Monitoring Behaviour in a Domestic Environment

Addressing the Needs of an Aging Population

Marte Elisabeth Bakken Skjønsfjell
Aslak Ringvoll Normann

Master of Science in Engineering Cybernetics
Submission date: June 2012
Supervisor: Amund Skavhaug, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

It is in the interest of both society and individuals that the increasingly elderly population can live at home as long as possible.

To achieve this, the subjective feeling of security as well as the actual medical needs of the individual must be satisfied.

The candidates shall study if this can be achieved through remote sensing technology and monitoring.

The assignment consists of:

- Study relevant relevant background theory and, if possible, study existing systems.
- Propose possible solutions and implement a system.
- Evaluate the implemented system and results.

What shall be done and to which degree as well as the importance of each subtask is to be decided by the candidates.

Summary

An increasingly aging population will challenge the current organization of society and will require new technological solutions for assisting as well as maintaining the health of this demographic. In this thesis, some solutions for remote monitoring of a domestic environment are researched, implemented in the home of an elderly citizen, and evaluated.

Background theory related to wireless remote sensing, data interpretation and presentation is studied and presented. Some design concepts for handling a multitude of varied sensor information together with ways to harvest information in a health context are discussed and implemented.

A system comprised of several types of both stationary and body-worn sensors together with a framework for collecting, interpreting and presenting the gathered data in a useful manner is developed, and a pilot-study is conducted.

The results show that it is possible to get useful information via remote monitoring, and that it is possible to implement a low cost monitoring system using off-the-shelf components. Different areas of applicaton for remote monitoring are proven or made plausible.

It is shown that such a system can have significance in assisting health personnel, both as it is presented here as well as with suggested further work.

Sammendrag

Den stadige økende eldre populasjonen vil komme til å utfordre den nåværende samfunnsorganisasjonen og vil fordre nye teknologiske løsninger i forhold til assistanse og opprettholdelse av denne helsen til dennegruppen. I denne masteroppgaven er noen løsninger for fjernovervåkning av et hjemmemiljø undersøkt, implementert i hjemmet til en eldre borger og evaluert.

Bakgrunnsteori relatert til trådløs fjernovervåkning, informasjonstolkning og -presentering er studert og vist. Designprinsipper for håndtering av en mengde variert sensorinformasjon i tillegg til måter å innhøste informasjon på i en helsesammenheng er diskutert og implementert.

Et system bestående av flere typer stasjonære og kroppsbårne sensorer er utviklet sammen med et rammeverk for innsamling, tolkning og presentering av informasjon på en nyttig måte. Dette danner tilsammen basisen for et utført pilotprosjekt.

Resultatene viser at det er mulig å få en objektiv oversikt over et monitorert individ, og ikke minst at det er mulig å implementere et billig overvåkningssystem med lett tilgjengelige komponenter. Forskjellige bruksområder for fjernmonitorering er både vist og sannsynliggjort.

Det er vist at et slikt system kan ha betydning for assistanse av helsepersonell, både i den formen den er presentert i her, så vel som med foreslått videre arbeid.

Acknowledgements

The candidates would like to thank Svein Skavhaug for his cooperation and enthusiasm as a human guinea pig in our field trial, and for giving us unrestricted access in installing sensors in his house. We hope he retains his good health and humor for many years to come.

We would also like to thank Amund Skavhaug for inspiration, guidance and practial assistance in this thesis. His coffee-fund for underfunded students is also appreciated.

Norsk Automatisering must be thanked for funding parts of the system, the Department of Engineering Cybernetics for lending us equipment and Jon Petter Skagmo for assembling his MultiTRX device on short notice and giving us access to his source code.

Lastly, we would like to thank the always smiling and helpful cleaning staff for helping maintain our office a liveable place to spend most of our time for the past months.

Contents

List of Figures

List of Tables

Listings

List of Abbreviatons

AAL	Ambient Assited Living
AAL JP	Ambient Assistant Living Joint Programme
ADT	Android Development Tools
AM	Amplitude Modulation
AMP	Alternate MAC/PHY
API	Application Programming Interface
ARC	Automatic Reference Counting
ASK	Amplitude Shift Keying
AVD	Android Virtual Device
BLE	Bluetooth Low Energy
BR	Basic Rate
EDR	Enhanced Data Rate
EU	European Union
GAP	Generic Access Profile
GATT	Generic Attribute Profile
HCI	Host Controller Interface
IAR	Intertransaction Association Rules

ICT	Information and Communication Technology
IDE	Integrated Development Environment
ISM	Industrial Scientific Medical
L2CAP	Logical Link Control and Adaptation Protocol
LE	Low Energy
LL	Link Layer
MAC	Media Access Control
NDK	Native Development Kit
OOK	On-Off Keying
OQPSK	Offset Quadrature Phase Shift Keying
PHY	Physical
RF	Radio Frequency
RFID	Radio Frequency Identification
SDK	Software Development Kit
SIG	Special Interest Group
SMP	Security Manager protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
TDD	Time-Division Duplex
UI	User Interface
UWB	Ultra Wide Band
WLAN	Wireless Local Area Network
WSN	Wireless Sensor Network

Part I

Introduction

1 Introduction

1.1 Motivation

As the population ages, we are now facing a shift in demographics where more people enter their golden years, live longer, and are entitled to their due welfare benefits. With this shift, the neo-traditional institutionalization of elderly is no longer tenable on the scale we will need in a few years.

Welfare technology is an emerging field of research which therefore is becoming more and more relevant. One of the key areas of application for welfare technology is to assist elderly living at home longer, in safety and comfort, as well as assisting primary caregivers in identifying potential health issues, both developing trends and acute illness. Signs of deteriorating health can include appetite loss, insomnia and reduced activity level.

Remote activity monitoring can give health personnel indicators on the condition of the patients and give an objective overview of how their days have been. As health personnel may only come by once a day or less, it is beneficial to be able to supervise the patients' condition when they are alone.

