ratio decreases slightly which is explained by the higher amount of transferred data. This also slightly increases the memory usage. In total, increasing event sizes have only a low impact on the messaging system. For the following tests, heart rate events including an `ID`, `type`, `timestamp`, `producer ID`, `sensor type`, `time of measurement`, and 3 Integers values payload are injected.

6.2.2 Changing Number of Event Producers and Subscribers

The support of multiple applications is a major advantage of having a middleware as a layer between sensors and applications. Besides the mediation between both, it avoids the retransmission of sensor readings in case sensors connections can only exclusively be established. The achieved energy savings were already analyzed in Section 6.1. In this section, we will focus on the impact on the system's performance of multiple applications being subscribed to the myHealthHub middleware. In Chapter 3, two general approaches of message forwarding were presented: open communication channels and application-specific communication channels. Both Message Handler implementations are evaluated within this section. Furthermore, in Section 4.3 two ways

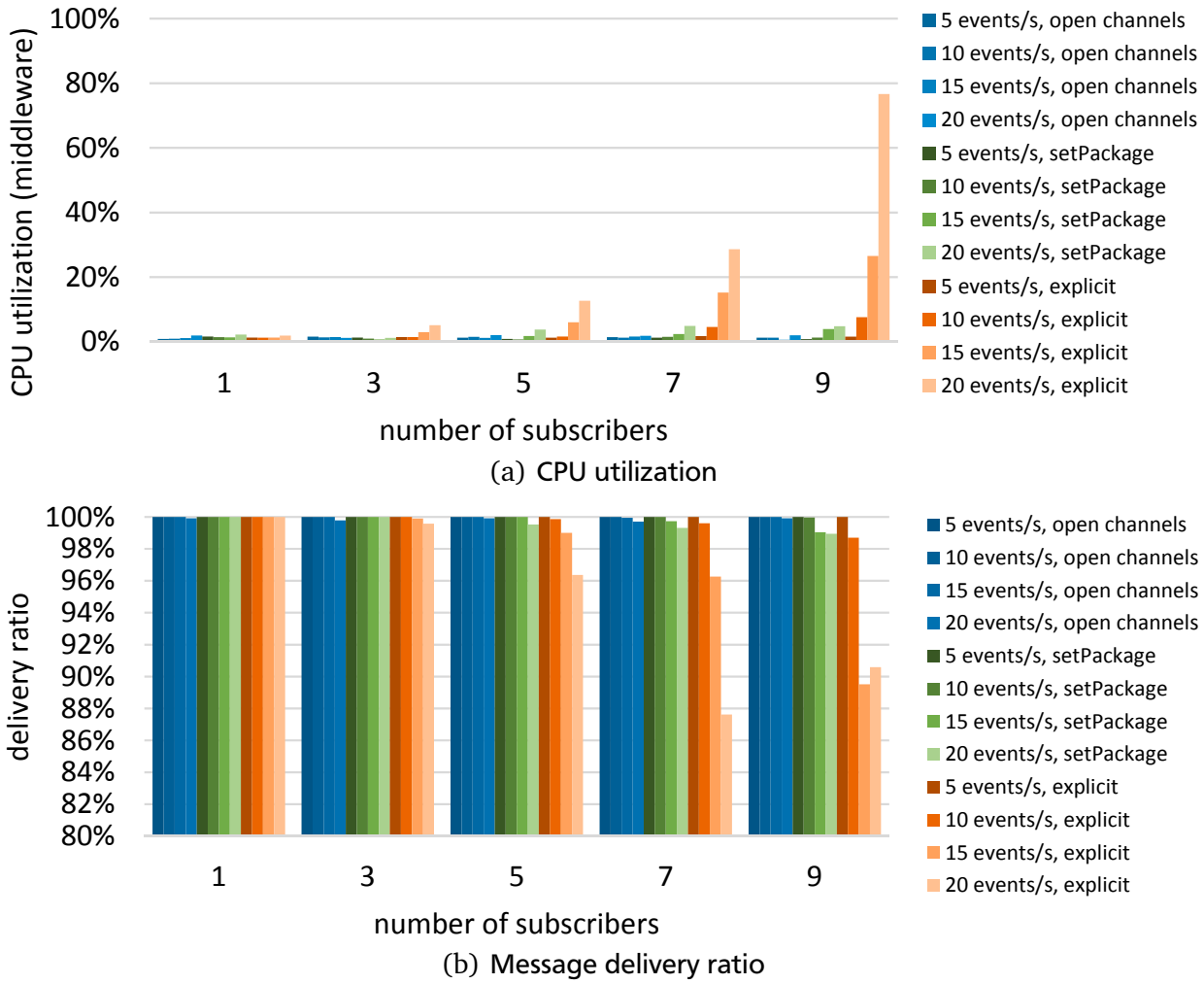


Figure 6.5: Performance evaluation of three Message Handler implementations: I) *open* communication channels, II) implicit broadcasts Intents using the *setPackage* option, and III) *explicit* broadcast Intents. For all implementations up to 9 subscribers and a workload of 5, 10, 15, and 20 events per second are tested. The explicit addressing performs the worst.

of implementing secure application-specific communication were proposed: explicit broadcast Intents and implicit Intents using the setPackage() method for defining the target application. Both implementations use the secure inter-application communication provided by Android (cp. Section 2.3.3).

Figure 6.5 depicts the CPU utilization and the delivery ratio for all three Message Handler implementations on phone #5. For each implementation, four different workload configurations (i.e., 5, 10, 15, and 20 events/s) and one to nine subscribers were tested. The CPU utilization depicted in Figure 6.5(a) shows a significant impact of an increasing workload on the implementation with explicit broadcast Intents. The same applies to the message delivery ratio shown in Figure 6.5(b). For a workload of 20 events per second and 9 applications being subscribed, the message delivery ratio drops to about 90% which is about 9% less than for the implicit broadcast Intents using the setPackage method. Therefore, we decided to discard the implementation of explicit broadcast Intents and focus only on open communication channels and the more efficient setPackage-option

for application-specific communication. In addition to this, the setPackage implementation is easier to use for application developers (cp. Section 4.3).

Figure 6.6 depicts the performance analysis for *open* communication channels using implicit broadcast Intents and for the secure application-specific communication using implicit broadcasts with the *setPackage* method for defining the receiving application. For open communication channels, an increasing number of subscribers has no influence on the CPU utilization as shown in Figure 6.6(a) since the Message Handler broadcasts events independently of the number of subscribed consumers. In case of heart rate events (as used for this evaluation), an incoming event is forwarded in three different channels along the event tree. A direct impact of an increased message workload on the CPU utilization cannot be observed. In contrast to this, the CPU utilization of the application-specific approach is influenced by both the number of subscribers and the message workload. With each additional subscriber, the Message Handler has to forward incoming events an additional time. In case of 9 applications being subscribed to heart rate events and a workload of 20 events per second, this results in a messaging workload of 180 events per second for the Message Handler. This is, compared to 60 events per second for open channels, a threefold higher workload which explains the higher CPU utilization. Figure 6.6(b) depicts the message delivery ratio for both approaches and shows that for an HTC One V phone with 1.0 GHz CPU clock rate, around 60 events per second form the upper bound for a message delivery ratio of more than 99.7% (cp. open channels, 7 subscribers, 20 events/s). For a higher workload the delivery ratio drops. In case of application-specific channels, 15 or 20 events per second and 9 applications subscribed, the delivery ratio decreased to less than 99%. Nevertheless, Section 6.2.3 shows that a faster CPU increases the system's messaging performance. In Figure 6.6(c) the memory usage of both approaches is compared. The 34.5% higher amount of memory used for the applications-specific approach (i.e., 7.4 MB on average) compared to the open channels (i.e., 5.6 MB on average) is explained by the more complex Message Handler and, more importantly, by the additionally running Transformation Manager in the second approach. The Transformation Manager provides an OSGi run-time environment which requires additional memory. For both approaches it can be observed that with a rise in message workload more main memory is required.

6.2.3 Impact of the Hardware Performance

After analyzing the system's behavior on different workload setups, Figure 6.7 depicts the performance gain in faster hardware. Both the CPU utilization and the message delivery ratio for three different Android phones is shown. The HTC Desire C (cp. phone #3 in Table 6.1) is a low-end phone with a 600 MHz CPU speed. The devices #4 (HTC Desire S) and #8 (Motorola RAZR i) are mid-range phones with 1.0 GHz respectively 2.0 GHz CPU speed. Furthermore, the devices HTC Desire C and Motorola RAZR i are Android 4.x devices whereas the HTC Desire S is an older device running Android 2.3.5. The CPU utilization of the faster phones #4 and#8 fluctuates around 2% respectively 1% for the whole test which shows that the middleware is not or only marginally affected by an increasing workload. That corresponds to the results obtained from the previous section. For the slower HTC Desire C device, the CPU utilization increases with a higher workload. More important, the message delivery ratio starts decreasing rapidly for more than 22 events per

