Master's Thesis

Mobile Applications for Fall Detection in the Area of Ambient Assisted Living

Stefan Almer, BSc
stefan@almer.cc

Graz University of Technology

Institute for Information Systems and Computer Media
Advisor: Univ.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner
Co-Advisor: Dipl.-Ing. Dr.techn. Josef Kolbitsch

Central European Institute of Technology
Advisor: Dipl.-Ing. Dr.techn. Johannes Oberzaucher

Graz, December 2011

Masterarbeit

Mobile Anwendungen zur Sturzerkennng im Bereich Umgebungsunterstütztes Leben

Stefan Almer, BSc
stefan@almer.cc

Technische Universität Graz

Institut für Informationssysteme und Computer Medien
Begutachter: Univ.-Doz. Dipl.-Ing. Dr.techn. Martin Ebner
Betreuer: Dipl.-Ing. Dr.techn. Josef Kolbitsch

Central European Institute of Technology
Betreuer: Dipl.-Ing. Dr.techn. Johannes Oberzaucher

Graz, Dezember 2011

Abstract

With an increasing population of elderly people the number of falls and fall-related injuries is on the rise. This will cause changes for future health care systems, and both fall detection and fall prevention will pose a major challenge. Ambient Assisted Living (AAL) is a research area in which concepts and information systems for assisting elderly individuals are developed. Fall detection, as an important discipline of AAL, investigates a broad range of approaches including wearable devices. With their growing popularity, mobile devices with their embedded motion sensors, their software capabilities and cost-efficiency are well-suited for fall detection.

This thesis presents a test framework for fall detection with the aim of easily setting up assessment tests for collecting motion data and analyzing the data regarding fall detection. The framework consists of a RESTful Web service, a relational database and a Web-based backend. It offers an open interface to support a variety of devices. The system architecture is based on the state-of-the-art theoretical background of AAL and on the evaluation of an existing software.

In order to test the framework, a mobile device client recording accelerometer and gyroscope sensor data is implemented on the iOS platform. The evaluation, which includes three mobility assessment tests, verifies the required functionality, flexibility, availability and data integrity. An investigation of the recorded motion data shows that the iOS client fulfills the requirements regarding sensor accuracy for movement analysis and further feature extraction.

Keywords *ambient assisted living, restful web service, mobile device, mobile application, fall detection*

Zusammenfassung

Mit dem steigenden Anteil der älteren Menschen steigt auch der Anteil von Stürzen und den damit verbundenen Verletzungen. Dies wird zu Veränderungen in zukünftigen Gesundheitssystemen führen und auch Herausforderungen an Sturzerkennung und Sturzprävention setzen. Ambient Assisted Living (AAL) (Umgebungsunterstütztes Leben) beschäftigt sich mit Konzepten und Technologien um ältere Menschen zu unterstützen sowie die Lebensqualität und Selbstständigkeit zu erhöhen. Sturzerkennung, ein zentrales Forschungsthema in AAL, umfasst den Bereich "Sturzerkennung mit getragenen Systemen". Mit der steigenden Popularität von mobilen Geräten, deren Kosteneffizienz und eingebauten Bewegungssensoren, eignen sich diese zur Sturzerkennung.

Diese Masterarbeit zeigt die Entwicklung eines Testsystems zur Sturzerkennung, mit dem Ziel, Sturztests einfach durchführen zu können. Die erfassten Bewegungsdaten werden hinschlicht Sturzerkennung zu einem späteren Zeitpunkt ausgewertet. Das Testsystem umfasst eine offene Schnittstelle (Web Service), eine Datenbank zu Speicherung von Bewegungsdaten und eine webbasierte Administrationsoberfläche. Über die Schnittstelle können verschiedenste Geräte zur Aufzeichnung von Bewegungsdaten eingebunden werden. Die Softwarearchitektur basiert auf dem theoretischen Hintergrund von AAL und der Evaluierung eines bestehenden Systems.

Das gesamte System wurde anhand von drei klinischen Mobilitätstests evaluiert. Dazu wurde eine mobile Applikation für die iOS Plattform entwickelt, welche die Bewegungssensoren des Gerätes benutzt. Die Evaluierung umfasst die Kommunikation der iPhone-Anwendung mit dem Web Service, sowie die korrekte Verarbeitung der aufgezeichneten Daten. Eine Analyse der Bewegungsdaten zeigt, dass sich die Daten der Sensoren zur Sturzerkennung und zur weiteren Verarbeitung eignen.

Keywords *ambient assisted living, umgebungsunterstütztes leben, restful web service, mobile geräte, mobile anwendungen, sturzerkennung*

Statutory Declaration

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

Place, date	Signature

Eidesstattliche Erklärung

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

Ort, Datum	Unterschrift

Acknowledgements

I would like to express my gratitude and appreciation to all those who gave me the support and possibility to write this thesis.

I want to thank the team at CEIT RALTEC in Schwechat for making this master's thesis possible and giving me an insight into research, especially in the area of Ambient Assisted Living. Additional thanks go to Dipl.-Ing. Dr.techn. Johannes Oberzaucher for providing me support and help for evaluating this thesis.

Furthermore, I would like to thank Dipl.-Ing. Dr.techn. Josef Kolbitsch for his encouragement, suggestions for improvement, support and the opportunity to write this master's theses. Moverover, I would like to thank Dipl.-Ing. Dr.techn. Univ.-Doz. Martin Ebner for advising this thesis.

Very special thanks go to my dear colleagues and friends, Korab, Bernhard, Matthias, Günter and Hannes, for providing ideas, room for never ending discussions and their professional feedback.

My dearest thanks go to Julia for her endless patience, encouragement and support while working on this thesis. Thank you so much.

Finally, I would like to express my most heartfelt thanks to my parents, Irmtraud and Richard, who always believed in me and their endless support all over the years.

Stefan Almer, BSc
Graz, Austria, December 2011

Contents

List of Figures

List of Tables

List of Listings

Chapter 1

Introduction

Falls in the group of elderly and disabled people are very common. A demographic trend shows that the average age of Europeans inhabitants will increase and therefore the number of serious and fatal falls will also rise [WHO, 2007; van den Broek et al., 2009].

Ambient Assisted Living (AAL) is a research area motivated by the demographic change. AAL deals with concepts and non invasive support technologies to extend the people's life, increase their independence and reduce their incapabilities. A broad range of approaches, including wearable devices for fall detection, are described. The current approaches and techniques for performing fall detection and the theoretical background of a fall is discussed.

The aim of this thesis is to provide a complete testing framework for setting up assessment tests in order to collect motion data for analyzing the data regarding fall detection. This framework can store motion data from different kinds of motion sensors especially mobile devices. It includes an open interface for adding novel devices in the future, and an administrative backend for easily managing tests. The collected data is analyzed regarding fall detection afterwards. In order to demonstrate the functionality of the framework, a mobile device client was implemented. This proof-of-concept implementation runs natively on iOS devices, particularly the iPhone 4, and records gait and fall data during assessment tests for research purposes.

This thesis is structured in three main parts. Part I covers a detailed description of Ambient Assisted Living (AAL) and its main concepts. It is discussed how the main stakeholders of Ambient Assisted Living are involved in the innovation process. Moreover, the main trends towards AAL and barriers raised by the stakeholder are discussed. It is shown which enabling technologies are offered to perform fall detection.

Chapter 3 provides an introduction to fall detection. The anatomy, characteristics and the risk factors of a fall are illustrated in order to differentiate between different types of a fall. A classification of fall detection methods and current approaches are provided. The main principle of fall detection algorithms using the approach of motion sensors in mobile and wearable devices are described.

The requirements on mobile devices used for fall detection are described in Chapter 4.

A brief history of mobile devices, up to today's smart phones, is presented and an overview of the current and future mobile device market highlights the importance of mobile devices in the future. Part I concludes with the benefits of mobile devices for fall detection, their hardware and software capabilities, which make them particularly well-suited for fall detection.

Part II deals with the implementation of a flexible test framework and its components. The first chapter in this part, Chapter 5, evaluates the currently used test software, analyzes each component, and identifies drawbacks. Based on these findings, the functional requirements for a new framework are specified. The proposed new system is based on a 3-tier architecture. Each tier is described in greater detail with its main functionality and technological base. Chapters 6–9 deal with a detailed description of the data, application and client tiers. It is elaborated how the proposed architecture is implemented and how the tiers interact. The client tier is divided in two parts: the administrative backend and a mobile client for recording motion data.

Part III presents the evaluation of the implemented test framework. The evaluation is contacted by performing common clinical mobility assessment tests. The proof-of-concept is shown by using the administrative backend for creating tests and the smart phone client for recording gait data. Finally, the recorded data is analyzed according feature extraction.

Appendix A provides details about the techniques used during the development of the test framework, backend and mobile device client. The guide in Appendix B shows the basic usage and possibilities of the administrative backend. Finally, Appendix C provides a detailed overview of the performed assessment tests.

PART I

MOTIVATION

This part focuses on the theoretical background of this theses. A brief overview of Ambient Assisted Living (AAL) and its concepts on how information and communication technologies can help improve the quality of life is given. AAL is an important aspect in a society where people become increasingly older. AAL defines enabling technologies for supporting and assisting elderly and disabled people. Moreover, the three main domains for "aging well at home", "aging well at work" and "aging well in the community" are discussed. The demographic trend shows that the number of falls rises in the future [WHO, 2007; van den Broek et al., 2009]. It is shown why fall detection and prevention becomes an important topic in the future.

Due to the importance of fall detection, the definition of a fall as well as the classification of different kinds of falls are illustrated. Thus, the importance of how and why people fall to perform fall detection are discussed. Moreover, recent approaches for fall detection and their technical background to enable fall detection especially with wearable devices are presented.

An approach involving mobile devices for fall detection and their capabilities is introduced. It is shown how the current and future market of mobile devices and networks will develop and what the effects will be. Advantages in the meaning of hardware, costs and user acceptance are discussed over custom-made hardware. Moreover, the two "big players" are discussed regarding software and hardware capabilities in order to perform fall detection with mobile devices.

Part I is structured as follows: chapter 2 details the concepts of AAL, chapter 3 focus on the definition and the technical background of fall detection, and finally, chapter 4 introduces mobile devices for fall detection.

Chapter 2

Ambient Assisted Living

The concept of Ambient Assisted Living (AAL) focus on quality of life for older people. AAL defines support and assistance services and information systems which are enabled by various technologies. The different approaches to AAL and its three main application domains are described briefly in this chapter.

Moreover, it is discussed how demographic, economic and technological trends influence AAL. Especially the demographic trend is discussed in more detail showing the impact on the social environment besides AAL. The main stakeholders are identified and it is depicted which barriers towards Ambient Assisted Living exist.

The last part of this chapter introduces one of the common aspects of AAL: fall detection. It illustrates the aftermath of falls for the society, which fall prevention methods exist and finally which enabling technologies in AAL are used to detect a fall.

2.1 Definition of Ambient Assisted Living

AAL defines information systems which support older people who need assistance in their everyday life in an non invasive way. Ambient Assisted Living is motivated by the demographic change towards the increasing group of the elderly and in need of care-services for the individuals. Therefore, AAL should:

- extend the people's lives in their preferred environment,
- increase and cultivate their independence,
- reduce incapabilities and
- ensure quality improvement of healthcare services and systems as well.

More precisely, AAL defines concepts, products, services as well as appropriate policies for improving the quality of live and technologies in the social environment [van den Broek et al., 2009; Jähnichen, 2008; Georgieff, 2008].

Information and Communication Technology (ICT) systems are a major part in the area of AAL and are also referred to as ambient intelligence systems. These systems are sensitive and responsive to the presence of the user by combining the concept of ubiquitous computing and intelligent social user interfaces [van den Broek et al., 2009]. Thus, such systems and technologies are linked between social actors, their environment and healthcare services where the following aspects by [van den Broek et al., 2009] have to be taken into account:

- *Embeddable:* Devices have to be non invasive or even invisible, distributed throughout the environment or directly integrated into appliances or furniture.
- *Personalized:* Devices have to fit the user's needs.
- *Adaptive:* Devices have to be responsive to the user and the user's environment.
- *Anticipatory:* Devices have to anticipate user's desires as far as possible without conscious mediation.

As a result assistance systems and technologies are developed and implemented in close cooperation with the people who utilize them. Therefore, during the entire product development cycle all involved parties (users, developers, hardware vendors, reseller) are included to yield a product which fits the requirements [Georgieff, 2008].

This innovation process is user-centered. This leads to a wide information spectrum and a high degree of interdisciplinarity. Besides the primary target group of the elderly or disabled people, further involved groups can be identified. This includes family members, neighbors, home-care nurses, community center staff, and emergency personnel [van den Broek et al., 2009]. Consequently, the system is also used by additional users who need to interact with it and with the target group. In order to offer high mobility and increase independence, ambient intelligence systems need to deal with different locations. Those locations are divided into *physical* and *virtual* spaces [van den Broek et al., 2009]. Physical spaces include home, car, workplace or outdoor where virtual places include e-shopping, gaming or activity planing for example.

The innovation model by [Steg et al., 2006] depicted in Figure 2.1 shows the demands and needs of AAL systems considering the relationship of influencing factors and coherences. The model distinguishes the *demand* side (user perspective), consisting of the primary user group defining their needs, requirements and acceptance, and the *supply* side, including the technological options, developers, reseller [Steg et al., 2006; Georgieff, 2008].

New Ambient Assisted Living systems, technologies and products result from matching demand and supply. The demand and supply side as well as matching AAL systems are influenced by their context factors including socio-economic, political, economic and technological factors [Steg et al., 2006]. Therefore, context factors could influence the whole innovation process of an AAL solution.

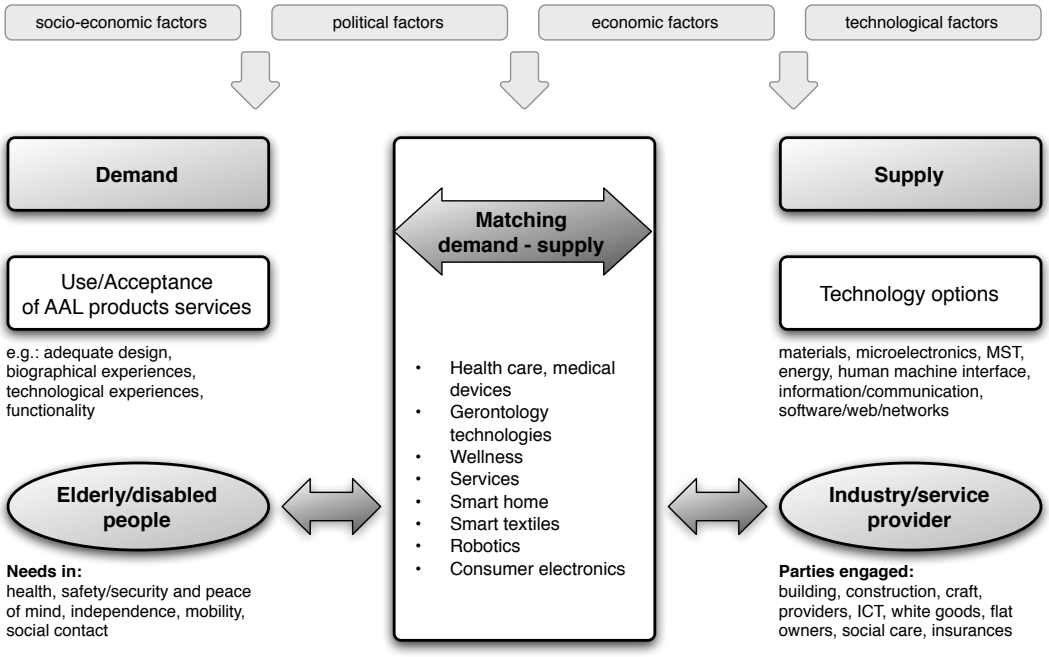


Figure 2.1: AAL Innovation Model [Steg et al., 2006]

2.2 Trends Towards AAL

This section discusses the important trends towards Ambient Assisted Living. It is shown how

- demographic,
- economic and
- technological

trends influence AAL from different perspectives. The demographic trend illustrates the impact of the aging society in Europe, whereas the economic trend shows how the health-care companies will change. Future technology trends show how they will influence the development of new AAL systems.

2.2.1 Demographic Trends

The demographic trend is one of the most influencing trends in the area of Ambient Assisted Living. In fact, the average age of the European inhabitants will increase [van den Broek et al., 2009]. This will raise new challenges for the healthcare, care systems and retirement plans. As shown in Figure 2.2, it is projected that in 2060, the number of people aged 80 or over is three times larger than in 2008 [Eurostat, 2008]. Moreover, the European Office for Statistics projects that the number of deaths will outnumber births in the EU27 from 2015.

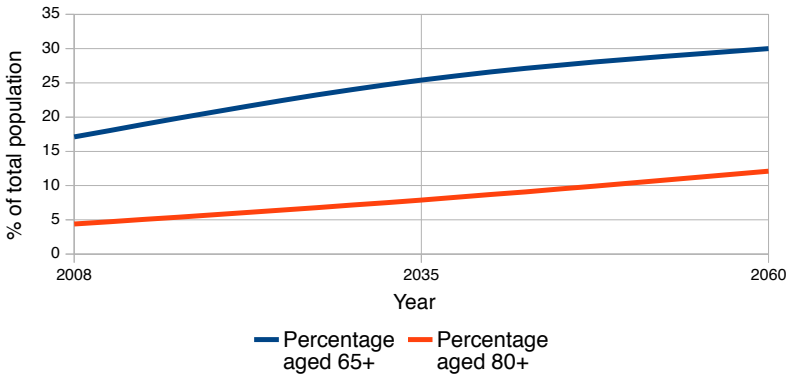


Figure 2.2: Ageing Population, EU 27, 2008–2060 [Eurostat, 2008]

According to [Eurostat, 2008], the rate of people aged 65+ will double to 30 percent and the rate of the 80+-aged will be nearly tripled to about 12 percent in 2060. Therefore, about 150 million people will be aged 65 and over in 2060 in Europe. This can be traced back to a decreasing birth-rate as well as an increasing life span which increased by about 10 years since 1960 [Georgieff, 2008]. In contrast to this, the whole population will decline. In 2035 a growth of 5.1 percent is expected in comparison to 2008, while in 2060 only a growth rate of 2.1 percent is expected compared to 2008 [Eurostat, 2008].

Since people will become older, requirements will change. Social and consumer behavior will change and also the retirement ages will rise. This will lead to a lager number of older people at work. For this reason, people will remain self-sufficient for a longer time but more people will need assistance or support especially in the group of aged 85+ [van den Broek et al., 2009]. A higher life expectancy results in a higher frequency of diseases, and activities of the daily living will become more challenging. This demographic trend will also impact the current and future retirement, healthcare and care systems as well as the labour markets. This will lead to a decreasing "income" for the social framework but, on the contrary, it will lead to increasing costs for healthcare systems [van den Broek et al., 2009].

Therefore, Europe will change radically in the demographic and socio-economy context. This change will raise technological and socio-economic opportunities in order to improve the quality of life for the elderly and disabled people. For this reason, AAL systems in cooperation with Information and Communication Technology (ICT) will strongly influence the aging generation and therefore become a big part of everyday life in the future.

2.2.2 Economic Trends

Companies will focus on individualized services to address new user groups as well as integrating services of several suppliers. According to [van den Broek et al., 2009] several services need to be individualized for Ambient Assisted Living:

- *Hospitals:* They will try increasingly to differentiate from others and offer a more

individual portfolio for the customers.

- *Tele-medicine companies:* This will play an important role in the future since companies are completing existing stationary and ambulant treatments.
- *Care-delivery organizations:* For example, community centers will become more important than equipment, which will result in a business-2-business model.

2.2.3 Technology Trends

Future technological trends will help to develop and design new AAL innovations. This will include easier-to-use systems with the benefit of focusing more on the user. The Internet will be available on every device even on mobile and embedded devices with the result of a more powerful and complex health monitoring. Thus, the networking capacity will increase enabling broadband communication at home with the result of enabling video communication with the healthcare services for example. Standardization will lead to a better integration and communication between a variety of devices. Moreover, Radio-Frequency Identification (RFID)-enabled devices will gain market share, user location awareness or user state recognition will be improved [van den Broek et al., 2009].

2.3 Deployment Barriers

Several barriers between the four AAL stakeholders (Figure 2.3) can be identified. These barriers highlight the problems which hinder the deployment of AAL.

Figure 2.3: AAL Stakeholder

The obvious target group of AAL, elderly and disabled people as well as caregivers, are the *primary stakeholders*. Due to psychological factors, such as habits older people have a general reluctance to use technology that could improve quality of life and do not understand the benefits. Thus, older people have strong prejudices against new technologies, often caused by not fulfilling the user's requirements. To overcome these barriers for

primary stakeholders it is necessary to inform people about AAL and its benefits as well as involving the elderly in the crucial process. [van den Broek et al., 2009]

The *secondary stakeholders* include organizations (care-service, security, community centers) generally providing services to the primary stakeholders, and the *tertiary stakeholders* supply goods and services to the secondary stakeholders. [van den Broek et al., 2009] identifies three main obstacles for the secondary and tertiary stakeholders. First, the requirements and objectives of devices and services do not reflect the actual user needs. This can be avoided by involving the end users in every stage of the design. Second, the lack of certain standards. Using standards would lead to open-reference architectures for efficient service integration for example. Third, the missing broadband coverage in various areas. Broadband coverage is a fundamental part of AAL in order to remotely monitor the subjects as well as including them in the social and service network. Therefore it is demand that the broadband coverage in Europe is improved.

Organizations analyzing the economical and legal context are the *quaternary stakeholders*. Those have to deal with the broadest range of barriers and problems. One of the biggest problems is the diversity of social, welfare and healthcare systems across Europe. Every country has its system which hinders the market for AAL solutions. Therefore, AAL solutions have to be adapted to accommodate the countries' requirements. In oder to facilitate and harmonize every social and healthcare system in Europe, a common social policy is needed. Moreover, [van den Broek et al., 2009] states additional problems:

- *Lack of visible value chains:* Currently, only a few AAL solutions are on the market, and little knowledge of user acceptance is available.

- *Lack of standards and certification:* Standards and certification is needed to provide reliability and foster trust in buyers and users. These standards are also needed for non-technical domains such as quality management and service quality.

- *Funding and reimbursement of AAL services:* AAL services and products are expensive and therefore the target groups are often not able to buy them. There is a need to raise awareness in the governments and politicians to raise funds for AAL related services.

2.4 Enabling Technologies

The enabling technologies describe the technological base used by the three applications domains described in Section 2.5. The following technologies are used and defined to develop applications and functionalities in the area of Ambient Assisted Living: sensing, reasoning, acting, communicating and interacting.

2.4.1 Sensing

Sensing refers to the use of several sensors which are able the measure different activities. Such sensors are placed in-body or on-body, in-appliance or on-appliance or in the environment and take place in anything and anywhere [van den Broek et al., 2009]. In the area of AAL, sensors are referred to as *"smart sensors"* which consist of a conventional sensor and a signal processing hardware. The processing of the information is either embedded or discrete:

- *Vital sign data and activity sensors:* This covers wearable sensors and sensors which are embedded in the user's environment for human activity recognition.
- *Sensor networks:* They combine sensors, appliances, reasoning and actuators for feedback into a communication network or a single device.
- *Sensors for environment, safety and security:* They include, for example, fire and domestic gas detection.

2.4.2 Reasoning

[van den Broek et al., 2009] defines reasoning as *"a core function of AAL systems and the conclusion of knowledge about the activities of the user and the current situation in the environment from low-level sensor data."* Therefore the user's activity is classified into five classes according to [van den Broek et al., 2009]:

- *Activities of daily living:* real sleeping, personal hygiene (washing activities), cooking activities, etc.;
- *Emergency situations:* lying on the floor, motionless, indicators of falls, etc.;
- *Psychosocial behavior:* going out, meeting people, communication;
- *Motion:* occupation of rooms, locomotion (walking, standing, lying, falling), quality and quantity of motion (walking speed, distance);
- *Vital parameters:* pulse rate, blood pressure, body weight, etc.;

The main goal of reasoning is to differentiate activities of the daily living and emergency situations by analyzing and processing user's activity. For example, a lying position could be interpreted as sleeping as well as the result of a fall.

2.4.3 Acting

Acting technologies describe applications and technologies that support the user in his/her everyday life by using intelligent robots and automatic control through actuators. The most important approaches by [van den Broek et al., 2009] are:

- *Rehabilitation:* Mechatronic technologies like prostheses, wearable robots for rehabilitation or artificial muscles are used to support the disabled.

- *Neural-machine interface:* This includes processing of neural information using surface electrodes, invasive methods or non invasive methods.

- *Internet-connected sensor and actuators:* Wireless sensor networks monitor different parameters and are integrated with the Internet. The sensor parameters are monitored remotely.

- *Service and companion robots:* Artificial robots can assist by fulfilling tasks that the disabled is not able to do any more.

2.4.4 Communicating

With pervasive infrastructures and an increasing number of distributed devices the communication among them will increase - like the communication with decentralized services. Moreover, sensors and actuators are connected with reasoning systems and the user's local or remote services [van den Broek et al., 2009].

By connecting several devices, systems and services the aspect of connectivity protocols (IPv4 vs. IPv6), data exchange (messaging format), the data itself (definition), security (confidentiality) and physical network properties (speed) should be kept in mind. Moreover, [van den Broek et al., 2009] defines three basic network contexts:

- *Personal or body network:* Communication within Personal Area Network (PAN) devices and Local Area Network (LAN).

- *Local or home network:* Communication within LAN devices.

- *Public area:* Connection of home or mobile devices with services on the Internet.

2.4.5 Interacting

Interacting describes the human-computer interaction. Intelligent interfaces in the user's environment will track current activities and will respond to all kinds of actions. The interaction with the services and systems needs to fit the requirements of the user's abilities. In order to develop easy to use AAL systems, the end users are involved during the design process in several iterations. Due to the increasing capabilities of networks, devices and services will be inter-connected where each one provides its own unique user interface. Therefore the user has the possibility to choose the appropriate interface [van den Broek et al., 2009].

2.5 Application Domains

Ambient assisted living approaches are user-centered and provide content-aware assistance. This assistance takes place in living locations (home, community center), mobile locations (walking, driving car) and visiting locations (workplace, shops) [van den Broek et al., 2009]. These varying locations lead to a classification of three main application domains according to [van den Broek et al., 2009]:

- *AAL@home*,
- *AAL@work*,
- *AAL@community*.

The application domain "ageing well at home" (AAL@home) focuses on better quality of life for a longer time by using assisting technologies. Through the use of supporting technology the elderly remain more independent and autonomous.

Work is an important factor in the daily life and requires active participation of each individual. Therefore, "ageing well at work" (AAL@work) deals with the improvement of quality of work and work-life balance, and adaptable workplaces as well.

"Ageing well in the community" (AAL@community) deals with the decreasing social participation of the elderly. AAL@community focuses on improving the quality of life by staying socially active for a longer time, reducing social isolation and therefore avoiding the decreasing social participation [van den Broek et al., 2009].

2.6 Falls in AAL

Falls are a relevant factor in the society especially in the in group of elderly and disabled people and therefore an important topic in AAL. With the depicted demographic trend (see Section 2.2.1) it is obvious that automatic fall detection will become a major technology for supporting the target group.

The demographic trend shows that the ageing population in European will increase. This trend will also impact the estimated fall incidence every year. According to [WHO, 2007], approximately 28–35% of people aged 65 and over fall each year about 2 to 4 times while 32–42% of the people aged 70 fall. It should be noted, that the percentage of falls in nursery homes is higher than of people living in the community (30–50%). Moreover, it can be observed (see Figure 2.4) that men have a higher rate of falling than women.

With this trend, the importance of fall prevention and fall detection will rise. This section highlights basic prevention methods as well as fall detection in the area of Ambient Assisted Living.

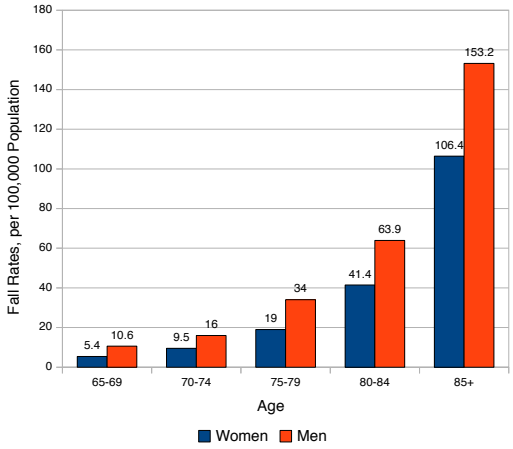


Figure 2.4: Fatal Fall Rates by Age and Sex, United States, 2001 [Cameron et al., 2005]

2.6.1 Fall Prevention

Fall prevention is a wide research topic and varies in the ways a fall is prevented or predicted. Most research topics focus on fall prevention for the elderly people. Several prevention methods differ in usage based on context such as nursing home, hospital, home or living in the community. According to [Todd and Skelton, 2004; WHO, 2007; Tremblay Jr. and Barber, 2005; LeMier et al., 2002; BRAID, 2010] the following most common fall prevention methods can be identified:

- *Assessment tests:* By performing common clinical mobility assessment test, such as the 2-Minute Walk, Sit-to-Stand 5 or Timed Up and Go test, the fall risk of a user can be determined.
- *Adjustment of environment and walking aids:* Potential tripping hazards can be removed and by using walking aids the risk of falling can be reduced.
- *Gait Analysis:* Based on the analysis of the gait pattern, a potential fall can be predicted and the user is alarmed.
- *Education:* By clarifying the fall risk factors (Figure 3.1) to the involved parties many of these risks can be reduced.
- *Exercise/training:* Specifically developed training sessions and exercises strengthen the patient's body and thus reduce the risk of falling.
- *Medications*: The use of the wrong medication can lead to reduced alertness, balance and gait.

2.6.2 Fall Detection

Current methods use sensing technologies to detect a fall-like behavior. Hence, sensors installed in the user's environment track the movement, analyze and process the collected

information in order to determine a fall. It can be distinguished between *user-activated alarms* (where the user manually activates an alarm after the fall event) and *automatic fall detectors* (mostly used by wearable devices) [BRAID, 2008, 2009]. The following two basic fall detection approaches can be identified:

- *Wearable approach:* The detection device is worn by the user. It incorporates accelerometer and gyroscope sensors, for example. Such wearable sensors include smart phones that use the supplied embedded sensors for fall detection.
- *Environmental approach:* Several sensors are installed in the user's usual environment, such as the floor or a wall, and constantly analyze the user's motion regarding a fall. Environmental approaches are location dependent.

Chapter 3 discusses *fall detection* in more detail by describing a fall, the characteristics of a fall as well as an overview of current fall detection methods and their classification.

2.7 Summary

Ambient Assisted Living defines systems to support the elderly and disabled people in order to extend their life, increase their independence, support living standards and reduce their incapabilities. Therefore, selected policies for sensitive and responsive systems are developed in a user-centered process with the primary target group of the elderly and disabled people. This relationship is presented in the innovation model which considers the supply of technology and several context factors besides the demands.

Three main trends towards AAL can be identified: the demographic trend shows that the average age of the European population will increase. This will lead to a larger number of people who need assistance while the number of diseases increase. The economic trend leads to more individualized services, and technology trends will help to develop AAL solutions.

To make AAL innovations more accessible, some barriers need to overcome: benefits are not understood, actual needs and requirements are not reflected and the diversity of the social and healthcare systems hinder the development. Three application domains are classified: AAL@home (ageing well at home), AAL@work (ageing well at work) and AAL@community (Ageing well in the community). Enabling technologies define the base technology used by the domains. *Sensing* deals with the use of sensors, *reasoning* classifies sensor data, *acting* deals with technologies that support the user, *communicating* deals with the communication between devices and services and *interacting* describes the human-compute interaction.

Fall detection is one of the most important topics in the area of Ambient Assisted Living. People 65+-aged fall several times a year and the number of falls increase. AAL suggest sensing and reasoning technologies for detecting a fall-like behavior. The wearable device approach uses embedded sensors for fall detection where the environmental

approach uses sensors located in the user's environment. Falls cannot be eliminated but fall prevention will help to reduce the number of falls.

Chapter 3

Fall Detection

This chapter deals with the term *fall* and tries to find an adequate definition. In order to develop an algorithm for detecting a fall, the risk factors and the phases of a fall are of great importance. Further, the main characteristics such as *fall from sleeping*, *fall from sitting* and *fall from standing* are explained with their properties and characteristics.

The different approaches in the area of fall detection have led to a classification with three main classes. Each class consists of sub-classes which are illustrated with their current most common approaches. The last section introduces the most used technique for fall detection with a body-attached sensor from a theoretical point of view. The problems and opportunities for improvement are also discussed.