1.2 Aim of This Thesis

There are many ailments specific to elderly, such as dementia, chronic obstructive pulmonary disease, heart conditions, Parkinson's, restricted movement, generally poor constitution and others. Each of these have their own set of parameters which may need monitoring and assessment.

In the candidates' specialization project, Normann/Skjønsfjell 2011[66], a body-worn sensor platform for monitoring a few such parameters was developed and integrated with a mobile device for display.

As mentioned in [66] and discussed further in this thesis, many types of sensors are of interest for monitoring the condition of a subject under observation, and

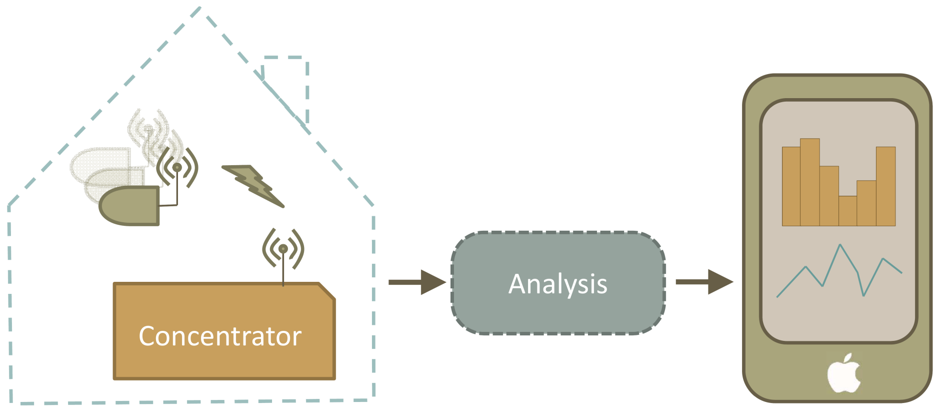


Figure 1.1: Simplified overview of the developed system. Analysis can be performed on-site to increase privacy.

a combination of heterogenous sensor data may be required to assess a subject's condition. Our aim in this thesis will therefore be to show how information from various sources may be handled and combined, develop a framework for collecting, interpreting and presenting heterogenous data and finally to realize and test such a system.

An entire monitoring system, as opposed to focusing on smaller details, has been realized to get a broad insight into the different aspects of creating a platform for welfare monitoring, as well as to be able to present a more overall picture of how a monitored environment can be used and developed further.

1.3 Wireless

Body-worn sensors must necessarily be wireless if they are to give a live feed of the wearer's condition without restricting movement or otherwise demanding undue mental effort. Stationary sensors also benefit from communicating wirelessly, the main advantages being modularity and cost savings both for installation and for maintainance throughout the lifetime of a sensor network.

Considerations with regards to wireless technologies and characteristics of wireless communication will therefore be covered in the background chapters and affect the implementation of our realized system.

1.4 Implementation of a Total System

A subject's condition may be sensed either directly via e.g. cardio-pulmonary monitors, or indirectly via the subject's activity level, mobility and interaction with the environment. The advent of "Smart Houses" as ever more mature commercial products which by their nature monitor activity in a home makes the latter indirect method a promising venue for exploiting synergies, lowering costs and risks.

Both an ambulant and a stationary platform will be implemented and discussed. The system will be implemented in a domestic environment and monitor a volunteered test subject.

Part III of this thesis details the implemented system. First, an overview will be presented, and component choices will be explained and discussed. Second, the test-environment, deployment of the stationary sensor nodes used for “Smart House” location- and activity tracking, and the development of a body-worn sensor will be described.

Then the capture of sensor data and our interpretation framework and as well as its components will be presented, followed by a description of an iPad application used for presenting the gathered data. The system presented in this report is illustrated in Figure 1.1.

1.5 Presentation

Presentation of the aggregated data is a crucial part of making the implemented system available for health personell, relatives or others without them needing to know the particularities of the system. The presentation will need to be easy to comprehend and not be cluttered with data.

To achieve this, an iPad application will be developed as a means to giving users an intuitive way of getting an objective overview of a monitored person’s activity in the domestic environment.

1.6 To the Reader

This thesis is intended as an in-depth presentation of the development of such a system, but parts of it may also be of interest to health personnel or relatives who would like an understanding of the topic of monitoring elderly.

A CD including all source code, sensor data logged during this pilot project and some background material is enclosed with this thesis.

Part II

Background

2 Welfare Technology

Welfare technology is technology assisting users in their daily life. This can contribute to an increase in life quality and help users become more independent. Examples of welfare technology include cleaning robots, smart house technology, positioning technology (GPS) as well as body-worn sensor systems and means of increasing the comfort of health care patients.

2.1 Welfare Technology in Norway

Several technologies are already put into use in the Norwegian health care system. Bærum kommune in cooperation with SINTEF and Abilia AS has a project called "Welfare technology for elderly living at home"[42]. This project has created a showroom apartment[45] with technical solutions to enable elderly to live at home as long as possible. The solutions include aids to help the user remember dates, appointments and medicines, keep in contact with family and health personnel as well as automatic lighting and alarms.

A few nursing homes use a robotic seal for social stimulation of patients with dementia[55] and "Hjelpemiddelsentralen" distributes fall sensors[1] with GPS[15].

2.2 Ambient Assisted Living

Ambient Assisted Living (AAL) is the form of welfare technology focusing only on the eldery part of the population.

2.2.1 Exisiting Projects

The European Union has a project called Ambient Assistant Living Joint Programme (AAL JP) that focuses on technological solutions for enhancing the life

[1]Fallofon from Cognita AS

quality of elderly in Europe. Focus of the project in spring 2012 is "ICT[2]-based Solutions for (Self-) Management of Daily Life Activities of Older Adults at Home"[2]

Earlier, three other focuses have resulted in funding of several projects under development:

- ICT based solutions for Prevention and Management of Chronic Conditions of Elderly People
- ICT based solutions for Advancement of Social Interaction of Elderly People
- ICT-based Solutions for Advancement of Older Persons Independence and Participation in the "Self-Serve Society"

A few of these projects are presented below, but as they are still under development, more information on their technology and methods was not found.