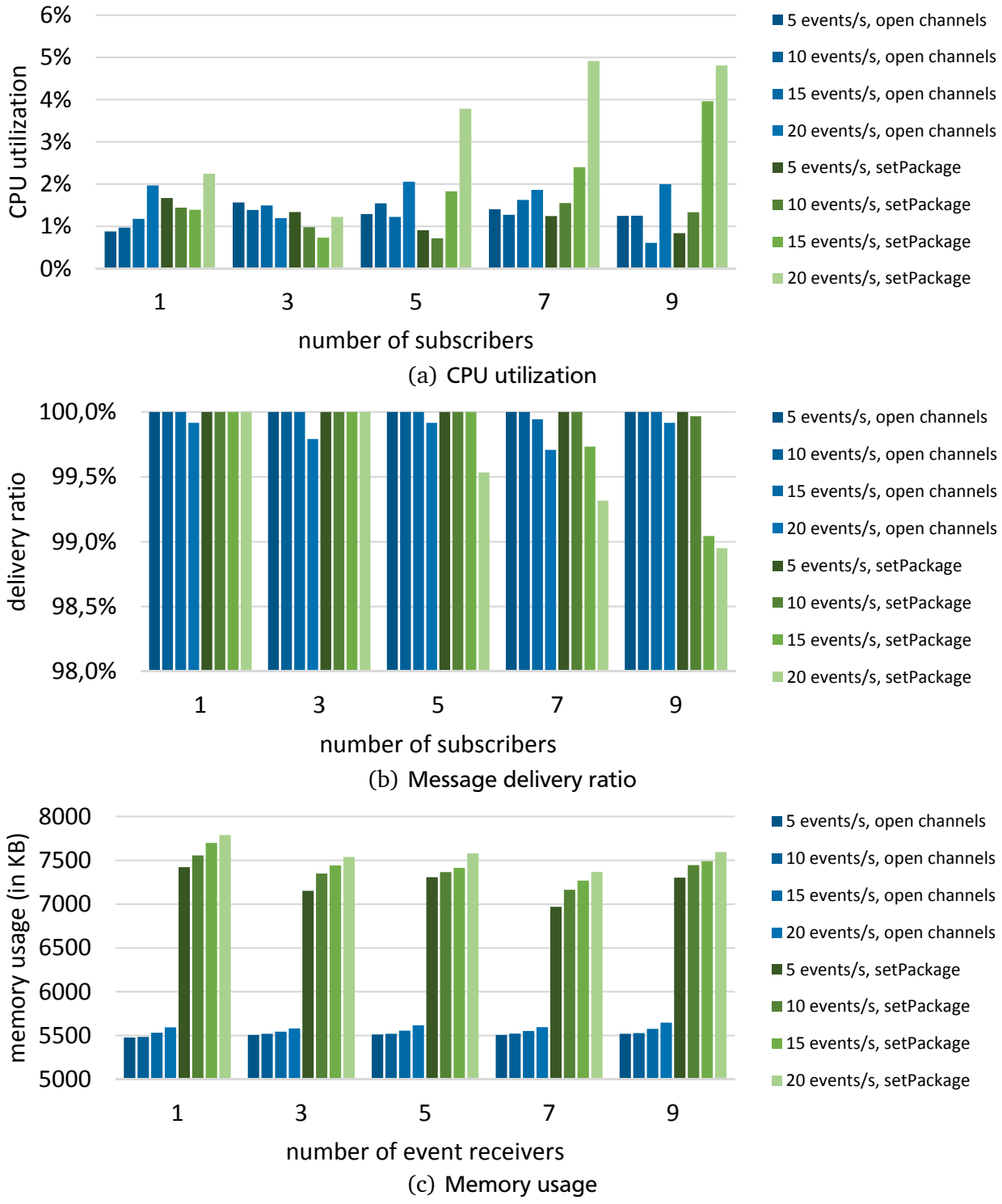
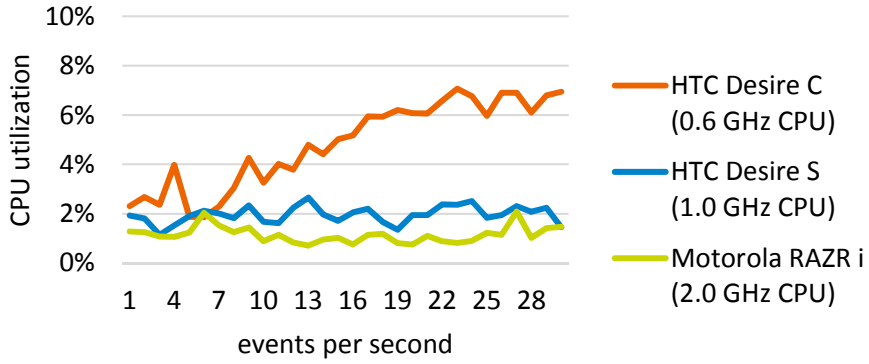
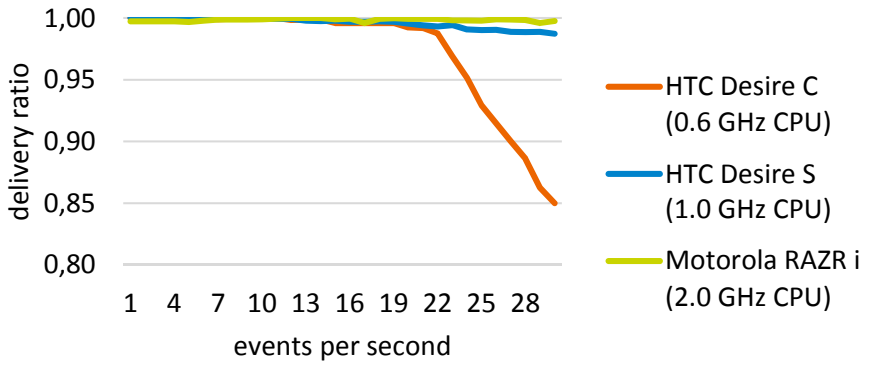


Figure 6.6: Performance evaluation of the two faster Message Handler implementations: I) *open* communication channels and II) implicit broadcasts Intents with the *setPackage* option. Again, for all implementations up to 9 subscribers and a workload of 5, 10, 15, and 20 events per second are tested. For an increasing amount of subscribers, the application-specific Message Handler (*setPackage*) is less efficient due to the higher amount of individually sent messages.



(a) CPU utilization



(b) Message delivery ratio

Figure 6.7: Performance analysis of three different phones for an increasing amount of events per second: The HTC Desire C and the Motorola RAZR i run Android 4.x and the HTC Desire S Android 2.3.5. For the slowest device, the message delivery ratio starts dropping rapidly if more than 22 events per second are injected.

second. This shows that phones have a maximum messaging throughput for broadcast Intents and once this threshold is passed, the message delivery ratio drops. The reason for the rather low messaging throughput of, for instance, the HTC Desire C device might be that Android broadcasts are not introduced for more frequent inter-application communication. On the other hand, 22 events per second is more than what a usual BASN application produces. For our faster phones this threshold was at 60 events per second. During the evaluation of our applications, those thresholds were never reached. As a final result, the hardware performance has a significant impact on the message passing performance. For applications requiring data in a high frequency, faster phones than the HTC Desire C are recommended.

6.3 Event Transformation

Referring back to the example described in the introduction in which a rehabilitation patient has four monitoring applications running on top of a body sensor network, one application requiring ECG data and the others requiring only heart rate information, we assumed that using a single ECG sensor and transforming an ECG stream into heart rate information requires less energy than using both sensor types. In order to validate our assumption, we implemented and installed a transformation from ECG to heart rate events and compared the energy consumption of different

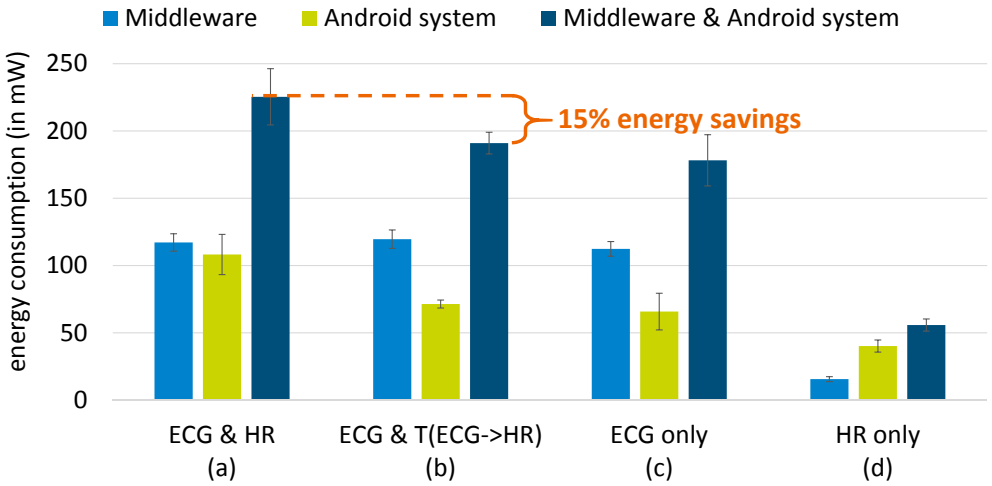


Figure 6.8: Energy consumption of different sensor constellations. Having a transformation converting ECG streams to heart rate data (b) saves more than 15 % energy compared to having both sensors connected (a). Compared to an ECG sensor (c), a heart rate sensor (d) consumes perceptibly less energy.

sensor constellations as shown in Figure 6.8. Figure 6.8 (a) shows the energy consumption in case of individual sensors for each application. In total, both the middleware and the Android system consume about 225 mW. Compared to this, the combination of an ECG sensor and a transformation that transforms ECG streams into heart rate events ($T_{ECG \rightarrow HR}$) consumes only about 191 mW and, thus, saves more than 15% of energy. In this case, the overall system lasts longer and does not require the user to wear an additional sensor. The comparison between (b) and (c) shows the additional energy overhead for performing the transformation which is less than 7.5%. Figure 6.8 (d) depicts the energy consumption for having only a heart rate sensor connected to the system. Since the Zephyr HxM Bluetooth heart rate sensor sends only 59 Bytes with 1 Hz frequency, the energy consumption is much lower than the one for the CorScience 1-lead ECG sensor sending 112 Bytes with 2 Hz frequency. The energy consumption produced by the other transformations is marginally higher.

As a remark, we have experienced that our ECG sensor is very prone to movement. As soon as the user was moving, the ECG curve started fluctuating with the result that our $T_{ECG \rightarrow HR}$ transformation was not able to detect QRS complex anymore and, thus, stopped delivering heart rate events.

6.4 User Experience

For the fitness diary application, sensors have to be attached to three positions on a subject's body (cp. Figure 5.3 on page 80). The heart rate sensor together with one accelerometer are attached to the chest. Since the heart rate sensor's strap is very comfortable and both sensors are small, these sensors are unobtrusive and attaching them is easy to do. The wrist sensor is combined with a weight lifting glove which makes it also easy to attach. Only attaching the leg's accelerometer happened to be slightly difficult for subjects because it is not clear were the sensor has to be. This was solved by labeling the strap with an arrow for the fitness trail application as shown in Figure 5.14 on page 97. Nevertheless, once we had shown how to attach the leg's sensors,

attaching it was no more a problem. Some subjects had the problem that the strap was slipping during the cardio exercises, especially during a run. For a new deployment, a dimpled rubber strap should avoid slipping. Overall, attaching the sensors was fairly easy and readjustments were not necessary. From an end-user's point of view, the myFitnessDiary seemed to be easy and intuitive to use. Except for the leg's accelerometer, the sensors are unobtrusive and wearing the system in a gym was reported to be inconspicuous. The rehabilitation application uses the same body sensors as the fitness diary.