3.1 Definition of a Fall

For better analysis and comparison of a fall, it is required to define a fall in the context of fall detection. This is important in order to be able to differentiate a fall from activities of daily living. It is also necessary to clearly define which events could be included into a fall and classify different types of falls [Zecevic et al., 2006]. In literature, many researchers individually defined a fall but a general definition is still missing [Zecevic et al., 2006]. There are many definitions of a fall, which try to describe a fall in more detail or with only very basic characteristics. The most popular and most adapted definition of a fall is by [Gibson et al., 1987]:

"A fall is an event which results in a person coming to rest inadvertently on the ground or other lower level and other than as a consequence of the following: Sustaining a violent blow, Loss of consciousness, Sudden onset of paralysis, as in a stroke, An epileptic seizure."

[Zecevic et al., 2006] conducted a survey in a group of seniors, the main "target group", to get a better meaningful description of what older people think of a fall. In this telephone survey 477 people (aged 55 and above) asked to define a fall and the main reasons for

falling. The participants tended to focus on the antecedents (for example lost balance or weather) and consequences (injury, environmental landing) of falls instead of defining a fall.

For seniors, the most mentioned reasons for falling are [Zecevic et al., 2006]:

1. Balance
2. Weather
3. Inattention
4. Medical conditions
5. Indoor obstacles
6. Surface hazards
7. Slip-trip-stumble

According to the results of the survey it can be said that a fall has different meanings for different people based on their knowledge, experience and psychological mindset [Zecevic et al., 2006]. Among the above noted reasons for falling, [Abbate et al., 2010] defines a number of risk factors which can lead to a fall. Risk factors are classified into four categories: *Intrinsic*, *Extrinsic*, *Internal Environment* and *External Environment*. Figure 3.1 depicts a number of potential risk factors for each category.

3.2 Anatomy and Characteristics of a Fall

A fall is triggered by an unpredictable event such as slipping on the floor, and usually happens during activities of daily life. A fall ends with the impact on the floor with the person lying on the floor. Nevertheless, it should be noted that activities of daily living, for example "falling on the bed" or "sitting down on a chair fast", could be spuriously detected as a fall [Abbate et al., 2010]. According to [Abbate et al., 2010] a fall can be separated into five phases:

1. Activity of Daily Living (Figure 3.2a)
2. Hard-predictable event (Figure 3.2b)
3. Free-fall (Figure 3.2c)
4. Impact (Figure 3.2d)
5. Optional recovery (the fall could be fatal, therefore the person would be unable to recover, Figure 3.2e)

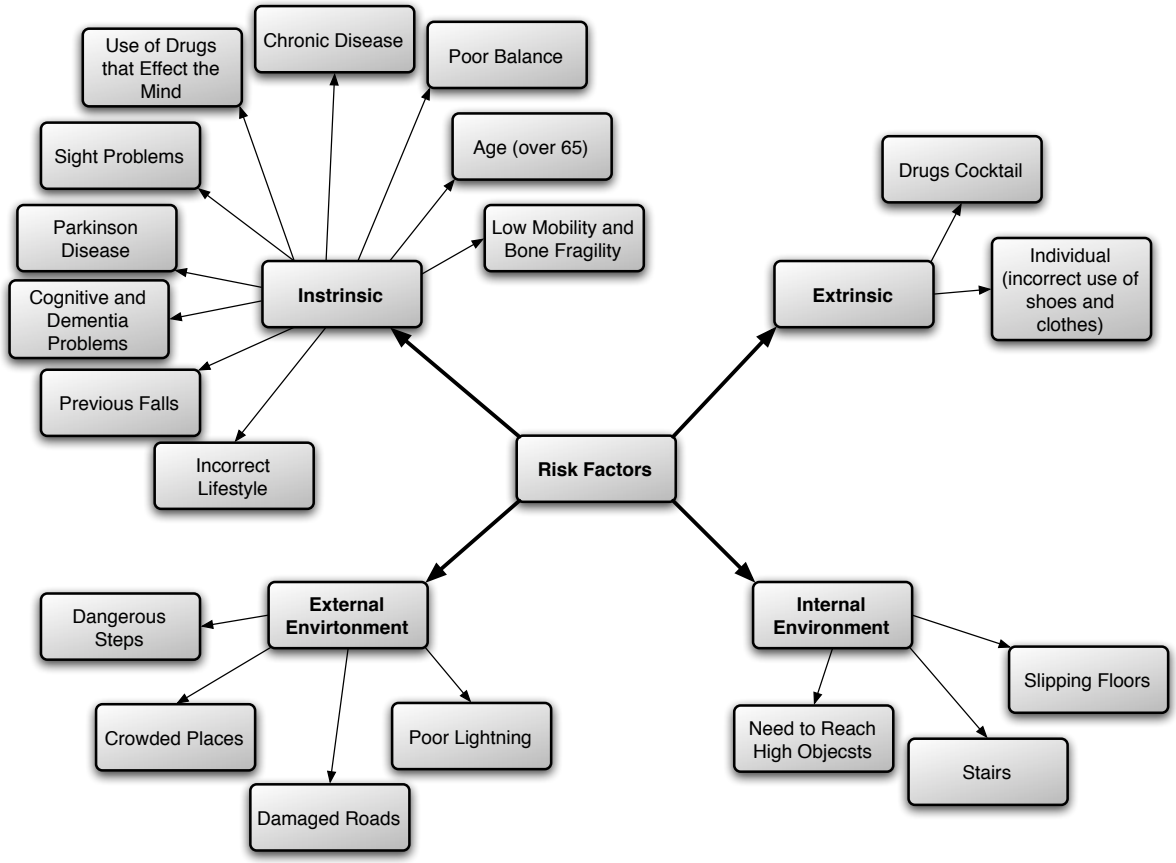


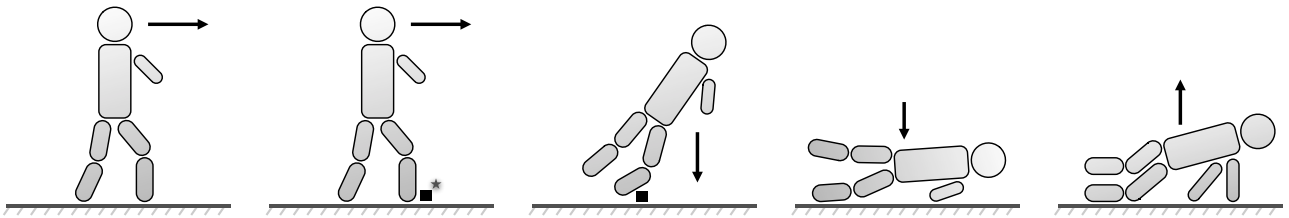
Figure 3.1: Classification of Fall Risk Factors [Abbate et al., 2010]

[Yu, 2008] was the first who specified the characteristics of a fall. These characteristics facilitate developing better methods for analyzing the different types of a fall. [Yu, 2008] defines the duration, the position of the person before and after the fall as well as the range and direction of the fall. Generally, the four defined fall scenarios are: *fall from sleeping*, *fall from sitting*, *fall from walking or standing on the floor*, and *fall from standing on supports* [Yu, 2008]. Figure 3.3 illustrates the first three scenarios, *fall from standing on supports* goes with *fall from standing*.

3.2.1 Fall from Sleeping

The scenario *fall from sleeping* defines a fall from lower height for example a bed (see Figure 3.3a). The person is in a lying position either asleep or not. [Yu, 2008] defines the characteristics as follows:

1. *Duration:* 1–3 seconds.
2. *Starting position:* Person lying in the bed.
3. *Fall:* The body reduces its height from the bed height to the lying height (the body is in free-fall).



(a) Activity of Daily Living **(b)** Hard-predictable event **(c)** Free-fall **(d)** Impact **(e)** Optional Recovery

Figure 3.2: Anatomy of a Fall [Abbate et al., 2010]

4. *End position:* The body lying on the floor is nearby the bed.

3.2.2 Fall from Sitting

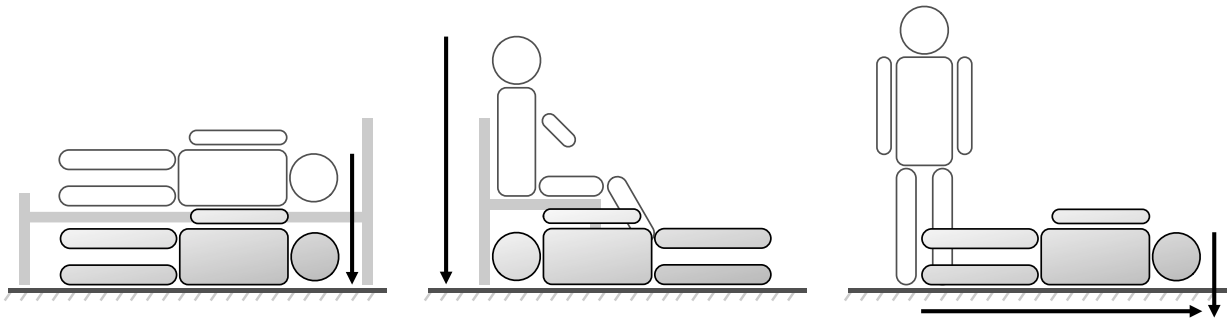
A *fall from sitting* starts with the person in a sitting position, for example, on a chair (see Figure 3.3b) and is defines as follows [Yu, 2008]:

1. *Duration:* 1–3 seconds.
2. *Starting position:* Person sitting in the chair.
3. *Fall:* The head reduces its height form the sitting height to the lying height (the head is in free-fall).
4. *End position:* The body lying on the floor is nearby the chair.

3.2.3 Fall from Standing, Walking or Standing on Supports

This scenario covers the fall from standing, fall from walking and fall from standing on supports, which commonly occur while working (such as standing on the ladder). [Yu, 2008] and [Abbate et al., 2010] defines the fall from standing to a lying position on the floor, while normally a fall does not include standing (see Figure 3.3c).

1. *Duration:* 1–2 seconds.
2. *Starting position:* Person standing.
3. *Fall:* The head reduces its height from the standing height to the lying height (the head is in free-fall). The person falls in one direction (the head and and the weight center of the person move along one plane).
4. *End position:* The lying head is within a virtual circle that is centered at the person's feet before the fall, with the radius of the circle being the persons height.



(a) Fall from Sleeping **(b)** Fall from Sitting **(c)** Fall from Standing

Figure 3.3: Position of the Body Before and After the Fall [Abbate et al., 2010]

3.3 Classification of Fall Detection Methods and Current Approaches

[Yu, 2008] divides fall detection methods for elderly and patients into three main approaches: *wearable device*, *camera-based* and *ambience device*. Further, each class is divided into several sub-classes to give a more detailed overview of the approaches. Figure 3.4 shows the complete hierarchy of all methods. Each method is described in more detail in the following sections.

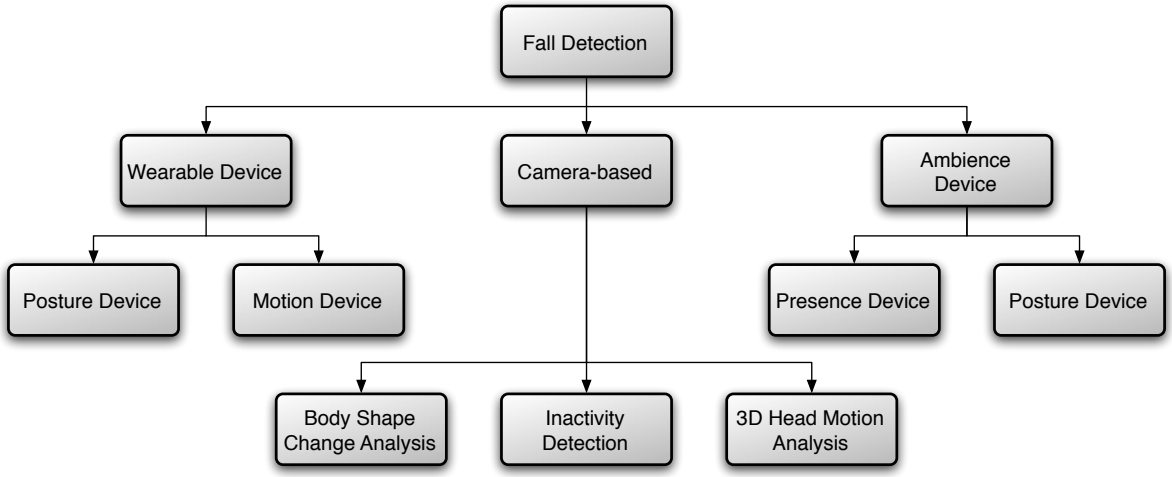


Figure 3.4: Classification of Fall Detection Methods [Yu, 2008]

3.3.1 Wearable Devices

Within this approach the user "wears" one or more devices with embedded sensors. These devices are able to detect a fall or the posture of a person using accelerometers and gyroscopes (see Section 3.4). The advantage of this approach is that it is independent from the location of the user. A fall can be directly assigned to the user since he/she

wears the device.

Wearable Devices are further divided into:

- *Posture device:* the posture after and during the fall is monitored.
- *Motion device:* only the motion (acceleration) during the fall is monitored.

The most used technique to detect a fall is by measuring the acceleration. If the acceleration exceeds a threshold a fall is detected [Dai et al., 2010]. This basic implementation has the disadvantage that even picking up the device from the table could be detected as a fall. The goal is to eliminate such events [Sposaro and Tyson, 2009]. According to [Kangas et al., 2008], the sensor (for example a mobile phone) needs to be carried on specific body parts to reduce false alarms. For example wearing the mobile phone on the trunk reduces false alarms whereas on the arm false alarms are increased since the arm activity is greater.

[Lopes et al., 2009], [Dai et al., 2010] and [Sposaro and Tyson, 2009] use mobile phones equipped with accelerometers to detect a fall. This approach has several advantages like cost-efficient hardware, portability and acceptance even for the elderly [Dai et al., 2010].

Another approach by [Li et al., 2009] uses two tri-axial accelerometers and gyroscopes at separate body parts, for example, located on the chest and thigh. Using sensors on different parts on the body can reduce false alarms. [Li et al., 2009] divides human activity into two categories: static postures and dynamic transitions between these postures. A fall is defined as an unintentional transition to the lying posture measured with acceleration and angular velocity.

[Lindemann et al., 2005] developed an accelerometer which is worn on the head fixed behind the ear. The accelerometer is integrated into the housing of a hearing-aid. Placing the accelerometer on the head has the advantage that the movements are more sensitive since the human tries to protect the head. Therefore high acceleration means unpleasant moves which can lead to a fall.

The *"vitaliSHOE"* project by [Oberzaucher et al., 2010] makes use of a variety of sensors installed in an insole. Those insoles measure the bipedal gait and transmit the data wirelessly for analysis. A base station is used to process and analyze the transmitted data according to gait and moving behavior. The main goal of the project is fall risk estimation also fall detection can be performed. Since the current approach makes use of a base station the insole is not ready for everyday use. A benefit is that users do not feel influenced by the insole [Jagos et al., 2010].

3.3.1.1 Performing Fall Detection with Wearable Devices

As mentioned previously, fall detection with mobile devices is basically performed by measuring acceleration and extracting the acceleration peaks. Figure 3.5 shows the recorded acceleration of a Sit to Stand 5 (STS5) test (see Chapter 10). Regarding fall detection, the acceleration peaks depicted in Figure 3.5 can be used to detect a fall-like behavior.

Such peaks need to be differentiated from activities of daily living in order to perform accurate fall detection.

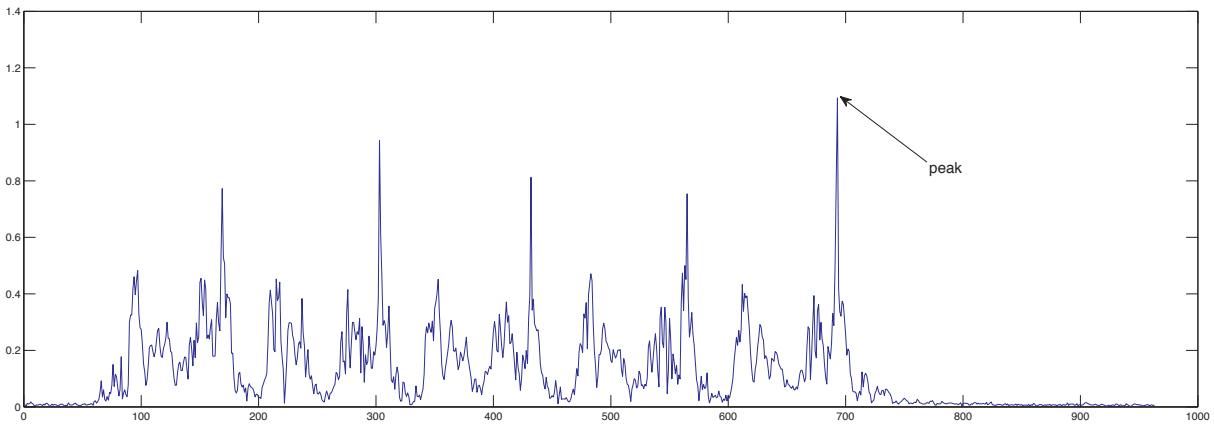


Figure 3.5: Recorded Motion Data of the "Sit-to-Stand 5" Test

In the context of this thesis only the theoretical background for fall detection algorithms are discussed (see Section 3.4).

3.3.2 Camera-based

Camera-based methods can be divided into *Body Shape Change Analysis*, *Inactivity Detection* and *3D Head Motion Analysis*. The main approach of this method is to track the movement of the user with stationary installed cameras. Specific algorithms analyze the recorded video and are able to detect fall patterns. [Abbate et al., 2010] summarize the three sub-categories as follows:

- *Body Shape Change Analysis:* based on the change of posture after the fall.
- *Inactivity Detection:* based on the assumption that after a fall, the patient lies on the floor without moving.
- *3D Head Motion Analysis:* based on monitoring the position and velocity of the head.

The main advantage compared with other methods mentioned is that it is possible to detect multiple events simultaneously. Since the cameras are stationarily installed these methods are less intrusive. For both inactivity and body shape detection the computational complexity is small whereas with 3D head motion analysis the computational complexity is higher and needs more cameras. Therefore 3D head motion analysis is currently not reliable [Yu, 2008].

3.3.3 Ambience Devices

This method makes use of sensors which are installed in the environment of the patient. The advantage of those devices is that they are unobtrusive and best used in houses.

One approach used by [Alwan et al., 2006] makes use of sensors which are located on the floor. The device detects vibration patterns which indicate a fall. Activities of daily living like walking have a significantly different vibration signature than those generated by a fall. For high detection rates it is required to install a sensor in every room.

A second approach by [Sixsmith et al., 2005] uses an array of pyroelectric IR sensors mounted on the wall. The array of sensors only "sees" warm moving objects and not the static background. In contrast to camera-based methods the background has to be eliminated by the algorithm used to extract the moving object. The system analyses the target motion to detect characteristics of a fall. In the second step the target inactivity is analyzed regarding the period of time since the last movement and the time since when the tracker last tracked the object.

The disadvantage of ambience device based fall detection is that they all have the requirement to install several sensors around the patient. Moreover, the rate of false alarms (false positive falls) is high. This is caused by the fact that within pressure sensors it is hard to differentiate if a fall is issued by the pressure of the person's weight or for example an heavy falling object [Yu, 2008].

3.4 Fall Detection Algorithms

In order to develop an algorithm to detect a fall it must be clarified how a human body falls and which forces influence the fall. [Abbate et al., 2010] defines that a body only falls either into the sagittal or frontal plane. According to that the body falls forward or backward which is the sagittal plane or to the left or right which is therefore the frontal plane.

This chapter deals with the most commonly used techniques to detect a fall in the area of wearable devices.

3.4.1 Accelerometer-based Fall Detection

When a body is falling it is accelerated into one direction. The acceleration in the falling plane increases and can be measured with a sensor. Usually the sensor used is an *tri-axis accelerometer* which monitors acceleration in three different axes (x, y, z).

An accelerometer contains a tiny mass. If the accelerometer is moved, tilted or shaken the mass moves and the movement of the mass is measured and converted to an analog or digital signal [Epstein and O'Leary, 2006]. The corresponding signals can be analyzed to detect a fall-like behavior.

Many of the current approaches (see Section 3.3) in the area of fall detection use body attached accelerometers to detect a fall. Since a body is in a free fall-like situation for a short period of time the vertical speed increases due to gravitational acceleration [Noury et al., 2007]. To distinguish between activities of daily living and a fall it is necessary to measure the acceleration within such activities. Compared to the acceleration within a fall,

which is usually higher, it is possible to differentiate between a fall and a "normal activity" and define an appropriate threshold [Noury et al., 2007]. Hence, if the acceleration of an activity exceeds the threshold a possible fall has been detected. Defining the threshold is rather complex. If the threshold is too low, "normal activities" such as sitting down are detected as fall. Otherwise, if the threshold is too high, falls are not detected. Which leads to many false positives [Abbate et al., 2010].

The placement of the accelerometer is also important for reliable fall detection. [Kangas et al., 2007] tried to determine thresholds for accelerometers placed at the waist, wrist and head between intentional falls and activities of daily living. The study shows that each position has different parameters in order to differentiate between a fall and activities of daily living. Hence, the threshold also varies. [Kangas et al., 2007] shows that the most reliable position for the sensor is the head where it is possible to completely distinguish between a fall and "normal activities". When placing the accelerometer at the waist it is not possible to completely differentiate between a fall and activities of daily living. The wrist is the least reliable position for detecting a fall.

[Kangas et al., 2008] evaluated three fall detection algorithms (with different complexity) with the following detection rates:

- *Head:* 47%–98%
- *Waist:* 76%–97%
- *Wrist:* 55%

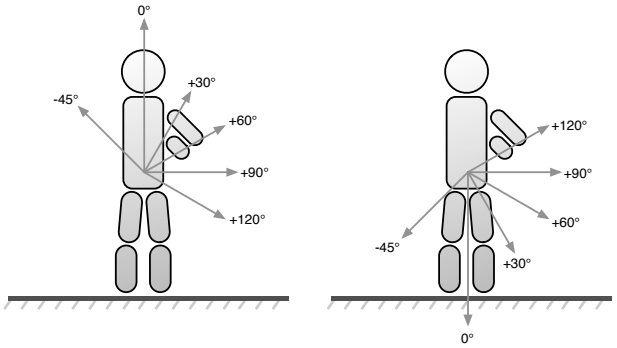
Although the algorithm is rather easy to implement its detection rate highly depends on the placement of the accelerometer, the used algorithm and the corresponding threshold for reliable fall detection. Moreover it can be said that a general threshold for accelerometer based fall detection does not exist.

3.4.2 Gyroscope based Fall Detection

After a fall the human body is for example lying in a specific position on the floor (see Figure 3.3). The orientation of the body can be monitored in order to improve fall detection [Kangas et al., 2007]. The sensor to monitor such positions is a gyroscope. A gyroscope is a sensor to measure orientation. It consists of a spinning wheel whose axes can freely move in any direction. The movement of the axes is measured along three axes (x, y, z) [Luštrek and Kaluža, 2009]. Therefore the gyroscope can be used the determine the orientation of the body.

In the meaning of fall detection the measured angular velocity helps to increase the fall detection accuracy and reduces the rate of false-positives. [Abbate et al., 2010]. The posture is determined by the inclination angels of the trunk and thigh which are depicted in Figure 3.6.

Several different approaches by [Li et al., 2009; Kangas et al., 2007] used accelerometer based fall detection in addition to posture detection. After a possible fall-like behavior



(a) Trunk **(b)** Thigh

Figure 3.6: Posture within Falls [Abbate et al., 2010; Li et al., 2009]

has been detected the current posture of the body is monitored. If the body is in a lying posture (depending of the implementation of the detection algorithm) a fall is detected. With mere accelerometer based fall detection it is not possible to detect the posture since only the acceleration is measured.

3.4.3 Quality Criteria of Fall Detection Algorithms

To compare and evaluate fall detection algorithms, [Noury et al., 2007] proposed two criteria: *sensitivity* and *specificity*. Sensitivity is the capacity to detect a fall and specificity is the capacity to detect only a fall [Noury et al., 2007]. These statistical values depend on the following four cases [Noury et al., 2007]:

1. *True Positive (TP):* A fall occurs and the algorithm detects it.
2. *False Positive (FP):* The algorithm detects a fall but no fall occurred.
3. *True Negative (TN):* A normal movement is performed and the algorithm does not detect a fall.
4. *False Negative (FN):* A fall occurs but the algorithm does not detect it.

With the four mentioned cases, sensitivity (3.1) and specificity (3.2) are calculated. "Good" algorithms will result with a sensitivity of 1 (= 100%) and a specificity of 1 (= 100%).

$$Sensitivity = \frac{TP}{TP + FN} \tag{3.1}$$

$$Specificity = \frac{TN}{TN + FP} \tag{3.2}$$

For better comparison of different implementations, [Noury et al., 2007] developed several test scenarios which are listed in Table 3.1.

Category	Name	Outcome
Backward fall (both legs straight or with knee flexion)	Ending sitting	*Positive*
	Ending lying	*Positive*
	Ending in lateral position	*Positive*
	With recovery	*Negative*
Forward fall	On the knees	*Positive*
	With forward arm protection	*Positive*
	Ending lying flat	*Positive*
	With rotation, ending in the lateral right position	*Positive*
	With rotation, ending in the lateral to the left position	*Positive*
	With recovery	*Negative*
Lateral fall to the right	Ending lying flat	*Positive*
	With recovery	*Negative*
Lateral fall to the left	Ending lying flat	*Positive*
	With recovery	*Negative*
Syncope	Vertical slipping against a wall finishing in sitting position	*Negative*
Neutral	To sit down on a chair then to stand up (consider the height of the chair)	*Negative*
	To lie down on the bed then to rise up	*Negative*
	Walk a few meters	*Negative*
	To bend down, catch something on the floor, then to rise up	*Negative*
	To cough or sneeze	*Negative*

Table 3.1: Scenarios for Evaluating a Fall Detection Algorithm [Noury et al., 2007]

3.5 Conclusion

Fall detection as a general term has very widespread meaning. Many researches are working on finding a general definition of a fall or using existing definitions with their own adaptions. Moreover, according to a survey in a group of seniors, people are focusing more on the antecedents and consequences of falling instead of defining it. Therefore no general definition of a fall exists. Among the reasons for falling such as balance, weather or inattention there are four main risk factors for falling: *intrinsic*, *extrinsic*, *internal environment* and *external environment*.

Generally a fall is triggered by an unpredictable event and ends with the impact on the floor. A fall is divided into five phases: *activity of daily living* (for example walking), *hard-predictable event* (such as slipping on the floor), *free-fall*, *impact* and optionally *recovery* from the fall. In order to develop methods for analyzing a fall four scenarios have been developed. Each scenario describes the duration, start and end position of the human body and the fall itself. Therefore, the scenarios are *fall from sitting* (which is a fall from lower height, the person is lying), *fall from sitting* (the person is sitting) and finally *fall from standing* (the person falls from standing).

Current fall detection methods are divided into three main classes. Each class is divided into several sub-classes which classifies the approaches in more detail. The *wearable devices* approach covers devices with embedded sensors which are attached to the body. Such sensors are able to measure the posture and acceleration of the body and therefore are able to detect a fall-like behavior. The second approach covers *camera-based* fall detection. With such methods the movement of a person is tracked with stationary installed cameras and the recorded video is analyzed for fall patterns. The third class is called *am-*

bience device. This approach makes use of sensors which are installed in the environment of the person. Different approaches install sensors in the floor to detect fall-like vibration patterns or install IR sensors on the wall to track moving objects, therefore a person.

To develop a fall detection algorithm it is important to know that the human body falls only in one direction. Either into the sagittal (forward/backward) or frontal (left-/right) plane. The most commonly used technique to detect a fall is to measure the body acceleration. This is done by using an accelerometer which is able to measure acceleration in any direction. The acceleration of activities of daily living differentiates from a fall. Since the acceleration of a fall is usually higher it is possible to define an acceleration threshold. If the measured acceleration exceeds the threshold a possible fall has been detected. Defining a threshold is complex. It depends on the activity and also on the placement of the sensor. If the threshold is too high falls are not detected, if it is too low normal activities are detected as fall. To compare and evaluate an algorithm the criteria sensitivity (capacity to detect a fall) and specificity (capacity to detect only a fall) are of great importance. To improve accelerometer-based fall detection, posture detection can be added. After a fall the posture of the body is monitored and if it is in a lying position a fall has been detected. This approach reduces the rate of false-positives.

Due to the fact that fall detection algorithms depend an a variety of parameters such as threshold, placement or threshold there is no general algorithm for detecting a fall.

Chapter 4

Mobile Devices

During the past years, mobile devices such as smartphones, tablets and netbooks gained much popularity. This chapter shows a classification of mobile devices according to flexibility and portability. The history of smartphones from the beginning in the early 90's until now is discussed. It is shown which devices caused the breakthrough of today's smartphones. Moreover, the currently most used mobile device platforms are discussed.

The second section analyzes the current mobile device market and highlights the current "big players" in the business. Moreover, the growth of the market by looking at the current state and future trends as well as mobile networks are discussed.

Finally, mobile devices for fall detection are discussed. It is detailed why smartphones fit in the area of wearable devices according to analyzed trends, hardware and software capabilities.

4.1 Mobile Devices

Mobile devices define a category of device types which offer great flexibility and portability. In the past years, mobile devices such as netbooks or smartphones became an important market in the field of mobile devices. Figure 4.1 illustrates the coherence between flexibility and portability according to the form factor by categorizing them into *transportable*, *portable*, *pocketable*, *personal* and *implanted*.

Transportable devices cover netbooks (smaller notebooks), *portable devices* cover tables, *pocketable devices* cover smartphones, *personal devices* cover devices for personal health monitoring (e.g., blood pressure) and *implanted devices* cover chips implanted in the human body.

Mobile devices, especially smartphones in the context of this thesis, combine the features of a Personal Digital Assistant (PDA) with those of a mobile phone. Modern smartphones are usually equipped with a touchscreen, an easy to use operating system and are pocket-sized. In addition to the common PDA functions such as e-mail, address book or calendar, current smartphone operating systems allow users to install additional

Figure 4.1: Flexibility and Portability within Mobile Device Categories [Wallin, 2010]

software also referred to as "application" or "app". Nowadays smartphones provide the mobile possibilities of business applications, Internet access, entertainment such as games and music.

The first smartphone, "Simon", was designed in 1992 by IBM and BellSouth. This device had the capabilities of making calls, create faxes and memos. It was equipped with a touchscreen for entering telephone numbers with a finger or with a stylus [Intelligence, 1993]. In the middle of the 90's Nokia released the Nokia 9000 Communicator product line. It provided an integrated keyboard, access to e-mail, fax and Internet. The communicator could be opened in order to present a bigger monochrome display and a keyboard [Retrobrick, 2011].

The actual term *smartphone* was first used in 1997 by Ericsson. The R380 smartphone introduced 2000 by Ericsson used the Symbian operating system and used a touchscreen and a hinged keyboard and a pen. Subsequently the Symbian OS became one of the most popular smartphone operating system used by a variety of vendors.

A new era of smartphones began in 2007 when Apple released the first iPhone. The device integrated a multi-touch interface with a single touchscreen which is used with fingers - no additional input devices are needed. With the introduction of the new input method and the underlying iPhone operating system, Apple started a new hype. The easy to use operating system, user interface and possibility to develop, distribute and install third-party applications through the "App Store" helped the iPhone to succeed.

Apples first competitor, Google released the Android operating system in 2008. Android is an open source operating system and integrates several Google services. The first device using the Android operating system was the HTC Dream with a full QWERTY keyboard. Google's mobile operating system can be freely used and therefore it is adopted by several device vendors as operating system. Figure 4.2 shows the devices mentioned in a timeline.

While Google only provides the operating system, Apple delivers both hardware and the operating system and a couple of applications. In order to allow developers to create and sell third party stores, Apple and Google offer a Software Development Kit (SDK)

Figure 4.2: Smartphone Device History, 1992–2008

which provides the basic tools for creating applications. Such third party applications can be distributed through the application stores.

Apple and Google, offer their customers a special service where they can download and install applications onto their mobile device. Apple calls it *"App Store"*, Google *"Android Market"*. In general, both stores offer the same functionally and are the common market places for downloading payed or free applications. According to [Scott, 2011] the App Store counts about 500,000 applications including 125,000 iPad specific applications. There have been 18 billion app downloads so far and $3 billion paid to developers. On contrary, the Android Market counts about 200,000 applications with a total of 4.5 billion installed applications [Barra, 2011]. A forecast by [Gartner, 2011b] predicts that there will be 185 billion applications download by the end of 2014 from all mobile app stores.