H@H - Health at Home

The H@H[19] project focuses on elderly with Cronic Heart Failure by remotely monitoring cardiovascular and respiratory parameters by automatic systems as well as health personnel. The aim is to reduce re-hospitalization by detecting situations that might become critical at an early stage. Total budget is €2,699,799.

HOPE - Smart Home for Elderly People

The HOPE[20] project is focusing on giving elderly with Alzheimer's disease a more independent life. This is achieved through a smart home solution with functionality for fall detection, security and communication. The budget is €2,019,199.

CARE - Safe Private Homes for Elderly Persons

CARE "aims to realise an intelligent monitoring and alarming system for independent living of elderly persons"[14]. This is to be accomplished by automatically detecting critical sitations such as a fall or immobility, by using stationary sensors. The total budget is €2,380,000.

2.3 Safety and Privacy

One of the main concerns when it comes to ICT solutions in health care is the issue of privacy and safety of patients. It is illegal for handlers of personal data to distribute or use this in a way not approved by the individual source of the data. The Norwegian Data Inspectorate (Datatilsynet), recommends that sensitive information about a user is stored as close to the user as possible, as opposed to in a central server system, to ensure privacy of the user.

The law of personal data (Personopplysningsloven[32]) states how personal data should be handled and has as an objective to protect the individual right to privacy. Information about a persons's health condition is considered to be sensitive, and

[2]Information and Communication Technology

is subject to strict regulation when it comes to harvesting and use of data. One can not gather sensitive personal data without the knowledge and approval of the affected person.

Considering these rules and regulations, it is obvious that the collecting and storing of personal data needs to be thoroughly evaluated to make sure the data is secure to unauthorized access. This include hackers as well as overly curious health personnel.

2.3.1 Electronic Patient Journals

The introduction of electronic patient journals in the Norwegian health care system has led to a public debate. Concerns regarding the protection of privacy of individuals has proven to be legimate. An example was when two NTNU students working on electronic journal software was given access to 110,000 journals from their professor[38] without the approval or knowledge of the involved patients. Another example is of health personnel having accessed electronic journals of patients out of curiosity, which have resulted in several being fired[38].

This debate and these incidents have led to a greater focus on the handling of sensitive information in the public health service, which will be more and more relevant as health services shift into a more electronic manner.

3 Wireless Technologies

Wireless technology has in recent years seen a revolution in terms of size, quality and cost of components to the point where it's a viable replacement for wired communication. A significant advantage is the ease of deployment and installation without the need for existing infrastructure or costly modifications to existing buildings.

A wireless sensor node consists primarily of three parts: The sensor hardware, data processor and communication interface, and it's the latter that will be the focus in this chapter.

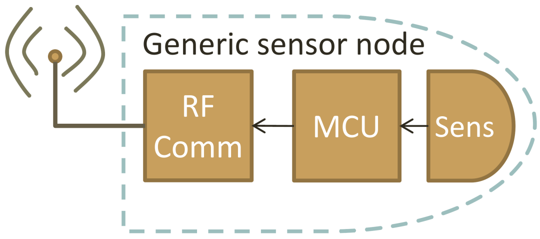


Figure 3.1: Important parts of a wireless sensor node: Communication, processing and sensing.

Wireless solutions on the market differ in many ways and this chapter will describe the most fundamental ways solutions can differ, before the next chapters will look closer at a selection of vendor-specific wireless communication solutions.

When considering different wireless sensor network technologies, it is helpful to consider them with regards to the layers in the OSI communication systems model seen in Figure 3.2 on the following page. In the following, the three media layers will be discussed in a wireless context as they are what most sets wireless communication apart from wired communication.

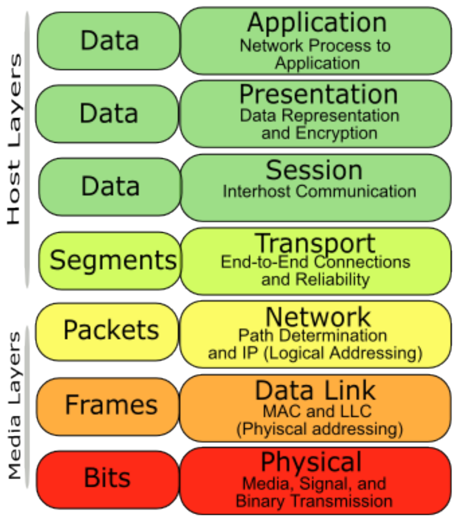

Host Layers
Media Layers
Data
Application
Network Process to Application
Data
Presentation
Data Representation and Encryption
Data
Session
Interhost Communication
Segments
Transport
End-to-End Connections and Reliability
Packets
Network
Path Determination and IP (Logical Addressing)
Frames
Data Link
MAC and LLC (Phyiscal addressing)
Bits
Physical
Media, Signal, and Binary Transmission

Figure 3.2: OSI network layer model

3.1 Physical Layer

Implementation of the physical layer is the most distinguishing factor betweeen wireless technologies. This is how a wireless unit modulates electromagnetic fields in the ether to set up corresponding oscillations in a receiving module.

Thus, there are two key factors differentiating how such signals traverse the intervening space:

- Base frequency of the oscillations
- Modulation technique

Higher signal carrier frequencies are more susceptible to attenuation when travelling through a medium, but offer higher throughput because they can be modulated at a proportionally higher frequency. Within a given wireless standard, a pre-determined list of base frequencies can sometimes be referred to as channels.

Conversely, the modulation technique can be said to determine how robust a signal transmission will be, at the cost of transmission speed and spectral[1] properties.

3.1.1 Frequency Bands

Commercial wireless sensor nodes usually operate in the Unlicensed Industrial Scientific and Medical (ISM) bands which are harmonized across the members of the International Telecommunication Union.