For the fitness trail application, participants were given a questionnaire assessing the comfort of wearing the accelerometers. As a result, the wrist and ankle sensors were the best accepted ones. Most participants had difficulties wearing a sensor at the chest or the upper leg. Since the analysis on the best fitting sensor positions for detecting fitness trail activities resulted in ankle and wrist or ankle, wrist, and chest sensors, using only the ankle and wrist sensors forms a very good trade-off between system performance and user comfort. Further results of the questionnaire are that the maximal number of accepted sensors is four and comfortable sensor attachment is crucial for such a system. With respect to usefulness of the fitness trail application, the participants would have preferred to get information about how well the repetitions were performed instead of the amount of repetitions. This would be an interesting extension of myFitnessTrail.

During the development of the myEdlerlyCare application we realized that environmental sensors need to be easy to understand in terms of their capabilities and their functionality. For most people, it is not clear what a sensor is capable of, especially for the PIR sensors. PIR sensors need to be in line-of-sight in order to work properly. Often, subjects needed to be assured that this sensor is not a camera that captures pictures or videos especially when the sensors were placed in the bathroom. Furthermore, in order to install a sensor correctly, a certain understanding about how a sensors works is required. Reed sensors that operate on a magnetic field were the most difficult sensors in terms of installing them correctly. Magnetic fields are not visible and, thus, it is not trivial to see whether the sensor is within the magnetic field or not. In addition to this, those sensors consist of two parts (i.e., a magnet and the sensor) whereas the other sensors consist of only one box. As in many cases it is easy to replace reed sensors by ball-in-tube sensors, we decided to use only PIR and ball-in-tube sensors. Difficulties with PIR sensors were experienced when it came to their visual range. Regarding the ball-in-tube sensors that simply detect the movement of an object, participants were sometimes confused about where to place it. For instance, placing the sensor on the body of a coffee machine does not work. Instead, it needs to be put on a moving object such as the coffee pot. We created a manual that sketches the do's and don'ts of attaching the sensors. It helps users to understand how the sensors work and how to attach them properly. It is a first step but further evaluations are required in order to provide a system that is easy to set-up.

For demonstrating and visualizing the system's capabilities, we implemented a simple demonstrator that showed a picture of the detected activity and an animated heart that pounds with respect to the user's heart rate. While showing this demonstrator to an elderly person who was wearing the sensors, the animated heart stopped working for about three seconds. As a result, the person was shocked, because he thought his actual heart had really stopped for three seconds.

This situation made us aware that users might misinterpret data and animations. As a result, we removed the animated heart from the user interface and started displaying only a heart rate value.

6.5 Summary

Several aspects of the middleware were analyzed. Starting with the energy consumption, the system is capable of monitoring a user's heart rate and daily activities for more than 16 hours using an off-the-shelf heart rate sensor and custom-made body and ambient sensors. This is enough time for monitoring during the day and charging the batteries at night. We also analyzed the impact of adding wireless sensors to the system: tripling the amount of sensors roughly doubled the overall energy consumption. This shows that for a usual Android smart phone and Bluetooth 2.x sensors the maximum number of sensors is about 4 sensors sending with 1 Hz frequency such as our accelerometers or a heart rate sensor. For Bluetooth LE, the maximum number is expected to increase. Unfortunately, we were not able to analyze this due to some difficulties regarding Android's Bluetooth LE compatibility (cp. Section 5.3).

For evaluating the middleware's messaging performance, we tested three implementations of the Message Handler. Since the explicit addressing approach was quite inefficient, we focused on the open communication channels and the application-specific channels using the setPackage method. The rather low-end phone we used for our evaluation has a maximum number of outgoing messages of about 60 before the message delivery ratio starts dropping. Since the application-specific channels require sending messages for each application individually, the messaging workload is higher than for the open channels. Thus, the open communication channels are the more powerful approach in terms of messaging performance. A major drawback of the open approach are the lacking security features. The application-specific approach handles up to 20 events per second with three applications or up to 10 events per second with nine applications subscribed and still provides 100% delivery ratio. With respect to the energy consumption of wireless communication and a recommended maximum number of 4 sensors, the application-specific approach is still fast enough. In addition to this, our performance analysis on different hardware showed that with faster phones, the messaging performance is expected to increase with a low CPU utilization of about 1-2%.

By transforming ECG data to heart rate information, we showed a reduction in energy consumption of about 15% compared to a system with an ECG and a heart rate sensor connected. In addition to this, reducing the amount of sensors increases the user's comfort level. This example highlights one benefit of event transformations. Two further important aspects are the unit transformations converting for instance kilograms to pounds and the inferring of new event types. The additional energy consumption for unit conversions is too small to be observed. Inferring can introduce a higher CPU load depending on the type of inferring (e.g. activity detection). On the other hand, it provides an additional service which might require less resources than running an additional application. Event transformations help reducing the energy consumption, but more importantly, they make the system more flexible.

We received good feedback regarding the user experience. Nonetheless, having sensors and an application that work perfectly does not mean that users are satisfied. Applications have to

be intuitive and absolutely not misleading such as the animated heart that stopped. Therefore, we put extra effort into a clear and easy to understand application design and asked users of different age and background knowledge to use the application. Sensors have to be easy to wear, comfortable, and not stigmatizing, especially for daylong health monitoring. We therefore tried to find unobtrusive and easy to wear sensor positions such as accelerometers integrated in a weight lifting glove or attached next to the heart rate sensor.

Overall, the system worked quite reliably. Regarding the sensors, the heart rate sensors had some difficulties when the skin-sensor contact became too dry and the ECG sensor is prone to body movements. The scales and blood pressure sensors worked very reliably. Except for the leg's accelerometer which disconnected once or twice within a day, the accelerometers were dependable. In case the leg's sensor disconnected, our middleware automatically triggered a re-connection. The middleware worked reliably for time spans of several days and by using it as a remote service, users were not bothered by starting or managing an additional application.

7 Conclusions and Outlook

Our health systems are facing two major challenges: an increasing number of chronic diseases and the demographic change. A promising way to cope with these challenges is equipping people with a personal health assistant that, on one hand, provides information and motivation in living healthier and avoiding chronic diseases and, on the other hand, supports people in their daily living when they are getting older or when their chronic diseases need to be monitored. A vision stated by the Continua Health Alliance is to provide such a personalized health solution as an assembly of interoperable health devices and services [33]. In this dissertation, a middleware architecture was presented that mediates between sensors and applications and, thus, provides the basis for future health solutions that are assembled from multiple health devices and services.

The research question tackled by this work is how to mediate between the increasing number of body and ambient sensors, sensor types, and applications. For providing personalized health solutions, many different entities need to collaborate and work on a common data structure. The first contribution of this dissertation was a generic middleware architecture running on a personal device that collects raw sensor readings, abstracts from the raw readings, and provides them to applications in a common data structure. In addition, it seamlessly handles changes in on-body and ubiquitous sensor constellations and, thus, provides the best possible information basis to applications. The benefit for applications is a loose coupling to sensors which minimizes the dependence on specific sensory devices. Furthermore, it provides the support of multiple applications running simultaneously and operating on the same set of sensors. All entities of the system are allowed to contribute which allows a high cooperation among entities.

When dealing with health information, security and privacy become an issue. A system that mediates between entities of different origins has to provide mechanisms to control the data dissemination within the system. Therefore, application-specific communication channels that provide tailor-made information dissemination were introduced as the second contribution. Applications need to subscribe to the information they are interested in. If the user or the care giver does not grant the dissemination of a specific information type, it is not forwarded. Furthermore, subscriptions are used to switch off unrequired sensors which saves valuable resources especially in the area of sensor networks.

Monitoring requirements change and, thus, new applications and sensors are connected to the system. Furthermore, sensors break and might be replaced by another sensor or a newer type of sensor. Those cases place new demands on the mediating middleware. For adapting the system to new requirements at run-time, an on-the-fly loading of transformations was introduced as the third contribution. Transformations allow converting sensor readings from one type to another such as body weight in kilogram to pound or an ECG stream to heart rate values. In addition to that, event transformations provide the derivation of new information types such as detecting a

user activity based on acceleration data. Since no user interaction is required, adaptations to new requirements are seamlessly handled at run-time.

For evaluating our system and demonstrating its capabilities, we have developed several applications operating on sensor data from on-body and ambient sensors. Real-time activity recognition provides new modalities to care givers that help to better interpret health measurements than with conventional systems. In a setup of a smart phone and on-body and ambient sensors for monitoring a person's physical and physiological state as well as activities and interaction with the environment, daylong health monitoring is provided. This should be sufficient for most telemedical applications.

In summary, we developed a system that fulfills the following requirements on a mediator for future health care solutions as listed in Section 1.2:

- **Multiple Application Support**: The proposed middleware serves multiple (preventive) health care applications (cp. Section 5) simultaneously while showing good performance results (cp. Section 6.2).
- **Support of Different Requirements on Data Representations**: By introducing Event Transformations (cp. Section 4.4), different application requirements are met and energy is saved (cp. Section 6.3).
- **Application-specific Security Requirements**: The combination of a security manager and application-specific communication channels provides control over the data dissemination (cp. Sections 2.3.3, 3.4, and 4.3).
- **Changing Application and Sensor Setups**: The support of independently running applications (cp. Sections 4.1 and 5) and the abstraction from individual sensors (cp. Sections 3.2, 3.5 and 4.1) allow swift adaptations to changing requirements.
- **Handling of Changing Sensor Constellations**: Changes in on-body and ambient sensor constellations are automatically handled and the best possible information basis is presented to the applications (cp. Section 5).
- Furthermore, an evaluation of the energy consumption (cp. Section 6.1) emphasizes the system's capabilities for full-fledged **health monitoring throughout the day**.

7.1 Outlook

Sensor technology is evolving and leads to an increasing amount of ubiquitous sensors as well as body sensors integrated in clothing and accessories. As sensors are becoming more unobtrusive, the stigma of being monitored vanishes and, therefore, the willingness of using sensory devices for personal assistance and safety increases. Furthermore, almost every person will have a personal device for daily assistance such as today's smart phones or enhanced objects like the Google Glass. In order to provide the best possible service to its user, those devices will be well-connected and, thus, can easily connect to on-body and ambient sensors. With every new sensor, sensor

type, and hardware platform, the opportunities increase, but so does the hardware diversity. For providing users and application developers a plug-and-play like system behavior that allows for easy addition of new sensors and applications and, thus, to adapt the system to a user's needs, mediating between the diverse entities is required. In our work, we provide a first step towards a flexible software platform that allows the composition of assistance solutions based on different hardware and software in a plug-and-play manner.