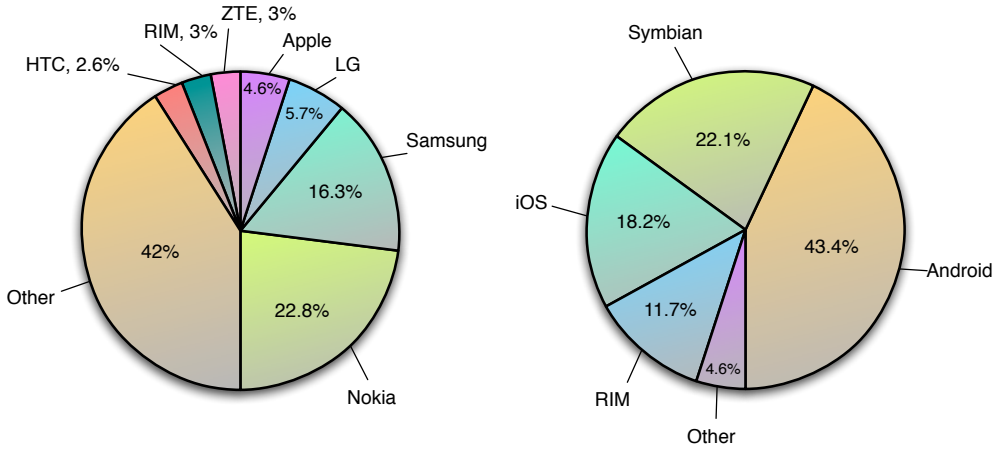
The top categories in Apple's *App Store* are "Games" (15%), "Books" (14%) and "Entertainment" (11%) [Schroeder, 2011]. While the top three categories in the Google *Android Market* are "Entertainment", "Personalization" and "Books & Reference" [AppBrain, 2011]. Moreover, [Gartner, 2009] identified future applications trends based on their impact on consumers and industry players as well as revenue and consumer value. The trend includes applications for *"Money Transfer"*, *"Location-Based Services"*, *"Mobile Health Monitoring"*, *"Mobile Payment"* or *"Near Field Communication Services"*.

4.2 Current and Future Mobile and Network Market

According to [Gartner, 2011a], the worldwide sales of mobile devices in the second quarter of 2011 totaled about 428 million devices sold to end users. Figure 4.3a shows the device sales by vendors, whereas Figure 4.3b shows the sales by operating system to the end user. In comparison to the second quarter in 2010 this is an 16.5% increase in sales.

According to [Gartner, 2011a] the current top vendors are Nokia and Samsung far ahead of LG and Apple. In contrast, the sales by operating systems show that Android is nearly installed on 50 percent of the sold devices, followed by Symbian and iOS. By combining both shares it can be observed that Google with its Android operating system and Apple are the *big players* in the "mobile world" [Gartner, 2011a].

The combined share of the iOS and Android operating systems leads to nearly 62 percent market share by those two operating systems. Therefore it can be said that iOS



(a) by Vendor **(b)** by Operating System

Figure 4.3: Worldwide Mobile Device Sales in Q2-11 to End Users [Gartner, 2011a]

and Android dominate the mobile operating system market. [Gartner, 2011a] explains that this share results from the usability those operating systems offer, the offered services as well as the applications which are offered for each platform.

The mobile device market is an increasingly fast growing market as illustrated by the device shipment trend in Figure 4.4. In 2010 about 288 million devices including smartphones and media tablets have been sold. Gartner predicts that their will be more than 1 billion devices sold in 2015 [Columbus, 2011]. According to this forecast, it can be said that mobile devices will, in the future, play an even bigger role in everybody's daily living than now.

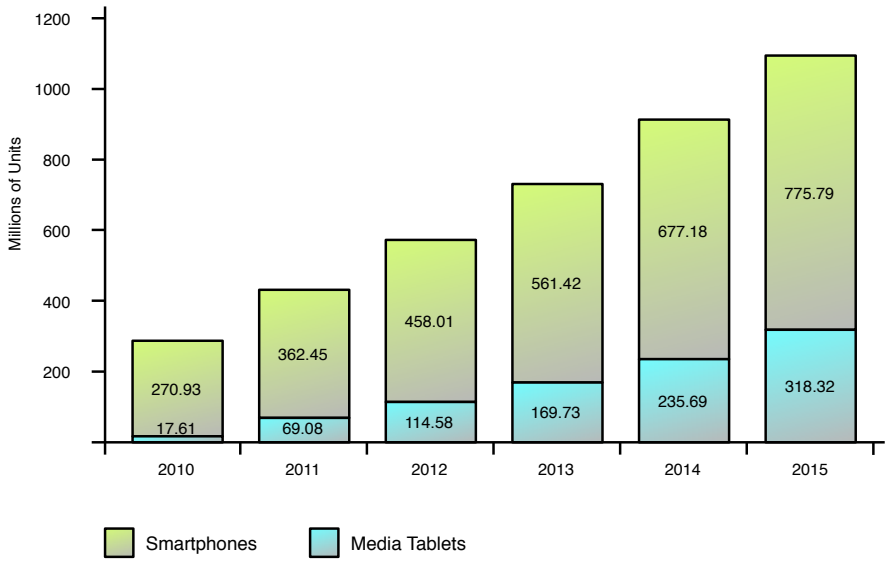


Figure 4.4: Worldwide Smartphone and Media Tablet Shipment Trend, 2010–2015 [Columbus, 2011]

In addition to the mobile device market the number of mobile Internet connections

will grow with the market. This is due to the fact that mobile devices have the necessary connectivity interfaces such as 3G or Wi-Fi for connecting to the Internet. To illustrate the growth rate of the mobile Internet, the global mobile traffic in 2010 was three times greater than the entire Internet traffic in 2000 [Cisco, 2011].

Mobile networks are continuously expanded, new technology standards enforced and therefore networks are becoming faster. By 2015 [Cisco, 2011] predicts a combined annual growth rate of about 92 percent in mobile data traffic. Moreover, the average mobile network connection speed will exceed 2.2 megabits per second with corresponds to an annual growth rate of about 60 percent. Additionally, [Meeker, 2010] predicts that in 2015 accessing the Internet mobile will overtake desktop Internet access.

4.3 Mobile Devices for Fall Detection

As described in Section 3.3, current fall detection approaches include wearable devices, camera-based and ambience devices. This section deals with mobile devices in the context of fall detection in the area of wearable devices. To recap, a wearable device uses embedded sensors such as accelerometers and gyroscopes in order to detect a fall or the posture of a person. In addition to the basic requirements of a wearable device, modern mobile devices offer several advantages which make them suitable for fall detection.

Mobile device trend. As depicted in the previous section, a growing trend for mobile devices can be observed. Mobile devices will increasingly be used in everyday life and will become a constant "companion". With the possibility to access information any time and anywhere in combination with the services offered (such as applications, social network integration) and an easy to use interface mobile devices will replace most users traditional devices. Thus, according to [Wallin, 2010] the mobile market will become people-centric, i.e., development will focus on people instead of devices.

Better networks. Mobile network connections will become increasingly fast. Mobile network carriers will expand the network which will also lead to a better network coverage. As a result, users will be connected to the network at any time. This factor will lead to better and more fail-safe alarm chains because of a more stable connection to the health services.

Hardware. With the rising success of mobile devices the embedded hardware is becoming more efficient. The used mobile processor technology is constantly improved which leads to faster and smaller chips. As a result, the form factor of devices change and will become smaller and increasingly powerful. Thus, the required motion sensors for fall detection are more precise. Initially only accelerometers and Global Positioning System (GPS) chips have been included. Today, modern devices also include a gyroscope and a compass (see Table 4.1). Furthermore, mobile device vendors usually use standard components for building their devices. This leads to more robust, stable and cheaper hardware.

Costs. The costs of mobile devices will be reduced which is caused by several reasons. Mobile device vendors want to gain market share and keep the initial costs low as well. Moreover, a cheaper device price should entice customers as well making the transition to a new vendor more palatable. Another reason is that, with millions of devices sold, hardware can be produced much cheaper.

User acceptance. Since mobile devices become more ubiquitous the acceptance for such devices also grows. Users are becoming more familiar with such devices and are willing to wear them all the time.

All mentioned facts show that a mobile device perfectly fits into the body area network (see also Section 3.3.1). Mobile devices deliver all necessary components to perform fall detection as well as the acceptance of the user to carry a device. Thus, the winning factor for a mobile device is not only the fact that it fits into the wearable device approach but also the fact that it is a future-proof "concept". Therefore, it is not required that the user wears an additional device for fall detection purposes.

4.3.1 Hardware and Software Capabilities

Modern state of the art smartphones such as the *Apple iPhone 4S* or *Samsung Galaxy S II* (shown in Figure 4.5) are well suited for implementing and performing fall detection algorithms.

(a) Apple iPhone 4S [Apple, 2010b]

(b) Samsung Galaxy S II [Samsung, 2011]

Figure 4.5: Smartphones by Apple and Samsung

As shown in Table 4.1, current modern devices are equipped with high performance Central Processing Units (CPUs) and a large amount of memory to offer the best performance to the user. In addition to the rich multimedia capabilities, both devices include necessary orientation sensors to monitor the user's motion.

Beyond that, the devices include a GPS module for positioning, all kinds of cellular and Wi-Fi modules as well. Those modules can be used to build a preferably complete alarm chain. For example, if a fall is detected, it would be possible to place a call or send a message, additionally the current GPS location could be send to the health services.

	Apple iPhone 4S	**Samsung Galaxy S II**
Platform	iOS 5	Android 2.3
Processor	Apple A5 Dual-Core (1GHZ*)	Samsung Exynos (1.2GHz) or Qualcomm Snapdragon S3 (1.5Ghz)
RAM	512MB*	1GB
Storage	16/32/64GB	16/32GB, micro SD expansion
Cellular	quadband GSM, quadband HSPA+ 14.4 / EDVO Rev.A	quadband GSM, quadband HSPA+ 21/42 or CDMA / EDVO Rev.A, WiMax
Wi-Fi	802.11b/g/n	802.11a/b/g/n
Location sensor	AGPS, compass	AGPS, compass
Orientation sensors	accelerometer, gyroscope	accelerometer, gyroscope
Price	starting at 629 €	starting at 399 €

* estimated

Table 4.1: Hardware Comparison of Apple iPhone 4S and Samsung Galaxy S II [Qazi, 2011]

In order to implement an application that performs fall detection, Apple and Google distribute a SDK to developers. Such development kits provide documentation and a set of tools that allow the developer to create applications. Moreover, an Application Programming Interface (API) is provided for accessing each individual motion sensor. The delivered motion data can be processed and passed to the fall detection algorithm in order to detected a fall-like behavior.

Apple's iOS SDK provides access to accelerometers for *x, y, z* axis acceleration, to the gyroscopes for *x, y, z* axis rotation rate as well as to the devices roll, pitch and yaw. Apple's iOS platform allows an application to use the motion sensors during its application life-time. Since the power consumption of motion sensors is higher than other equipped hardware, Apple does not allow applications to "listen to" motion sensor events - the battery would drain too quickly. [Apple, 2011] suggests the following event frequencies:

- *10-20Hz:* Suitable for determining the device's orientation.
- *30-60Hz:* Suitable for real-time user input such as games.
- *70-100Hz:* Suitable for applications which need to detect high-frequency motions.

4.4 Summary

Mobile devices define a category of devices with high flexibility and portability and offer access to the Internet via mobile networks. Mobile devices are categorized into *transportable*, *portable*, *pocketable*, *personal* and *implanted* devices. The history began back in

1992 with IBMs “Simon” and reached the highest popularity in 2007 when Apple released the iPhone. Such smartphones combine the functionality of a PDA and a mobile phone offering the possibility of installing third-party software.

The mobile device market is increasingly growing, with estimates that in 2015 about 1 billion devices will be sold. Associated with that, the mobile network will become faster and expanded. Hence, the mobile Internet access will overtake the desktop Internet access in 2015.

By analyzing the current mobile device market share, can be said that Apple and Google are the obvious big players. Both gained much users by offering them “superb” usability and services as well as a wide range of third-party applications offered through their applications stores. Currently, both application markets combined include about 700,000 applications while it is predicted that there will be about 185 billion downloads by the end of 2014.

Furthermore, it can be observed that mobile devices are well suited for fall detection. The trend shows that mobile devices will play a big role in the future and therefore will increasingly be used by people. Even mobile networks will become better which leads to a much better and fail-safe alarm chain. Modern mobile devices offer the required hardware for fall detection such as accelerometer and gyroscope. Thus, the sensors will become more precise and deliver data at a higher sample rates.

Software Development Kits (SDKs) provide access to each sensor and therefore applications can be built for implementing an algorithm for fall detection.

By combining all facts, it can be said that mobile devices meet all requirements for fall detection and show that users will be likely to wear one single device instead of an additional device for fall detection.

PART II

IMPLEMENTATION

This part describes the current test framework architecture by analyzing its components and interaction. Moreover, the drawbacks of the current architecture regarding sustainability, flexibility, availability and integrity are evaluated. According to these drawbacks functional requirements describing the basic functionality are identified. A proposed new system architecture is described with its technological base.

The proposed architecture is implemented using a 3-tier architecture. First, the entity relationship model of the developed database is described in detail. Subsequently, the implementation of the database using Java persistence is shown. Based on the database model the required API is defined with its parameters, permissions and responses. Since the API is designed as RESTful Web service , the Java frameworks used and their configuration are explained in more detail.

To provide a better user experience, a backend using JavaScript has been developed. It is shown how the Rich Internet Application (RIA) is built using a JavaScript toolkit. The backend provides an easy to use user interface taking advantage of asynchronous requests. The last section describes the implementation details of the mobile client. It is described how motion data is gathered using the mobile device platform. Finally, a generic application life-cycle on which the client is built, is presented.

Chapter 5

System Architecture

This chapter focuses on two main parts, first the current architecture of the testing framework and secondly the proposed system architecture. The first section describes the assessment framework currently used and its components as well as the interaction of the components. Further, the requirements for a new architecture are discussed. This includes highlighting the drawbacks of the current state according to flexibility, availability, integrity and sustainability. Moreover, the user requirements for the new architecture are described using user stories.

Finally, the proposed system architecture is described from a technical point, based on the evaluated requirements. The main parts of the proposed 3-tier architecture are illustrated, describing each layer in more detail as well as its technological base.

5.1 Current Architecture

The current architecture shows which components are used to recored motion data in order to detect a fall-like behavior among older people. The project *"vitaliSHOE"* by [Oberzaucher et al., 2010; Jagos et al., 2010] uses a five component approach depicted in Figure 5.1 which consists of a sensor which is embedded in the insole of a shoe, the transceiver, the base station, a database server and the algorithm server.

A usual assessment test requires the proband to wear the insole which is connected wirelessly to the transceiver. The motion data is transmitted to the transceiver (also called "coordinator") [Jagos et al., 2010]. The transceiver is connected via a serial interface to the base station. This base station stores, displays and analyzes the received data [Jagos et al., 2010]. Finally, the data is stored in a database. Moreover, the algorithm server implements various algorithms for analyzing the stored gait data.

5.1.1 Component Description

As depicted in Figure 5.1, the current architecture is comprised of the following five components. A detailed description is provided in the framework paragraphs.

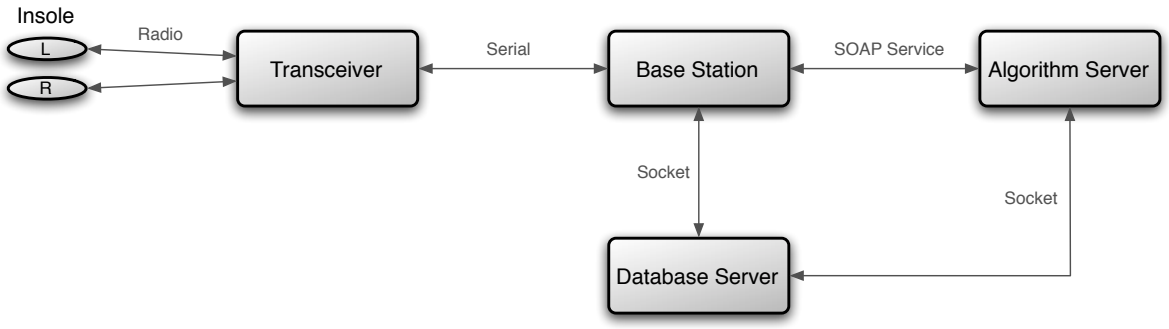


Figure 5.1: Components of the Current Architecture

5.1.1.1 Insole

The instrumented shoe insole consists of motion sensors, an embedded system and a wireless module [Jagos et al., 2010]. The insole is divided into three functional blocks which are also shown in Figure 5.2 [Jagos et al., 2010]:

1. *Force Sensitive Resistors (FSR)* for plantar pressure measurement;
2. *Inertial Measurement Unit (IMU)* for measuring pitch, row angles and acceleration;
3. *Processing Unit*;
4. *Transceiver* for wireless data transmission.

[Jagos et al., 2010] specifies that the sensor data is acquired with a sample rate of 200Hz. The embedded system filters and processes the data down to 50Hz and finally transmits it wirelessly to the base station.

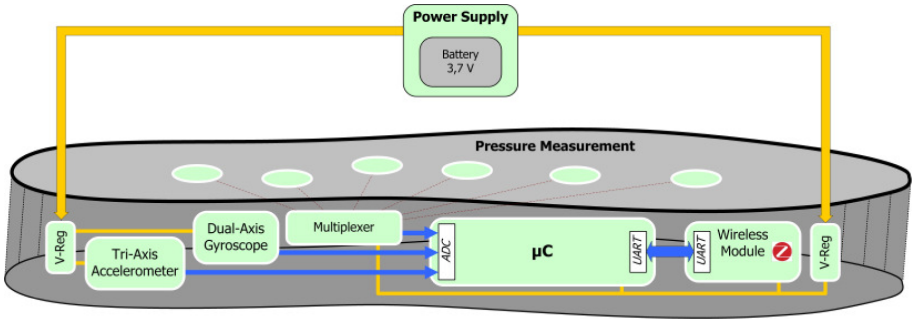


Figure 5.2: Schematic Diagram of the Insole's Functional Blocks [Jagos et al., 2010]

5.1.1.2 Transceiver

The transceiver is responsible for the wireless data transmission between the insole and the base station. In general, this component can be seen as the connector between the insole and the base station. The base station is connected through the serial interface to the transceiver.

5.1.1.3 Base Station

The base station is a Personal Computer (PC) running a client implemented in the Java programming language. This client is responsible for communicating with the insole for, for example, sending the start and stop commands as well as storing the received data in a local database and displaying it.

After the assessment is performed the data is stored in a database table on the database server in the local network. The connection is established via a simple socket connection. For each assessment a new database table is created with the attributes in Table 5.1. To identify the data of a assessment, the name of the table follows a naming convention containing the test type, proband, researcher and the timestamp. The naming convention is constructed by the pattern "`TestType_Propand_Researcher_Timestamp`" which results in "`TUG_SA_SA_10.10.2011_12:33:13`", for example.

Field	Type
pitch_angle_left	text
dynamic_acceleration_left	text
X_acceleration_left	text
Y_acceleration_left	text
Z_acceleration_left	text
FSR_heel_left	text
FSR_meta_one_left	text
FSR_meta_five_left	text
FSR_toe_left	text
pitch_angle_right	text
dynamic_acceleration_right	text
X_acceleration_right	text
Y_acceleration_right	text
Z_acceleration_right	text
FSR_heel_right	text
FSR_meta_one_right	text
FSR_meta_five_right	text
FSR_toe_right	text
Timestamp	text

Table 5.1: Structure of the Database Table used to Store Motion Data

5.1.1.4 Database Server

A MySQL[1] database server is used as storage engine for recorded gait data as well as for computed data by the algorithm server. The server is available on the local network and over a Virtual Private Network (VPN).

5.1.1.5 Algorithm Server

The algorithm server is used to process the recorded data with a defined algorithm to detect a fall-like behavior. The server implements several different algorithms for evaluating different kind of gait data. Additionally, the algorithm server offers a Simple Object Access Protocol (SOAP) interface implemented in Java and accepts an identifier of the data to be processed.

[1] `http://mysql.com/`

Therefore the data is retrieved through a socket connection to the local database server. The interface passes the data to a local instance of MATLAB[2] which actually performs the computation. Finally, the result is stored in a database on the database server.

5.2 Requirements for a new Architecture

The previous section focused on the current system architecture which showed that there are some drawbacks regarding sustainability, flexibility, availability and integrity. This section deals with these drawbacks and furthermore the basic requirements for the new architecture is discussed. The following drawbacks of the current system are analyzed according to:

Flexibility: The current design does not allow to add new wearable devices into the framework. The database as well as the client needs to be adopted to be able to integrate a new device.

Availability: Data is first saved locally and then stored on the database server. Which means that new assessment data is only available if the researcher manually saves it to the database server. Thus, the data is only available in the local network or via VPN. No access is given at any time.

Integrity: Using the current database schema, data integrity and consistency is not guaranteed. A new database table is used for each assessment. The gait data is stored locally on the base station.

Sustainability: Without the description of the database table naming convention and its abbreviations it is no longer possible to identify the stored assessment data and the involved persons. Moreover, the device used and its sensor descriptions are not mapped to the data.

5.2.1 Functional Requirements

In addition to evaluating the current architecture, functional requirements have been identified. According to [Malan and Bredemeyer, 2001], functional requirements capture the indented behavior of the architecture. The following requirements show the demands of the new testing framework:

Device sensors: A device has a variable number of sensors (such as “acceleration x”).

Device type: A device is identified by a device type like “iPhone 4”.

Sensor data type: A sensor is defined through a data type such as “integer”, “double”, “boolean” or “text” and has a description.

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

Data storage: Data is stored centrally.

Remote data access: Stored data can be accessed (in a secure way) remotely.

Data processing: Collected data can be evaluated with several algorithms. The result is saved in a database.

Multiple wearable devices: Several devices can be worn at the same time during an assessment.

Tests: A test is the relationship between the person who performs the test and the data which is recorded with a device during the test.

Wearable device: Gait data is recorded with (a wearable) device such as a mobile phone or an insole.

Users: User are defined who have access to the framework and are assigned to a assessment.

User roles: A user can have multiple roles. The roles are "admin", "proband" and "observer".

User properties: A variable number of properties such as weight, height or fat level can be used to characterize a user.

Confidentiality: Data can be accessed by identifying a user with his/her credentials and roles. Users only have access to their data.

Programming interface: A programming interface should be provided for storing and accessing data.

5.2.2 Technical Requirements

By combining the identified drawbacks of the current system discussed in Section 5.2 and the user requirements in Section 5.2.1 the technical requirements have been evaluated:

- *Integrity, consistency:* A relational database should be used for storing and querying data in the meaning of *integrity* and *consistency*. This database is connected to a well defined Application Programming Interface (API) which makes it easy possible to access the stored data. This approach leads to a clear separation between the device (client) and the stored data (server).

- *Flexibility, extensibility:* This interface provides additional logic for processing the data. The interface is well defined and should provide the greatest possible flexibility, in terms of *availability* and extensibility (*flexibility*). Changing the interfaces in the "background" (logic) should not affect the usability for the user. Therefore interfaces remain unchanged in the meaning of *sustainability*.

- *Openness, extensibility:* Consequently, the new adopted framework is open for additional devices. Various devices used within the body area network can be connected, all its status data can be processed and analyzed.

5.3 Proposed System Architecture

Based on the evaluated requirements an architecture for the new test framework is proposed. The proposed framework consists of three main components: database, interface and client. Therefore a client-server architecture is chosen. The framework is built on a 3-tier architecture. The parts of the 3-tier architecture are illustrated in Figure 5.3.

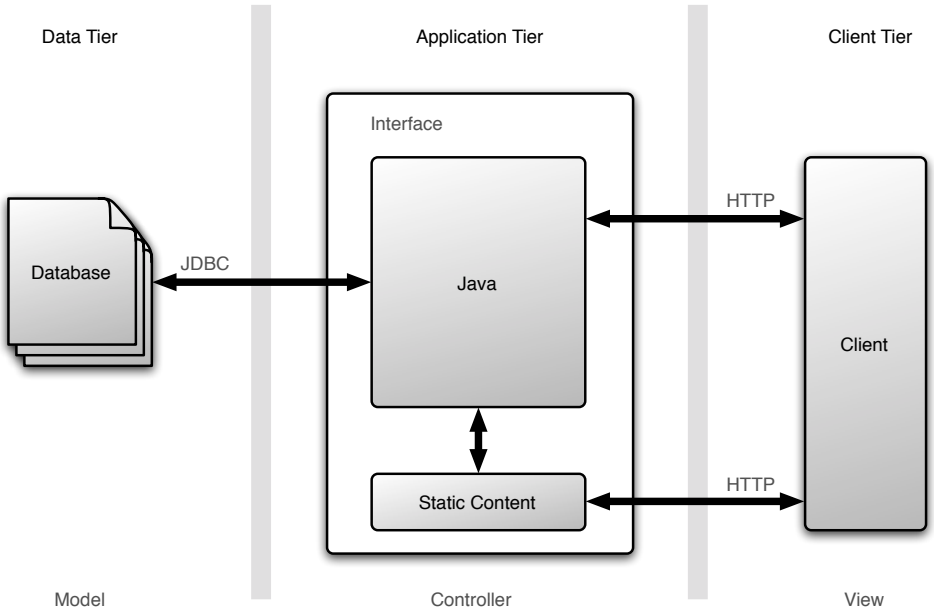


Figure 5.3: 3-Tier System Architecture

[Helic, 2008] describes an n-tier architecture as a modern client-server architecture which is separated into presentation, application and data layers. A 2-tier architecture consists of a client layer providing the user interface and the data server for storing data. The disadvantage of the 2-tier architecture is that the client needs to know how to communicate with the data servers and also implements the application logic [Helic, 2008].

To overcome the drawbacks of the 2-tier architecture a middle layer called "application tier" is inserted and therefore called 3-tier architecture. The application tier resides typically on the server side and holds application or business logic. [Helic, 2008] highlights the following advantages of the 3-tier architecture:

- improved scalability,
- "thinner" clients,

- clean separation of presentation, application and data layers,
- easier client maintenance (middle layer can be updated in isolation),
- additional layers can be added in the middle layer, and
- isolation of data layer specifics in the meaning of extensibility and configuration

Figure 5.4 shows the communication process within the 3-tier architecture. The client presents the user interface and displays data requested from the application layer to the user. The application layer processes the request from the client by performing additional logic and requests the required data from the data tier. Finally, the requested data is passed back to the client. The data tier is commonly a database server running a Database Management System (DBMS).

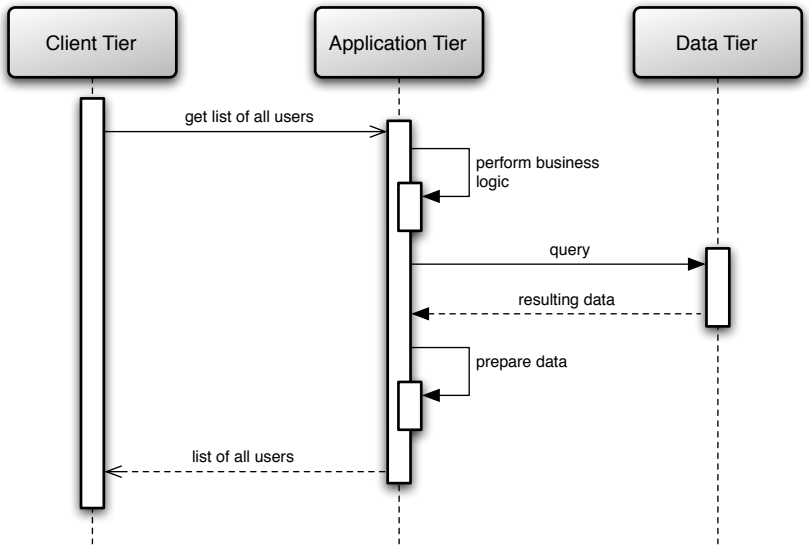


Figure 5.4: Exemplary Client-Server Interaction in a 3-Tier Architecture

The following sections describe each component for the 3-tier architecture used within the framework. Moreover, it is explained which technologies are used to provide flexibility, maintainability, availability and sustainability.

5.3.1 Data Tier

The data tier as described previously is the data layer of the 3-tier architecture and mainly consists of databases. A database is designed to store and manage information in an organized structure. [Scerbakov, 2008] states that a database is an information model containing facts relevant to some area of interest (called domain of interest).

To be more precisely, a relational database is used to store the required data entities and relations among them. The so called relational data model specifies a data structure (relation) and several languages to manipulate relations [Scerbakov, 2008]. In general,

the relationship between tuples (data objects) is described. A relational database ensures entity integrity by means of primary keys and referential integrity with foreign keys [Scerbakov, 2008].

5.3.2 Application Tier

The application tier implements the interface described in Section 5.2.2. The interface provides mechanisms for simply accessing and modifying data in the data store. Additional logic is added for parsing and processing client requests and responses. The interface is designed as a Web service which is according to [Haas and Brown, 2004]: *"... a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."*

A Web service can be accessed by different applications regardless of the programing language the application is implemented in. The Web service offers an Application Programming Interface (API) which is a set of HTTP messages with defined responses. [McCabe et al., 2004] identifies two major classes of Web services:

- *REST-compliant Web services:* with the primary purpose to manipulate Web resources using a uniform set of "stateless" operations (like create, retrieve, update and delete).
- *Arbitrary Web services:* the service exposes an arbitrary set of operations used by Simple Object Access Protocol (SOAP) or Remote Procedure Call (RPC) for example.

The interface for this application is implemented as resource-oriented Representational State Transfer (REST) Web service because of its simplicity in contrast so Service Oriented Architectures (SOAs). Moreover, REST Web services are easier to implement on a wide range of devices including mobile devices.

The application tier is implemented in the Java programing language as servlet container using an appropriate application server.

5.3.2.1 RESTful Web services

REST is an architectural style first defined and introduced by [Fielding, 2000]. REST stands for *Representational State Transfer* and deals with resources on the Web. Resources are the key abstraction of information where every information named can be a resource [Fielding, 2000]. Resources are uniquely identified by an Uniform Resource Identifier (URI). REST uses the Hypertext Transfer Protocol (HTTP) for transferring resource representations. [Fielding, 2000] defines a representation as the current or intended

state of a resource. Such a representation could be Hypertext Markup Language (HTML), Extensible Markup Language (XML) or even JavaScript Object Notation (JSON).

Thus, the communication is stateless, which means that all necessary information is transferred with no client relevant information stored on the server [Fielding, 2000]. REST provides a *uniform interface* which is defined by four constraints [Fielding, 2000]:

- identification of resources,
- manipulation of resources through representations,
- self descriptive messages, and
- hypermedia as the engine of application state.

RESTful Web services combine the principles of REST with HTTP. A URI defines a resource while HTTP methods are used to create, read, update and delete them (following the CRUD principle). Table 5.2 shows the HTTP method equivalents of CRUD:

CRUD	HTTP	Description
Create	POST	create a new resource
Read	GET	retrieve a representation of a resource
Update	PUT	modify an existing resource
Delete	DELETE	delete an existing resource

Table 5.2: CRUD and its HTTP Method Equivalents

5.3.2.2 Authentication and Authorization

Authentication is used to identify and verify the user's access permission to the Web service. A simple authentication system based on user name and password is provided. Authentication is based on the principle of a shared secret which is only known by the individual and the authentication system.

Besides authentication, authorization is used to determine the user's access right on a specific resource, once identified. Resources are protected by defining user roles which are allowed to access them. If the user is part of the role, access is granted.

5.3.3 Client Tier

The client is the presentation layer of the 3-tier architecture. With n-tier architectures the client is responsible for displaying data retrieved from the application tier and only required user interface logic since the applications tier handles the business logic.

Decoupling the client from processing data and business logic leads to more flexible and "thinner" clients as well as decreasing the implementation time. Therefore it is possible to implement various clients which perfectly fit their area of application. For the current base station (see Section 5.1.1.3), this means to implement only the required API calls to

save the motion data. This leads to a focus on implementing hardware and algorithms instead of setting up a framework on every device.

Moreover, the client can be implemented independent from the other layers, which leads to increased modularity.

5.3.3.1 Administrative Backend

For performing administrative tasks such as creating users, devices and tests an administration backend is required (see Section 5.2.1). This backend is a special type of a client using the Web service API. It is implemented as Rich Internet Application (RIA). RIAs are web applications which run on the client side and eliminate the presentation and interactive layers from the server side [Preciado et al., 2005]. In general, [Preciado et al., 2005] describes RIAs as the fusion of desktop applications with Web applications according to user interface functionality. This is achieved using technologies like Adobe Flex[3], Microsoft Silverlight[4] or JavaScript.

The backed is protected giving access to users with certain roles. This is required because only researchers should be able to administrate the framework. The backend covers the administration of users, devices and tests with all their necessary relationships among each other.

5.4 Summary

The current architecture consists of five components. The wearable device (insole), a transceiver for wireless data transmission between the insole and the base station. The base station runs a Java client for communicating with the insole and storing the received motion data. A local database server is used to store the acquired data and finally, an algorithm server is used to evaluate the assessment data.

By evaluating the current assessment framework several drawbacks could be identified. In terms of flexibility, the current architecture is not "open" enough for new wearable devices for fall detection. Test data is only saved manually to the database server which limits the availability of the data for further evaluation. Moreover, sustainability is not guaranteed since relevant information such as the used device, proband or assessment location is not clearly saved.