In Norway, this is embedded in the regulation *Forskrift om generelle tillatelser til bruk av frekvenser (fribruksforskriften)*[18]. This regulation stipulates which

[1]Said simply, how much space a signal takes

bands are free to use, the spectral properties of the signals and the allowable duty cycle of transmitters. Table 3.1 shows some frequency bands commonly used in off the shelf wireless devices. Some popular wireless sensor systems such as Insteon sold in the US use frequencies around 315 MHz, but these devices are not allowed for use in Norway as they clash with reserved NATO bands.

Band	Type[18]	Regulatory limits[18]
312-322 MHz	NATO	Not free to use
433.050-434.790 MHz	ISM	1 mW transmit effect
434.040-434.790 MHz	ISM	10 mW. 10% DS[a] if $>$ 250kHz bandwidth
868.6-869.7 MHz	Alarms	10-25 mW. $<$ 25kHz. DS 0.1, 1 or 10%.
2400.0-2483.5 MHz	ISM	10 mW

Table 3.1: Some frequency bands commonly used in off-the-shelf wireless devices.

[a]Duty Cycle: active transmission period

3.1.2 Modulation Techniques

Some modulation techniques will be covered briefly to aid later discussion. Popular modulation techniques for wireless networks include:

- ASK - Amplitude Shift Keying
- FSK - Frequency Shift Keying
- PSK - Phase Shift Keying
- DSSS - Direct Sequence Spread Spectrum
- FHSS - Frequency-hopping Spread Spectrum

Each of the three first exist in several derived and combined forms, such as Binary Phase Shift Keying, 2-FSK, QAM-4, QAM-16, BPFSK, OQPSK etc. ASK and FSK will receive som extra attention as they are relevant to the implementation chapters.

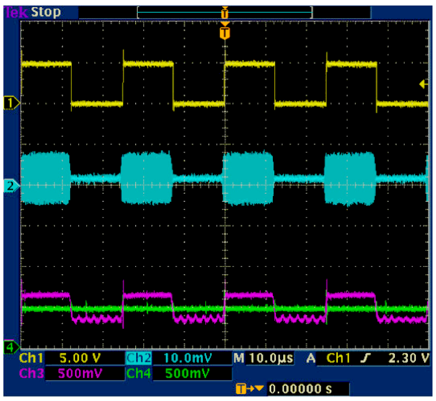
Amplitude Shift Keying

ASK works by varying the amplitude of the transmitted signal between a finite number of amplitude levels corresponding to data symbols. In the case of On/Off Keying[2] (OOK), a variation of ASK used in the implementation in this thesis, the symbol size is 1 bit and bits are represented by the presence or absence of a carrier wave.

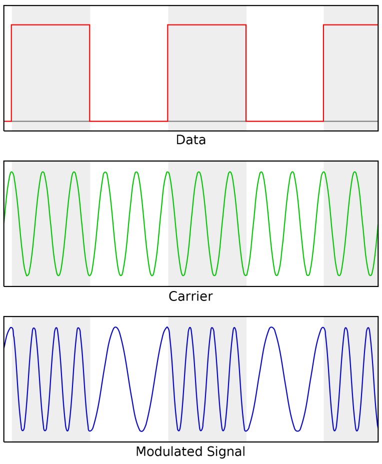
Figure 3.3a from a Maxim Application Note[34] shows how an OOK signal is received and interpreted by a typical receiver. For generic ASK, the low bit is not necessarily the absence of a carrier, but any lower amplitude. This increases noise immunity compared with OOK, at the cost of higher power consumption.[34]

[2]Keying is a remnant from CW/Morse code, meaning an actual key was pressed

OOK is relatively inexpensive hardware-wise compared with other techniques, is purely a modulation technique, and therefore not subject to any industry standard and is popular in cheap off-the-shelf radio devices operating in unlicensed frequency bands. But like analog AM, ASK is sensitive to atmospheric fading[3], and OOK in particular is sensitive to transient fading and noise.



(a) Envelope detection of OOK signal. Digital output (yellow), carrier (blue) and internal comparators (pink/green)



(b) Modulation of data onto a carrier wave using FSK. (GNU FD Licence)

Figure 3.3: Illustrations of ASK and FSK signals.

Frequency Shift Keying

FSK works by changing the frequency of the carrier instead of the amplitude. FSK is comparable with analog FM, and is more robust against noise and almost immune to fading as the decoding is independent of the received signal amplitude. However, the spectral efficiency is much lower than for ASK and hence needs 150% of the bandwidth and the energy cost is 200% that of OOK, as the transmitter must always be powered when transmitting[16].

Figure 3.3b shows how a bit-stream is encoded onto the carrier wave, resulting in a FSK signal. The modulated signal from Figure 3.3b corresponds to the blue signal in Figure 3.3a.

Phase Shift Keying

PSK transmits information by letting a number of discrete phase offsets from a common reference signal represent symbols, and the signal is then modulated by discontinuously changing the phase of the carrier. A decoder can rely either on

[3]Temporary increase in signal attenuation

comparison with a reference signal or changes within the received signal (Differential PSK).

An advantage of PSK over FSK is that the bandwidth that is needed for transmission is as low as half (for binary symbols) and the main advantage over ASK is fading resistance, but these advantages come at the expense of complexity and cost.

Direct Sequence Spread Spectrum

Spreading is the process of multiplying a data-signal with a pseudorandom sequence PN of $\{1,-1\}$ at a much higher rate than the data-rate before digitally modulating and transmitting the data. On the receiving end, the signal is demodulated and de-spread with the same sequence PN.

This makes the signal spectrally much wider, ideally comparable to gaussian white noise, and achieves a higher signal-to-noise ratio as the de-spreading can be seen as band-pass or autocorrelation with PN.

The technique is used among other things in GPS, CDMA, IEEE 802.15.4 and IEEE 802.11b. A further advantage is that on the Media Access layer, the autocorrelation reduces co-channel interference when applications use different PN. This is most readily seen in the cellular CDMA standard.

Frequency Hopping Spread Spectrum

In FHSS, the spectrum is spread uniformly by frequently re-tuning the carrier wave of the transmitter and receiver. The channel cohabitation advantages are the same as for direct sequence.