The integration of sensor data from on-body and ambient sensors should be as simple as connecting to today's Internet services. By providing this service to application developers, new application types will benefit from detailed information about a user's context and, as a result, a new generation of assistance solutions will arise. In order to provide such a seamless integration of applications and devices, we see four possible extensions of our approach:

Adaptive Device Driver Model

In our work, a mechanism for loading and installing missing functionality based on required and provided event types was presented. This mechanism takes advantage of the introduced abstraction layer. In case of new hardware, not only adaptations based on this abstraction layer might be required but also new modules that translate the raw sensor data into data abstractions such as events. Based on device identifiers like device and service descriptors, manufacturer information, and MAC addresses, missing device drivers can be queried and downloaded, similar to the presented event transformation approach. This would allow for the introduction of new device types and retrofitting, while avoiding re-installation of the middleware.

Multiple Data Representations

Different applications might operate on different data abstractions. Our system is not limited to a specific data representation but thus far it operates only on a single data representation. In order to remove the limitation of a single representation that might hinder swift adaptations to new applications and developments, the mediator could operate on multiple data representations simultaneously. Event transformations could then mediate between those representations. Such a system would overcome the drawbacks of having to agree to a fix set of data types.

Sensor Configuration

As sensory devices become more powerful and flexible, they might provide certain configuration parameters. An ECG sensor, for instance, could provide different sampling rates and an accelerometer could support calculating different feature sets. In order to save energy, future sensors should be configured to use as little energy as possible but still meet the information requirements of all applications operating on them. Since the middleware knows the application's individual requirements and directly connects to sensors, performing the sensor configuration is an interesting extension of our middleware.

Multi-Platform Support

Although many of the presented ideas can easily be applied to other hardware and software platforms, we mostly focused on off-the-shelf sensors and implemented the middleware on an Android system. Some parts of the system such as the secure inter-application communication uses Android-specific techniques which are not provided by other systems such as iOS. Porting the system to another operating system might require new approaches for secure application-specific communication channels. A possible solution would be encrypted communication over the system's network stack. In principle, this technique should work on any system but encryption will introduce additional overhead that has to be analyzed.

7.2 Epilogue

For health care solutions to make full use of this technology, new clinical studies need to be conducted and new monitoring parameters defined and discovered. Most of today's health care methods rely on the traditional medicine using only a little continuous real-time monitoring outside the hospital. As soon as new monitoring parameters are found and unobtrusive sensors developed, new health monitoring solutions will arise that allow us to deal better with chronic diseases, multi-morbidity, and the demographic change. In our vision, physicians will simply equip patients with the required sensory devices and set the monitoring parameters. The future health care system will then automatically deploy and configure the required applications.

A. For Developers to Use the Middleware

This chapter describes how Android applications connect to our middleware, subscribe to event types, and how to send and receive events using the middleware. In the following, we assume that the middleware implementation, called *myHealthHub*, is installed on the system and the `myHealthHub.jar` library is referenced.

Connecting to the Middleware

Before an application can subscribe to event types, it needs to bind the myHealthHub `Remote Service` (cp. Section 2.3.2) as shown in Listing 7.1. For this, a corresponding Intent (line 1) needs to be instantiated and a `Service Connection` (line 2: `myHealthHubConnection`) provided.

Listing 7.1: Binding the myHealthHub `Remote Service` and starting the middleware.

```
1 Intent myHealthHubIntent = new Intent(IMyHealthHubRemoteService.class.getName());
2 bindService(myHealthHubIntent, myHealthHubConnection, Context.BIND_AUTO_CREATE);
```

The provided `Service Connection` allows the detection of an established service connection as well as the lost of the service connection. An implementation is shown in Listing 7.2: it displays on the user interface whether the application is currently connected or not.

Listing 7.2: Monitoring the Android `Service Connection` to myHealthHub.

```
1  /** Interface for monitoring the state of an application service. */
2  private ServiceConnection myHealthHubConnection = new ServiceConnection() {
3      public void onServiceConnected(ComponentName name, IBinder service) {
4          // I am connected to myHealthHub
5          setTextInTV(R.id.connectedToMHH, "yes", Color.GREEN);
6      }
7
8      public void onServiceDisconnected(ComponentName name) {
9          // I lost connection to myHealthHub
10         setTextInTV(R.id.connectedToMHH, "no", Color.RED);
11     }
12 };
```

Listings 7.1 and 7.2 demonstrated how to connect to the middleware and how to monitor myHealthHub's liveliness.

Subscribing to Event Types

After an successful start of the middleware, applications have to subscribe to their required event types. Listing 7.3 depicts how to create a subscription message to blood pressure events: the

Android method `getPackageName()` (line 2) returns the unique package name of the subscribing application which is used for the secure communication as described in Section 4.3.

Listing 7.3: Creating a subscription to blood pressure readings.

```
1 Subscription mySubscription = new Subscription(     // create a subscription event
2     getEventID(), getTimestamp(), "BloodPressureApp", getPackageName(),
3     SensorReadingEvent.BLOOD_PRESSURE);
```

After the subscription message is created, it needs to be sent to myHealthHub. For this, the subscription is encapsulated in an Android `Intent` as shown in Listing 7.4 (lines 1 & 2) and sent to the middleware by defining the corresponding channel (line 3) and receiver package (line 4). Afterwards, the `Intent` is sent by invoking the Android `sendBroadcast()` method.

Listing 7.4: Sending a subscription to myHealthHub's management channel with explicit addressing using the `setPackage()` method.

```
1 Intent i = new Intent();                                    // create Intent
2 i.putExtra(Event.PARCELABLE_EXTRA_EVENT, mySubscription);   // add event
3 i.setAction(MyHealthHub.MANAGEMENT_CHANNEL);                // set channel
4 i.setPackage(MyHealthHub.PACKAGE_NAME);                     // set receiver package
5 sendBroadcast(i);                                           // send Intent
```

Receiving Events

The previous sections described how to start the middleware and how to subscribe to blood pressure reading events. For receiving incoming events, applications have to implement an Android `Broadcast Receiver` and register it to the required reading types. Listing 7.5 depicts an example of a `Broadcast Receiver` that checks incoming events for being of type blood pressure (line 9). Upon an incoming event, the method `onReceive()` is invoked (line 4) and the encapsulated event (`intent`) given. The method `getParcelableExtra` (line 6) returns the event.

Listing 7.5: Android `BroadcastReceiver` that receives events from myHealthHub. Incoming events are processed based on their event type (e.g. blood pressure events).

```
1  /** Event receiver implemented as an Android BroadcastReceiver. */
2  private class ReadingEventReceiver extends BroadcastReceiver {
3      @Override
4      public void onReceive(Context context, Intent intent) {
5          /* Retrieving the event from intent */
6          Event myEvent = intent.getParcelableExtra(Event.PARCELABLE_EXTRA_EVENT);
7
8          /* Check for blood pressure event */
9          if(myEvent.getEventType().equals(SensorReadingEvent.BLOOD_PRESSURE)) {
10             // process blood pressure event
11         }
12     }
13 };
```

Listing 7.6 registers the `Broadcast Receiver` created in the previous listing for events of type blood pressure (line 3). Assuming a successful subscription, this application will receive blood pressure readings as soon as they become available to the middleware.

Listing 7.6: Registering the *ReadingEventReceiver()* to blood pressure events.

```
1 mReadingEventReceiver = new ReadingEventReceiver();      // create event receiver
2 registerReceiver(mReadingEventReceiver,                  // register listener for
3   new IntentFilter(SensorReadingEvent.BLOOD_PRESSURE);  // blood pressure readings
```

Sending Events

Sending (reading) events to myHealthHub is similar to sending management messages (cp. Listing 7.4). The only difference in the selection of another event channel (cp. `myHealthHub.RECEIVER_CHANNEL` in line 3).

Listing 7.7: Sending a reading event to myHealthHub.

```
1 Intent i = new Intent();                                  // create Intent
2 i.putExtra(Event.PARCELABLE_EXTRA_EVENT, myReadingEvent); // add event
3 i.setAction(MyHealthHub.RECEIVER_CHANNEL);                // set channel
4 i.setPackage(MyHealthHub.PACKAGE_NAME);                   // set receiver package
5 sendBroadcast(i);                                         // send Intent
```

Index

A

Activity Recognition
- Daily Activities, 81
- Fitness Trail Activities, 97
- Gym Exercises, 83
- Rehabilitation Activities, 91
- Repetition Counting, 84

Adaptive Transformation Manager, *see* Event Transformation

Ambient Sensors, 16

Ambulant Care, *see* Out-patient Care

Android, 27
- Activity, 31
- Broadcast Receiver, 31
- Component, 30
- Content Provider, 32
- Intent, 28
- Permissions, 33
- Security Attacks, 34
- Service, 31

B

Beyond-BAN Communication, 20

Bluetooth, 19
- Low Energy, 19

Bluetooth Sensor Module, 62

Body Sensors, 17, 79

C

Chronic Diseases, 1

Communication Channels
- Application-specific, 47, 67
- Open, 45, 65

D

Demographic Change, 1

E

Elderly Care, 6, 99

Event Composer, 45, 64

Event Transformation, 49, 71
- Life-Cycle, 51, 73
- OSGi Bundle, 73
- Remote Repository, 53, 74
- Request, 51, 74
- Transformation Manager, 50, 72

H

HedgeHog Sensor, 96

I

InfraWOT, 75

Inter-BAN Communication, 19

Intra-BAN Communication, 19

M

Mediator, 20

Medical Sensor Network Applications, 3

Message Handler, 43, 60

Middleware for Medical Sensor Networks, 21
- Distributed, 21
- Mediator-only, 24

Motivational Example, 8

myElderyCare, 99

myFitnessDiary, 78

myFitnessTrail, 95

myHealthHub, 57

myReha, 89

O

OSGi, 72
- ApacheFelix, 72
- Bundle, 73

Out-patient Care, 4

P

Porcupine Sensor, 79

Preventive Care, 3

R

RedPin, 74

S

Security Manager, 44, 63

Sensor Module Manager, 44, 60

Sensor Module, 44, 61

Active, 60

Passive, 60

System Monitor, 44, 63

T

Telemedical Platform, 90

Training Instruction Module, 92

Transformation Manager, *see* Event Transformation

U

User Motivation Module, 98

User Repository, 45, 64

W

Wi-Fi Sensor Module, 63

Z

ZigBee, 19

List of Abbreviations

Acc	Accelerometer
ADL	Activities of Daily Living
AP	Access Point
BASN	Body and Ambient Sensor Network
BiT	Ball-in-Tube Sensor
BLE	Bluetooth Low Energy
BMBF	German Ministry of Education and Research
BMI	Body Mass Index
BP	Blood Pressure
BSN	Body Sensor Network
CVD	Cardiovascular Disease
DoS	Denial-of-Service
ECA	Event Condition Action
ECG	Electrocardiogram
EEG	Electroencephalogram
EMG	Electromyogram
EOG	Electrooculogram
GPS	Global Positioning System
GSR	Galvanic Skin Response
HR	Heart Rate
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
MAC	Media Access Control

PIR Passive Infrared

Reed Magnetic Switch

RFID Radio-frequency Identification

SpO_2 Blood Oxygen Saturation

SPP Serial Port Profile

WHO World Health Organization

WPA Wi-Fi Protected Access

WSN Wireless Sensor Network

XML Extensible Markup Language

Bibliography

[1] M. Abousharkh and H. Mouftah. A SOA-based middleware for WBAN. *International Workshop on Medical Measurements and Applications Proceedings (MeMeA)*, pages 257–260, 2011.