Requirements for the new framework have been collected using functional requirements used to evaluate the technical requirements for the proposed system architecture. The framework is built on a 3-tier architecture consisting of data, application and presentation layers. This client-server architecture improves scalability and provides a clean separation of the three layers.

The data tier uses a relational database to guarantee integrity and consistency of

[3] `http://www.adobe.com/products/flex.html`
[4] `http://www.microsoft.com/silverlight/`

the saved data. The application tier offers an interface for accessing and storing data. The interface is based on the Representational State Transfer (REST) architectural style. Therefore the Web service API covers best flexibility and availability. The Web service is protected with an authentication and authorization mechanism. The client itself is responsible for data representation and only needs to implement Web service calls to save the recorded data. Thus, a client providing an administrative backend, implemented as Rich Internet Application is required.

Chapter 6

Database

This chapter discusses the implementation details of the database used. Figure 6.1 depicts the database component in the proposed system architecture.

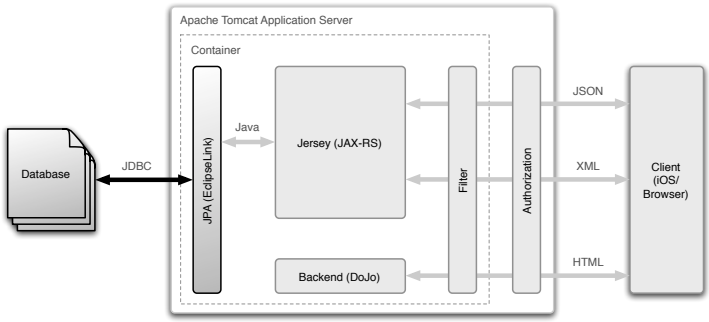


Figure 6.1: Architectural Overview of the Database Component

The relational diagram in Figure 6.2 shows the mapping and relations between the used entities. The database design covers all relationships depicted in Figure 6.2 and is implemented in an object-oriented design. Furthermore the term *"Java persistence"* in the context of the Java Persistence API (JPA) is discussed. An exemplary usage of these techniques and libraries is also specified. Also optimization settings are discussed. The last section describes all entities and the relationships depicted in Figure 6.2.

6.1 Java Persistence with EclipseLink

Java persistence means storing Java objects in a relational database by using the Java Persistence API. In general, JPA is a "Plain Old Java Object (POJO)" based framework offering object-relational mapping [Keith and Schincariol, 2009]. JPA is only a specification (JSR 317[1]) which defines how to persist, access and manage data between a relational database and Java objects [DeMichiel, 2009]. Version 2.0 of the specification was released in 2009 and is the most recent one.

[1] `http://jcp.org/en/jsr/detail?id=317`

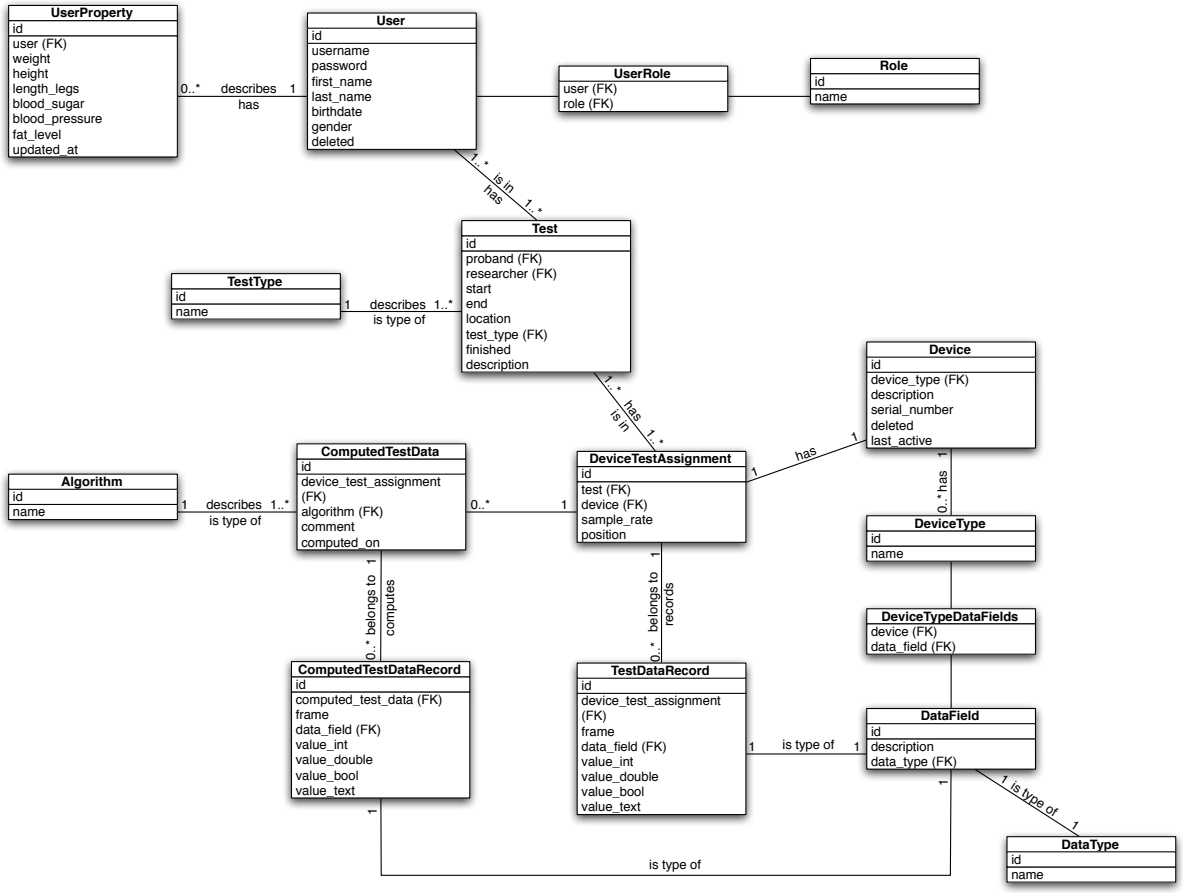


Figure 6.2: Relational Database Model

JPA works with a set of special annotations to "transform" a Java object into a database entity. [Keith and Schincariol, 2009] characterizes an entity with:

- **Persistability:** Entities are *persistable* which means that their state can be represented in a data store and can be accessed later. Entities are not automatically persisted, the application decides when to persist.
- **Identity:** A key that uniquely identifies an entity instance and distinguishes it from all other instances - equivalent to the primary key in the database.
- **Transactionality:** Entities are created, updated and deleted within a transaction. Transactions are committed in the database and are atomic.
- **Granularity:** Entities are meant to be fine-grained objects with an aggregated state stored in a table and should be designed as lightweight as possible.

Since entities do not persist themselves in the meaning of *persistability*, JPA provides an interface called `EntityManager` for this purpose. The `EntityManger` provides basic operations for creating, reading and writing an entity. The managed entities of the `EntityManager` are called *persistence context* [Keith and Schincariol, 2009]. Entity

managers are created by the `EntityManagerFactory` which is defined by the *persistence unit*. This unit defines which classes are used by all entity mangers and correspond to a single `EntityMangerFactory` [Keith and Schincariol, 2009]. The stated concepts and their relation are shown in Figure 6.3.

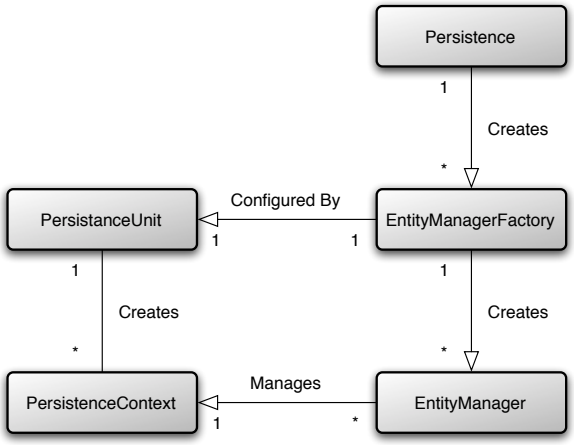


Figure 6.3: Relationships between JPA Concepts [Keith and Schincariol, 2009]

The JPA 2.0 specification is implemented by various vendors, Table 6.1 provides an overview of JPA 2.0 implementations.

Product	Vendor	Version	Release
DataNucleus	DataNucleus	3.0.1	August 2011
EclipseLink	Eclipse Foundation	2.3.0	June 2011
Hibernate	Red Hat	4.0.0.CR3	September 2011
OpenJPA	The Apache Software Foundation	2.1.1	July 2011
TopLink	Oracle	11g Release 1 (11.1.1.4.0)	January 2011

Table 6.1: Java Persistence API 2.0 Implementations

The *EclipseLink*[2] implementation was chosen since it is the reference implementation of the Java Persistence API [Foundation, 2008]. Moreover, *EclipseLink* offers a good documentation and community. *MySQL Community Server*[3] 5.5 is used as production and development database server. The official *MySQL Connector/J*[4] driver is used as Java DataBase Connectivity (JDBC) connector.

6.1.1 Sample Usage of the Java Persistence API

As described perviously, POJOs (Listing A.2 shows a sample class) are transformed to entities with JPA annotations. Applying the entity annotation tells the persistence engine that this class is an entity. Furthermore, also the `@Id` annotation needs the be added to a property to identify it as Primary Key (PK).

[2] `http://www.eclipse.org/eclipselink/`
[3] `http://www.mysql.com/downloads/mysql/`
[4] `http://dev.mysql.com/usingmysql/java/`

To illustrate the entity creation, the relation between the `User` and `Role` (see Figure 6.2) is described. The relation between these two entities is a "n:m" relation, meaning that many users are in many roles. "Transforming" the relationship into Java classes will lead to only two classes shown in Figure 6.4. There is no need to create the junction (join) table (holds the relationship) shown in Figure 6.2. After applying the annotations, the persistence engine will automatically create the junction table.

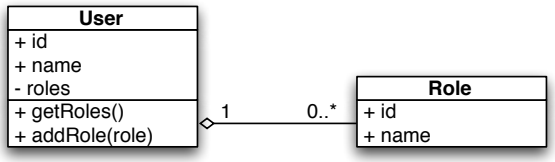


Figure 6.4: Class Diagram of the `User` ↔ `Role` Relationship

To model the "n:m" relationship between the two classes in the persistence context, the `@ManyToMany` annotation is applied to the `roles` property. This bidirectional mapping also requires the usage of a "join table" since it is not possible to save the foreign keys in a single row. The join table contains only the foreign keys of each entity, identifying the relationship. This is archived by applying the `@JoinTable` annotation. Figure 6.5 illustrates the "n:m" relationship using a "join table" and the resulting database tables generated by the persistence engine.

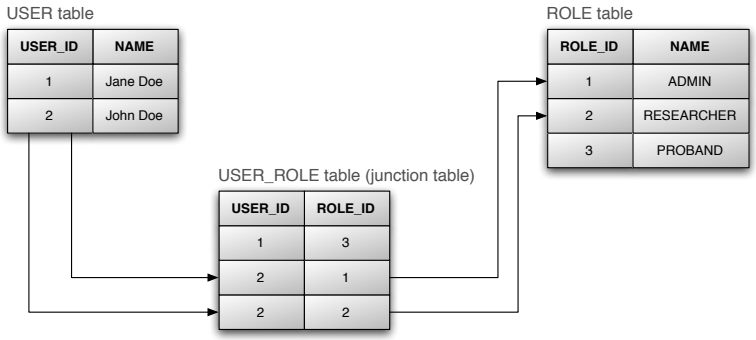


Figure 6.5: Resulting relational database of `User` ↔ `Role` relationship

Listing A.1 and Listing A.3 show the basic implementation of the many-to-many relationship using the annotations mentioned earlier. Listing A.1 also shows the additional parameter required by the `@JoinTable` annotation: `name` (name of the join table), `joinColumns` (join column on the owning side) and `inverseJoinColumns` (join column on the inverse side). The following list provides a short summary by [DeMichiel, 2009] of the JPA annotations used:

- `@Entity`: Specifies that the class is an entity. This annotation is applied to the entity class.
- `@Id`: Specifies the primary key property or field of an entity.

- `@GeneratedValue`: Specification of generation strategies for the values of primary keys. The `GeneratedValue` annotation may be applied to a primary key property.
- `@ManyToMany`: Defines a many-valued association with many-to-many multiplicity. Every many-to-many association has two sides, the owning side and the non-owning (inverse) side.
- `@ManyToOne`: Defines a single-valued association to another entity class.
- `@OneToOne`: Defines a single-valued association to another entity that has one-to-one multiplicity.
- `@OneToMany`: Defines a many-valued association with one-to-many multiplicity.
- `@Temporal`: Must be specified for persistent fields or properties of type `java.util.Date` and `java.util.Calendar`.
- `@Column`: Used to specify a mapped column for a persistent property or field as *unique* or *nullable* for example.
- `@MappedSuperclass`: Designates a class whose mapping information is applied to the entities that inherit from it. A mapped superclass has no separate table defined for it.

Finally, the code in Listing 6.1 shows how to persist an object with the `EntityManager`.

```
1  EntityManagerFactory factory = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);
   EntityManager em = factory.createEntityManager();
3
   em.getTransaction().begin();
5
   User user = new User();
7  user.name = "Jane Doe";
   em.persist(user);
9
   em.getTransaction().commit();
11 em.close();
```

Listing 6.1: Persisting an JPA Entity

6.1.2 Performance Optimization

To improve the performance of EclipseLink, the following persistence unit properties inside the `persitence.xml` (see Listing A.4) should be set [Guide, 2010; Sutherland, 2011; Matthews, 2009]:

- `eclipselink.jdbc.batch-writing`: "`JDBC`", enables batch writing, which means that `INSERT`, `UPDATE` and `DELETE` statements are grouped instead of executing each individually.
- `eclipselink.jdbc.cache-statements`: "`true`", enables statement caching to avoid parsing same statements again.

- `rewriteBatchedStatements`: "`true`", by adding this option to the database Uniform Resource Locator (URL), prepared statements are rewritten with MySQL Connector/J.

With these above settings enabled, a simple test inserting about 120.000 records was reduced from 60 seconds to about 30 seconds.

6.2 Entity Description and Relations

The following section describes each entity shown in Figure 6.2 in more detail with its relation to other entities. The data types depicted in this section are automatically generated by the JPA.

6.2.1 Algorithm

This table contains the name of the algorithm with whom the computed data is generated. This table is referenced by the foreign key `algorihm` of the entity `ComputedTestData`.

Entity: **Algorithm**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
name	varchar(255)	name of the algorithm, unique

Table 6.2: Description of Database Entity `Algorithm`

6.2.2 ComputedTestData

This table describes the computed data which belongs to a `DeviceTestAssignment`.

Foreign keys:

- `algorihtm`: The algorithm used to compute data (`Algorithm`),
- `device_test_assignment`: Foreign Key (FK) which references to the `DeviceTestAssignment`.

Entity: **ComputedTestData**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
deviceTestAssignment (FK)	bigint	device/test the data record belongs to
algorithm (FK)	bigint	algorithm used to compute data
comment	varchar(255)	comment about the data
computedOn	date	when the data has been computed

Table 6.3: Description of Database Entity `ComputedTestData`

6.2.3 ComputedTestDataRecord

A `ComputedTestDataRecord` holds a single value of a set of computed data. The result of a vector with the values a, b, c would cause three records with the same frame number to be inserted. A single record belongs to just one data field which holds the information of the data type.

Foreign keys:

- `computed_test_data`: Reference to the computed data record (`ComputedTestData`),
- `data_field`: Reference to the used data field (`DataField`).

Entity: **ComputedTestDataRecord**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
computed_test_data (FK)	bigint	reference to the computed test data assignment
frame	int	frame number the value belongs to
data_field (FK)	bigint	data type (field) this record holds
integerValue	int(11)	the corresponding integer value
doubleValue	double	the corresponding double value
boolValue	tinyint	the corresponding boolean value
textValue	varchar(11)	the corresponding text value

Table 6.4: Description of Database Entity `ComputedTestDataRecord`

6.2.4 DataType

This table holds the available data types and is referenced by the FK `data_type` of the entity `DataField`. There are only four fixed data types: *Integer*, *Double*, *Boolean* and *Text*.

Entity: **DataType**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
name	varchar(255)	name of the data type, unique

Table 6.5: Description of Database Entity `DataType`

6.2.5 DataField

A `DataField` describes a sensor (for example "Accelerometer X") with the corresponding data type.

Foreign keys:

- `data_type`: Reference to the used data type (`DataType`).

Entity: **DataField**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
data_type	bigint	data type of the field
description	varchar(255)	description of the field, unique

Table 6.6: Description of Database Entity `DataField`

6.2.6 Device

This table holds information about a concrete device and is defined through the device type. It is referenced by the FK `device` of the entity `DeviceTestAssignment`.

Foreign keys:

- `device_type`: Reference to the device type (`DeviceType`)

Entity: **Device**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
device_type (FK)	bigint	type of the device
description	varchar(255)	description of the device
serialNumber	varchar(255)	serial number of the device
deleted	tinyint	flag that indicates if the device is deleted, default `false`
last_active	date	timestamp of the device's last activity

Table 6.7: Description of Database Entity `Device`

6.2.7 DeviceTestAssignment

A `DeviceTestAssignment` holds the relationship between a `Test` and `Device` (therefore the devices of a test) and the used device settings.

Foreign keys:

- `test`: Reference to the test (`Test`),
- `device`: Reference to the device (`Device`).

Entity: **DeviceTestAssignment**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
test (FK)	bigint	reference to test
device (FK)	bigint	reference to device
sampleRate	int(11)	sampling rate of the device
position	varchar(255)	position of the device

Table 6.8: Description of Database Entity `DeviceTestAssignment`

6.2.8 DeviceType

This table holds the available device types such as *"iPhone 4"* or *"Insole"* for example. It also defines the device data field through a many-to-many association to `DataField` through the junction table `DeviceTypeDataField`.

Entity: **DeviceType**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
name	varchar(255)	name of the device type, unique

Table 6.9: Description of Database Entity `DeviceType`

6.2.9 Role

This table holds the available user roles which are *"ADMIN"*, *"RESEARCHER"* and *"PROBAND"*.

Entity: **Role**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
name	varchar(255)	name of the role, unique

Table 6.10: Description of Database Entity `Role`

6.2.10 Test

A `Test` holds all necessary information about taken tests. It references to the proband who took the test, the researcher who observed the test and the type of test. All devices which were used are referenced by the `test` FK in the table `DeviceTestAssignment`.

Foreign keys:

- `proband`: Reference to `User`,
- `researcher`: Reference to `User`,
- `type`: Reference to the test type (`TestType`).

Entity: **Test**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
proband (FK)	bigint	the person to be tested
researcher (FK)	bigint	the person observing the test
start	date	start time of the test
end	date	end time of the test
location	varchar(255)	the location of the test
status	varchar(255)	status of the test
type (FK)	bigint	the type of the test
finished	tinyint	indicates if the test is finished
description	varchar(255)	description of the test

Table 6.11: Description of Database Entity `Test`

6.2.11 TestDataRecord

This table stores the captured data from a device. A `TestDataRecord` holds a single value of a set of device data. The result of a vector with the values a, b, c would cause three records with the same frame number to be inserted. A single record belongs to just one data field which holds the information of the data type.

Foreign keys:

- `device_test_assignment`: Reference to the device test assignment (`DeviceTestAssignment`)
- `data_field`: Reference to the used data field (`DataField`)

Entity: **TestDataRecord**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
device_test_assignment (FK)	bigint	device/test the data record belongs to
frame	int	the frame number the values belong too
data_field (FK)	bigint	data type (field) this record holds
integerValue	int(11)	the corresponding integer value
doubleValue	double	the corresponding double value
boolValue	tinyint	the corresponding boolean value
textValue	varchar(11)	the corresponding text value

Table 6.12: Description of Database Entity `TestDataRecorded`

6.2.12 TestType

This table contains the test types. It is referenced by the FK `test_type` of the entity `Test`.

Entity: **TestType**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
name	varchar(255)	name of the test type, unique

Table 6.13: Description of Database Entity `TestType`

6.2.13 User

This table holds the available users. A user belongs to a number of roles through a many-to-many association to `Roles` through the junction table `UserRole`.

Entity: **User**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
username	varchar(255)	name of the user, used to identify a user, unique
password	varchar(255)	password of the user, saved as `MD5()` hash value
firstName	varchar(255)	first name of the user
lastName	varchar(255)	last name of the user
birthdate	date	the date of birth of the user
gender	varchar(255)	the gender of the user
deleted	tinyint	flag that indicates if the user is deleted, default `false`

Table 6.14: Description of Database Entity `User`

6.2.14 UserProperties

This table holds additional properties of a user. A user can have a dynamic number of property entities. The `updated_at` field indicates when the record has been created or updated.

Foreign keys:

- `user`: Reference to the user the record belongs to (`User`)

Entity: **UserProperties**		
Field	**Type**	**Description**
id	bigint	primary key, auto-increment
user (FK)	bigint	the whom this property belongs to
height	float	the body height of the person in cm
weight	float	the users weight in kg
length_legs	float	the length of the user's legs in cm
blood_sugar	varchar(255)	blood sugar
blood_pressure	varchar(255)	blood pressure in the format systolic/diastolic
fat_level	float	the useres fat level in %
updated_at	date	indicates when the record has been last updated

Table 6.15: Description of Database Entity `UserProperties`

6.3 Summary

This chapter covered the development of the object relational database through the Java Persistence API using the *EclipseLink* reference implementation. The persistence API specifies how entities are persisted and managed between a relational database and Java objects. Thus, entities characterized as *persistable*, *transactional*, *granular* and *unambiguous* and persisted through the entity manager. An example showed how JPA annotations are used to "transform" an object in to an entity.

Chapter 7

RESTful Web Service

This chapter defines the services implemented as well as the techniques and libraries used. The first section provides an architectural overview and the next section details the Web service definition. The definitions of the custom error codes are listed in Section 7.10 and the complete definition of the HTTP Status Codes in Section 7.11. If not mentioned otherwise, all parameters are required. Figure 7.1 shows the components discussed in this chapter.

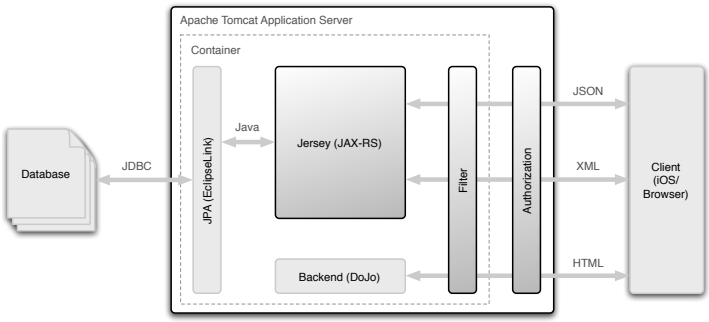


Figure 7.1: Architectural Overview of the Service Components

7.1 Restful Web Services with JAX-RS

As defined in the previous chapter, the API is implemented as a REST Web service in Java. The JSR 311[1] specification, also called Java API for RESTful Web Services (JAX-RS), defines a set of annotations used for the development of Web services according to the Representational State Transfer architectural style [Hadley and Sandoz, 2009].

The implementation of the Web service uses the *Jersey*[2] framework, which is according to [Community, 2011] the production quality reference implementation of the JSR 311 specification. The annotations defined in JSR 311 are simply applied to Java objects. The listing in Appendix A.5 shows the basic usage of the following annotations:

[1] `http://jcp.org/en/jsr/detail?id=311`
[2] `http://jersey.java.net`

- `@GET`, `@POST`, `@PUT`, `@DELETE`: Defines the HTTP request type of the resource.
- `@Path`: Defines a root resource.
- `@QueryParam`: Extracts the value of a URI query parameter.
- `@PathParam`: Extracts the value of a URI template parameter.

The two required output formats, XML and JSON, are automatically generated (*marshalling*) by enabling the JSON support of Jersey in the configuration options in the project's `web.xml` (see Listing 7.1).

```
<init-param>
  <param-name>com.sun.jersey.api.json.POJOMappingFeature</param-name>
  <param-value>true</param-value>
</init-param>
```

Listing 7.1: Enabling JSON Support in Jersey

The client needs to set the HTTP header field "`ACCEPT`" to either "`application/xml`" or "`application/json`" in order to get the response in the desired format. Otherwise the Web service returns the HTTP error 406 (*"Not Acceptable"*). To enable GNU Zip (GZIP) encoding the container filters in Listing 7.2 are added to `web.xml`.

```
<!-- For handling gzip-ed request bodies -->
<init-param>
  <param-name>com.sun.jersey.spi.container.ContainerRequestFilters</param-name>
  <param-value>com.sun.jersey.api.container.filter.GZIPContentEncodingFilter</param-value>
</init-param>

<!-- For creating gzip-ed responses -->
<init-param>
  <param-name>com.sun.jersey.spi.container.ContainerResponseFilters</param-name>
  <param-value>com.sun.jersey.api.container.filter.GZIPContentEncodingFilter</param-value>
</init-param>
```

Listing 7.2: Enabling GZIP Support in Jersey

As application server Apache Tomcat[3] is used. It serves also the static backend (see Section 8). Moreover Apache Tomcat is also responsible for client authentication through HTTP Basic Authentication which is defined in Section 7.8.

The Web service uses the ISO 8601[4] standard for date and time representation which is accomplished with the *Joda Time*[5] library.

It should be noted, that the current Web service returns *all* requested records. This can be avoided by using a paging mechanism where only a smaller set of the requested records is returned, if the number of records becomes very large.

7.1.1 Accepted Query Parameters

The Web service API accepts the input query parameters according to [Berners-Lee et al., 1994]. The HTTP URL scheme is defined as `http://<host>:<port>/<path>?<searchpart>` where [Berners-Lee et al., 1994] defines

[3] `http://tomcat.apache.org`
[4] `http://www.rfc-editor.org/rfc/rfc3339.txt`
[5] `http://joda-time.sourceforge.net`

- <host>: as the fully qualified domain name or IP address,
- <port>: as the port number to connect to,
- <path>: as an HTTP selector, and
- <searchpart>: as the query string.

The query string is composed by a variable number of key-value pairs (key=value) and separated by &. The following example shows the usage of two query parameters: http://example.com:8080/users/username=UniqueUsername&password=SecurePassword.

The Web service accepts the following value data types:

- String (string): Used for text, e.g., /location=Graz.
- Integer (int): Integer numbers, e.g., /frame=1.
- Float (float): Real numbers, e.g., /weight=88.34.
- Boolean (bool): Represents just two states, true and false, e.g., /deleted=true.

7.2 Service: /tests

The service /tests and its sub-services are defined to retrieve and create tests, store computed test data and device data. The following services are explained in more detail in this section:

- /tests: Returns a list of available tests or creates new a test.
- /tests/computedrecords/<device_assignment_id>: Returns a list of computed test data assignments or creates a new assignment by appending computed data.
- /tests/computedrecords/<device_assignment_id>/record/<id>: Returns submitted data of or deletes computed data.
- /tests/devicerecords/<device_assignment_id>: Stores recorded device data or retrieves stored device data of a test.
- /tests/running: Returns a list of device assignments for current running tests.
- /tests/test/<id>: Get information about specific test, update or delete test.
- /tests/test/<id>/devices: Returns a list of device assignments of a test or adds a new device to a test.
- /tests/test/<id>/devices/device/<device_assignment_id>: Delete device from test.

- `/tests/test/<id>/start|finish|stop`: Starts, finishes or cancels a test.
- `/tests/testtypes`: Returns a list of test types or creates a new test type.
- `/tests/testtypes/testtype/<id>`: Get information about specific test type, update or delete test type.

7.2.1 /tests

Returns a list of available tests or creates a new test. The user can only retrieve a test where he is assigned (ADMIN gets all tests, PROBAND and RESEARCHER get only tests where they are attendees). The supplied proband and researcher user have to be in the supplied role; deleted users are not accepted.

Parameter (GET):

`exclude_finished` **(bool) (optional):** Exclude already finished tests, `true` or `false`, default `false`.

Parameter (POST):

`proband_user_id` **(int):** The ID of the user who performs the test.

`researcher_user_id` **(int):** The ID of the user who observes the test.

`location` **(string) (optional):** Location of the test.

`description` **(string):** Short description of the test.

`test_type_id` **(int):** The ID of the test type.

	GET	POST	PUT	DELETE
Description	List of tests.	Create new test.	Not supported.	
Permission	ALL	ADMIN		
HTTP Status Code	200, 404	201, 400, 403	405	
Error Code	103	101, 102	100	

Permissions and Supported HTTP Methods of Service `/tests`

7.2.2 /tests/computedrecords/<device_assignment_id>

Returns a list of computed test data assignments or creates a new assignment by appending computed data.

To submit newly computed data, the data is attached in the request body either as plain text or gzip-ed CSV file. To submit data, the HTTP header field "`Accept`" needs to be "`text/plain`" for plain text data or "`application/gzip`" for gzip-ed body data. The data fields for the submitted data can be freely chosen as long the field is unique and

exists. For additional requirements see service definition /tests/devicerecords/<device_assignment_id>.

Parameter (GET):

<device_assignment_id> (int): The device assignment ID (holds the information which device belongs to a test).

Parameter (POST):

<device_assignment_id> (int): The device assignment ID (holds the information which device belongs to a test).

algoritm_id (int): ID of the algorithm used for the computed results.

comment (string) (optional): Comment of the record.

	GET	POST	PUT DELETE
Description	Get list of computed test data.	Save new computed data.	Not supported.
Permission	ADMIN	ADMIN	
HTTP Status Code	200, 403, 404	200, 400, 403, 404, 409	405
Error Code	102, 103	101, 102, 103, 104	100

Permissions and Supported HTTP Methods of Service /tests/computedrecords/<device_assignment_id>

Example: *Append newly computed data to device assignment with ID = 2. The device data is computed with the "Band Bass" algorithm. The computed result is of the data type "Text" with the value "No Result.". Listing 7.3 shows the corresponding HTTP message.*

```
1 POST /tests/computedrecords/2/?algorithm_id=51&comment=Short comment HTTP/1.1
  Host: www.example.com
3
  Text
5 No Result.
```

Listing 7.3: Example HTTP Message for Submitting Computed Data

Response XML

```
1 <computedTestData>
    <id>51</id>
3   <algorithm>
      <id>51</id>
5     <name>Band Bass</name>
    </algorithm>
7   <comment>Short comment</comment>
    <computedOn>2011-09-14T09:15:47.569+02:00</
        computedOn>
9 </computedTestData>
```

Response JSON

```
1 {
    "id": 51,
3   "algorithm": {
      "name": "Band Bass",
5     "id": 51
    },
7   "comment": "Short comment",
    "computedOn": "2011-09-14T09:15:47.569+02:00"
9 }
```

Example: GET /tests/computedrecords/2

Get list of computed data for device assignment with ID = 2.

Response XML

```
<computedTestDatas>
  <computedTestData>
    <id>1</id>
    <algorithm>
      <id>1</id>
      <name>Simple Fall Detection</name>
    </algorithm>
    <comment>Evaluation of fall detection</comment
        >
    <computedOn>2011-09-13T16:10:48.000+02:00</
        computedOn>
  </computedTestData>
  <computedTestData>
    <id>51</id>
    <algorithm>
      <id>51</id>
      <name>Band Bass</name>
    </algorithm>
    <comment>Short comment</comment>
    <computedOn>2011-09-14T09:15:47.000+02:00</
        computedOn>
  </computedTestData>
</computedTestDatas>
```

Response JSON

```
[
  {
    "id": 1,
    "algorithm": {
      "name": "Simple Fall Detection",
      "id": 1
    },
    "comment": "Evaluation of fall detection",
    "computedOn": "2011-09-13T16:10:48.000+02:00"
  },
  {
    "id": 51,
    "algorithm": {
      "name": "Band Bass",
      "id": 51
    },
    "comment": "Short comment",
    "computedOn": "2011-09-14T09:15:47.000+02:00"
  }
]
```

7.2.3 /tests/computedrecords/<device_assignment_id>/record/<id>

Get submitted data or delete computed data. The HTTP header field "`Accept`" needs to be "`text/plain`" or "`application/gzip`" to retrieve the stored data.

Parameter:

`<device_assignment_id>` **(int):** The device assignment ID (holds the information which device belongs to a test).

`<id>` **(int):** The ID of the computed record.

	GET	POST	PUT	DELETE
Description	Get details of computed record.	Not supported.		Delete computed record.
Permission	ADMIN			ADMIN
HTTP Status Code	200, 400, 403, 404	405		200, 400, 403, 404
Error Code	101, 102, 103	100		101, 102, 103

Permissions and Supported HTTP Methods of Service `/tests/computedrecords/<device_assignment_id>/record/<id>`

Example: `GET /tests/computedrecords/2/record/51`
Get computed data, "`Accept`" is set to "`text/plain`".

Response "`text/plain`"

```
Text
No result.
```

7.2.4 /tests/devicerecords/<device_assignment_id>

Stores recorded device data or returns stored device data of a test/device assignment.