From a regulatory perspective, FHSS and DSSS are allowed to use higher transmit effect, because the spreading reduces harmful interference with other applications.

3.2 Media Access Layer

A protocol determining who has access to the transmission medium and when is important for wireless networks, as all nodes operating on the same frequency in a vicinity share the same medium.

Sharing of the medium can be achieved in several ways, but the following are common in wireless networks[57]:

Carrier Sense Multiple Access w/ Collision Avoidance Devices check for clear channel before transmitting. If an acknowledgement is not received, a collision is assumed to have taken place, and an exponentially increasing delay is inserted before the device tries again.

Code Division Multiple Access As described in the previous section, this is the same as Spread Spectrum Multiple Access.

Time Division Multiple Access Devices may only transmit in appointed time-slots.

Frequency Division Multiple Access Devices transmit in separate frequency bands (channels) with little or no overlap.

No Access Control / Spray and Pray Devices transmit whenever they need to, without regards to the wireless environment. For cheap uni-directional transmitters this is the only option. For some measure of robustness, packets are repeated a finite number of times.

3.3 Network Layer

Common services afforded by wireless sensor node stacks are:

Broadcast The most basic form of routing, typically used during integration of a node into a network.

Master/Slave Nodes have different roles, and slaves may not interconnect.

Point to Point Any node can address any node if it's within reach.

Mesh A routing algorithm is in place to allow any node to address any node even without a direct radio-link.

3.4 Security

All radio transmissions are easily interceptable. The only thing that needs to be known is the operating frequency and the modulation technique. For open standards such as Bluetooth and IEEE 802.15.4 this is true even with pseudorandom spectrum-spreading, because the spreading sequences are known or can be obtained by listening to channel setup. Projects at NTNU have even shown how to intercept and decrypt mobile GSM communications[63].

3.4.1 Encryption

Hence, applications must rely on encryption instead of obscurity. Because wireless nodes are often severly limited in capacity, symmetric encryption such as AES is often used, for instance in IEEE 802.15.4 or an algorithm called SAFER+ for Bluetooth.

Limitations of Symmetric Encryption

Because asymmetric encryption is too computationally expensive, both the sender and receiver must use the same encryption key. When using only radio for communication, this key can either be pre-stored or exchanged in the clear.

Even though the encryption itself may be considered safe, the practice is intrinsically unsafe because a dedicated attacker can always dump the memory of a device or snoop the exchange and thus gain access to the entire network[53].

Out of Band

A viable solution is to exchange keys through some other means that are complementary to the RF communication and may not be intercepted via RF, such as:

- Infrared or light
- Inductive communication
- Electrical contact
- Push-buttons

These methods require extra hardware-components and in some cases physical proximity, but will guarantee secure communication as long as:

- The encryption scheme is secure
- The OOB exchange is not eavesdropped
- Nodes use different keys between different peers

The last item is important because otherwise a fully integrated node could be tampered with and expose the whole network[53].

3.4.2 Privacy

Even with encryption it should be noted that the location of a transmitter is easily detectable via triangulation. Additionally, the fact that a transmission takes place can in many cases be enough to infer the characteristics of the node, thereby guessing what data must have been transmitted.

The only solution to this would be to periodically transmit data even if no new information is sent and accepting the resulting drop in battery life and possible increase in latency.

4 Existing Wireless Communication Solutions

4.1 ZigBee

The ZigBee standard is based on the IEEE 802.15.4 standard[60]. ZigBee features low power consumption and simple implementation, link-layer mechanisms for implementing security encryption, and supports a and high density of nodes per network. This technology is marketed as suited for sensor- and control networks. Operation is at 2.4 GHz and 868/915 MHz and the data rates offered are respectively 250 kbps, 20 kbps and 40 kbps.

The ZigBee stack is shown in Figure 4.1, and we can see the physical and media access layers which use OQPSK[1], direct sequence spread spectrum and CSMA/CA mentioned in the previous chapter. In the middle we see the network layer which supports a number of routing topologies and at the top, the ZigBee Device Object, a special application endpoint present on all Zigbee devices, which keeps Application Objects – similar to the concept of device profiles seen in Bluetooth – that are defined by the ZigBee Alliance.

There are two different device types that can be implemented in a ZigBee network:

- Full function device (FFD) — Can function in any topology and talk to any other device. It is also capable of being the network coordinator and a function as a routing device.
- Reduced function device (RFD) — Has a very simple implementation and can only function in a star topology. It is only capable of talking with an FFD.

Supported network topologies are star, peer-to-peer and mesh and an example of a ZigBee network can be seen in Figure 4.2.

In a star topology, at least one FFD is required in the network to function as a network coordinator. The other devices could be RFDs, which have the advantage

[1]Offset Quadrature Phase Shift Keying, a variation of PSK with two bits per symbol.

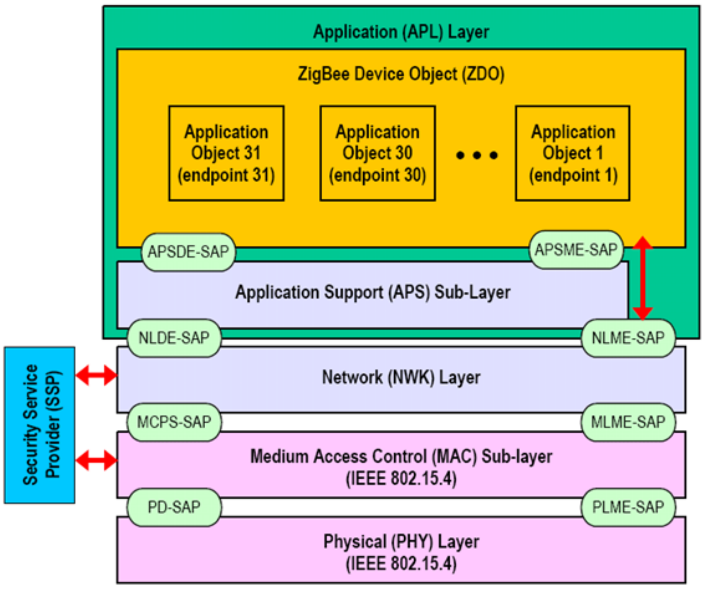

Application (APL) Layer
ZigBee Device Object (ZDO)
Application Object 31 (endpoint 31)
Application Object 30 (endpoint 30)
Application Object 1 (endpoint 1)
APSDE-SAP
APSME-SAP
Application Support (APS) Sub-Layer
NLDE-SAP
NLME-SAP
Security Service Provider (SSP)
Network (NWK) Layer
MCPS-SAP
MLME-SAP
Medium Access Control (MAC) Sub-layer (IEEE 802.15.4)
PD-SAP
PLME-SAP
Physical (PHY) Layer (IEEE 802.15.4)