[2] Hande Alemdar and Cem Ersoy. Wireless sensor networks for healthcare: A survey. *Computer Networks*, 54(15):2688–2710, October 2010.

[3] Mohammed F. Alhamid, Jamal Saboune, Atif Alamri, and Abdulmotaleb El Saddik. Hamon: An activity recognition framework for health monitoring support at home. *IEEE International Instrumentation and Measurement Technology Conference*, pages 1–5, May 2011.

[4] Oliver Amft, Martin Kusserow, and Gerhard Tröster. Bite weight prediction from acoustic recognition of chewing. *IEEE transactions on bio-medical engineering*, 56(6):1663–72, June 2009.

[5] Oliver Amft and Gerhard Tröster. Recognition of dietary activity events using on-body sensors. *Artificial intelligence in medicine*, 42(2):121–36, February 2008.

[6] Urs Anliker, Jamie A. Ward, Paul Lukowicz, Gerhard Tröster, François Dolveck, Michel Baer, Fatou Keita, Eran B. Schenker, Fabrizio Catarsi, Luca Coluccini, Andrea Belardinelli, Dror Shklarski, Menachem Alon, Etienne Hirt, Rolf Schmid, and Milica Vuskovic. AMON: a wearable multiparameter medical monitoring and alert system. *IEEE Transactions on Information Technology in Biomedicine: A Publication of the IEEE Engineering in Medicine and Biology Society*, 8(4):415–27, December 2004.

[7] H.H. Asada, P. Shaltis, A. Reisner, Sokwoo Rhee, and R.C. Hutchinson. Mobile monitoring with wearable photoplethysmographic biosensors. *Engineering in Medicine and Biology Magazine, IEEE*, 22(3):28–40, 2003.

[8] Louis Atallah, Benny Lo, and Frank Siegemund. Wirelessly accessible sensor populations (WASP) for elderly care monitoring. *Second International Conference on Pervasive Computing Technologies for Healthcare*, pages 2–7, January 2008.

[9] O. Aziz, L. Atallah, B. Lo, M. Elhelw, L. Wang, G. Z. Yang, and A. Darzi. A pervasive body sensor network for measuring postoperative recovery at home. *Surgical innovation*, 14(2):83–90, June 2007.

[10] Jens Barth, Michael Sünkel, Katharina Bergner, Gerald Schickhuber, Jürgen Winkler, Jochen Klucken, and Björn Eskofier. Combined analysis of sensor data from hand and gait motor function improves automatic recognition of Parkinson's disease. In *International Conference on the IEEE EMBS*, pages 5122–5125, 2012.

[11] Amel Bennaceur, Gordon Blair, Franck Chauvel, Huang Gang, Nikolaos Georgantas, Paul Grace, Falk Howar, Paola Inverardi, Valérie Issarny, Massimo Paolucci, Animesh Pathak, Romina Spalazzese, Bernhard Steffen, and Bertrand Souville. Towards an architecture for runtime interoperability. In *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part II*, ISoLA'10, pages 206–220, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] Gordon S. Blair, Amel Bennaceur, Nikolaos Georgantas, Paul Grace, Valérie Issarny, Vatsala Nundloll, and Massimo Paolucci. The role of ontologies in emergent middleware: supporting interoperability in complex distributed systems. In *Proceedings of the 12th ACM/IFIP/USENIX International Conference on Middleware*, Middleware'11, pages 410–430, Berlin, Heidelberg, 2011. Springer-Verlag.

[13] Philipp Bolliger. Redpin-adaptive, zero-configuration indoor localization through user collaboration. In *1st ACM International Workshop on Mobile Entity Localization and Tracking in GPS-less Environments*, pages 55–60, 2008.

[14] Remi Bosman, Johan Lukkien, and Richard Verhoeven. An integral approach to programming sensor networks. In *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference*, CCNC'09, pages 269–273, Piscataway, NJ, USA, 2009. IEEE Press.

[15] Vicente Boton-Fernandez and Adolfo Lozano-Tello. Learning Algorithm for Human Activity Detection in Smart Environments. *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pages 45–48, August 2011.

[16] M. Boulmalf, A. Belgana, T. Sadiki, S. Hussein, T. Aouam, and H. Harroud. A lightweight middleware for an e-health WSN based system using Android technology. In *International Conference on Multimedia Computing and Systems (ICMCS)*, pages 551–556, 2012.

[17] Pedro Brandão and Jean Bacon. Body Sensor Networks : Can we use them? In *International Workshop on Middleware for Pervasive Mobile and Embedded Computing*, pages 1–4, 2009.

[18] Vinh T. Bui, Johan J. Lukkien, and Richard Verhoeven. Toward a trust management model for a configurable body sensor platform. In *Proceedings of the 6th International Conference on Body Area Networks*, BodyNets '11, pages 23–26, ICST, Brussels, Belgium, Belgium, 2011. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[19] Vinh T. Bui, Richard Verhoeven, and Johan J. Lukkien. A Body Sensor Platform for concurrent applications. *IEEE Second International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 38–42, September 2012.

[20] Bundesministerium für Bildung und Forschung (BMBF) – Federal Ministry of Education and Research. *Information and Communication Technologies (ICT) Agenda 2020*. Federal Ministry of Education and Research, Public Relations Division, 2007.

[21] Bundesministerium für Bildung und Forschung (BMBF) – Federal Ministry of Education and Research. *The New Future of Old Age*. BMBF, 2011.

[22] Bundesministerium für Bildung und Forschung (BMBF) – Federal Ministry of Education and Research. Actively shaping a future of demographic change. `http://www.bmbf.de/en/4657.php`, 2013. [online; accessed 04-July-2013].

[23] Bundesministerium für Bildung und Forschung (BMBF) – Federal Ministry of Education and Research. Was ist AAL? Marktpotenziale. `http://www.aal-deutschland.de/aal-1/marktpotenziale`, 2013. [online; accessed 10-July-2013].

[24] Dominic Carr, M.J. O'Grady, G.M.P. O'Hare, and Rem Collier. SIXTH: A Middleware for Supporting Ubiquitous Sensing in Personal Health Monitoring. In *International Workshop on Advances in Personalized Healthcare Services*. Springer, 2012.

[25] Rajiv Chakravorty. MobiCare : A Programmable Service Architecture for Mobile Medical Care. In *Pervasive Computing and Communications Workshops (PerCom)*. IEEE, 2006.

[26] Marie Chan, Daniel Estève, Christophe Escriba, and Eric Campo. A review of smart homes- present state and future challenges. *Computer Methods and Programs in Biomedicine*, 91(1):55–81, July 2008.

[27] Keng-hao Chang, M. Chen, and John Canny. Tracking free-weight exercises. *IEEE Ubiquitous Computing (UbiComp)*, pages 19–37, 2007.

[28] Rohit Chaudhri, Jonathan Lester, and Gaetano Borriello. An RFID based system for monitoring free weight exercises. In *ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2008.

[29] Min Chen, Sergio Gonzalez, Athanasios Vasilakos, Huasong Cao, and Victor C. M. Leung. Body Area Networks: A Survey. *IEEE Mobile Networks and Applications*, 16(2):171–193, August 2010.

[30] Xiang Chen, A.B. Waluyo, I. Pek, and Wee-Soon Yeoh. Mobile middleware for wireless body area network. In *International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 5504 –5507. IEEE, September 2010.

[31] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in Android. *ACM Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services - MobiSys '11*, 2011.

[32] ComScore Inc. Future in Focus Digitales Deutschland 2013. Technical report, ComScore Inc., 2013.

[33] Continua Health Alliance. Personal Telehealth Overview. `http://www.continuaalliance.org/connected-health-vision`, 2013. [online; accessed 05-July-2013].

[34] P. Corbishley and E. Rodriguez-Villegas. Breathing detection: Towards a miniaturized, wearable, battery-operated monitoring system. *IEEE Transactions on Biomedical Engineering*, 55(1):196–204, 2008.

[35] Shirley Coyle, King-Tong Lau, Niall Moyna, Donal O'Gorman, Dermot Diamond, Fabio Di Francesco, Daniele Costanzo, Pietro Salvo, Maria Giovanna Trivella, Danilo Emilio De Rossi, Nicola Taccini, Rita Paradiso, Jacque-André Porchet, Andrea Ridolfi, Jean Luprano, Cyril Chuzel, Thierry Lanier, Frdéric Revol-Cavalier, Sébastien Schoumacker, Véronique Mourier, Isabelle Chartier, Reynald Convert, Henri De-Moncuit, and Christina Bini. BIOTEX–biosensing textiles for personalised healthcare management. *IEEE Transactions on Information Technology in Biomedicine: A Publication of the IEEE Engineering in Medicine and Biology Society*, 14(2):364–70, March 2010.

[36] Foad Dabiri, Hyduke Noshadi, Hagop Hagopian, Tammara Massey, and Majid Sarrafzadeh. Lightweight Medical BodyNets (Invited Paper). *Proceedings of the ICST 2nd International Conference on Body Area Networks (BodyNets)*, 2007.