For storing device data, the data is attached in the request body either as gzip-ed or plain text Comma-separated Values (CSV) file. It is not possible to append device records if there is already existing data, the returned HTTP status code would be *409 ("Conflict")*. The attached CSV file should met the following requirements:

- The values (including header) are separated by comma (",").
- The header fields have the same name as the device data fields.
- Each single device data field has to be submitted.
- The number of values (columns) is equal to the number of device data fields.
- The value and the data field data type should belong together.
- The status of the test needs to be finished to submit data.

If one of the requirements is not met, a corresponding error is raised. To submit data, the HTTP header field "`Accept`" needs to be "`text/plain`" for plain text data or "`application/gzip`" for gzip-ed body data.

Parameter:

`<device_assignment_id>` **(int):** The device assignment ID (holds the information which device belongs to a test).

	GET	POST	PUT	DELETE
Description	Get recorded device data.	Store device data.	Not supported.	
Permission	ADMIN	ADMIN, PROBAND		
HTTP Status Code	200, 400, 403, 404	201, 400, 403, 404, 406, 409, 500	405	
Error Code	101, 102, 103	101, 102, 103, 104, 110	100	

Permissions and Supported HTTP Methods of Service `/tests/devicerecords/<device_assignment_id>`

Example: *This example shows how to submit recorded device data. We assume that the device behind the test/device assignment has the following data fields: "`Accelerometer X`", "`Accelerometer Y`" and "`Accelerometer Z`". The CSV file is contained in the HTTP message like the example in Listing 7.4. For success, the HTTP status code 201 (Created) is returned.*

```
POST /tests/devicerecords/1 HTTP/1.1
Host: www.example.com

Accelerometer X,Accelerometer Y,Accelerometer Z
1.0,2.0,4.0
5.4,4.4,5.6
```

Listing 7.4: Example HTTP Message for Submitting Device Data

7.2.5 /tests/running

Returns a list of device assignments for the given device were the following conditions are met:

- The device is assigned to a test.
- The current user is assigned as test user in the test (except ADMIN).
- The test is not finished.
- The test is started (e.g. the start time is set)

This service only returns one single assignment, since a unique device can only be part of one running test. This service is usually called by the device itself, therefore the device `last_active` date is updated.

Parameter:

`for_device_id` **(int):** Device ID used to search for running tests.

	GET	POST	PUT	DELETE
Description	Returns device test assignment.	Not supported.		
Permission	ADMIN, PROBAND			
HTTP Status Code	200, 400, 403, 404	405		
Error Code	101, 102, 103	100		

Permissions and Supported HTTP Methods of Service `/tests/running`

Example: `GET /tests/running/?for_device_id=1`
Get device test assignment for device with `ID = 1`.

Response XML

```
<deviceTestAssignment>
  <id>3</id>
  <device>
    <deleted>false</deleted>
    <description>iPhone 4 Black</description>
    <deviceType>
      <id>1</id>
      <name>iPhone 4</name>
    </deviceType>
    <id>1</id>
    <lastActive>2011-09-13T16:51:27.744+02:00</
        lastActive>
    <serialNumber>1234567890</serialNumber>
  </device>
  <sampleRate>50</sampleRate>
  <position>Hip</position>
</deviceTestAssignment>
```

Response JSON

```
{
  "id": 3,
  "device": {
    "id": 1,
    "serialNumber": "1234567890",
    "description": "iPhone 4 Black",
    "deleted": false,
    "lastActive": "2011-09-13T16:50:26.090+02:00",
    "deviceType": {
      "id": 1,
      "name": "iPhone 4"
    }
  },
  "sampleRate": 50,
  "position": "Hip"
}
```

7.2.6 /tests/test/<id>

Get information about specific test, update or delete test. ADMIN has permission to all tests, PROBAND and RESEARCHER only permission to tests where they are assigned. If a test is deleted all assigned devices and computed data are deleted too.

Parameter:

`<id>` **(int):** The ID of the test.

`proband_user_id` **(int) (optional):** The ID of the user who performs the test.

`researcher_user_id` **(int) (optional):** The ID of the user who observes the test.

`location` **(string) (optional):** Location of the test.

`description` **(string) (optional):** Short description of the test.

`test_type_id` **(int) (optional):** The ID of the test type.

	GET	POST	PUT	DELETE
Description	Details about test	Not supported.	Update test.	Delete test.
Permission	ALL		ADMIN	ADMIN
HTTP Status Code	200, 400, 403, 404	405	200, 400, 403, 404	204, 403, 404
Error Code	101, 102, 103	100	101, 102, 103	102, 103

Permissions and Supported HTTP Methods of Service `/tests/test/<id>`

7.2.7 /tests/test/<id>/devices

Returns a list of device assignments of a test or adds a new device to a test.

Parameter (GET):

`<id>` **(int):** The ID of the test.

Parameter (POST):

`<id>` **(int):** The ID of the test to add a device.

`device_id` **(int):** The ID of the device to add.

`sample_rate` **(int):** Sample rate of the device (Hz).

`position` **(string):** Position of the device (usually body location).

	GET	POST	PUT	DELETE
Description	List of assigned devices.	Add new device to test.	Not supported.	Delete device assignment.
Permission	ADMIN	ADMIN		ADMIN
HTTP Status Code	200, 403, 404	201, 400, 403, 404	405	204, 403, 404
Error Code	102, 103	101, 102, 103	100	101, 103, 105

Permissions and Supported HTTP Methods of Service `/tests/test/<id>/devices`

Example: `POST /tests/test/1/devices/?device_id=1&sample_rate=100&position=Hip`
Add new device with a sample rate of 100Hz located on the hip to test with `ID = 1`.

Response XML

```
<deviceTestAssignment>
  <id>51</id>
  <device>
    <deleted>false</deleted>
    <description>iPhone 4 Black</description>
    <deviceType>
      <id>1</id>
      <name>iPhone 4</name>
    </deviceType>
    <id>1</id>
    <lastActive>2011-09-13T16:51:27.744+02:00</
        lastActive>
    <serialNumber>1234567890</serialNumber>
  </device>
  <sampleRate>100</sampleRate>
  <position>Hip</position>
</deviceTestAssignment>
```

Response JSON

```
{
    "id": 51,
    "device": {
        "id": 1,
        "serialNumber": "1234567890",
        "description": "iPhone 4 Black",
        "deleted": false,
        "lastActive": "2011-09-13T16
            :51:27.744+02:00",
        "deviceType": {
            "id": 1,
            "name": "iPhone 4"
        }
    },
    "sampleRate": 100,
    "position": "Hip"
}
```

7.2.8 /tests/test/<id>/devices/device/<device_assignment_id>

Delete device from test.

Parameter:

`<id>` **(int):** The ID of the test.

`device_assignment_id` **(int):** The ID of the device assignment.

	GET	POST	PUT	DELETE
Description	Not supported.			Delete device assignment.
Permission				ADMIN
HTTP Status Code	405			204, 403, 404
Error Code	100			102, 103

Permissions and Supported HTTP Methods of Service /tests/test/<id>/devices

7.2.9 /tests/test/<id>/start|finish|stop

Starts, finishes or cancels a test. Cancel simply resets the start/end time and the finished flag of the given test. If it is not possible to set a test in the required status, a HTTP error is returned with a detailed error message (see Section 7.10).

- *Cancel* test: Only tests with status *started* or *no* status can be canceled.
- *Start* test: Only test with *no* status can be started.
- *Finish* test: Only tests which are *started* can be finished.

Parameter:

<id> (int): The ID of the test.

	GET	POST	PUT	DELETE
Description	Not supported.		Start, cancel, finish test.	Not supported.
Permission			ADMIN	
HTTP Status Code	405		200, 403, 404, 500	450
Error Code	100		102, 103, 105, 106	100

Permissions and Supported HTTP Methods of Service /tests/test/<id>/start|finish|stop

Example: PUT /tests/test/1/start
Start test with ID = 1.

Response XML

```
<error>
  <id>107</id>
  <message>Internal Server Error</message>
  <detailedMessage>Unable to start test. Test
      already started or finished.</
      detailedMessage>
</error>
```

Response JSON

```
{
  "id": 107,
  "message": "Internal Server Error",
  "detailedMessage": "Unable to start test. Test
      already started or finished."
}
```

7.2.10 /tests/testtypes

Returns a list of test types or creates a new test type.

Parameter (POST):

`name` **(string):** Unique name of the test type.

	GET	POST	PUT	DELETE
Description	List of test types	Create new test type.	Not supported.	
Permission	ADMIN	ADMIN		
HTTP Status Code	200, 403	201, 400, 403, 409	405	
Error Code	102	101, 102, 104	100	

Permissions and Supported HTTP Methods of Service `/tests/testtypes`

Example: `GET /tests/testtypes`
Get list of all test types.

Response XML

```
<testTypes>
  <testType>
    <id>3</id>
    <name>Running</name>
  </testType>
  <testType>
    <id>2</id>
    <name>Shaking</name>
  </testType>
  <testType>
    <id>1</id>
    <name>STS5</name>
  </testType>
</testTypes>
```

Response JSON

```
[
  {
    "name": "Running",
    "id": 3
  },
  {
    "name": "Shaking",
    "id": 2
  },
  {
    "name": "STS5",
    "id": 1
  }
]
```

7.2.11 /tests/testtypes/testtype/<id>

Get information about specific test type, update or delete test type.

Parameter:

`<id>` **(int):** The ID of the test type.

`name` **(optional) (string):** Unique name of the test type.

	GET	POST	PUT	DELETE
Description	Details about the test type.	Not supported.	Update test type.	Delete test type.
Permission	ALL		ADMIN	ADMIN
HTTP Status Code	200, 400, 403, 404	405	200, 400, 403, 404, 409	204, 400, 403, 404
Error Code	101, 102, 103	100	101, 102, 103, 104	101, 102, 103

Permissions and Supported HTTP Methods of Service `/tests/testtypes/testtype/<id>`

Example: `GET /tests/testtypes/testtype/2`
Get test type with `ID = 2`.

Response XML

```
<testType>
  <id>2</id>
  <name>Shaking</name>
</testType>
```

Response JSON

```
{
  "name": "Shaking",
  "id": 2
}
```

7.3 Service: /status

The `/status` service is used to get the current status of a test:

- `/status`: Returns the status of a test or the assigned test of a device assignment.

7.3.1 /status

Returns the status of a test or the assigned test of a device assignment. PROBAND and RESEARCHER are only allowed to get the status of tests where they are assigned, ADMIN has permission to all tests. If the request contains the `for_device_assignment_id` parameter, the assigned device last activity field is updated. The returned status codes are defined as follows:

- `0`: Test has no status.
- `1`: Test is finished.
- `2`: Test is running.

Parameter:

`for_test_id` **(int):** The ID of the test type.

`for_device_assignment_id` **(int):** The ID of the test/device assignment.

	GET	POST	PUT	DELETE
Description	Status of the test.	Not supported.		
Permission	ALL			
HTTP Status Code	200, 400, 403, 404	405		
Error Code	101, 102, 103, 105, 106, 109	100		

Permissions and Supported HTTP Methods of Service `/status`

Example: `GET /status/?for_test_id=1`
Get status of test with `ID = 1`.

Response XML

```
<testStatus>
  <status>2</status>
  <testID>1</testID>
</testStatus>
```

Response JSON

```
{
  "status": 2,
  "testID": 1
}
```

7.4 Service: `/users`

The `/users` services are used to create new users and user properties. Access to all services is only granted to the ADMIN role with one exception: `/users/user` which returns the current user. The following services are provided:

- `/users`: Returns a list of available users or creates a new user.
- `/users/user/<id>`: Get information about a specific user, update or delete a user.
- `/users/roles`: Returns a list of available roles.
- `/users/user`: Returns the current active (logged in) user.
- `/users/user/<id>/properties`: Returns list of all user properties or creates a new property.
- `/users/user/<id>/properties/property/<property_id>`: Get information about a specific user property, update or delete a property.

7.4.1 /users

Returns a list of available users or creates a new user.

Parameter (GET):

`role` **(int) (optional):** Filter users by role ID.

`deleted` **(bool) (optional):** Include deleted users (`true` or `false`, default: `false`).

Parameter (POST):

`username`**:** The unique username of the user.

`password`**:** The password of the user.

`first_name`**:** First name of the user.

`last_name`**:** Last name of the user.

`birthdate`**:** Birthdate of the user (format YYYY-MM-DD).

`gender`**:** Gender of the user: `Male` or `Female`.

`roles`**:** Comma separated list of role IDs.

	GET	POST	PUT	DELETE
Description	List of users.	Create new user.	Not supported.	
Permission	ADMIN	ADMIN		
HTTP Status Code	200, 400, 403, 404	201, 400, 403, 409	405	
Error Code	101, 102, 103	101, 102, 104	100	

Permissions and Supported HTTP Methods of Service `/users`

Example: `POST /users/?username=new.user&first_name=New&last_name=User&birthdate=1970-12-25&roles=1,3&gender=Male&password=test`
Create new user.

Response XML

```
<user>
  <id>51</id>
  <username>new.user</username>
  <firstName>New</firstName>
  <lastName>User</lastName>
  <birthdate>1970-12-25</birthdate>
  <deleted>false</deleted>
  <gender>Male</gender>
  <roles>
    <role>
      <id>1</id>
      <name>ADMIN</name>
    </role>
    <role>
      <id>3</id>
      <name>RESEARCHER</name>
    </role>
  </roles>
</user>
```

Response JSON

```
{
  "id": 51,
  "username": "new.user",
  "firstName": "New",
  "lastName": "User",
  "birthdate": "1970-12-25",
  "deleted": false,
  "gender": "Male",
  "roles": [
    {
      "name": "ADMIN",
      "id": 1
    },
    {
      "name": "RESEARCHER",
      "id": 3
    }
  ]
}
```

7.4.2 /users/roles

Returns a list of available roles.

	GET	POST	PUT	DELETE
Description	List of all roles.	Not supported.		
Permission	ADMIN			
HTTP Status Code	200, 403, 404	405		
Error Code	102, 103	100		

Permissions and Supported HTTP Methods of Service `/users/roles`

Example: `GET /users/roles`
Get list of all roles.

Response XML

```
<roles>
  <role>
    <id>1</id>
    <name>ADMIN</name>
  </role>
  <role>
    <id>2</id>
    <name>PROBAND</name>
  </role>
  <role>
    <id>3</id>
    <name>RESEARCHER</name>
  </role>
</roles>
```

Response JSON

```
[
  {
    "name": "ADMIN",
    "id": 1
  },
  {
    "name": "PROBAND",
    "id": 2
  },
  {
    "name": "RESEARCHER",
    "id": 3
  }
]
```

7.4.3 /users/user

Returns the current active (logged in) user.

	GET	POST	PUT	DELETE
Description	Current user.	Not supported.		
Permission	ALL			
HTTP Status Code	200, 403, 500	405		
Error Code	102, 111	100		

Permissions and Supported HTTP Methods of Service `/users/user`

7.4.4 /users/user/<id>

Get information about specific a user, update or delete a user.

Parameter (at least on optional parameter is required):

`<id>` **(int):** The ID of the user.

`username` **(string) (optional):** The unique username of the user.

`password` **(string) (optional):** The password of the user.

`first_name` **(string) (optional):** First name of the user.

`last_name` **(string) (optional):** Last name of the user.

birthdate (string) (optional): Birthdate of the user (format YYYY-MM-DD).

gender (string) (optional): Gender of the user: `Male` or `Female`.

roles (string) (optional): Comma separated list of role IDs.

deleted (bool) (optional): Mark user as deleted (`true`) or undeleted (`false`).

	GET	POST	PUT	DELETE
Description	Details about the user.	Not supported.	Update user.	Mark user as deleted.
Permission	ADMIN		ADMIN	ADMIN
HTTP Status Code	200, 400, 403, 404	405	200, 400, 403, 404, 409	200, 403, 404
Error Code	101, 102, 103	100	101, 102, 103, 104	102, 103

Permissions and Supported HTTP Methods of Service `/users/user/<id>`

Example: `PUT /users/user/101/?username=new.user`
Update the username of user with `ID = 101` (duplicate entry, username already taken).

Response XML

```
<error>
  <id>104</id>
  <message>Conflict</message>
  <detailedMessage>user with username 'new.user'
      already exists</detailedMessage>
</error>
```

Response JSON

```
{
  "id": 104,
  "message": "Conflict",
  "detailedMessage": "user with username 'new.user
      ' already exists"
}
```

7.4.5 /users/user/<id>/properties

Returns a list of all user properties or creates a new property. If a property is created of updated the `created_on` field is updated to the current time to keep track when the property has been changed.

Parameter (POST):

<id> (int): The ID of the user.

weight (float) (optional): The weight of the user (kg).

height (float) (optional): The height of the user (cm).

length_legs (float) (optional): The length of the user's legs (cm).

blood_sugar (string) (optional): The user's blood sugar.

blood_pressure (string) (optional): The user's blood pressure (usually `systolic/diastolic`)

fat_level (float) (optional): The user's fat level.

	GET	POST	PUT	DELETE
Description	Get all properties of the user.	Create new user property.	Not supported.	
Permission	ADMIN	ADMIN		
HTTP Status Code	200, 403, 404	201, 400, 403, 404	405	
Error Code	102, 103	101, 102, 103	100	

Permissions and Supported HTTP Methods of Service `/users/user/<id>/properties`

Example: `GET /users/user/2/properties`
Get properties of user with `ID = 2`.

Response XML

```
<userProperties>
  <userProperty>
    <id>2</id>
    <height>180.0</height>
    <weight>99.0</weight>
    <lengthLegs>90.0</lengthLegs>
    <updatedAt>2011-09-13T16:10:17.000+02:00</
        updatedAt>
  </userProperty>
  <userProperty>
    <id>3</id>
    <weight>101.0</weight>
    <updatedAt>2011-09-13T16:10:17.000+02:00</
        updatedAt>
  </userProperty>
</userProperties>
```

Response JSON

```
[
  {
    "id": 2,
    "height": 180,
    "weight": 99,
    "lengthLegs": 90,
    "updatedAt": "2011-09-13T16:10:17.000+02:00"
  },
  {
    "id": 3,
    "weight": 101,
    "updatedAt": "2011-09-13T16:10:17.000+02:00"
  }
]
```

Example: `POST /users/user/2/properties/?weight=88.2`
Create new user property with `weigt = 88.2`kg.

Response XML

```
<userProperty>
  <id>51</id>
  <weight>88.2</weight>
  <updatedAt>2011-09-13T18:09:36.798+02:00</
      updatedAt>
</userProperty>
```

Response JSON

```
{
  "id": 51,
  "weight": 88.2,
  "updatedAt": "2011-09-13T18:09:36.798+02:00"
}
```

7.4.6 /users/user/<id>/properties/property/<property_id>

Get information about a specific user property, update or delete a property.

Parameter:

`<id>` **(int):** The ID of the user.

`<property_id>` **(int):** The ID of the property.

`weight` **(float) (optional):** The weight of the user (kg).

`height` **(float) (optional):** The height of the user (cm).

`length_legs` **(float) (optional):** The length of the user's legs (cm).

`blood_sugar` **(string) (optional):** The user's blood sugar.

`blood_pressure` **(string) (optional):** The user's blood pressure (usually *"systolic/diastolic"*).

`fat_level` **(float) (optional):** The user's fat level.

	GET	POST	PUT	DELETE
Description	Get information of property.	Not supported.	Update property.	Delete property.
Permission	ADMIN		ADMIN	ADMIN
HTTP Status Code	200, 403, 404	405	200, 400, 403, 404	204, 400, 403, 404
Error Code	102, 103	100	101, 102, 103	101, 102, 103

Permissions and Supported HTTP Methods of Service `/users/user/<id>/properties/property/<property_id>`

7.5 Service: /devices

The `/devices` services are used to create, update and delete devices. Moreover, the device types and its data fields can be configured. The following services are defined:

- `/devices`: Returns a list of devices or creates a new device.
- `/devices/device/<id>`: Get information about a specific device, update or delete a device.
- `/devices/device/<id>/datafields`: Returns a list of data fields of a device.
- `/devices/devicetypes`: Returns a list of device types or creates a new device type.
- `/devices/devicetypes/devicetype/<id>`: Delete a device type.
- `/devices/devicetypes/devicetype/<id>/datafields`: Returns a list of the data types' data fields.

7.5.1 /devices

Returns a list of devices or creates a new device.

Parameter (GET):

`serial_number` **(string) (optional):** Returns devices with a given serial number (`deleted` is ignored)

`deleted` **(bool) (optional):** Include deleted devices (`true` or `false`, default: `false`)

Parameter (POST):

`description` **(string):** Short description of the device.

`serial_number` **(string):** Serial number of the device.

`device_type_id` **(int):** The ID of the device type.

	GET	POST	PUT	DELETE
Description	List of devices.	Create new device.	Not supported.	
Permission	ADMIN, PROBAND	ADMIN		
HTTP Status Code	200, 400, 403, 404	201, 403, 404	405	
Error Code	101, 102, 103	102, 103	100	

Permissions and Supported HTTP Methods of Service `/devices`

Example: `GET /devices`
Get the list of all devices.

Response XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<devices>
  <device>
    <deleted>false</deleted>
    <description>iPhone 4 Black</description>
    <deviceType>
      <id>1</id>
      <name>iPhone 4</name>
    </deviceType>
    <id>1</id>
    <lastActive>2011-09-13T16:51:27.000+02:00</lastActive>
    <serialNumber>1234567890</serialNumber>
  </device>
  <device>
    <deleted>false</deleted>
    <description>iPhone 4 White</description>
    <deviceType>
      <id>1</id>
      <name>iPhone 4</name>
    </deviceType>
    <id>2</id>
    <serialNumber>09876543210</serialNumber>
  </device>
  <device>
    <deleted>false</deleted>
    <description>Insole</description>
    <deviceType>
      <id>2</id>
      <name>Insole</name>
    </deviceType>
    <id>3</id>
    <serialNumber>001</serialNumber>
  </device>
</devices>
```

Response JSON

```
[
  {
    "id": 1,
    "serialNumber": "1234567890",
    "description": "iPhone 4 Black",
    "deleted": false,
    "lastActive": "2011-09-13T16:51:27.000+02:00",
    "deviceType": {
      "id": 1,
      "name": "iPhone 4"
    }
  },
  {
    "id": 2,
    "serialNumber": "0987654321",
    "description": "iPhone 4 White",
    "deleted": false,
    "lastActive": null,
    "deviceType": {
      "id": 1,
      "name": "iPhone 4"
    }
  },
  {
    "id": 3,
    "serialNumber": "001",
    "description": "Insole",
    "deleted": false,
    "lastActive": null,
    "deviceType": {
      "id": 2,
      "name": "Insole"
    }
  }
]
```

Example: `POST /devices/?description=iPhone 4 Black 1&serial_number=789 &device_type_id=1`
Create a new device.

Response XML

```
<device>
  <deleted>false</deleted>
  <description>iPhone 4 Black 1</description>
  <deviceType>
    <id>1</id>
    <name>iPhone 4</name>
  </deviceType>
  <id>101</id>
  <serialNumber>789</serialNumber>
</device>
```

Response JSON

```
{
  "id": 101,
  "serialNumber": "789",
  "description": "iPhone 4 Black 1",
  "deleted": false,
  "lastActive": null,
  "deviceType": {
    "id": 1,
    "name": "iPhone 4"
  }
}
```

7.5.2 /devices/device/<id>

Get information about a specific device, update or delete a device.

Parameter:

<id> (int): The ID of the device.

description (string) (optional): Short description of the device.

serial_number (string) (optional): Serial number of the device.

device_type_id (int) (optional): The ID of the device type.

	GET	POST	PUT	DELETE
Description	Detailed device information.	Not supported.	Update device.	Mark device as deleted.
Permission	ADMIN, PROBAND		ADMIN	ADMIN
HTTP Status Code	200, 403, 404	405	200, 400, 403, 404	200, 400, 403, 404
Error Code	102, 103	100	101, 102, 103	101, 102, 103

Permissions and Supported HTTP Methods of Service /devices/device/<id>

Example: GET /devices/device/101
Get details of the device with ID = 101.

Response XML

```
<device>
  <deleted>false</deleted>
  <description>iPhone 4 Black 1</description>
  <deviceType>
    <id>1</id>
    <name>iPhone 4</name>
  </deviceType>
  <id>101</id>
  <serialNumber>789</serialNumber>
</device>
```

Response JSON

```
{
  "id": 101,
  "serialNumber": "789",
  "description": "iPhone 4 Black 1",
  "deleted": false,
  "lastActive": null,
  "deviceType": {
    "id": 1,
    "name": "iPhone 4"
  }
}
```

7.5.3 /devices/device/<id>/datafields

Returns a list of data fields of a device.

	GET	POST	PUT	DELETE
Description	List of device data fields.	Not supported.		
Permission	ADMIN, PROBAND			
HTTP Status Code	200, 403, 404	405		
Error Code	102, 103	100		

Permissions and Supported HTTP Methods of Service /devices/device/<id>/datafields

Example: `GET /devices/device/101/datafields`
Get the data fields of the device with `ID = 101`.

Response XML

```
<dataFields>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>accX</description>
    <id>2</id>
  </dataField>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>accY</description>
    <id>3</id>
  </dataField>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>accZ</description>
    <id>4</id>
  </dataField>
</dataFields>
```

Response JSON

```
[
  {
    "id": 2,
    "description": "accX",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 3,
    "description": "accY",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 4,
    "description": "accZ",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  }
]
```

7.5.4 /devices/devicetypes

Returns a list of device types or creates new a device type.

Parameter (POST):

`name` **(string):** Unique name of the device type.

`data_fields` **(string):** Comma separated list of data field IDs.

	GET	POST	PUT	DELETE
Description	List of device types.	Create new device type.	Not supported.	
Permission	ADMIN	ADMIN		
HTTP Status Code	200, 403, 404	201, 400, 403, 409	405	
Error Code	102, 103	101, 102, 104	100	

Permissions and Supported HTTP Methods of Service `/devices/devicetypes`

Example: `POST /devices/devicetypes/?name=iPhone%203&data_fields=2,3,4`
Create a new device type with `name = iPhone 3` and its data fields.

Response XML

```
<deviceType>
  <id>51</id>
  <name>iPhone 3</name>
</deviceType>
```

Response JSON

```
{
  "id": 51,
  "name": "iPhone 3"
}
```

7.5.5 /devices/devicetypes/devicetype/<id>

Delete or update a device type. Data fields cannot be deleted.

Parameter (PUT):

`<id>` **(int):** ID of the device type.

`name` **(string) (optional):** New name of the device type.

`data_fields` **(string) (optional):** Comma separated list of data field IDs to be added.

	GET POST	PUT	DELETE
Description	Not supported.	Update device type.	Delete device type.
Permission		ADMIN	ADMIN
HTTP Status Code	405	200, 403, 404	204, 403, 404
Error Code	100	102, 103	102, 103

Permissions and Supported HTTP Methods of Service `/devices/devicetypes/devicetype/<id>`

7.5.6 /devices/devicetypes/devicetype/<id>/datafields

Returns a list of the data types' data fields.

Parameter:

`<id>` **(int):** ID of the device type.

	GET	POST PUT DELETE
Description	List of data fields.	Not supported.
Permission	ADMIN	
HTTP Status Code	200, 403, 404	405
Error Code	102, 103	100

Permissions and Supported HTTP Methods of Service `/devices/devicetypes/devicetype/<id>/datafields`

Example: `GET /devices/devicetypes/devicetype/51/datafields`
Get the data fields of the device type with `ID = 51`.

Response XML

```
<dataFields>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>accX</description>
    <id>2</id>
  </dataField>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>accY</description>
    <id>3</id>
  </dataField>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>accZ</description>
    <id>4</id>
  </dataField>
</dataFields>
```

Response JSON

```
[
  {
    "id": 2,
    "description": "accX",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 3,
    "description": "accY",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 4,
    "description": "accZ",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  }
]
```

7.6 Service: `/datafields`

The `/datafields` service is used to define data fields for device types and computed data. The following services are provided:

- `/datafields`: Returns a list of available data fields or creates a new data field.
- `/datafields/datafield/<id>`: Get information about a specific data field, update or delete a data field.
- `/datafields/datatypes`: Returns a list of available data types.

7.6.1 /datafields

Returns a list of available data fields or creates a new data field.

Parameter (POST):

`description` **(string):** Short description of the data field.

`data_type_id` **(int):** The ID of the data type.

	GET	POST	PUT	DELETE
Description	List of data fields.	Create new data field.	Not supported.	
Permission	ADMIN	ADMIN		
HTTP Status Code	200, 403, 404	201, 400, 403, 409	405	
Error Code	102, 103	101, 102, 104	100	

Permissions and Supported HTTP Methods of Service /datafields

Example: GET /datafields
Get list of all data fields.

Response XML

```
<dataFields>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>FSR_heel_right</description>
    <id>28</id>
  </dataField>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>FSR_meta_one_right</description>
    <id>29</id>
  </dataField>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>FSR_meta_five_right</description>
    <id>30</id>
  </dataField>
  <dataField>
    <dataType>
      <id>2</id>
      <name>Double</name>
    </dataType>
    <description>FSR _ toe_right</description>
    <id>31</id>
  </dataField>
  <dataField>
    <dataType>
      <id>4</id>
      <name>Text</name>
    </dataType>
    <description>Text</description>
    <id>32</id>
  </dataField>
</dataFields>
```

Response JSON

```
[
  {
    "id": 28,
    "description": "FSR_heel_right",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 29,
    "description": "FSR_meta_one_right",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 30,
    "description": "FSR_meta_five_right",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 31,
    "description": "FSR _ toe_right",
    "dataType": {
      "name": "Double",
      "id": 2
    }
  },
  {
    "id": 32,
    "description": "Text",
    "dataType": {
      "name": "Text",
      "id": 4
    }
  }
]
```

Example: POST /datafields/?description=Gyroscope X&data_type_id=1
Create a new data type with ***name = Gyroscope X*** *of data type "**Integer**".*

Response XML

```
<dataField>
  <dataType>
    <id>1</id>
    <name>Integer</name>
  </dataType>
  <description>Gyroscope X</description>
  <id>51</id>
</dataField>
```

Response JSON

```
{
  "id": 51,
  "description": "Gyroscope X",
  "dataType": {
    "name": "Integer",
    "id": 1
  }
}
```

7.6.2 /datafields/datafield/<id>

Get information about specific a data field, update or delete a device data field. Only the description of the data type can be updated. Changing the data type would cause that all saved (associated) data becomes invalid.

Parameter:

`<id>` **(int):** The ID of the data field.

`description` **(string) (optional):** Short description of the data field.

	GET	POST	PUT	DELETE
Description	Details about data field.	Not supported.	Update data type.	Delete data type.
Permission	ADMIN		ADMIN	ADMIN
HTTP Status Code	200, 400, 403, 404	405	200, 400, 403, 404, 409	204, 403, 404
Error Code	101, 102, 103	100	101, 102, 103, 104	102, 103

Permissions and Supported HTTP Methods of Service `/datafields/datafield/<id>`

7.6.3 /datafields/datatypes

Returns a list of available data types. Data types cannot be modified through API. The following data types are defined:

- *Boolean*: `true` or `false`
- *Text*: Text of variable length
- *Double*: Double-precision 64-bit IEEE 754 floating point[6]
- *Integer*: 32-bit signed two's complement integer[6]

	GET	POST | PUT | DELETE
Description	List of data types.	Not supported.
Permission	ADMIN	
HTTP Status Code	200, 403, 404	405
Error Code	102, 103	100

Permissions and Supported HTTP Methods of Service `/datafields/datatypes`

[6] `http://download.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html`

Example: `GET /datafields/datatypes`
Get the list of all data types.

Response XML

```
<dataTypes>
  <dataType>
    <id>3</id>
    <name>Boolean</name>
  </dataType>
  <dataType>
    <id>2</id>
    <name>Double</name>
  </dataType>
  <dataType>
    <id>1</id>
    <name>Integer</name>
  </dataType>
  <dataType>
    <id>4</id>
    <name>Text</name>
  </dataType>
</dataTypes>
```

Response JSON

```
[
  {
    "name": "Boolean",
    "id": 3
  },
  {
    "name": "Double",
    "id": 2
  },
  {
    "name": "Integer",
    "id": 1
  },
  {
    "name": "Text",
    "id": 4
  }
]
```

7.7 Service: /algorithms

The `/algorithms` service deals with the defined algorithms used by the computed data to indicate which algorithm was used to compute the data.

Overview of services:

- `/algorithms`: Returns a list of available algorithm or creates a new algorithm.
- `/algorithms/algorithm/<id>`: Get information about specific algorithm, update or delete algorithm.

7.7.1 /algorithms

Returns a list of available algorithm or creates a new algorithm.

Parameter (POST):

`name` **(string):** Unique name of the algorithm.

	GET	POST	PUT	DELETE
Description	Get list of algorithm.	Create new algorithm.	Not supported.	
Permission	ADMIN	ADMIN		
HTTP Status Code	200, 403, 404, 409	201, 400, 403, 409	405	
Error Code	102, 103, 104	101, 102, 104	100	

Permissions and Supported HTTP Methods of Service `/algorithms`

Example: `GET /algorithms`
Get the list of all algorithms.

Response XML