Figure 4.1: ZigBee Stack system requirements, from [60]

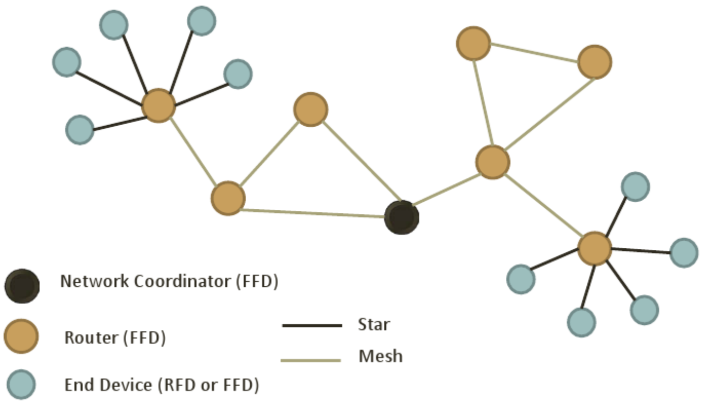


Figure 4.2: ZigBee network model

of being able to spend a lot of time in sleep, thereby reducing power consumption.

Any topology requires a network coordinator which is responsible for setting up the network, transmitting network beacons, managing network nodes and route messages between nodes. The network nodes search for available networks, transfer data when needed and request data from the network coordinator. If the topology is not star, FFDs are needed to act as routing devices as well.

Several home automation systems that use the ZigBee standard exist today, examples include the AlertMe[1] system, along with as a wide variety of single units that can be included in a self-made system.

4.2 System NEXA

System NEXA is a brand of Swedish-produced home automation products that operate at 433.92Mhz. These are available for purchase from Clas Ohlson at low cost. The system is based on point-to-point communication from a transmitter to a receiver where one receiver can be controlled by several transmitters. Transmitters include motion detectors, magnetic contacts and light switches, while receivers are mainly on/off relays or dimmers for light sources.

The System NEXA receivers have implemented what is called self-learning code. When a receiver is set in self-learning mode and the unit it should listen to must be activated. This pairs the units, and lets the transmitter control the receiver.

Wall outlet plugs that can be controlled by a remote control or a wireless wall-mounted light switch which can be seen in Figure 4.3.

[a]Images from www.nexa.se

Figure 4.3: Nexa wall outlet plugs with remote control and wireless light switch[a]

4.3 Z-Wave

Z-Wave is a home automation network with low bandwidth designed for low cost networks[44]. It has a mesh topology and operates at 868MHz with data rates of 9600bit/s or 40 kbits/s. A Z-Wave network consists of two types of devices:

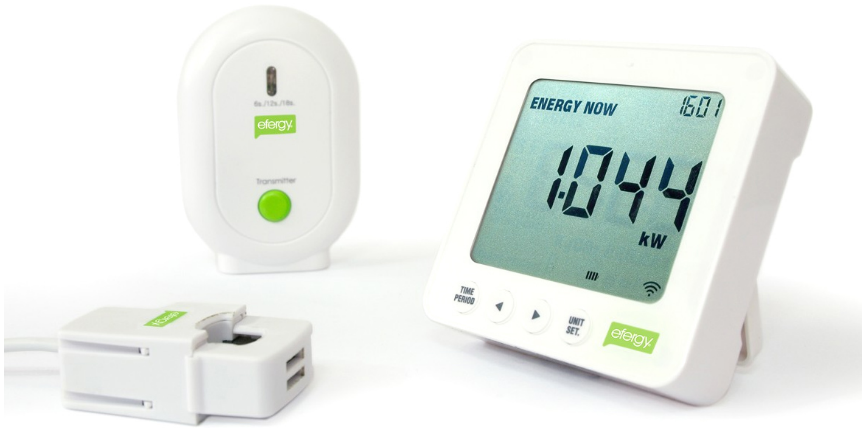
- Controlling devices that initiate control commands and send these to other nodes in the network.
- Slave devices that execute the commands. Slave devices are also capable of forwarding commands to other nodes out of reach of the controlling device.

PC software can communicate with the controlling devices via a serial interface using a dongle. Nodes are added to a network by simply pressing a button which pairs the devices. The same procedure can be used for the removal of nodes.

The protocol of Z-Wave is used by many home automation software solutions such as IP Symcon, Homeseer, Embedded Automation, 4HomeMedia and Control-Think.

4.4 Efergy

Efergy is a company that sells wireless energy consumption meters to markets all over the world. Their meters are sold as pairs of receiver and transmitter, and are not affiliated with or integrated in any other wireless system. Figure 4.4



[a]Product shot from www.efergy.eu

Figure 4.4: Efergy e2 Energy-Now Powermeter retailed at Clas Ohlson. Features wireless transmission of recorded power consumption[a]

shows a display unit and remote sensor with a transformer clamp intended for attachment to three-phase leads inside a consumer fuse-box. The display unit can store historical data and transfer this to a computer via an USB connection, but only in summarized form, and cannot transfer live data this way.

4.5 Bluetooth

This section is an excerpt from Normann/Skjønsfjell[66].

4.5.1 Classic Bluetooth

Bluetooth is a wireless technology standard which is used for transfer of data over short distances. Invented by Ericsson in 1994[13], the radio technology operates in the globally unlicensed industrial, scientific and medical band (ISM) at 2.4 to 2.485 GHz.