[37] Jiangpeng Dai, Xiaole Bai, Zhimin Yang, Zhaohui Shen, and Dong Xuan. Mobile phone-based pervasive fall detection. *Personal and Ubiquitous Computing*, 14(7):633–643, April 2010.

[38] Anthony Dalton, Shyamal Patel, A.R. Chowdhury, Matt Welsh, Trudy Pang, Steven Schachter, G. OLaighin, and Paolo Bonato. Detecting epileptic seizures using wearable sensor technologies. In *AMA Medical Technology Conference*, 2010.

[39] Fabienne Daniel. *Pilotstudie zur Untersuchung der Übereinstimmung des "estimated Continuous Cardiac Output" (esCCO™) mit der Pulskonturanalyse der arteriellen Druckkurve (PiCCO™) bei Leberteilresektionen*. Dissertation, Medizinische Fakultät Heidelberg der Ruprecht-Karls-Universität, 2013.

[40] R. DeVaul, M. Sung, J. Gips, and a. Pentland. MIThril 2003: applications and architecture. *Seventh IEEE International Symposium on Wearable Computers*, pages 4–11, 2003.

[41] Marco Di Rienzo, Francesco Rizzo, Gianfranco Parati, Gabriella Brambilla, Maurizio Ferratini, and Paolo Castiglioni. MagIC System: a New Textile-Based Wearable Device for Biological Signal Monitoring. Applicability in Daily Life and Clinical Setting. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society.*, 7:7167–9, January 2005.

[42] Hala ElAarag, David Bauschlicher, and Steven Bauschlicher. System Architecture of HatterHealthConnect: An Integration of Body Sensor Networks and Social Networks to Improve Health Awareness. *International Journal of Computer Networks & Communications*, 5(2):1–22, March 2013.

[43] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, June 2003.

[44] European Commision Staff. *Demography report 2010*. European Commission, 2011.

[45] FitnessKeeper, Inc. HealthGraph. `http://developer.runkeeper.com/healthgraph`, 2013. [online; accessed 10-July-2013].

[46] FitnessKeeper, Inc. RunKeeper. `http://runkeeper.com/`, 2013. [online; accessed 10-July-2013].

[47] Tobias Freudenreich, Stefan Appel, Sebastian Frischbier, and Alejandro Buchmann. Actress - automatic context transformation in event-based software systems. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, DEBS, pages 179–190, New York, NY, USA, July 2012. ACM.

[48] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesc language: A holistic approach to networked embedded systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pages 1–11, New York, NY, USA, 2003. ACM.

[49] Toni Giorgino, Paolo Tormene, Federico Lorussi, Danilo De Rossi, and Silvana Quaglini. Sensor evaluation for wearable strain gauges in neurological rehabilitation. *IEEE Transactions on Neural Systems and Rehabilitation Engineering: A Publication of the IEEE Engineering in Medicine and Biology Society*, 17(4):409–15, August 2009.

[50] Toni Giorgino, Paolo Tormene, Giorgio Maggioni, Caterina Pistarini, and Silvana Quaglini. Wireless support to poststroke rehabilitation: MyHeart's neurological rehabilitation concept. *IEEE Transactions on Information Technology in Biomedicine: A Publication of the IEEE Engineering in Medicine and Biology Society*, 13(6):1012–8, November 2009.

[51] Google. Android OS - Fundamentals. `http://developer.android.com/guide/components/fundamentals.html`, 2013. [online; accessed 02-August-2013].

[52] Google Play. Apps from NorthPark. `https://play.google.com/store/apps/developer?id=NorthPark`, 2013. [online; accessed 10-July-2013].

[53] Raffaele Gravina, Antonio Guerrieri, Giancarlo Fortino, Fabio Bellifemine, Roberta Giannantonio, and Marco Sgroi. Development of Body Sensor Network applications using SPINE. *2008 IEEE International Conference on Systems, Man and Cybernetics*, pages 2810–2815, October 2008.

[54] Norbert Gyrbíró, Ákos Fábián, and Gergely Hományi. An Activity Recognition System For Mobile Phones. *Mobile Networks and Applications*, 14(1):82–91, November 2008.

[55] Abdelkrim Hadjidj, Marion Souil, Abdelmadjid Bouabdallah, Yacine Challal, and Henry Owen. Wireless sensor networks for rehabilitation applications: Challenges and opportunities. *Journal of Network and Computer Applications*, 36(1):1–15, January 2013.

[56] W.B. Heinzelman, a.L. Murphy, H.S. Carvalho, and M.a. Perillo. Middleware to support sensor network applications. *IEEE Network*, 18(1):6–14, January 2004.

[57] Farrukh Hijaz, Nabeel Afzal, Talal Ahmad, and Osman Hasan. Survey of fall detection and daily activity monitoring techniques. *2010 International Conference on Information and Emerging Technologies*, pages 1–6, June 2010.

[58] Timothy W. Hnat, Vijay Srinivasan, Jiakang Lu, Tamim I. Sookoor, Raymond Dawson, John Stankovic, and Kamin Whitehouse. The hitchhiker's guide to successful residential sensing deployments. *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems - SenSys '11*, page 232, 2011.

[59] Xin Hong, Chris Nugent, Maurice Mulvenna, Sally McClean, Bryan Scotney, and Steven Devlin. Evidential fusion of sensor data for activity recognition in smart homes. *Pervasive and Mobile Computing*, 5(3):236–252, June 2009.

[60] Enamul Hoque and John Stankovic. AALO: Activity recognition in smart homes using Active Learning in the presence of Overlapped activities. *Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare*, 2012.

[61] Danny Hughes, Klaas Thoelen, Wouter Horré, Nelson Matthys, Javier Del Cid, Sam Michiels, Christophe Huygens, and Wouter Joosen. LooCI: a loosely-coupled component infrastructure for networked embedded systems. *International Conference on Advances in Mobile Computing and Multimedia*, pages 195–203, 2009.

[62] Danny Hughes, Klaas Thoelen, Jef Maerien, Nelson Matthys, Wouter Horre, Javier Del Cid, Christophe Huygens, Sam Michiels, and Wouter Joosen. LooCI: The Loosely-coupled Component Infrastructure. *IEEE 11th International Symposium on Network Computing and Applications*, pages 236–243, August 2012.

[63] Ingrid Lunden, TechCrunch. Android, Led By Samsung, Continues To Storm The Smartphone Market, Pushing A Global 70% Market Share. `http://tiny.cc/01p60w`, 2013. [online; accessed 02-August-2013].

[64] Pekka Iso-Ketola, Tapio Karinsalo, and Jukka Vanhala. HipGuard: A wearable measurement system for patients recovering from a hip operation. *Second International Conference on Pervasive Computing Technologies for Healthcare*, pages 196–199, January 2008.

[65] Shanshan Jiang, Yanchuan Cao, Sameer Iyengar, Philip Kuryloski, Roozbeh Jafari, Yuan Xue, Ruzena Bajcsy, and Stephen Wicker. CareNet: An Integrated Wireless Sensor Networking Environment for Remote Healthcare. *Proceedings of the 3rd International ICST Conference on Body Area Networks*, 2008.

[66] V. Jones, A. van Halteren, and N. Dokovsky. *Mobihealth: Mobile health services based on body area networks*. Springer, 2006.

[67] V.M. Jones, H. Mei, T. Broens, I. Widya, and J. Peuscher. Context aware body area networks for telemedicine. In *Advances in Multimedia Information Processing (PCM 2007)*, pages 590–599. Springer Berlin Heidelberg, 2007.

[68] Emil Jovanov, Aleksandar Milenkovic, Chris Otto, and Piet C. de Groen. A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation. *Journal of Neuroengineering and Rehabilitation*, 2(1):6, March 2005.

[69] Emil Jovanov, Aleksandar Milenkovic, Chris Otto, and Piet C. de Groen. A wireless body area network of intelligent motion sensors for computer assisted physical rehabilitation. *Journal of Neuroengineering and Rehabilitation*, 2(1):6, 2005.

[70] Holger Junker, Oliver Amft, Paul Lukowicz, and Gerhard Tröster. Gesture spotting with body-worn inertial sensors to detect user activities. *Pattern Recognition*, 41(6):2010–2024, June 2008.

[71] Harri Kailanto, Esko Hyvärinen, and Jari Hyttinen. Mobile ECG Measurement and Analysis System Using Mobile Phone as the Base Station. *The Proceedings of the Second ICST International Conference on Pervasive Computing Technologies for Healthcare*, 2008.

[72] Rossi Kamal, NH Tran, and C.S. Hong. Event-based middleware for healthcare applications. *IEEE Journal of Communications and Networks*, 14(3):296–309, 2012.

[73] David Kantola, Erika Chin, Warren He, and David Wagner. Reducing attack surfaces for intra-application communication in android. *Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices - SPSM '12*, page 69, 2012.

[74] Sye Loong Keoh, N. Dulay, E. Lupu, K. Twidle, A.E. Schaeffer-Filho, M. Sloman, S. Heeps, S. Strowes, and J. Sventek. Self-managed cell: A middleware for managing body-sensor networks. In *International Conference on Mobile and Ubiquitous Systems: Networking Services (MobiQuitous 2007)*, pages 1–5, 2007.

[75] J.G. Ko, Chenyang Lu, M.B. Srivastava, John A. Stankovic, Andreas Terzis, and Matt Welsh. Wireless sensor networks for healthcare. *Proceedings of the IEEE*, 98(11), 2010.

[76] D. Konstantas and R. Herzog. Continuous monitoring of vital constants for mobile users: the mobihealth approach. In *Engineering in Medicine and Biology Society, 2003. Proceedings of the 25th Annual International Conference of the IEEE*, volume 4, pages 3728–3731, 2003.

[77] Srdjan Krco, Srdjan Kostic, Dejan Sakac, and Zoran Lukic. mSens Mobile Health Monitoring System. In *International Conference on Computer as a Tool*, pages 80–83, 2005.

[78] Kai Kunze and Paul Lukowicz. Dealing with sensor displacement in motion-based onbody activity recognition systems. In *Proceedings of the 10th international conference on Ubiquitous computing*, UbiComp '08, pages 20–29, New York, NY, USA, 2008. ACM.