```
<algorithms>
  <algorithm>
    <id>2</id>
    <name>Low Pass</name>
  </algorithm>
  <algorithm>
    <id>1</id>
    <name>Simple Fall Detection</name>
  </algorithm>
</algorithms>
```

Response JSON

```
[
  {
    "name": "Low Pass",
    "id": 2
  },
  {
    "name": "Simple Fall Detection",
    "id": 1
  }
]
```

Example: `POST /algorithms?name=Band Bass`
Create a new algorithm with "`name = Band Pass`"

Response XML

```
<algorithm>
  <id>51</id>
  <name>Band Pass</name>
</algorithm>
```

Response JSON

```
{
    "name": "Band Bass",
    "id": 51
}
```

7.7.2 /algorithms/algorithm/<id>

Get information about a specific algorithm, update or delete an algorithm.

Parameter:

`<id>` (int): ID of the algorithm.

	GET	POST	PUT	DELETE
Description	Get details of algorithm.	Not supported.	Update algorithm.	Delete algorithm.
Permission	ADMIN		ADMIN	ADMIN
HTTP Status Code	200, 403, 404	405	200, 400, 403, 404	204, 403, 404
Error Code	102, 103	100	101, 102, 103, 104	102, 103

Permissions and Supported HTTP Methods of Service `/algorithms/algorithm/<id>`

7.8 Authentication

Authentication is the process of identifying and verifying the identity of a user.

The Web service uses the *"HTTP Basic Authentication"* according to RFC2617[7]. [Franks et al., 1999] specifies RFC261 as a simple authentication scheme in which password are passed for each realm. HTTP Basic Authentication sends the username and password

[7] `http://www.rfc-editor.org/rfc/rfc2617.txt`

within the request header "`WWW-Authenticate`" as Base64-encoded string [Franks et al., 1999].

The Web service uses Apache Tomcat's JDBC Realm instead of the commonly used UserDatabaseRealm. The JDBC Realm uses a JDBC connection to the database which provides the users and their roles for authentication. [Chopra et al., 2007] defines the configuration in Listing A.8 which is added to Apache Tomcat's `server.xml` inside the `Host` node to enable the JDBC Realm. [Chopra et al., 2007] describes the options as follows:

- `connectionURL`: Points to the database that contains the authentication details.
- `connectionName`/`connectionPassword`: Credentials used to access the database.
- `userTable`/`userRoleTable`: Specifies which tables should be used to lookup the user and role.
- `digest`: The algorithm that Tomcat uses to digest the password entered by the user.

According to [Chopra et al., 2007] the JDBC Realm expects the tables depicted in Figure 7.2. Since the table *"user_roles"* does not exist in the expected definition, a virtual table (called *"view"*) is created to provide the user ↔ role relation (see Listing A.7) [Asseg, 2011].

Figure 7.2: Database Tables used by Apache Tomcat for Authentication

7.9 Authorization

In contrast to authentication, authorization deals with permissions: *"Is the user allowed to access the resource he/she requests?"* The Web service has three pre-defined roles a user can belong to:

- `ADMIN`: Has permission to access all resources as well as permission to access the backend.
- `RESEARCHER`: Person who usually observes a test; has only permissions to access several resources and no permission to access the backend.
- `PROBAND`: Person who is to be tested; has only permission to access several resources and no permission to access the backend.

Those roles are defined through the `<security-role>` element in the `web.xml` file of the Tomcat server application. The `<security-constraint>` elements specifies which roles (`<auth-constraint>`) and which HTTP methods (`<http-method>`) are allowed to access a specific resource (`<url-pattern>`) [Burke, 2010]. The complete authorization setup of the Web service is shown in Listing A.6.

At least the `RolesAllowedResourceFilterFactory` is enabled by adding Listing A.9 to the applications `web.xml`. This will enable `javax.security` annotations, which allows to specify access roles within method definitions. Listing A.10 shows the basic usage of the following annotations:

- `@PermitAll`: Specifies that all security roles are allowed to invoke the specified method(s) [Mordani, 2009].
- `@RolesAllowed`: Specifies the security roles permitted to access method(s) in an application. The value element of the RolesAllowed annotation is a list of security role names [Mordani, 2009].

7.10 Error Codes

The error codes offer additional information besides the appropriate 3xx, 4xx or 5xx HTTP status codes in the response header. The response depends on the requested format which could be either XML or JSON (see Listing 7.5 and Listing 7.6 for an example response). The following error codes and messages are available:

100: Method not allowed. (Typically goes with HTTP status 405.)

101: Bad Request. (Typically goes with HTTP status 400.)

102: Forbidden. (Typically goes with HTTP status 403.)

103: Resource not found. (Typically goes with HTTP status 404.)

104: Conflict. (Typically goes with HTTP status 409.)

105: Unable to start test. Test already started or finished.

106: Unable to finish test. Test not started.

107: Internal server error. (Typically goes with HTTP status 500.)

108: Unknown.

109: Unable to cancel test. Test is finished.

110: Not Acceptable. (Typically goes with HTTP status 406.)

```
<error>
  <id>104</id>
  <message>Conflict</message>
  <detailedMessage>user with username 'new.user'
      already exists</detailedMessage>
</error>
```

Listing 7.5: XML Response of Error Code 104

```
{
  "id": 104,
  "message": "Conflict",
  "detailedMessage": "user with username 'new.user
      ' already exists"
}
```

Listing 7.6: JSON Response of Error Code 104

7.11 HTTP Status Codes

[Fielding et al., 1999] defines a complete list of HTTP status code which are part of each HTTP response. HTTP status codes are divided into several "classes" where status codes of class 2xx ("Successful"), 4xx ("Client Error") and 5xx ("Server Error") are of greatest importance:

200 OK: *"The request has succeeded. The information returned with the response is dependent on the method used in the request."* [Fielding et al., 1999]

201 Created: *"The request has been fulfilled and resulted in a new resource being created."* [Fielding et al., 1999]

204 No Content: *"The server has fulfilled the request but does not need to return an entity-body."* [Fielding et al., 1999] (Mostly used when deleting a resource.)

400 Bad Request: *"The request could not be understood by the server due to malformed syntax."* [Fielding et al., 1999]

403 Forbidden: *"The server understood the request, but is refusing to fulfill it."* [Fielding et al., 1999]

404 Not Found: *"The server has not found anything matching the Request-URI. No indication is given of whether the condition is temporary or permanent."* [Fielding et al., 1999]

405 Method Not Allowed: *"The method specified in the Request-Line is not allowed for the resource identified by the Request-URI."* [Fielding et al., 1999]

406 Not Acceptable: *"The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request."* [Fielding et al., 1999]

409 Conflict: *"The request could not be completed due to a conflict with the current state of the resource."* [Fielding et al., 1999]

500 Internal Server Error: *"The server encountered an unexpected condition which prevented it from fulfilling the request."* [Fielding et al., 1999]

7.12 Summary

The API has been implemented as RESTful Web service using the Java API for RESTful Web Services (JAX-RS) specification. *Jersey* was chosen as the implementation framework since it is the reference implementation of the specification. Jersey comes with a set of annotations for defining the basic HTTP methods and Uniform Resource Locators (URLs). Furthermore, all required services haven been specified in detail defining the URL, authorization, response as well as error response codes.

Chapter 8

Backend

As described in Section 5.3.3, the backend is built as Rich Internet Application (RIA) using JavaScript. Today, the list of JavaScript libraries seems to be endless, which makes it hard to decide on the correct framework. The final decision to find the most appropriate framework is based on the application requirements.

While looking at various toolkits such as jQuery[1], Google Web Toolkit[2], ExtJS[3], Prototype[4] and YUI[5], it turned out that DoJo Toolkit[6] best fits the requirements: Dojo provides a wide range of layout possibilities and widget support for implementing the backend.

Thus, this section deals with the architecture of Dojo, the modules used by the backend and the interaction with the API. Appendix B provides a short guide for the backend.

As depicted in Figure 8.1, the backend is stored as static content on the Apache Tomcat application server. The client (browser) requests the backend from the server. All further Web service requests are performed by the backend running in the client's Web browser.

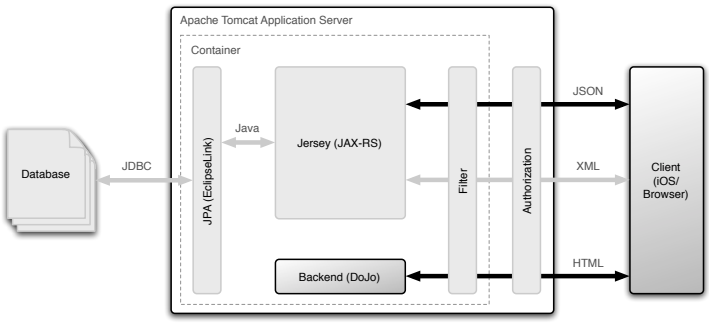


Figure 8.1: Architectural Overview of the Backend Components

[1] http://jquery.com/
[2] http://code.google.com/webtoolkit/
[3] http://www.sencha.com/products/extjs/
[4] http://www.prototypejs.org/
[5] http://developer.yahoo.com/yui/
[6] http://dojotoolkit.org/

8.1 JavaScript with the Dojo Framework

The *DoJo Toolkit* consists of five major components as depicted in Figure 8.2: *Base*, *Core*, *Dijit*, *DojoX* and *Util*. In addition to a JavaScript library, the toolkit provides feature-rich widgets such as form widgets or layout widgets [Russell, 2008]. Each component is discussed in more detail.

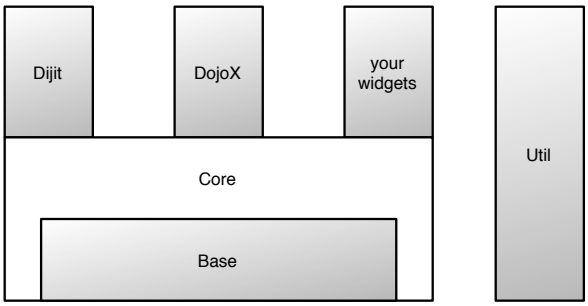


Figure 8.2: Overview of Dojo's Architecture [Russell, 2008]

8.1.1 Base

This component provides the foundations of the toolkit and can be referred to as the "kernel" of Dojo [Russell, 2008]. Besides, Base is responsible for *bootstrapping* the application, i.e, detecting the browser environment, "smoothing out" browser incompatibilities and finally loading the `dojo` namespace [Russell, 2008]. Thus, the Base provides relevant AJAX utilities for performing HTTP requests.

8.1.2 Core

Core is built upon Base and offers enhanced functionality which is not universal enough to be included in Base [Russell, 2008]. Such functionality includes widget parsing, animation or drag and drop, for example. Moreover, [Russell, 2008] states that there is no clear boundary between Core and Base.

8.1.3 Dijit

Dijit stands for *"Dojo widget"* and according to [Foundation, 2011] it provides *"a complete collection of user interface controls, giving you the power to create web applications that are highly optimized for usability, performance, internationalization, accessibility, but above all deliver an incredible user experience."*

With little effort, widgets can be created by applying the special `data-dojo-type` inside an HTML tag. Listing 8.1 shows how to "transform" a simple HTML `<div>` element into a Dojo dialog widget.

```
1 <div id="dialog" data-dojo-type="dijit.Dialog" data-dojo-props="..."></div>
```

Listing 8.1: Example usage of `data-dojo-type` Property

8.1.4 DojoX

DojoX stands for *"Dojo Extensions"* and contains widgets which do not fit into Core or Dijit [Russell, 2008]. DojoX also includes extensions which are still under development and therefore DojoX is often called *"Extensions and Experimental"*. DojoX includes for example, the grid widget.

8.1.5 Util

The Util component provides a unit-testing framework and build tools. The build tools are used for production releases of an application. The code is compressed via ShrinkSafe and furthermore the code is aggregated into layers (which is a collection of JavaScript files) [Russell, 2008].

8.2 The Backend and Dojo

Starting with a simple mockup as shown in Appendix A.5.1, the final layout can be seen in Figure 8.3. The tab-based layout is created by arranging various `dijit.layout.*` containers in a hierarchical structure. The main containers are shown in Figure 8.3, Listing A.11 shows the proper HTML markup. To summarize, the backend additionally uses the following modules:

- `dojo.data.ItemFileWriteStore`: An abstraction layer for storing data. Offers the possibility to modify the underlying data. An `ItemFileWriteStore` is connected to an `EnhancedGrid` which displays the containing data.
- `dojox.grid.EnhancedGrid`: This grid is based on the `dojox.grid` with enhanced features such as context menu.
- `dijit.form.*`: This components "transform" a simple HTML form into Dojo form widgets allowing, for example, validating the form.

8.3 API Communication

The communication with the Web service is achieved by using Dojo's XMLHttpRequest (XHR) function which performs asynchronous requests [Russell, 2008]. According to the RESTful architecture the following methods are used for the HTTP methods:

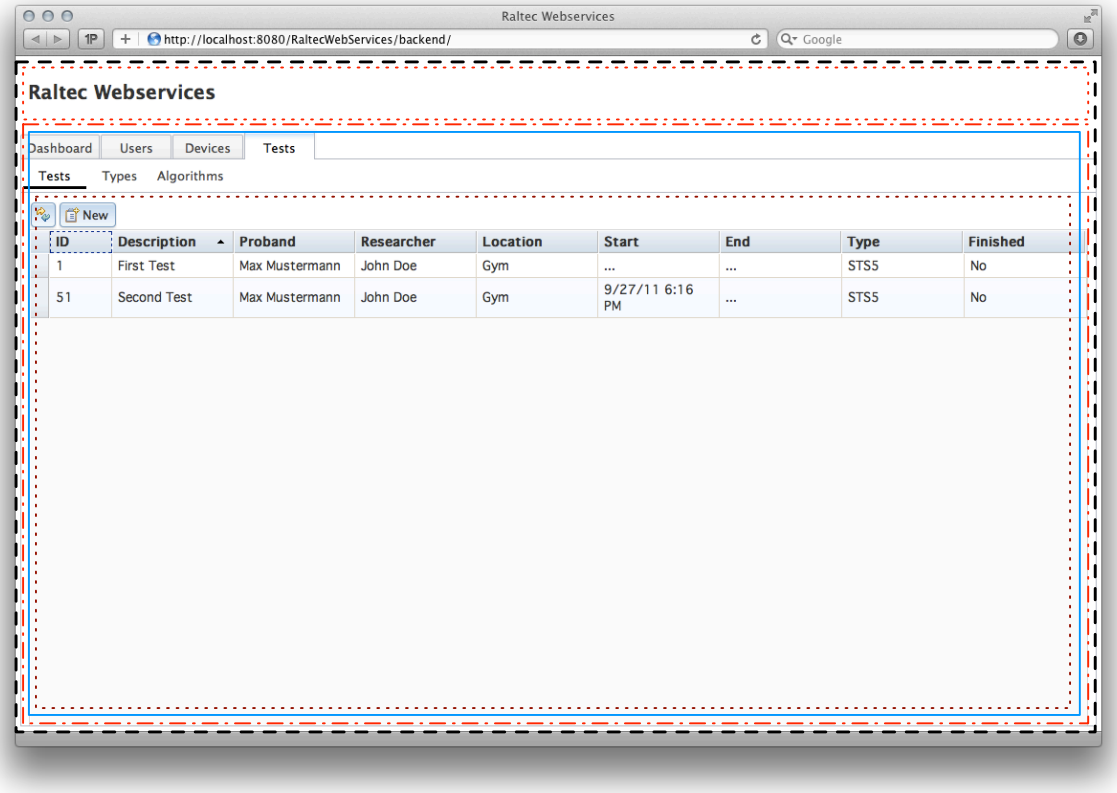


Figure 8.3: Used `dijit.layout.*` Components for Layout

- `GET`: `dojo.xhrGet()`, performs an XHR GET request.
- `POST`: `dojo.xhrPost()`, performs an XHR POST request.
- `PUT`: `dojo.xhrPut()`, performs an XHR PUT request.
- `DELETE`: `dojo.xhrDelete()`, performs an XHR DELETE request.

In more detail, a parameter object is passed to the method to configure the request:

```
var xhrArgs = {
  url: /* String: URL of the request */,
  headers: /* Object: Additional header fields */,
  handleAs: /* String: How to handle the response */,
  load: /* Function: Called funtion on success */,
  error: /* Function: Called funtion on failure */
}
```

Listing 8.2: Parameter for XHR Request

The *load callback* is only called if the request succeeds, providing the response data as argument. The implemented callback processes the received data and attaches it to

an `ItemFileWriteStore`. This store is finally bound to the corresponding `EnhancedGrid` which forces the grid to refresh and display the data. Listing A.12 illustrates an example request.

8.4 Summary

To accomplish a desktop-like backend application, the backend is built as Rich Internet Application (RIA) using JavaScript. The *Dojo Toolkit* was chosen since it best fitted the backend requirements. Dojo provides a wide range of layout containers as well as widgets. Dojo is module-based and therefore can be optimized best for production. API calls are realized through Dojo's asynchronous XMLHttpRequest (XHR) function using the API's JSON response.

Chapter 9

Mobile Client

In order to demonstrate the functionality of the framework and the recording of motion data with a mobile device, a client using the iOS platform has been implemented. The client runs specifically on the iPhone 4 since this device supplies both an accelerometer and a gyroscope for recording device motion. The client uses the Web services as defined in Section 7. Figure 9.1 illustrates the interaction of the mobile device client and the Web service.

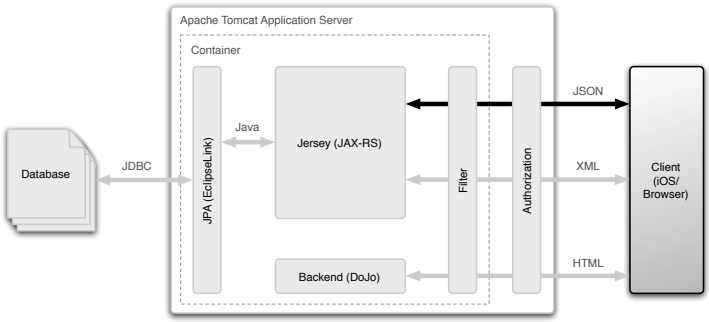


Figure 9.1: Architectural Overview of the Mobile Client Interaction

iOS applications are built according to the Model-View-Controller (MVC) pattern. This pattern divides code into functional areas where the model defines the underlying data, the view the user interface and the controller acts as bridge between the model and view [Apple, 2010b].

Moreover, the user interface and interaction are discussed together with the basic methods of gathering device data. The last section of this chapter deals with the API communication and shows which services are used.

9.1 User Interface and Interaction

The iOS client application consists of three main views. The view in Figure 9.2a shows the applications main view where the *"My Data"* section contains information about the

current user and *"My Device"* contains information about the device used. *"Reload"* invokes reloading login and device data. *"My Tests"* opens the view depicted Figure 9.2c which gives an overview of the user's tests.

Touching the magnifier button opens the *"Active Test"* view (Figure 9.2b). This view is immediately opened after the application launch and polls for running tests. To avoid unintentional touching, a "lock view" is overlayed (not shown). A running test is indicated by the green background in the first cell with the appropriate status description. Moreover, the device settings used for the current test is shown within *"Test Settings"*. *"Cancel"* aborts the current task and closes the view. If a test is running, cancel does not cancel the test on the server side. The application was designed to require as little user interaction as possible.

(a) Main View **(b)** Active Test View **(c)** User's Tests View

Figure 9.2: Screenshots of the iOS Client

9.2 Device Motion

Besides detecting motion as gestures, iOS offers the possibility to receive high-rate continuous motion data with the framework called "Core Motion" [Apple, 2011]. This framework obtains motion data from the device's sensors such as accelerometer and gyroscope. The framework consists of four parts, illustrated in Figure 9.3.

- `CMAccelerometerData`: measures device acceleration (accelerometer).
- `CMGyroData`: the device's rotation rate (gyroscope).

- `CMDeviceMotion`: encapsulates device-motion data from the accelerometer and the gyroscope.

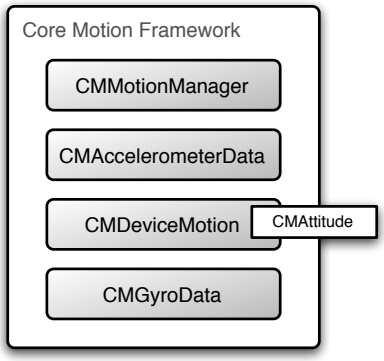


Figure 9.3: Core Motion Framework [Apple, 2011]

The `CMMotionManger` is a manager class providing the gateway to the motion services. It provides access to three motion types: raw accelerometer data, raw gyroscope data, and processed device-motion data [Apple, 2010a].

Motion data can be obtained by using either the *push* or *pull* approach. The *pull* approach means periodically requesting the most reset motion data measurement from the motion manager, whereas with *push*, an update interval and a block are defined. Core motion delivers each motion update on the defined interval to the block. The block is responsible for handling the retrieved motion data. Listing A.13 shows the basic implementation of the client's motion recording. [Apple, 2011] suggests using the *push* approach for data-collection applications and the *pull* approach for most applications

9.3 API Communication

The communication with the API is achieved by using the ASIHTTPRequest[1] library. This library is well-suited for basic HTTP requests and interacting with REST-based services [Copsey, 2011]. The response of the Web service in the JSON format is handled by the SBJson[2] library. SBJson converts JSON objects to its counterpart in Objective-C.

Web service interaction is handled by the `RTConnectionManager` which is built on the delegation pattern[3], i.e., a class which sends a message to `RTConnectionManager` implements the `RTConnectionManagerDelegate`. For each Web service call a corresponding method is implemented, constructing the request, and finally sending the request to the API.

[1] `http://allseeing-i.com/ASIHTTPRequest/`
[2] `https://github.com/stig/json-framework/`
[3] `http://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/CocoaFundamentals/CocoaFundamentals.pdf`

If the request succeeds the associated delegate method is called, telling the delegate that the request has finished. The delegate needs to implement the `requestFailed:request` method, which is called if the request fails for some reason. The client makes use of the following services:

1. `/users/user`: For authentication and user details.
2. `/devices/?serial_number=<serial>`: For loading device information. `<serial>` is replaced by the device's Unique Identifier (UDID).
3. `/tests/running/?for_device_id=<device_id>`: Periodically requesting current running tests. `<device_id>` is returned by the previous service.
4. `/status/?for_device_assignment_id=<assignment_id>`: If a running test is found, this service is periodically invoked. It returns the current status of the test. `<assignment_id>` is supplied in the response of the former service call.
5. `/tests/devicerecords/<assignment_id`: Finally, if the test has finished, the recorded device motion data is uploaded.

The interaction mention in the previous list of the iPhone application, the API and the database is illustrated in Figure 9.4.

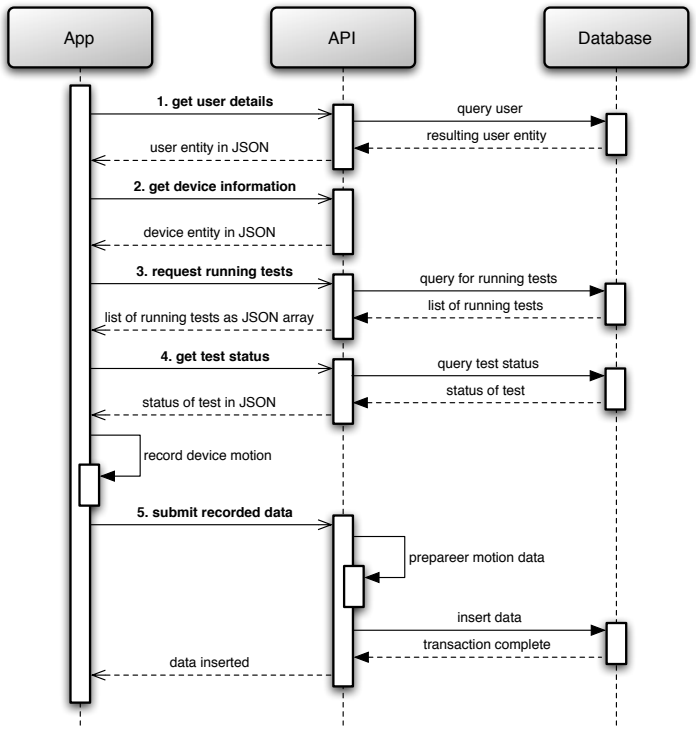


Figure 9.4: Exemplary Client ↔ API ↔ Database Interaction

The flow chart in Appendix A.6 illustrates the application life-cycle for performing a test. This design requires a permanent Internet connection. The application life-cycle

highly depends on the requirements of the client. Therefore, for example, within an offline client it may not be necessary the poll for the test status every time. The device data could be stored on the device and uploaded later on.

9.4 Summary

A mobile client using Apples iOS platform was developed for recording motion data. The application itself is very light-weight and uses the device's accelerometer and gyroscope for recording motion data. The user interface and interaction is reduced to a bare minimum and consists only of three views. Device data is recorded using the provided "motion manager" which offers two approaches to acquire data: *pull* and *push*. The application uses the *push* approach which means that at the given interval the motion data is delivered from the system whereas *pull* requires manually requesting motion data. The implemented client shows an generic application life-cycle for performing test-based motion recording. Since the API offers a maximum of flexibility, the implementation of the client highly depends on its requirements.

PART III

EVALUATION

Part III show the evaluation of the new test framework architecture discussed in Part II. The following components of the architecture are evaluated:

- database,
- Web service API,
- administrative backed, and
- mobile device client

regarding flexibility, availability, integrity and sustainability. The evaluation should ensure that the implemented framework fulfills the requirements and is open enough for adding new devices and sensors with its different types of data. The mobile device client application should prove that fall detection on mobile devices is possible.

Chapter 10

Evaluation of the Framework

The architecture described in Part II has been evaluated by running three common clinical mobility assessment tests. The *2-Minute Walk*, *Sit-to-Stand 5* and *Timed Up and Go* test were performed [Lewis and Shaw, 2005; Whitney et al., 2005; Podsiadlo and Richardson, 1991]. The test scenarios comprised creating the tests and their dependencies using the backend. For each test two mobile devices using the implemented client were used for recording the motion data during the test to demonstrate the idea. A body area network, both devices were used at the same time. During the assessment the motion data is stored locally on the device and is transmitted after the test is finished to the Web service. Appendix C shows a more detailed description of each scenario.

Moreover, this chapter lists the used settings and procedures used for each test. Finally, each assessment test is described and analyzed. To verify and analyze the recorded device data, the data has been visualized using MATLAB[1].

10.1 Test Settings

The tests have been performed in the "Senior Citizen Center Schwechat" in a well equipped room for testing (see Figure 10.1). A notebook using the Mozilla Firefox browser was used for performing backend relevant tasks such as creating users, devices and tests. The notebook as well as the two mobile phones were connected though Wi-Fi with the Internet. As mobile devices running the client application, an iPhone 4 (using accelerometer and gyroscope) and an iPhone 3 (using accelerometer only) have been connected to the Web service using the proband's user credentials. Both devices were running all the tests with a sample rate of 50Hz. The iPhone 4 was located on the right hip and the iPhone 3 on the left hip (see Figure 10.2).

[1] `http://www.mathworks.de/products/matlab/index.html`

Figure 10.1: Test Room (Gym) at "Senior Citizen Center Schwechat"

Figure 10.2: Proband Wearing the Test Device at Hip Height

10.2 Test Procedure

The test procedure for all three performed tests was as follows:

1. The researcher creates the test by assigning devices and a user in the backend.
2. The proband is fitted with the mobile devices connected to the Web service and waiting for the test to be started.
3. The researcher explains the test procedure to the proband. The proband is in start position.
4. Start of test: The proband performs test.
5. The researcher finishes test by using the backend.
6. Evaluation of the recorded device data:
   (a) Retrieving the data from the Web service and
   (b) Visualizing the data in MATLAB.

10.3 Test Scenarios and Evaluation

For evaluating the test framework, three clinical mobility assessment tests have been performed: *2-Minute Walk Test*, *Sit-to-Stand 5 Test* and *Timed Up and Go Test*. Each test was performed by the same proband using the settings in Section 10.1. Appendix C provides a detailed overview of the performed tests.

10.3.1 2-Minute Walk Test

The 2-Minute Walk Test (2MWT) is intended to measure physical endurance and function [Lewis and Shaw, 2005].

- *Starting position:* The proband is standing.
- *Procedure:* The proband walks without assistance for two minutes.

In Figure 10.3, data, recorded during the performance of a classical 2-Minute Walk test, can be seen. The test was performed on a 10 meter straight walk passage. During the test, the test-subject had to walk with "normal" (self-chosen) speed.

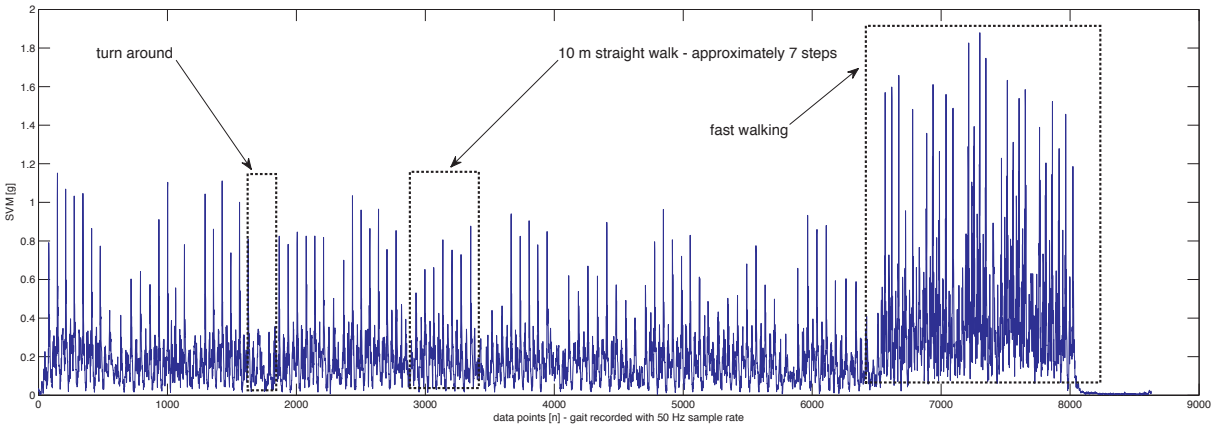


Figure 10.3: 2-Minute Walk - Fast Walk Phase

As depicted in Figure 10.4, the iPhone 4 data promises a good base for ongoing data analysis and movement feature extraction (based on simple peak detection as well as more sophisticated feature analysis, such as knowledge-based methods and statistical analysis). Single movement passages (for example 10m straight walk, turn around) are distinguishable by performing a simple Support Vector Machine (SVM) calculation and following peak detection. A calculation of peak distances offers the opportunity to extract important and fall risk describing parameters (for instance gait cycle times, variances over time) in the time domain.

After the 2 minutes of normal walking, a walking phase with an increased speed can be seen, which is characterized by a well increased value of the SVM peaks during the movement in x-direction (sagittal plane).

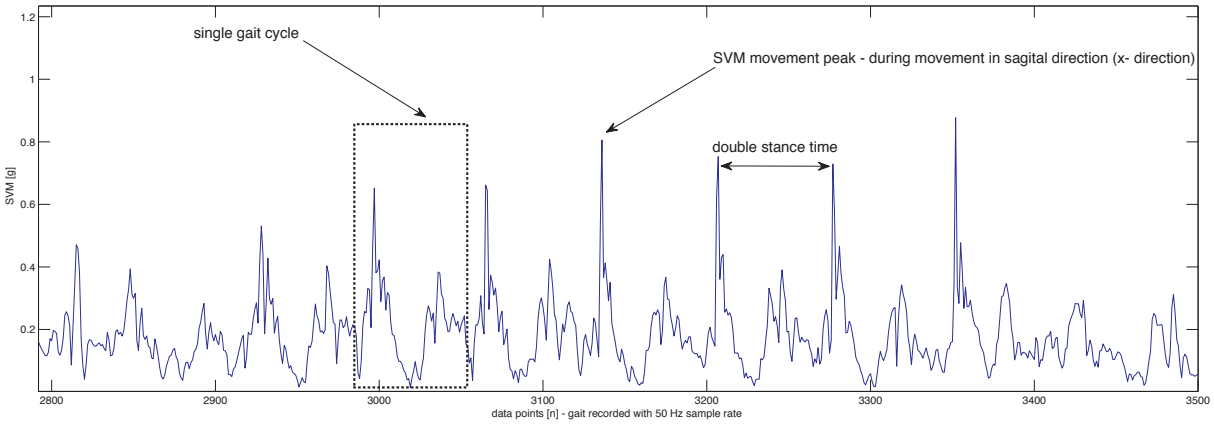


Figure 10.4: Detailed Gait Signal During 2-Minute Walk with Normal Gait Speed (Single Steps)

10.3.2 Sit-to-Stand Test

The Sit to Stand 5 (STS5) test is used to evaluate lower-extremity strength and balance [Whitney et al., 2005].

- *Starting position:* The proband is sitting on a chair.
- *Procedure:* The proband is timed while he stands up and sits down five times.

The data in Figure 10.5 shows a good base for feature extraction of classical assessment parameters like STS5 performance time as well as an expanded set of features like single movement times.

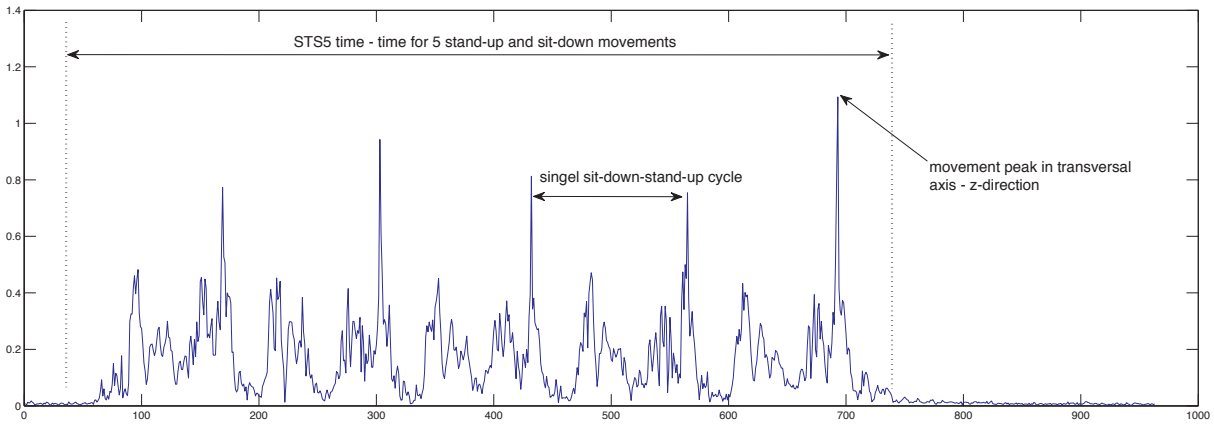


Figure 10.5: Sit-to-Stand 5 Test

10.3.3 Timed Up and Go Test

The Timed Up and Go (TUG) evaluates gait and balance of the proband [Podsiadlo and Richardson, 1991]. According to [Fuller, 2000] a proband is freely mobile if the test is

performed in less than ten seconds, and a proband is partially mobile if the test takes longer than 30 seconds.

- *Starting position:* The proband is sitting in a standard armchair.
- *Procedure:* The proband is timed while he gets up of the armchair walks three meter, turns around, walks back and sits down again [Podsiadlo and Richardson, 1991].

The evaluation data of the performance of the TUG can be seen in Figures 10.6 and 10.7. Wearing the iPhone 4 at hip height shows a promising base for feature extraction. Moreover, parameters for classical assessments like TUG performance time can be extracted as well as an expanded set of features (single movement times, gait cycle times, variances over time). Figure 10.7 illustrates the individual phases of the TUG test.

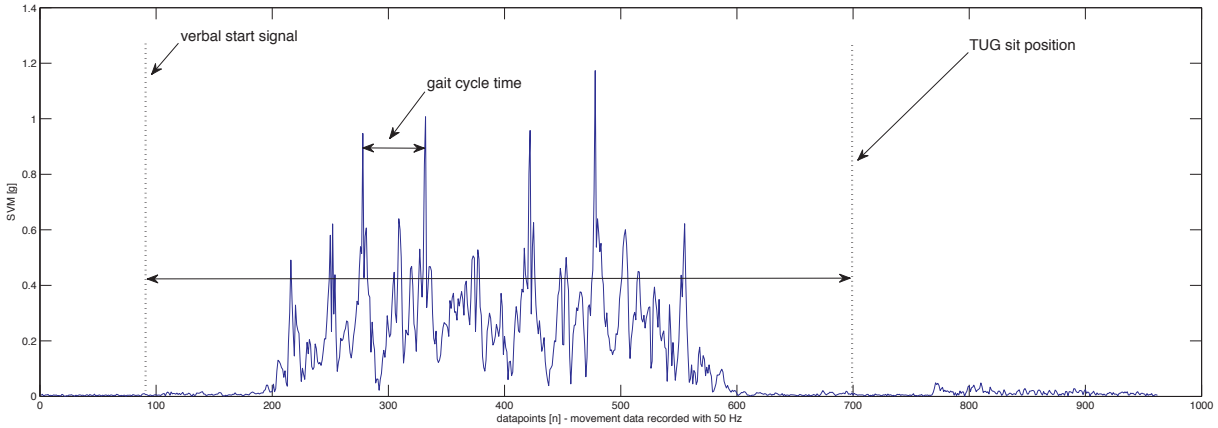


Figure 10.6: Timed Up and Go Test

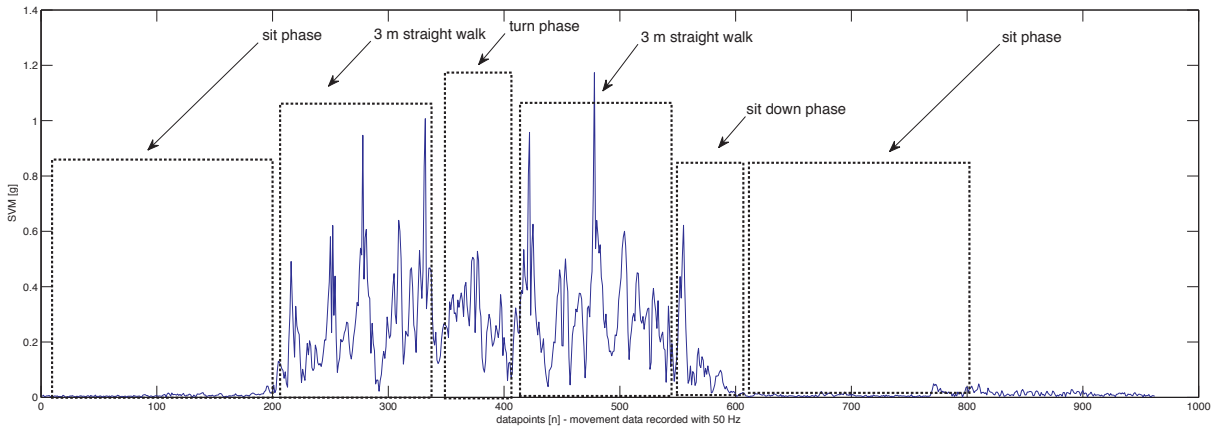


Figure 10.7: Timed Up and Go Test - Individual Phases

10.4 Conclusion

The system architecture proposed in Section 5.3 and the implementation were evaluated to ensure:

- flexibility,
- availability,
- integrity, and
- sustainability.

The backend was tested regarding usability, the Web service API regarding correct data transmission and storage and the mobile device client in terms of proper motion data recording and Web service communication. Finally the stored motion data has been evaluated.

The test procedure consisted of creating the used test device, the user and the actual test in the administrative backend. The proband was equipped with both test devices at hip height with the client application running. Subsequently the proband was instructed about the test procedure. Finally, the proband performed the assessment by starting and finishing the test using the backed.

The visualization and evaluation of the recorded iPhone 4 motion data showed that the device fulfills the requirements for movement analysis regarding the sensor data and accuracy. The recorded data promises a good base for ongoing data analysis and movement feature extraction. Furthermore, knowledge based methods and statistical analysis can be performed.

Thus, it can be said that:

1. the framework fulfills the requirements for data storage and processing and
2. the iPhone is well suited for implementing an application for movement analysis.

The evaluation also showed that the backend is easy to use and understandable and therefore all requirements have been met. The tests also proved the communication between the client and the Web service API, all data has been stored correctly. Moreover, the evaluated data shows that the saved data is returned correctly by the Web service.

Chapter 11

Summary, Conclusion & Outlook

This chapter summarizes and concludes the results gathered in this master's thesis. An outlook gives an overview of future implementation and extensibility opportunities of the proposed architecture.

11.1 Summary

This thesis dealt with the implementation of an assessment test framework for collecting fall related data using mobile devices. Thus, this theses presented relevant research in the field of Ambient Assisted Living, fall detection and mobile devices.

A currently used software for performing assessments tests has been evaluated and analyzed regarding flexibility, availability, integrity and sustainability. The evaluation showed that the framework is not flexible enough to integrate different devices. In terms of availability, data was saved locally and regarding integrity and sustainability, it was not possible to identify recorded data later on.

According to these drawbacks, a new system architecture based on the evaluation of the existing software and the theoretical background was proposed and implemented. The proposed system architecture consisted of three main parts: database, interface and client. Therefore a 3-tier architecture was chosen. The architecture is separated into the data, the application and the client tier. The data tier was used for storing related test data in a relational database. An open interface implemented by the application tier supports a variety of devices. The interface was designed as Web service according to the RESTful architectural style. The client tier covers the administrative backend built as Rich Internet Application running in the client's Web browser. The backend is used for easily managing assessment tests with their related users and devices.

A mobile device application using the iOS platform has been developed. The application used the accelerometer and gyroscope sensor of the iPhone 4 for recording motion data during the assessment test. An evaluation of the proposed and implemented architecture regarding flexibility (by adding the iPhone 4 into the framework), availability (communicating with the Web server over the internet), integrity (recorded data is saved

correct) and sustainability (access to recorded data is granted all the time) has been conducted. Moreover, the backend was tested regarding usability.

The evaluation was done by performing three clinical mobility assessment tests by using two iPhones for recording gait data. The visualization and evaluation of the recorded motion data proofed that the framework and the iOS client fulfills the requirements. Moreover, the analysis of the motion data showed that the sensor accuracy of the iPhone is accurate enough to perform movement analysis and further feature extraction.

11.2 Conclusion

Ambient Assisted Living is a wide research area and defines policies, concepts and technologies on which new support and assistance systems are built. AAL highlights the need of a common social policy for Europe in order to overcome the adaption of AAL solutions for each country.

The development of a fall detection algorithm depends on a variety of parameters such as the fall direction of the human body and the phases a fall. This parameters make it hard to develop a common fall detection algorithm for all different kinds of falls. Current approaches show how fall detection can be performed but only under certain circumstances with different drawbacks. Therefore, no general fall detection algorithm exist.

The evaluation of the implemented framework, backend and the iOS client application shows that:

- the implementation of the framework fulfills the requirements regarding flexibility, availability, integrity and sustainability;
- integrating a new device for recording motion data into the framework is easy and only requires a few steps;
- the implementation of a client application using the framework does not require expert knowledge because of the easy-to-use interface.
- mobile devices such as the iPhone 4 and their embedded hardware and software capabilities regarding sensor data accuracy are well-suited for further movement analysis and feature extraction;
- the backend is easy-to-use and performing assessment tests does not require to set-up the required hardware and software all the time;
- performing and creating assessment tests is less complex and time consuming, therefore more tests can be performed;
- assessment tests can be performed regardless of the location since the framework is available over the Internet.

11.3 Outlook

The system architecture proposed in Chapter 5.3 and its implementation offers an flexible and open interface for different devices used. As mentioned in Section 5.1, the current used test framework lacks in the meaning of availability, integrity and sustainability. To overcome these drawbacks, the base station (see Section 5.1.1.3) used by the *vitaliSHOE* project can be easily adopted and integrated into the new test framework.

Moreover, the proposed system architecture could be extended to integrate the algorithm server (see Section 5.1.1.5) as well. Consequently it would be possible to analyze newly transmitted or already existing data regarding fall detection. This would cause adopting existing services or defining new services depending on the new requirements. The basic services for storing the results is already implemented.

As depicted in Chapter 10, the iPhone 4 is well-suited for fall detection. Therefore, an independent application for fall detection could be implemented. With the current disadvantage that no general fall detection algorithm exists. A general fall detection algorithm could be developed on the foundation of the collected motion data.

Appendix A

Implementation

A.1 Example JPA Entity Annotation