Bluetooth uses adaptive frequency-hopping spread spectrum. This is a method of transmitting signals by switching a carrier wave among many frequency channels, and gives resistance to radio frequency interference by using only available frequencies. The signal appears as full-duplex through a Time-Division Duplex (TDD) scheme. The range can be from 1 - 100 meters, but a 10 meter minimum range is mandated by the Core Specification[12].

The Bluetooth protocol is packet-based and has a master-slave structure. One master can communicate with up to seven slaves in what is referred to as a piconet, and all the devices share a radio channel hop list and are synchronized to the master's clock. The frequency-hopping pattern is determined by the Bluetooth address and clock of the master.

Transmission of data between devices is divided into packages, which are placed into time slots in the time-multiplexed channel.

The Bluetooth core system consists of a Host and one or more Controllers. A Host is "..a logical entity defined as all of the layers below the non-core profiles and above the Host Controller Interface (HCI)."[12] A Controller is "..a logical entity defined as all of the layers below HCI."[12]

Classic Bluetooth includes Basic Rate (BR) and Enhanced Data Rate (EDR) and it offers data rates of 721.2 kbps for BR and 2.1 Mbps for EDR. There is also an Alternate Media Access Control (MAC) and Physical (PHY) layer extension which offers data rates up to 24 Mbps with the 802.11 AMP[2] (High Speed).

4.5.2 Bluetooth 4.0

The 4.0 specification was adopted June 30th, 2010. This includes the Classic Bluetooth and Bluetooth High Speed as well as the new edition: Bluetooth Low Energy (BLE). As Bluetooth Low Energy is more interesting than Classic Bluetooth and Bluetooth High Speed in a health sensor aspect, the rest of this section will focus on this specification.

Bluetooth Low Energy BLE is designed for applications with lower data rates, lower current consumption and lower complexity and cost than Classic Bluetooth devices. BLE has lower transfer rates than Classic Bluetooth, but lower latencies and power consumption makes it ideal for use in sensor devices.

The BLE Protocol stack is illustrated in figure 4.5. The LE Protocol stack consists of controller and host, as BR. The different layers in the stack will be presented here:

- PHY: The physical layer of BLE also operates in the 2.4 GHz ISM band and uses frequency hopping. To minimize transceiver complexity, a shaped, binary frequency modulation is used. BLE supports data rates of 1 Mbps.

  There are two LE physical channels defined:

[2]Alternate MAC/PHYs (AMP)

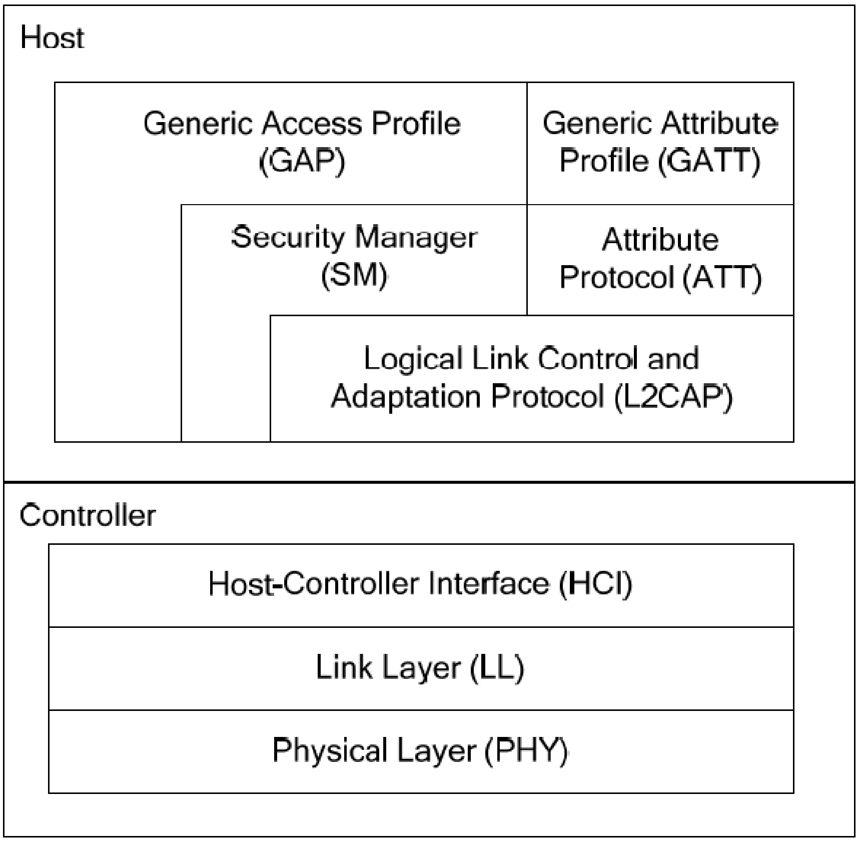

Host
Generic Access Profile (GAP)
Generic Attribute Profile (GATT)
Security Manager (SM)
Attribute Protocol (ATT)
Logical Link Control and Adaptation Protocol (L2CAP)
Controller
Host-Controller Interface (HCI)
Link Layer (LL)
Physical Layer (PHY)

Figure 4.5: BLE Protocol Stack[39]

- LE piconet channel: Used for communication between connected devices.
- LE advertisement broadcast channel: Used for broadcasting advertisements to devices.

A device can only use one of these channels at a time, and time-division multiplexing between the channels is used to support concurrent operations. In contrast to a BR channel, an LE channel is divided into events, where packets are placed for transmission. There are two types of events, Advertising and Connection events.

- Link Layer (LL) : The LL controls the RF[3] state of the device. There are five different states a device can be in: Standby, advertising, scanning, initiating and connected.

[3]Radio Frequency

Transmitters of advertising packets are called *advertisers* and devices receiving advertising without connecting to the advertiser are called *scanners*, and these are in advertising and scanning state.