[79] P. Kuryloski, A. Giani, R. Giannantonio, K. Gilani, R. Gravina, V.-P. Seppa, E. Seto, V. Shia, C. Wang, Posu Yan, A.Y. Yang, J. Hyttinen, S. Sastry, S. Wicker, and R. Bajcsy. Dexternet: An open platform for heterogeneous body sensor networks and its applications. In *Sixth International Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, pages 92–97, 2009.

[80] Peter Leijdekkers and Valérie Gay. A Self-Test to Detect a Heart Attack Using a Mobile Phone and Wearable Sensors. *21st IEEE International Symposium on Computer-Based Medical Systems*, pages 93–98, June 2008.

[81] Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 766–772, 2005.

[82] Cheng-Yuan Li, Yen-Chang Chen, Wei-Ju Chen, Polly Huang, and Hao-hua Chu. Sensor-embedded teeth for oral activity recognition. *International Symposium on Wearable Computers (ISWC)*, 2013.

[83] Qiang Li and John A Stankovic. Grammar-based, posture-and context-cognitive detection for falls with different activity levels. *Proceedings of the 2nd Conference on Wireless Health*, 2011.

[84] Zhi Li and Guanglie Zhang. A Physical Activities Healthcare System Based on Wireless Sensing Technology. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2007.

[85] Ching-hu Lu and LC Fu. Robust location-aware activity recognition using wireless sensor network in an attentive home. *IEEE Transactions on Automation Science and Engineering*, pages 598–609, 2009.

[86] Johan Lukkien, Frank Siegemund, Richard Verhoeven, Remi Bosman, Laurent Gomez, and Michael Hellenschmidt. The WASP Architecture for Wireless Sensor Networks. In *Constructing Ambient Intelligence SE*, volume 11 of *Communications in Computer and Information Science*, pages 430–447. Springer Berlin Heidelberg, 2008.

[87] J. Luprano, J. Sola, S. Dasen, J.-M. Koller, and O. Chetelat. Combination of body sensor networks and on-body signal processing algorithms: the practical case of MyHeart project. In *International Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, pages 4–7, 2006.

[88] A. Marco, R. Casas, J. Falco, H. Gracia, J.I. Artigas, and A. Roy. Location-based services for elderly and disabled people. *Computer Communications*, 31(6):1055–1066, April 2008.

[89] Simon Mayer and Dominique Guinard. An extensible discovery service for smart things. In *Proceedings of the 2nd International Workshop on the Web of Things (WoT 2011)*, San Francisco, CA, USA, June 2011.

[90] Simon Mayer, Dominique Guinard, and Vlad Trifa. Searching in a web-based infrastructure for smart things. In *Proceedings of the 3rd International Conference on the Internet of Things (IoT 2012)*, Wuxi, China, October 2012.

[91] Mayo Foundation for Medical Education and Research. Heart attack - Lifestyle and home remedies. `http://www.mayoclinic.com/health/heart-attack/DS00094/DSECTION=lifestyle-and-home-remedies`, 2013. [online; accessed 12-July-2013].

[92] A Milenković, Chris Otto, and Emil Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer communications*, pages 1–29, 2006.

[93] K. Montgomery, C. Mundt, G. Thonier, A. Tellier, U. Udoh, V. Barker, R. Ricks, L. Giovangrandi, P. Davies, Y. Cagle, J. Swain, J. Hines, and G. Kovacs. Lifeguard–a personal physiological monitor for extreme environments. *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 3:2192–5, January 2004.

[94] M. J. Morón, A. Gómez-Jaime, J. R. Luque, and E. Casilari. Development and Evaluation of a Python Telecare System Based on a Bluetooth Body Area Network. *EURASIP Journal on Wireless Communications and Networking*, 2011:1–10, 2011.

[95] M. J. Morón, J. R. Luque, A. A. Botella, E. J. Cuberos, and E. Casilari. A Smart Phone-based Personal Area Network for Remote Monitoring of Biosignals. *Body Sensor Networks*, 2007.

[96] M. J. Moron, J. R. Luque, A. Gomez-Jaime, E. Casilari, and A. Diaz-Estrella. Prototyping of a remote monitoring system for a medical Personal Area Network using Python. *Proceedings of the 3d International ICST Conference on Pervasive Computing Technologies for Healthcare*, 2009.

[97] Muhammad Mubashir, Ling Shao, and Luke Seed. A survey on fall detection: Principles and approaches. *Neurocomputing*, 100:144–152, January 2013.

[98] K. Murao and T. Terada. A motion recognition method by constancy-decision. In *International Symposium on Wearable Computers (ISWC)*, October 2010.

[99] National Library of Medicine - Medical Subject Headings. Preventive Medicine. `http://www.nlm.nih.gov/cgi/mesh/2011/MB_cgi?mode=&term=Preventive+Medicine`, 2013. [online; accessed 10-July-2013].

[100] Nike, Inc. Nike+ Accelerator. `http://nikeinc.com/digital-sport/news/nike-accelerator-companies-announced`, 2013. [online; accessed 10-July-2013].

[101] Gregory M. P. O'Hare, Conor Muldoon, Michael J. O'Grady, Rem W. Collier, Olga Murdoch, and Dominic Carr. Sensor Web Interaction. *International Journal on Artificial Intelligence Tools*, 21(02):1240006, April 2012.

[102] Omron Healthcare, Inc. HJ-112. `http://www.omronhealthcare.com/products/hj-112/`, 2013. [online; accessed 10-July-2013].

[103] OpenSignal. Android Fragmentation 2013. Technical Report July, OpenSignal, Inc., 2013.

[104] Joseph Oresko and Allen C. Cheng. HeartToGo: A Personalized medicine technology for cardiovascular disease prevention and detection. *IEEE/NIH Life Science Systems and Applications Workshop*, pages 80–83, April 2009.

[105] Benedikt Ostermaier, Matthias Kovatsch, and Silvia Santini. Connecting things to the web using programmable low-power wifi modules. In *Proceedings of the 2nd International Workshop on the Web of Things (WoT 2011)*, San Francisco, CA, USA, June 2011. ACM.

[106] Chris Otto, A. Milenkovic, Corey Sanders, and Emil Jovanov. System architecture of a wireless body area sensor network for ubiquitous health monitoring. *Journal of Mobile Multimedia*, 1(4):307–326, 2006.

[107] M. Pacelli, G. Loriga, N. Taccini, and R. Paradiso. Sensing Fabrics for Monitoring Physiological and Biomechanical Variables: E-textile solutions. *3rd IEEE/EMBS International Summer School on Medical Devices and Biosensors*, September 2006.

[108] P. S. Pandian, K. Mohanavelu, K. P. Safeer, T. M. Kotresh, D. T. Shakunthala, Parvati Gopal, and V. C. Padaki. Smart Vest: wearable multi-parameter remote physiological monitoring system. *Medical Engineering & Physics*, 30(4):466–77, May 2008.

[109] Alexandros Pantelopoulos and Nikolaos G. Bourbakis. A Survey on Wearable Sensor-Based Systems for Health Monitoring and Prognosis. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(1):1–12, January 2010.

[110] Shyamal Patel, Chiara Mancinelli, Anthony Dalton, Ben Patritti, Trudy Pang, Steven Schachter, and Paolo Bonato. Detecting epileptic seizures using wearable sensors. *IEEE 35th Annual Northeast Bioengineering Conference*, pages 1–2, April 2009.

[111] Shyamal Patel, Hyung Park, Paolo Bonato, Leighton Chan, and Mary Rodgers. A review of wearable sensors and systems with application in rehabilitation. *Journal of Neuroengineering and Rehabilitation*, 9(1):21, January 2012.

[112] J.A.C. Patterson, D.G. McIlwraith, and Guang-Zhong Yang. A flexible, low noise reflective ppg sensor platform for ear-worn heart rate monitoring. In *Sixth International Workshop on Wearable and Implantable Body Sensor Networks (BSN)*, pages 286–291, 2009.

[113] Thomas Pignede. Development of an activity recognition system for fitness trails. Bachelor of science thesis, Technische Universität Darmstadt, 2012.

[114] Thomas Plötz, Paula Moynihan, Cuong Pham, and Patrick Olivier. Activity recognition and healthier food preparation. In *Activity Recognition in Pervasive Intelligent Environments*, volume 4 of *Atlantis Ambient and Pervasive Intelligence*, pages 313–329. Atlantis Press, 2011.

[115] Polar Electro. Smart Coaching. `http://www.polar.com/en/smart_coaching`, 2013. [online; accessed 10-July-2013].

[116] PowerTutor. A Power Monitor for Android-Based Mobile Platforms. `http://ziyang.eecs.umich.edu/projects/powertutor/`, 2012. [online; accessed 04-October-2013].

[117] P. Rashidi and A. Mihailidis. A survey on ambient-assisted living tools for older adults. *IEEE Journal of Biomedical and Health Informatics*, 17(3):579–590, 2013.

[118] Jesús Rodríguez-Molina, José-Fernán Martínez, Pedro Castillejo, and Lourdes López. Combining wireless sensor networks and semantic middleware for an Internet of Things-based sportsman/woman monitoring application. *Sensors (Basel, Switzerland)*, 13(2):1787–835, January 2013.

[119] Runtastic GmbH. Runtastic makes sports funtastic. `http://www.runtastic.com`, 2013. [online; accessed 10-July-2013].

[120] Edward Sazonov, Stephanie Schuckers, Paulo Lopez-Meyer, Oleksandr Makeyev, Nadezhda Sazonova, Edward L. Melanson, and Michael Neuman. Non-invasive monitoring of chewing and swallowing for objective quantification of ingestive behavior. *Physiological Measurement*, 29(5):525–541, 2008.

[121] Edward S. Sazonov, Stephanie A. C. Schuckers, Paulo Lopez-Meyer, Edward L. Melanson, Michael R Neuman, and James O. Hill. Toward Objective Monitoring of Ingestive Behavior in Free-living Population. *Obesity (Silver Spring)*, 17(10):1971–1975, 2009.

[122] Philipp M. Scholl, Nagihan Kücükyildiz, and Kristof Van Laerhoven. When do you light a fire? capturing tobacco use with situated, wearable sensors. In *First Workshop on Human Factors and Activity Recognition in Healthcare, Wellness and Assisted Living (Recognize2Interact)*, Zurich, Switzerland, 09/2013 2013. ACM Press, ACM Press.

[123] Philipp M. Scholl and Kristof Van Laerhoven. A feasibility study of wrist-worn accelerometer based detection of smoking habits. In *esIoT 2012*, Palermo, Italy, 2012. IEEE Press, IEEE Press.