```
import ...;

@Entity
public class User {

  /**
   * Primary Key
   */
  @Id
  @GeneratedValue(strategy = GenerationType.TABLE, generator="USER_GEN")
  public Long id;

  /**
   * Name of the user
   */
  public String name;

  /**
   * Relation to the roles a user belongs to
   */
  @ManyToMany
  @JoinTable(name = "USER_ROLE", joinColumns = @JoinColumn(name = "USER_ID"), inverseJoinColumns = @JoinColumn(name
       = "ROLE_ID"))
  private List<Role> roles = new ArrayList<Role>();

  /**
   * Roles the user belongs to
   *
   * @return List of roles the user belongs to
   */
  public List<Role> getRoles() {
    return roles;
  }

  /**
   * Add role to user
   *
   * @param role The new department the student belongs to
   */
  public void addDepartment(Role role) {
    if(!getRoles().contains(role))
      getRoles().add(role);
  }

}
```

Listing A.1: JPA Annotations used for Entity `User`

```
import ...;

public class User {

  public Long id;

  public String name;

  private List<Role> roles = new ArrayList<Role>();

  /**
   * Roles the user belongs to
   *
   * @return List of roles the user belongs to
   */
  public List<Role> getRoles() {
    ...
  }

  /**
   * Add role to user
   *
   * @param role The new department the student belongs to
   */
  public void addDepartment(Role role) {
    ...
  }

}
```

Listing A.2: Java Class `User`

```
import ...;

@Entity
public class Role {

  /**
   * Primary Key
   */
  @Id
  @GeneratedValue(strategy = GenerationType.TABLE, generator="DEPARTMENT_GEN")
    public Long id;

  /**
   * Name of the department
   */
  public String name;

}
```

Listing A.3: JPA Annotations used for Entity `Role`

A.2 JPA Persistence Configuration

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.
       xsd"
  version="2.0" xmlns="http://java.sun.com/xml/ns/persistence">

  <persistence-unit name="<persistence name>" transaction-type="RESOURCE_LOCAL">
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

    <class>...</class>

    <properties>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://<host>:3306/<database>?
           rewriteBatchedStatements=true" />
      <property name="javax.persistence.jdbc.user" value="<username>" />
      <property name="javax.persistence.jdbc.password" value="<password>" />

      <!-- Optimize database writes to use batching. -->
      <property name="eclipselink.jdbc.batch-writing" value="JDBC" />
      <property name="eclipselink.jdbc.cache-statements" value="true"/>
      <property name="eclipselink.jdbc.batch-writing.size" value="1000" />
      <property name="eclipselink.ddl-generation" value="none" />

      <property name="eclipselink.logging.level" value="INFO" />
    </properties>

  </persistence-unit>
</persistence>
```

Listing A.4: JPA Persistence Configuration

A.3 Example JAX-RS Service Annotations

```
import ...;

@Path("/users")
public class UsersService {

  /**
   * Returns list of all all users.
   */
  @GET
  public List<Users> getUsers() {
    ...
  }

  /**
   * Create new user.
   *
   * @param username The user name of the user
   */
  @POST
  public User createUser(@QueryParam("username") String username) {
    ...
  }

  /**
   * Return single user with given ID.
   *
   * @param userID The ID of the user
   */
  @GET
  @Path("/user/{id: [0-9]+}")
  public User getUser(@PathParam("id") int userID) {
    ...
  }

  /**
   * Delete single user with given ID.
   *
   * @param userID The ID of the user
   */
  @DELETE
  @Path("/user/{id: [0-9]+}")
  public User deleteUser(@PathParam("id") int userID) {
    ...
  }

  /**
   * Update user with given ID.
   *
   * @param username The user name of the user
   */
  @PUT
  @Path("/user/{id: [0-9]+}")
  public User updateUser(@QueryParam("username") String username) {
    ...
  }

}
```

Listing A.5: Example JAX-RS Service Annotations

A.4 Apache Tomcat Authentication and Authorization

```
<!-- API, all users allowed -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>RaltecWebservice Protected</web-resource-name>

    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/*</url-pattern>

    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>

  <!-- Anyone with one of the listed roles may access this area -->
  <auth-constraint>
    <role-name>ADMIN</role-name>
    <role-name>RESEARCHER</role-name>
    <role-name>PROBAND</role-name>
  </auth-constraint>
</security-constraint>

<!-- BACKEND, only admin allowed -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>RaltecWebservice Protected</web-resource-name>

    <!-- Define the context-relative URL(s) to be protected -->
    <url-pattern>/backend/*</url-pattern>

    <http-method>DELETE</http-method>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
    <http-method>PUT</http-method>
  </web-resource-collection>

  <!-- Anyone with one of the listed roles may access this area -->
  <auth-constraint>
    <role-name>ADMIN</role-name>
  </auth-constraint>
</security-constraint>

<!-- Login configuration -->
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>RaltecWebservice Protected Area</realm-name>
</login-config>

<!-- Security roles referenced by this web application -->
<security-role>
  <role-name>ADMIN</role-name>
</security-role>
<security-role>
  <role-name>RESEARCHER</role-name>
</security-role>
<security-role>
  <role-name>PROBAND</role-name>
</security-role>
```

Listing A.6: Apache Tomcat Configuration for Authorization

```
create view TC_RALTEC_REALM_USERS as
select USER.username as USERNAME, ROLE.name as ROLENAME from USER_ROLE left join USER on USER_ROLE.user_id = USER.id
      left join ROLE on USER_ROLE.role_id = ROLE.id WHERE USER.deleted = 0
```

Listing A.7: SQL View for Tomcat's JDBC Authentication

```
<Context docBase="RaltecWebServices" path="/RaltecWebServices" reloadable="true">
  <Realm className="org.apache.catalina.realm.JDBCRealm"
      driverName="com.mysql.jdbc.Driver"
      connectionURL="jdbc:mysql://localhost:3306/raltec_webservices"
      connectionName="<username>" connectionPassword="<password>"
      userTable="user" userNameCol="username" userCredCol="password"
      userRoleTable="tc_raltec_realm_users" roleNameCol="rolename"
      digest="MD5"/>
</Context>
```

Listing A.8: Enabling JDBCRealm via HTTP Basic Authentication in Tomcat

```
<init-param>
  <param-name>com.sun.jersey.spi.container.ResourceFilters</param-name>
  <param-value>com.sun.jersey.api.container.filter.RolesAllowedResourceFilterFactory</param-value>
</init-param>
```

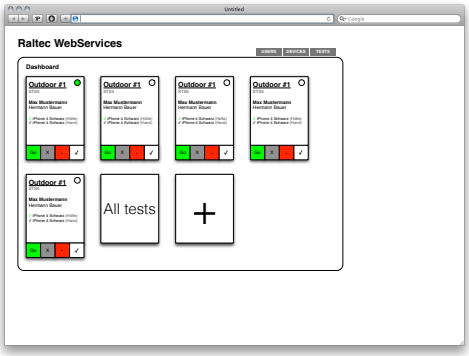
Listing A.9: Enabling `RolesAllowedResourceFilterFactory` in Apache Tomcat

```
import ...;

@Path("/users")
public class UsersService {

  /**
   * Returns list of all all users.
   *
   * All users allowed to access resource
   */
  @GET
  @PermitAll
  public List<Users> getUsers() {
    ...
  }

  /**
   * Create new user.
   *
   * Only user with role "ADMIN" allowed to access resource
   */
  @POST
  @RolesAllowed("ADMIN")
  public User createUser(...) {
    ...
  }

}
```

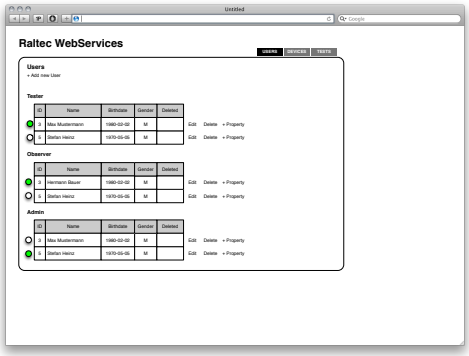
Listing A.10: Example usage of the `javax.security` Annotations

A.5 Backend

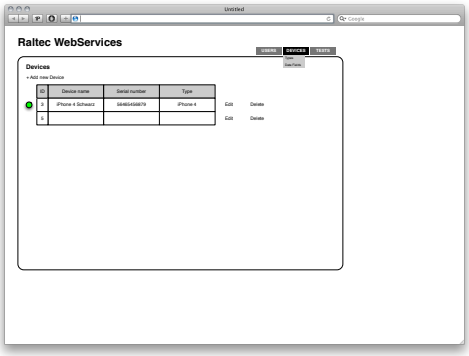
A.5.1 Mockups



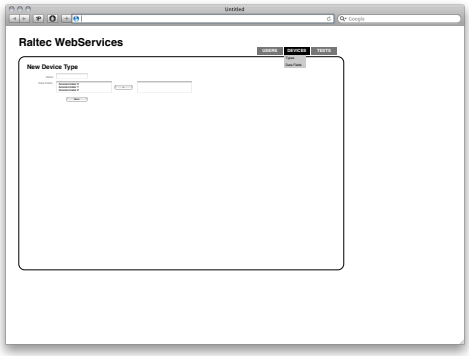
(a) Dashboard



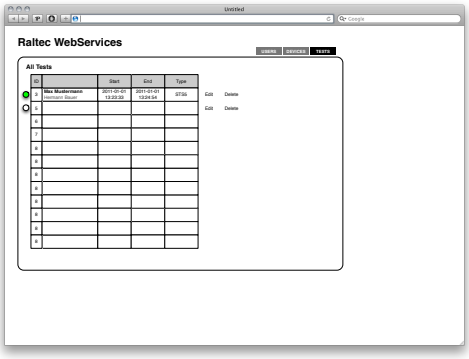
(b) User



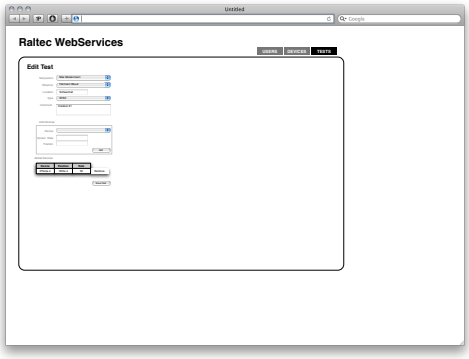
(c) Devices



(d) Create Device Type



(e) Tests



(f) Edit Test

Figure A.1: Mockups used for Backend Layout

A.5.2 Layout HTML Markup

```
<!DOCTYPE HTML>
<html lang="en">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>Raltec Webservices</title>
  <script src="js/dojo/dojo.js" data-dojo-config="isDebug: true"></script>
</head>
<body class="claro">
  <div id="wrapper" class="" data-dojo-type="dijit.layout.BorderContainer" data-dojo-props="design: 'headline'">

    <div id="header" data-dojo-type="dijit.layout.ContentPane" data-dojo-props="region: 'top'">
    </div>

    <div id="tabContainerParent" class="centerPanel" data-dojo-type="dijit.layout.TabContainer" data-dojo-props="
        region: 'center', tabPosition: 'top'">

      <div id="tabDashboard" data-dojo-type="dijit.layout.TabContainer" data-dojo-props="title: 'Dashboard', nested:
          'true'">
        <div id="tabDashboardTests" data-dojo-type="dijit.layout.ContentPane" data-dojo-props="title: 'Tests'">
        </div>
      </div>

      <div id="tabUsers" data-dojo-type="dijit.layout.TabContainer" data-dojo-props="title: 'Users', nested: 'true'"
          >
        <div id="tabUsersUsers" data-dojo-type="dijit.layout.ContentPane" data-dojo-props="title: 'Users'">
        </div>
      </div>


      <div id="tabDevices" data-dojo-type="dijit.layout.TabContainer" data-dojo-props="title: 'Devices', nested: '
          true'">
        <div id="tabDevicesDevices" data-dojo-type="dijit.layout.ContentPane" data-dojo-props="title: 'Devices'">
        </div>
      </div>

      <div id="tabTests" data-dojo-type="dijit.layout.TabContainer" data-dojo-props="title: 'Tests', nested: 'true'"
          >
        <div id="tabTestsTests" data-dojo-type="dijit.layout.ContentPane" data-dojo-props="title: 'Tests'">
        </div>
      </div>

    </div>

  </div>
</body>
</html>
```

Listing A.11: Layout HTML Markup

A.5.3 Example XHR Request

```
dojo.provide("raltec.Example");

dojo.declare("raltec.Example", [raltec.Main], {

  grid: {},

  /** Default constructor */
  constructor: function() {

    // create grid layout
    this.gridLayout = ...;

    // create a new grid
    this.grid = new dojox.grid.EnhancedGrid(...);
    this.grid.startup();

    this.load(); // load data for grid
  },

  /**
   * Load data from service
   */
  load: function() {

    var xhrArgs = {
      url: "http://www.example.com/service/x",
      headers: { "Accept": "application/json" },
      handleAs: "json",
      load: this.loadSuccess,
      error: this.loadFailed
    }

    var deferred = dojo.xhrGet(xhrArgs);
  },

  /*
   * Handle request success
   * @param receivedData Received data in JSON format
   */
  loadSuccess: function(receivedData) {

    // create store
    var store = new dojo.data.ItemFileWriteStore( { data:
      { identifier: 'id',
        items: receivedData }
    });

    // set new grid store
    this.grid.setStore(store);
  },

  /**
   * Handle request error
   * @param error Error message
   * @param ioArgs Addtional error information (http status code, ...)
   */
  loadFailed: function(error, ioArgs) {
    ...
  }
});
```

Listing A.12: Example Dojo XHR Request

A.6 Example Interaction of Client

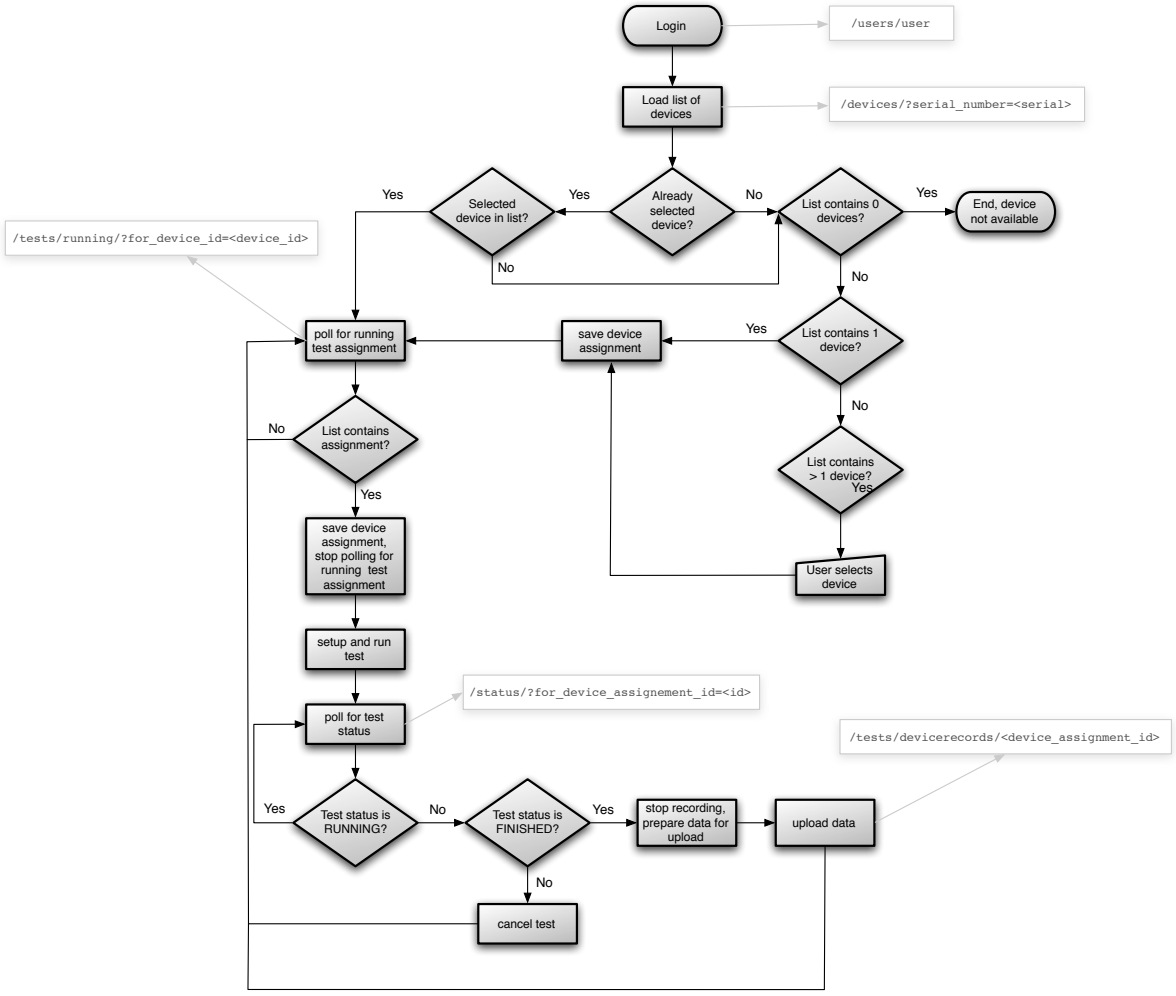


Figure A.2: Example Interaction of Client With API

A.7 iOS Device Motion Retrieval

```
- (void)startRecording
{
  int sampleRate = ...;

  // set sample rate
  motionManager.deviceMotionUpdateInterval = 1.0 / sampleRate;

  // start device motion updates
  [motionManager startDeviceMotionUpdatesToQueue:motionQueue withHandler:^(CMDeviceMotion *motion, NSError* error) {
    ...
  }];

}
```

Listing A.13: Using the Push Approach to Retrieve Device Motion Update

Appendix B

Backend Guide

The backend is an administration tool which allows researchers to easily administer assessments. In order to facilitate the workflow of creating users, devices and tests, the backend is a simple and easy-to-use web application running in every modern web browser. The backend allows the researcher to create devices with its sensors, users with additional properties and finally assessments with related devices. A dashboard provides a clear overview of current assessments which allows the researcher to start, finish or cancel an active assessment.

This guide provides a basic overview of how to create a complete test scenario. To illustrate the process, the guide is based on an Sit to Stand 5 (STS5) test use case. This generic use case consists of five steps:

- create device (Section B.1),
- create user (Section B.2),
- create test (Section B.3),
- start the test (Section B.4), and
- using the iPhone client for performing the assessment (Section B.5).

The table columns can be rearranged by simply dragging the column header to a different position. Records can be edited by double-clicking the required row. Additional options are offered through a context menu (right click) on the record such as “delete”, “add devices” or “user properties”.

It should be noted that users and devices cannot be deleted, they are *marked* as deleted by checking the “deleted” checkbox in the edit dialog.

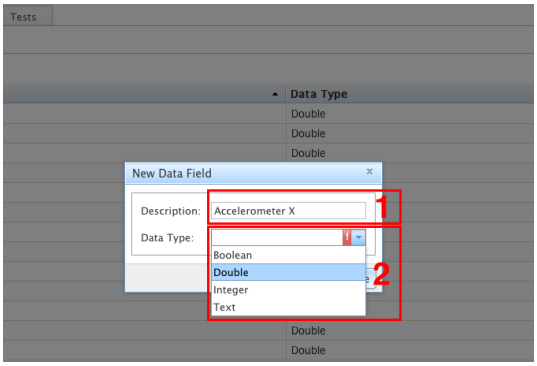
B.1 Create Device

If the desired device is already created, proceed to the next step. Before creating a device, it is important to know which sensor channels of the device should be used respectively which data fields the client uses.

According to use case: The use case illustrated in Appendix B makes use of an "Apple iPhone 4". This step involves also creating the desired data fields. Data fields are the used sensor channels of the device. For example, an accelerometer has three channels which would be "Accelerometer X", "Accelerometer Y", "Accelerometer Z".

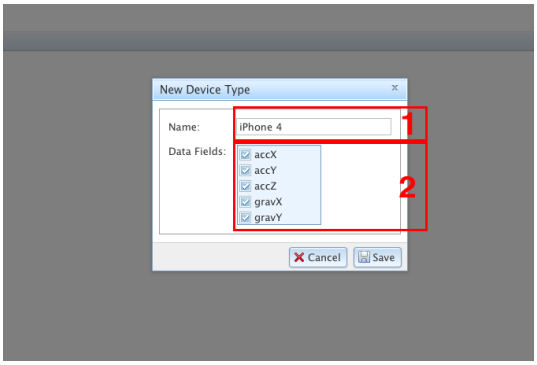
1. Create data fields for each sensor: *Devices* → *Data Fields* → *New*.
Note: Data fields can be used for several device types, so there is no need to create a data field twice.

1. Description of the data field, e.g., "Accelerometer X"
2. Desired data type of the data field, e.g., "Double"



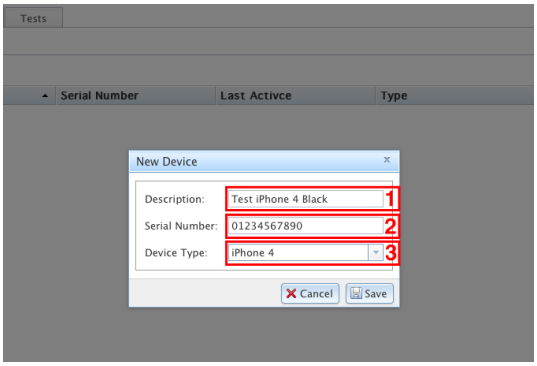
2. Create the desired device type "iPhone 4" and assign the used data fields from the list: *Device* → *Types* → *New*.
Note: The device type indicates a device family, not the actual device.