[124] Christian Seeger, Alejandro Buchmann, and Kristof Van Laerhoven. myhealthassistant: A phone-based body sensor network that captures the wearer's exercises throughout the day. In *The 6th International Conference on Body Area Networks*, Beijing, China, November 2011. ACM Press.

[125] Christian Seeger, Alejandro Buchmann, and Kristof Van Laerhoven. Poster abstract: Adaptive gym exercise counting for myhealthassistant. In *Body Area Networks (BodyNets)*, Beijing, China, 11 2011. ACM Press.

[126] Christian Seeger, Alejandro Buchmann, and Kristof Van Laerhoven. Wireless sensor networks in the wild: Three practical issues after a middleware deployment. In *the 6th International Workshop on Middleware Tools, Services and Run-time Support for Networked Embedded Systems (MidSens 2011)*, Lisbon, Portugal, 12 2011. ACM Press.

[127] Christian Seeger, Alejandro Buchmann, and Kristof Van Laerhoven. An event-based bsn middleware that supports seamless switching between sensor configurations. In *Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium (IHI 2012)*. ACM, December 2012.

[128] Christian Seeger, Kristof Van Laerhoven, Jens Sauer, and Alejandro Buchmann. A publish/subscribe middleware for body and ambient sensor networks that mediates between sensors ans applications. In *International Conference on Healthcare Informatics*, Philadelphia, PA, USA, September 2013. IEEE.

[129] P.A. Shaltis, A. Reisner, and H.H. Asada. Wearable, cuff-less ppg-based blood pressure monitor with novel height sensor. In *28th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBS '06)*, pages 908–911, 2006.

[130] Victor Shnayder, Bor-rong Chen, Konrad Lorincz, Thaddeus R. F. Fulford Jones, and Matt Welsh. Sensor networks for medical care. *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, page 314, 2005.

[131] Laura B. Shrestha and Elayne J. Heisler. *Changing Demographic Profile of the United States*. Congressional Research Service, 2011.

[132] Eduardo Souto, Germano Guimarães, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, Carlos Ferraz, and Judith Kelner. Mires: a publish/subscribe middleware for sensor networks. *Personal and Ubiquitous Computing*, 10(1):37–44, October 2005.

[133] Eduardo Souto, Germano Guimaraes, Glauco Vasconcelos, Mardoqueu Vieira, Nelson Rosa, and Carlos Ferraz. A message-oriented middleware for sensor networks. *Proceedings of the 2nd workshop on Middleware for Pervasive and ad-hoc Computing*, pages 127–134, 2004.

[134] Dimitar H. Stefanov, Zeungnam Bien, and Won-Chul Bang. The smart house for older persons and persons with physical disabilities: structure, technology arrangements, and perspectives. *IEEE Transactions on Neural Systems and Rehabilitation Engineering: A Publication of the IEEE Engineering in Medicine and Biology Society*, 12(2):228–50, June 2004.

[135] Peter Van Der Stok, Holger Karl, Fabio De Ambroggi, Jean-Dominique Decotignie, Frank Siegemund, Kees Lokhorst, and Michael Hellenschmidt. WASP - Wirelessly Accessible Sensor Populations: A Project Overview. In *Constructing Ambient Intelligence SE*, volume 11 of *Communications in Computer and Information Science*, pages 426–429. Springer Berlin Heidelberg, 2008.

[136] M. Sung, C. Marci, and A. Pentland. Wearable Feedback Systems for Rehabilitation. *Journal of NeuroEngineering and Rehabilitation*, 2005.

[137] M. Sung and A. Pentland. Livenet: Health and lifestyle networking through distributed mobile devices. *Proceedings of the International Conference on Mobile Systems, Applications, and Services (MobiSys)*, pages 2–4, 2004.

[138] Joseph Sventek, Nagwa Badr, Naranker Dulay, and Steven Heeps. Self-managed cells and their federation. In *Conference on Advanced Information Systems Engineering*, pages 1–10, 2005.

[139] E.M. Tapia, S.S. Intille, and K. Larson. Activity recognition in the home using simple and ubiquitous sensors. In *Pervasive Computing*, pages 158—-175. Springer Berlin / Heidelberg, 2004.

[140] A. Triantafyllidis, V. Koutkias, I. Chouvarda, and N. Maglaveras. An open and reconfigurable wireless sensor network for pervasive health monitoring. *Methods of Information in Medicine*, 47(3):229–34, January 2008.

[141] Andreas K. Triantafyllidis, Vassilis G. Koutkias, Ioanna Chouvarda, Georgios D. Giaglis, and Nicos Maglaveras. Personal health systems for patient self-management: Integration in pervasive monitoring environments. In *Wireless Mobile Communication and Healthcare*, volume 55 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 87–94. Springer Berlin Heidelberg, 2011.

[142] Tim van Kasteren, Athanasios Noulas, Gwenn Englebienne, and Ben Kröse. Accurate activity recognition in a home setting. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, pages 1–9. ACM, 2008.

[143] Kristof Van Laerhoven and Hans-Werner Gellersen. Spine versus porcupine: A study in distributed wearable activity recognition. In *International Symposium on Wearable Computers (ISWC)*, pages 142–149, 2004.

[144] Eduardo Velloso, Andreas Bulling, and Hans Gellersen. Towards qualitative assessment of weight lifting exercises using body-worn sensors. In *Proceedings of the 13th International Conference on Ubiquitous Computing*, UbiComp '11, pages 587–588, New York, NY, USA, 2011. ACM.

[145] Eduardo Velloso, Andreas Bulling, Hans Gellersen, Wallace Ugulino, and Hugo Fuks. Qualitative activity recognition of weight lifting exercises. In *Proceedings of the 4th Augmented Human International Conference*, AH '13, pages 116–123, New York, NY, USA, 2013. ACM.

[146] G. Virone, M. Alwan, S. Dalal, S.W. Kell, B. Turner, J.A. Stankovic, and R. Felder. Behavioral patterns of older adults in assisted living. *IEEE Transactions on Information Technology in Biomedicine*, 12(3):387–398, 2008.

[147] Vitaliberty GmbH. The Stress Balance app - this is how it works. `https://www.corporate-moove.de/en/product-modules/apps/stress-balance-app/`, 2013. [online; accessed 10-July-2013].

[148] M. Wagner, B. Kuch, C. Cabrera, P. Enoksson, and A. Sieber. Android based Body Area Network for the evaluation of medical parameters. In *Proceedings of the Tenth Workshop on Intelligent Solutions in Embedded Systems (WISES)*, pages 33–38, 2012.

[149] A.B. Waluyo, Song Ying, I. Pek, and Jian Kang Wu. Middleware for wireless medical body area network. In *Biomedical Circuits and Systems Conference (BIOCAS)*, pages 183 –186. IEEE, 2007.

[150] Agustinus Borgy Waluyo and Isaac Pek. LiteMWBAN: A lightweight middleware for wireless body area network. *5th International Summer School and Symposium on Medical Devices and Biosensors*, pages 141–144, 2008.

[151] Agustinus Borgy Waluyo, Isaac Pek, Xiang Chen, and Wee-Soon Yeoh. Design and evaluation of lightweight middleware for personal wireless body area network. *Personal Ubiquitous Computing*, 13(7):509–525, October 2009.

[152] Agustinus Borgy Waluyo, Wee-Soon Yeoh, Isaac Pek, Yihan Yong, and Xiang Chen. MobiSense : Mobile Body Sensor Network for Ambulatory Monitoring. *ACM Transactions on Embedded Computing Systems*, 10(1):1–30, August 2010.

[153] Frank Wartena, Johan Muskens, and Lars Schmitt. Continua: The Impact of a Personal Telehealth Ecosystem. *International Conference on eHealth, Telemedicine, and Social Medicine*, pages 13–18, February 2009.

[154] Wikipedia. Fitness Trail. `http://en.wikipedia.org/wiki/Fitness_trail`, 2013. [Online; accessed 18-September-2013].

[155] Anthony Wood, John Stankovic, Gilles Virone, Leo Selavo, Zhimin He, Qiuhua Cao, Thao Doan, Yafeng Wu, Lei Fang, and Radu Stoleru. Context-aware wireless sensor networks for assisted living and residential monitoring. *IEEE Network*, 22(4):26–33, July 2008.

[156] Anthony D Wood, Leo Selavo, and John A Stankovic. SenQ: An Embedded Query System for Streaming Data in Heterogeneous Interactive Wireless Sensor Networks. *IEEE Distributed Computing in Sensor Systems*, 4:531–543, 2008.

[157] World Health Organization. Integrated chronic disease prevention and control. `http://www.who.int/chp/about/integrated_cd/en/`, 2002. [online; accessed 04-July-2013].

[158] World Health Organization. *Global status report on noncommunicable diseases*. WHO Press, 2011.

[159] World Health Organization. Cardiovascular diseases (CVDs). `http://www.who.int/mediacentre/factsheets/fs317/en/index.html`, 2013. [online; accessed 09-July-2013].

[160] World Health Organization. Ten facts about chronic disease. `http://www.who.int/features/factfiles/chp/01_en.html`, 2013. [online; accessed 05-July-2013].

[161] Allen Y. Yang, Roozbeh Jafari, S. Shankar Sastry, and Ruzena Bajcsy. Distributed recognition of human actions using wearable motion sensor networks. *Journal of Ambient Intelligence and Smart Environments*, 1(2):103–115, April 2009.

[162] Jun Yang and Zhigang Liu. Adacem: automatic daily activity and calorie expenditure monitor on mobile phones. In *Proceedings of the Conference on Embedded Networked Sensor Systems*, SenSys '10, pages 409–410, 2010.

[163] Won-Jae Yi, Sufeng Niu, Thomas Gonnot, and Jafar Saniie. System architecture of intelligent personal communication node for body sensor network. *IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*, pages 1717–1720, May 2013.

[164] Jie Yin, Qiang Yang, and JJ Pan. Sensor-based abnormal human-activity detection. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–24, 2008.

[165] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/software Codesign and System Synthesis*, CODES/ISSS '10, pages 105–114, New York, NY, USA, 2010. ACM.

[166] Bin Zhou, Chao Hu, HaiBin Wang, Ruiwen Guo, and Max Q.-H. Meng. A Wireless Sensor Network for Pervasive Medical Supervision. *IEEE International Conference on Integration Technology*, pages 740–744, March 2007.