1. Name of the device type, e.g., "iPhone 4"
2. Data field assigned with the data type, e.g., "accX, accY, accZ, ..."



3. Create the actual device: *Devices* → *Devices* → *New*.
Note: This is the physical device used for the assessment.

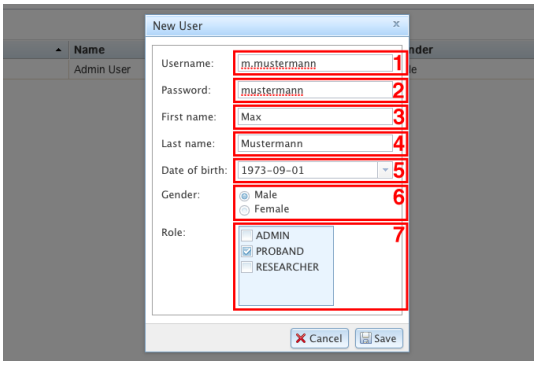
1. Description/name of the actual device, e.g., "Test iPhone 4 Black"
2. Serial number of the device, e.g., "0123456789"
3. Device type, e.g., "iPhone 4"



B.2 Create User

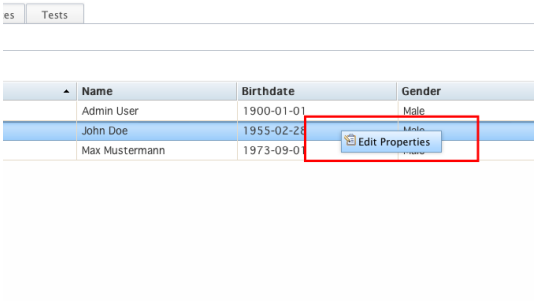
If the required users are already created, proceed to the next step. A test requires at least two users: one in the role *"PROBAND"* (the person who performs the test) and one *"RESEARCHER"* (the person who observes the test).

1. Create the two users in the desired roles. (*Users* → *Users* → *New*)



1. Unique user name (used for login), e.g., "m.mustermann"
2. The users password, e.g., "mustermann"
3. First name, e.g., "Max"
4. Last name, e.g., "Mustermann"
5. Date of birth, e.g., "1973-09-01"
6. Gender, e.g., "Male"
7. Desired roles, e.g., "PROBAND"

2. Optional: Add properties to the user by right clicking on the desired user and selecting "Edit Properties". A user can have a various number of properties. Therefore each property has an creation date assigned to identify the newest record.



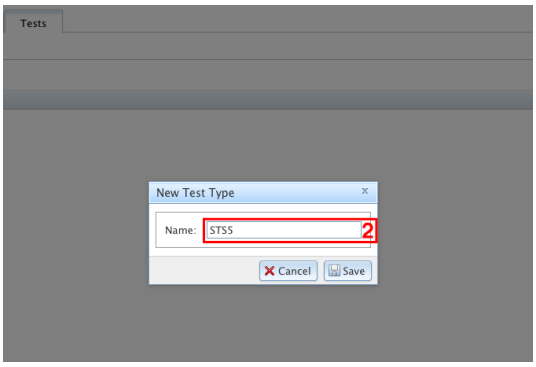
B.3 Create Test

A test is the actual relationship between users and the assigned device used within the test scenario.

According to use case: The first step involves creating the desired test type (STS5) and finally creating the test itself by assigning the previously created device, user and test type.

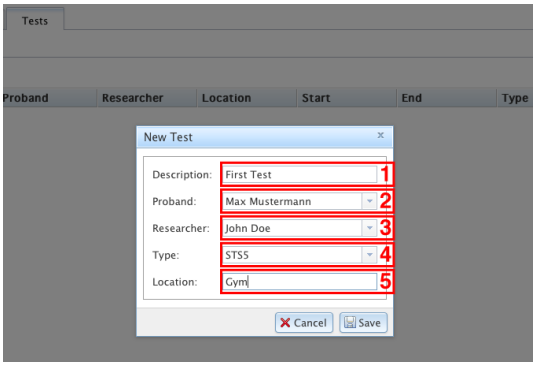
1. Create a new test type: *Tests* → *Types* → *New*.
Note: The type of a test indicates which test was performed (for example "STS5").
Optional if the desired test type is already created.

1. Name of the test type, e.g., "STS5"



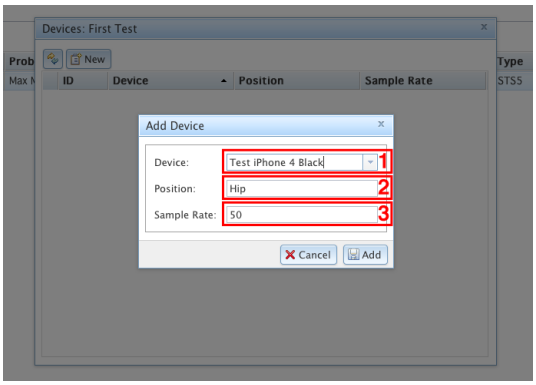
2. Create the active test to perform: *Tests* → *Tests* → *New*

1. Description of the test, e.g., "First Test"
2. Proband, e.g., "Max Mustermann"
3. Researcher, e.g., "John Doe"
4. Type, e.g., "STS5"
5. Location (where the test takes place), e.g., "Gym"



3. Add devices with their settings to the test. Devices can be added also later by right clicking on the test by selecting "Edit Devices".

1. Device to be added, e.g., "Test iPhone 4 Black"
2. Body location of the device, e.g., "Hip"
3. Used sample rate, e.g., "50"



B.4 Start/Stop Finish Test

The "Dashboard" offers an overview of all currently open or running tests. Tests can be arranged by simply dragging and dropping them. A green background color (see Figure B.1) indicates that a test is currently running. For each test, four buttons are present with the following actions (see Figure B.1):

1. *Start:* The test gets started, devices start recording.
2. *Cancel:* The test will be reset (e.g., stopped).
3. *Delete:* The test gets deleted, which means that it is completely removed from the database.
4. *Finish:* The test gets marked as finished and therefore removed from the dashboard, the device stops recording and submits the recorded data.

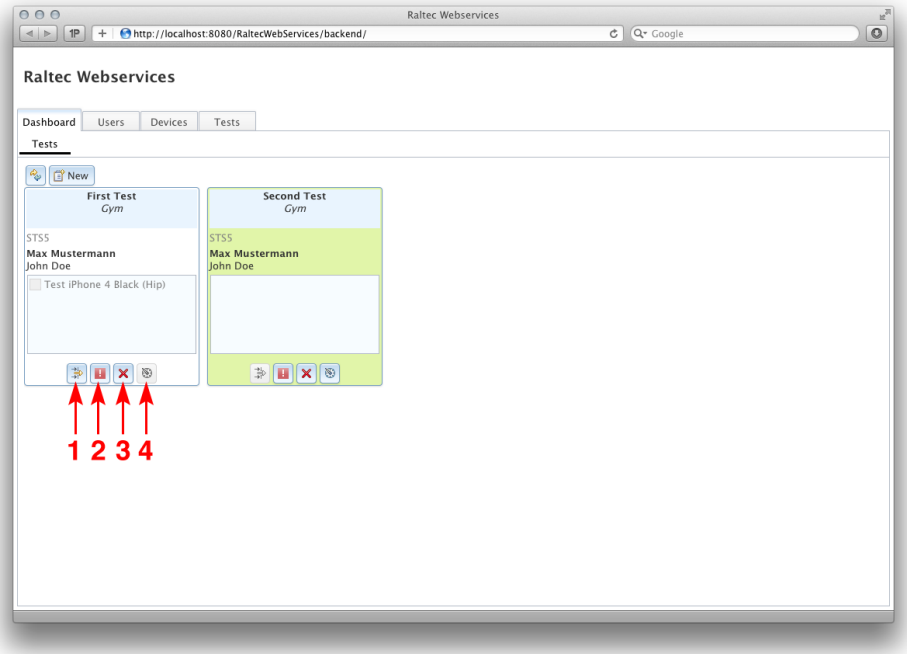


Figure B.1: Backend Dashboard

According to use case: The previously created test shows up in the dashboard and is ready to be started.

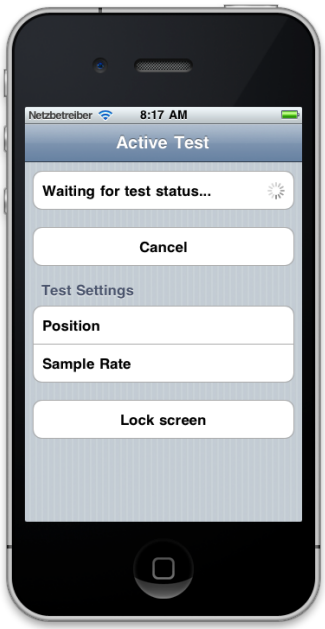
B.5 Start Client

The client is an application that is installed on the iPhone and is responsible for recording motion data by using its sensors. The client requires at least one test created in the backend as well as the actual device created. Otherwise it is not possible to run an assessment.

According to use case: The client is installed on the required device. First, the login credentials of the previously created proband user are entered and finally the application is started. If the test is already started, the client immediately begins recording device data otherwise it waits for the test to be started.

1. Enter login credentials of the proband in the "Settings" application on the device.

2. Start the client on the device. If everything is set up correctly, the proband needs no interaction with the application.



B.6 Summary

As illustrated in this appendix, creating new assessments requires three steps in the administrative backend: create a device, a user and the assessment. The mobile client is an application installed on the Apple iPhone and is responsible for recording motion data during the test. After the client is connected, the selected test can be started and stopped using the dashboard.

The backend is a Web application offering several features like column reordering through drag and drop. Moreover, tests shown in the dashboard can be reordered as well.

Appendix C

Evaluation Scenarios

#1 "Inexperienced User"	
Start:	2011-09-28 10:17
End:	2011-09-28 10:33
Proband:	Johannes Oberzaucher
Researcher:	Stefan Almer
Location:	Seniorenzentrum Schwechat, Projektraum
Description:	An inexperienced user should create a new "test scenario" within the backend. This covers creating a test, user and device and finally starting the test created.
Completion Criteria:	User clicks "start test".
Prerequisites:	Backend is opened in the browser.
Results:	Proband had no issues creating the test. Additionally two devices were added to the test. A short walk test was also performed.

Table C.1: Description of Test Scenario "Inexperienced User"

#2 "Two-Devices" (*Type: 2-minutes-walk*)	
Start:	2011-09-28 11:34
End:	2011-09-28 11:51
Proband:	Stefan Almer
Researcher:	Johannes Oberzaucher
Location:	Seniorenzentrum Schwechat, Gruppenraum
Description:	The proband is wearing two devices while he is performing the "2-minutes-walk" test. *Device 1:* iPhone 4 (position: right pants pocket, sample rate: 50Hz) *Device 2:* iPhone 3 (position: left pants pocket, sample rate: 50Hz)
Completion Criteria:	Timelimit
Prerequisites:	Test created in backend, user is equipped with both devices on start position.
Results:	Test took actually three minutes. Data recording on the phones and transmission the the API was no problem. The recorded data is OK.

Table C.2: Description of Test Scenario "Two-Devices"

#3 “Performance”	
Start:	2011-09-28
End:	2011-09-28
Proband:	Johannes Oberzaucher
Researcher:	Stefan Almer
Location:	Seniorenzentrum Schwechat
Description:	How long does it take up create a test?
Completion Criteria:	
Prerequisites:	
Results:	If all users and devices are created it takes under a minute to create a new test.

Table C.3: Description of Test Scenario “Performance”

#4 “Quick Test Series”	
End:	2011-09-28 11:34
Start:	2011-09-28 11:51
Proband:	Stefan Almer
Researcher:	Johannes Oberzaucher
Location:	Seniorenzentrum Schwechat
Description:	A series of tests is performed with two devices: 1. STS5 2. TUG *Device 1:* iPhone 4 (position: right pants pocket, sample rate: 50Hz) *Device 2:* iPhone 3 (position: left pants pocket, sample rate: 50Hz)
Completion Criteria:	All tests are done.
Prerequisites:	Tests created in backend (ready to start).
Results:	Quick evaluation was done, recorded data OK.

Table C.4: Description of Test Scenario “Quick Test Series”

Bibliography

Abbate, Stefano, Marco Avvenuti, Paolo Corsini, Alessio Vecchio, and Janet Light, 2010. *Monitoring of Human Movements for Fall Detection and Activities Recognition in Elderly Care using Wireless Sensor Network: A Survey*. InTech. ISBN 9789533073217.

Alwan, Majd, Prabhu Jude Rajendran, Steve Kelli, David Mack, Siddharth Dalali, and Matt Wolfe I, 2006. *A Smart and Passive Floor-Vibration Based Fall Detector for Elderly*. In *Proc. of the 2nd International Conference on Information & Communication Technologies: From Theory to Applications*, volume 2, pages 1003–1007. IEEE. doi:10.1109/ICTTA.2006.1684511.

AppBrain, 2011. *Most Popular Android Market Categories*. `http://www.appbrain.com/stats/android-market-app-categories`. Last accessed October 16, 2011.

Apple, 2010a. *CMMotionManager Class Reference*. `http://developer.apple.com/library/ios/#documentation/CoreMotion/Reference/CMMotionManager_Class/Reference/Reference.html`. Last accessed September 30, 2011.

Apple, 2010b. *iOS Application Programming Guide*. `http://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/iPhoneAppProgrammingGuide.pdf`. Last accessed September 30, 2011.

Apple, 2011. *Event Handling Guide for iOS*. `http://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/EventHandlingiPhoneOS.pdf`. Last accessed September 30, 2011.

Asseg, Frank, 2011. *Securing a Restful JPA WebApp With Tomcat's JDBCRealm and Jersey via HTTP Basic-Auth*. `http://objecthunter.congrace.de/tinybo/blog/articles/89`. Last accessed September 21, 2011.

Barra, Hugo, 2011. *Android: momentum, mobile and more at Google I/O*. `http://googleblog.blogspot.com/2011/05/android-momentum-mobile-and-more-at.html`. Last accessed October 12, 2011.

Berners-Lee, T., L. Masinter, and M. McCahill, 1994. *Uniform Resource Locators (URL)*. RFC 1738, Internet Engineering Task Force. `http://www.rfc-editor.org/rfc/rfc1738.txt`. Last accessed November 9, 2011.

BRAID, 2008. *Automatic Wearable Fall Detectors.* http://capsil.braidproject.eu/index.php?title=Automatic_wearable_fall_detectors&oldid=1738. Last accessed November 2, 2011.

BRAID, 2009. *User-Activated Alarms and Pendants.* http://capsil.braidproject.eu/index.php?title=User-activated_alarms_and_pendants&oldid=2899. Last accessed November 2, 2011.

BRAID, 2010. *Falls Prevention.* http://capsil.braidproject.eu/index.php?title=Falls_Prevention&oldid=6250. Last accessed November 2, 2011.

Burke, Bill, 2010. *RESTful Java with JAX-RS.* O'Reilly Media. ISBN 9780596158040, 320 pages.

Cameron, Kathleen A., Jon Pynoos, Debra J. Rose, and Judy A. Stevens, 2005. *Falls Free: Promoting a National Falls Prevention Action Plan.* Technical Report, National Council of the Aging. http://www.healthyagingprograms.org/resources/FallsFree_ReviewPaper_Final.pdf. Last accessed November 2, 2011.

Chopra, Vivek, Sing Li, and Jeff Genender, 2007. *Professional Apache Tomcat 6.* Wrox. ISBN 9780471753612, 672 pages.

Cisco, 2011. *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010–2015.* White Paper. http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.pdf. Last accessed October 12, 2011.

Columbus, Louis, 2011. *Gartner Releases Hype Cycle for Networking and Communications, 2011.* http://softwarestrategiesblog.com/2011/08/27/gartner-releases-hype-cycle-for-networking-and-communications-2011/. Last accessed October 12, 2011.

Community, Project Jersey, 2011. *Jersey: Restful Web Services Made Easy.* http://wikis.sun.com/display/Jersey/Main. Last accessed September 18, 2011.

Copsey, Ben, 2011. *What is ASIHTTPRequest?* http://allseeing-i.com/ASIHTTPRequest. Last accessed September 29, 2011.

Dai, Jiangpeng, Xiaole Bai, Zhimin Yang, Zhaohui Shen, and Dong Xuan, 2010. *PerFallD: A Pervasive Fall Detection System using Mobile Phones.* In *Proc. of the 8th IEEE International Conference on Pervasive Computing and Communications Workshops*, volume 10, pages 292–297. IEEE. ISBN 9781424466054. doi:10.1109/PERCOMW.2010.5470652.

DeMichiel, Linda, 2009. *JSR 317: Java™ Persistence API, Version 2.0.* Technical Report, Sun Microsystems, Inc. http://download.oracle.com/otndocs/jcp/persistence-2.0-fr-eval-oth-JSpec/. Last accessed September 22, 2011.

Epstein, Keith and Joseph O'Leary, 2006. *Motion Sensing with Accelerometers – Present and Future*. http://www.cs.cornell.edu/conferences/asee2006/ASEEPapers/Session4/motionsensingapplications_Epstein.pdf. Last accessed March 16, 2011.

Eurostat, 2008. *Population Projections 2008-2060*. http://europa.eu/rapid/pressReleasesAction.do?reference=STAT/08/119. Last accessed October 21, 2011.

Fielding, R., J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, 1999. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, Internet Engineering Task Force. http://www.rfc-editor.org/rfc/rfc2616.txt. Last accessed June 15, 2011.

Fielding, Roy Thomas, 2000. *Architectural Styles and the Design of Network-based Software Architectures*. PhD Thesis, University of California, Irvine. doi:10.1.1.91.2433.

Foundation, The Dojo, 2011. *Create Beautiful User Interfaces*. http://dojotoolkit.org/widgets. Last accessed September 29, 2011.

Foundation, The Eclipse, 2008. *Eclipse Announces EclipseLink Project to Deliver JPA 2.0 Reference Implementation*. http://www.eclipse.org/org/press-release/20080317_Eclipselink.php. Last accessed September 26, 2011.

Franks, J., P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, 1999. *HTTP Authentication: Basic and Digest Access Authentication*. Rfc, Internet Engineering Task Force. http://www.rfc-editor.org/rfc/rfc2617.txt. Last accessed September 21, 2011.

Fuller, Gorge F., 2000. *Falls in the Elderly. American Academy of Family Physicians*. http://www.aafp.org/afp/20000401/2159.html. Last accessed October 1, 2011.

Gartner, 2009. *Gartner Identifies the Top 10 Consumer Mobile Applications for 2012*. http://www.gartner.com/it/page.jsp?id=1230413. Last accessed October 12, 2011.

Gartner, 2011a. *Gartner Says Sales of Mobile Devices in Second Quarter of 2011 Grew 16.5 Percent Year-on-Year; Smartphone Sales Grew 74 Percent*. http://www.gartner.com/it/page.jsp?id=1764714. Last accessed October 12, 2011.

Gartner, 2011b. *Gartner Says Worldwide Mobile Application Store Revenue Forecast to Surpass $15 Billion in 2011*. http://www.gartner.com/it/page.jsp?id=1529214. Last accessed October 12, 2011.

Georgieff, Peter, 2008. *Ambient Assisted Living: Marktpotenziale IT-unterstützter Pflege für ein selbstbestimmtes Altern*. ISSN 18615066.

Gibson, M.J., R.O. Andres, B. Isaacs, T. Radebaugh, and J. Worm-Petersen, 1987. *The Prevention of Falls in Later Life. Danish Medical Bulletin*, 34(4), pages 1–24.

Guide, EclipseLink Users, 2010. *Optimizing the EclipseLink Application (ELUG)*. http://wiki.eclipse.org/index.php?title=Optimizing_the_EclipseLink_Application_%28ELUG%29&oldid=198358. Last accessed September 22, 2011.

Haas, Hugo and Allen Brown, 2004. *Web Services Glossary*. W3C note, W3C. http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/. Last accessed October 6, 2011.

Hadley, Marc and Paul Sandoz, 2009. *JAX-RS: Java™ API for RESTful Web Services*. Technical Report, Sun Microsystems, Inc. http://download.oracle.com/otndocs/jcp/jaxrs-1.1-mrel-eval-oth-JSpec/. Last accessed September 18, 2011.

Helic, Denis, 2008. *Software Architecture VO/KU*. http://coronet.iicm.tugraz.at/sa/s5/sa_styles1.html. Last accessed October 6, 2011.

Intelligence, Access, 1993. *Bellsouth, IBM Unveil Personal Communicator Phone*. http://research.microsoft.com/en-us/um/people/bibuxton/buxtoncollection/a/pdf/press%20release%201993.pdf. Last accessed October 15, 2011.

Jagos, Harald, Johannes Oberzaucher, Martin Reichel, Wolfgang L. Zagler, and Walter Hlauschek, 2010. *A Multimodal Approach for Insole Motion Measurement and Analysis*. *Procedia Engineering*, 2(2), pages 3103–3108. ISSN 18777058. doi:10.1016/j.proeng.2010.04.118.

Jähnichen, Stefan, 2008. *Ambient Assisted Living: Neues VDE-Positionspapier "Intelligente Assistenzsysteme im Dienst für eine reife Gesellschaft"*. http://www.vde.com/de/Verband/Pressecenter/Pressemappen/Documents/Hintergrundpapier%20AAL.pdf. Last accessed October 20, 2011.

Kangas, Maarit, Antti Konttila, Per Lindgren, Ilkka Winblad, and Timo Jämsä, 2008. *Comparison of Low-Complexity Fall Detection Algorithms for Body Attached Accelerometers*. *Gait & Posture*, 28(2), pages 285–291. doi:10.1016/j.gaitpost.2008.01.003.

Kangas, Maarit, Antti Konttila, Ilkka Winblad, and Timo Jämsa, 2007. *Determination of Simple Thresholds for Accelerometry-Based Parameters for Fall Detection*. In *Proc. of the 29th Annual International Conference of the Engineering in Medicine and Biology Society*, volume 2007, pages 1367–1370. doi:10.1109/IEMBS.2007.4352552.

Keith, Mike and Merrick Schincariol, 2009. *Pro JPA 2: Mastering the Java Persistence API*. Apress. ISBN 9781430219569, 500 pages.

LeMier, Mary, Ilene Silver, and Craig Bowe, 2002. *Falls Among Older Adults: Strategies for Prevention*. Technical Report, Washington State Department of Health. http://www.doh.wa.gov/hsqa/emstrauma/injury/pubs/FallsAmongOlderAdults.pdf. Last accessed November 2, 2011.

Lewis, Carole and Keiba Shaw, 2005. *Benefits of the 2-Minute Walk Test*. *Physical Therapy & Rehab Medicine*, 16(16). http://physical-therapy.advanceweb.com/Article/Benefits-of-the-2-Minute-Walk-Test.aspx. Last accessed October 1, 2011.

Li, Qiang, John A. Stankovic, Mark A. Hanson, Adam T. Barth, John Lach, and Gang Zhou, 2009. *Accurate, Fast Fall Detection Using Gyroscopes and Accelerometer-Derived Posture Information*. In *Proc. of the 6th International Workshop on Wearable and*

Implantable Body Sensor Networks, pages 138–143. IEEE. ISBN 9780769536446. doi:10.1109/BSN.2009.46.

Lindemann, U., A. Hock, M. Stuber, W. Keck, and C. Becker, 2005. *Evaluation of a Fall Detector Based on Accelerometers: A Pilot Study. Medical & Biological Engineering & Computing*, 43(5), pages 548–551. http://www.ncbi.nlm.nih.gov/pubmed/16411625. Last accessed March 13, 2011.

Lopes, Ivo C., Binod Vaidya, and Joel J. P. C. Rodrigues, 2009. *SensorFall - An Accelerometer Based Mobile Application*. In *Proc. of the 2nd International Conference on Computer Science and its Applications*, pages 1–6. IEEE. ISBN 9781424449453. doi:10.1109/CSA.2009.5404172.

Luštrek, Mitja and Boštjan Kaluža, 2009. *Fall Detection and Activity Recognition With Machine Learning. Informatica*, 33(2), pages 205–212. http://dis.ijs.si/mitjal/documents/Fall_detection_and_activity_recognition_with_machine_learning-Informatica-09.pdf. Last accessed April 4, 2011.

Malan, Ruth and Dana Bredemeyer, 2001. *Functional Requirements and Use Cases*. Technical Report, Bredemeyer Consulting. http://www.bredemeyer.com/pdf_files/functreq.pdf. Last accessed November 8, 2011.

Matthews, Mark, 2009. *A 10x Performance Increase for Batch INSERTs With MySQL Connector/J Is On The Way....* http://www.jroller.com/mmatthews/entry/speeding_up_batch_inserts_for. Last accessed September 22, 2011.

McCabe, Francis, David Booth, Christopher Ferris, David Orchard, Mike Champion, Eric Newcomer, and Hugo Haas, 2004. *Web Services Architecture*. W3C note, W3C. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. Last accessed October 6, 2011.

Meeker, Mary, 2010. *Internet Trends*. http://www.morganstanley.com/institutional/techresearch/pdfs/Internet_Trends_041210.pdf. Last accessed October 12, 2011.

Mordani, Rajiv, 2009. *Common Annotations for the Java™ Platform™*. Technical Report, Sun Microsystems, Inc. http://download.oracle.com/otndocs/jcp/common_annotations-1.1-mrel-eval-oth-JSpec/. Last accessed September 21, 2011.

Noury, N., A. Fleury, P. Rumeau, A.K. Bourke, G. Ó Laighin, V. Rialle, and J.E. Lundy, 2007. *Fall Detection - Principles and Methods*. In *Proc. of the 29th Annual International Conference of the Engineering in Medicine and Biology Society*, pages 1663–1666. IEEE. doi:10.1109/IEMBS.2007.4352627.

Oberzaucher, Johannes, Harald Jagos, Christian Zödl, Walter Hlauschek, and Wolfgang Zagler, 2010. *Using a Wearable Insole Gait Analyzing System for Automated Mobility Assessment for Older People*. In *Proc. of the 12th International Conference on Computers Helping People With Special Needs*, pages 600–603. Springer-Verlag. http://portal.acm.org/citation.cfm?id=1880751.1880852. Last accessed March 13, 2011.

Podsiadlo, D. and S. Richardson, 1991. *The Timed 'Up & Go': A Test of Basic Functional Mobility for Frail Elderly Persons. American Geriatrics Society*, 39(2), pages 142–148. ISSN 00028614. http://www.ncbi.nlm.nih.gov/pubmed/1991946. Last accessed October 1, 2011.

Preciado, J.C., M Linaje, F. Sánchez, and S. Comai, 2005. *Necessity of Methodologies to Model Rich Internet Applications*. pages 7–13. IEEE. doi:10.1109/WSE.2005.10.

Qazi, Saad, 2011. *Comparison between iPhone 4S Vs Samsung Galaxy S2 Vs Droid Bionic Vs Htc Titan*. http://mobilephones.pk/reviews/comparison-between-iphone-4s-vs-samsung-galaxy-s2-vs-droid-bionic-vs-htc-titan/. Last accessed October 13, 2011.

Retrobrick, 2011. *Nokia 9000i Communicator*. http://www.retrobrick.com/nokia9000.html. Last accessed October 15, 2011.

Russell, Matthew A., 2008. *Dojo: The Definitive Guide*. O'Reilly Media. ISBN 9780596516482.

Samsung, 2011. *Photos*. http://www.samsung.com/global/microsite/galaxys2/html/photos.html. Last accessed November 7, 2011.

Scerbakov, Nikolai, 2008. *Structured Data Management*. http://coronet.iicm.tugraz.at/wbtmaster/courses/LV706045_panel1.htm. Last accessed October 7, 2011.

Schroeder, Stan, 2011. *Apple's 500,000 Approved iOS Apps by the Numbers*. http://mashable.com/2011/05/24/app-store-500000-apps/. Last accessed October 16, 2011.

Scott, Jeff, 2011. *Apple Event Recap and Our Thoughts: iPhone 4S Annouced; iOS 5 and iCloud Release Dates*. http://www.148apps.com/news/iphone-4s-annouced-ios-5-icloud-release-dates/. Last accessed October 12, 2011.

Sixsmith, A., N. Johnson, and R. Whatmore, 2005. *Pyroelectric IR Sensor Arrays for Fall Detection in the Older Population*. In *Proc. of the Journal de Physique IV*, volume 128, pages 153–160. doi:10.1051/jp4:2005128024.

Sposaro, Frank and Gary Tyson, 2009. *iFall: An Android Application for Fall Monitoring and Response*. In *Proc. of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2009, pages 6119–6122.

Steg, Horst, Hartmut. Strese, Claudia Loroff, Jérôme. Hull, and Sophie Schmidt, 2006. *Europe is Facing a Demographic Challenge Ambient Assisted Living Offers Solutions. IST Project Report on Ambient Assisted Living*, pages 1–85. http://www.aal-europe.eu/Published/reports-etc/FinalVersion.pdf. Last accessed October 20, 2011.

Sutherland, James, 2011. *How to improve JPA performance by 1,825%*. http://java-persistence-performance.blogspot.com/2011/06/how-to-improve-jpa-performance-by-1825.html. Last accessed September 22, 2011.

Todd, C and D Skelton, 2004. *What are the main risk factors for falls among older people and what are the most effective interventions to prevent these falls?* Technical Report, WHO Regional Office for Europe. http://www.euro.who.int/document/E82552.pdf. Last accessed November 2, 2011.

Tremblay Jr., K.R. and C.E. Barber, 2005. *Preventing Falls in the Elderly.* http://www.ext.colostate.edu/pubs/consumer/10242.pdf. Last accessed November 2, 2011.

van den Broek, Ger, Filippo Cavallo, Luca Odetti, and Christian Wehrmann, 2009. *Ambient Assisted Living Roadmap.* http://www.aaliance.eu/public/documents/aaliance-roadmap/aaliance-aal-roadmap.pdf. Last accessed October 20, 2011.

Wallin, Leif-Olof, 2010. *Trends and Directions in Mobile and Wireless.* http://computersweden.idg.se/polopoly_fs/1.301097.1268145273!leifolofwallingartner.pdf. Last accessed October 13, 2011.

Whitney, Susan L., Diane M. Wrisley, Gregory F. Marchetti, Michael A. Gee, Mark S. Redfern, and Joseph M. Furman, 2005. *Clinical Measurement of Sit-To-Stand Performance in People With Balance Disorders: Validity of Data for the Five-Times-Sit-To-Stand Test. Physical Therapy*, 85(10), pages 1034–1045. ISSN 00319023. http://www.ncbi.nlm.nih.gov/pubmed/16180952. Last accessed October 1, 2011.

WHO, 2007. *WHO Global Report on Falls Prevention in Older Age.* http://www.who.int/ageing/publications/Falls_prevention7March.pdf. Last accessed November 2, 2011.

Yu, Xinguo, 2008. *Approaches and Principles of Fall Detection for Elderly and Patient.* In *Proc. of the 10th International Conference on e-Health Networking, Applications and Services*, pages 42–47. IEEE. doi:10.1109/HEALTH.2008.4600107.

Zecevic, Aleksandra A., Alan W. Salmoni, Mark Speechley, and Anthony A. Vandervoort, 2006. *Defining a Fall and Reasons for Falling: Comparisons Among the Views of Seniors, Health Care Providers, and the Research Literature. The Gerontologist*, 46(3), pages 367–376. http://www.ncbi.nlm.nih.gov/pubmed/16731875. Last accessed March 13, 2011.

Glossary

2MWT 2-Minute Walk Test.

AAL Ambient Assisted Living.

API Application Programming Interface.

CPU Central Processing Unit.

CSV Comma-separated Values.

DBMS Database Management System.

FK Foreign Key.

FSR Force Sensitive Resistors.

GPS Global Positioning System.

GZIP GNU Zip.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

ICT Information and Communication Technology.

IMU Inertial Measurement Unit.

JAX-RS Java API for RESTful Web Services.

JDBC Java DataBase Connectivity.

JPA Java Persistence API.

JSON JavaScript Object Notation.

LAN Local Area Network.

MVC Model-View-Controller.

PAN Personal Area Network.

PC Personal Computer.

PDA Personal Digital Assistant.

PK Primary Key.

POJO Plain Old Java Object.

REST Representational State Transfer.

RFID Radio-Frequency Identification.

RIA Rich Internet Application.

RPC Remote Procedure Call.

SDK Software Development Kit.

SOA Service Oriented Architecture.

SOAP Simple Object Access Protocol.

STS5 Sit to Stand 5.

SVM Support Vector Machine.

TUG Timed Up and Go.

UDID Unique Identifier.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

VPN Virtual Private Network.

XHR XMLHttpRequest.

XML Extensible Markup Language.