Devices listening to connectable advertising packets with the purpose of forming a connection to another device are called *initiators* and are in the initiating state. Upon receipt of a connectable advertising event on an advertising PHY channel, the initiator can make a connection request using the same channel. If the advertiser accepts the request, a connection is established and connection events take the place of advertising events. Now, the initiator becomes the master device and the advertiser the slave device, and they are both in the connected state.

- Host Controller Interface (HCI): The HCI layer provides a standardized interface for the host and controller to communicate. The separation is that between logical and physical layers, and can be seen in Figure 4.5

- Logical Link Control and Adaptation Protocol (L2CAP): The L2CAP layer "..provides data encapsulation services to the upper layers, allowing for logical end-to-end communication of data"[39]. L2CAP channels are used for general purpose communication.

- Security Manager protocol (SMP): This protocol is used to implement security functions between devices. Uses a fixed L2CAP channel.

- Generic Access Profile (GAP): GAP is a base profile which all Bluetooth devices must implement. GAP service includes device discovery, connection, security and authentication. GAP defines four different roles for LE: Broadcaster, Observer, Peripheral and Central.

- Attribute Protocol (ATT): This is the peer-to-peer protocol between attribute server and client, and it provides a method to communicate small amounts of data and to determine services and capabilities of other devices. Uses a fixed L2CAP channel.

- Generic Attribute Profile (GATT): GATT is built on top of ATT, and represents the functionality of the attribute server. It is used for BLE profile service discovery and defines server and client roles. Both GATT and ATT are mandatory in BLE as they are used for discovering services. A GATT server accepts commands and confirmations from a GATT client. It also sends asynchronous notifications to a client when preconfigured event types occur. As characteristics and descriptors from the server are available to clients, a client does not have to implement a specific profile to access information from the GATT server.

  A profile describes services, characteristics and attributes used in the attribute server and provides interfaces for discovering, reading and writing these.

Some characteristics and GATT profiles are already adopted by the Bluetooth SIG[4]. The adopted profiles are Alert Notification, Find Me, Health Thermometer, Heart Rate, Phone Alert Status, Proximity and Time. There are characteristics for blood pressure measurement, body sensor locations, time and date, heart rate measurement, temperature and many others. The adopted profiles and characteristics gives a clear indication that BLE is intended for use in health monitoring.

For a more thorough explanation of BLE, see the Bluetooth Core Specification[12] and Using Bluetooth Low Energy in Sensor Devices[46].

[4]Bluetooth Special Interest Group. Profiles and characteristics can be found at `http://developer.bluetooth.org/gatt/Pages/default.aspx`

5 Sensors

To track the activity patterns in a domestic environment, several types of information can be interesting. Examples of information types include position inside the house and amount of movement together with gait speed, which will be presented in the following sections.

5.1 Indoor Positioning

For determining indoor positioning, two main approaches can be distinguished; Stationary and ambulant/body-worn systems. The ambulant systems consist of body-worn transceivers in combination with beacons, while stationary systems rely exclusively on stationary transceivers.

5.1.1 Positioning with Ambulant/Body-Worn Transceivers

More or less any kind of receiver/transmitter can be used, for instance RFID, WLAN, UWB[1] and Bluetooth.

For this type of positioning system, four topologies exist[62]:

- Remote positioning system - Includes a mobile signal transmitter as well as several stationary units receiving the signal. The position of the mobile transmitter is calculated in a central unit based on the results from all the receiving units.
- Self-positioning systems - The receiving unit is mobile, and computes its own location based on signals from several stationary transmitters.
- Indirect remote positioning system - The receiving unit of a self-positioning system sends its results to a central unit, which calculates the position.

[1]Ultra Wide Band

- Indirect self-positioning - The results from the stationary receivers are sent to the mobile device and the position is calculated in the mobile device.

Determining Position

Proximity Algorithms When a mobile unit is detected by one base station, it is presumed to be in the near vicinity of this. If the mobile unit is detected by several base stations, the mobile unit is presumed to be in the vicinity of the base station detecting the strongest signal.

Triangulation Classic triangulation determines location by using the geometric properties of triangles. A location is found as the last point of a triangle with one known side and two known angles.

There are two types of triangulation; lateration and angulation. Lateration measures distance directly by using signal strength, time of arrival or time difference of arrival. Angulation estimates a location by somehow measuring the angles to it from two known points attached to a fixed baseline.

Scene Analysis/Location Fingerprinting A collection of signal strength measurements ("fingerprints") between different locations is created before starting the analysis. The location is later determined by comparing the current measurement with the characteristics of the fingerprints and finding the closest match[58].

5.1.2 Positioning with Stationary Transceivers

A stationary system may for example use cameras or motion detectors to determine position in certain zones. These systems depend upon the sensing devices being placed at appropriate locations in the indoor environment.

Video

For surveillance of poultry, a video tracking system that can track several birds exists[70]. The video tracking software distinguishes objects to be tracked and analyzes the bird's behavior such as position, movement speed and time spent in areas of interest. Video tracking systems such as this can monitor more than one tracking subject, but usually in only one room at a time.

Motion detectors

Other surveillance techniques include using motion detectors or other types of stationary sensors that provides information about activity in a certain zone. Normal motion detectors are not able to distinguish several subjects, and should therefore only be used for application designed for tracking one subject.

Motion detectors are normally active or passive, with passive sensors not emitting any energy. The most common type of sensor is PIR (Passive Infra-Red) sensors, which detect body heat.

Magnet Switches

Stationary transceivers can also include magnet switches on doors or closets to detect opening and closing of these. In a context aware system, as described in the next chapter, this information can be used to estimate the position of a subject, such that opening and closing an entrance door with no following activity in the house could be an indication of the subject being outside.

Pressure Mats

Pressure sensor mats[36] can also be used, e.g. to detect whether a person is lying in a bed, or sitting in a chair. These could also be placed on either side of a door, to directly detect a subject moving in or out of a zone with more confidence than magnet switches.

5.2 Other Means of Tracking Activity

5.2.1 Body-Worn Sensors

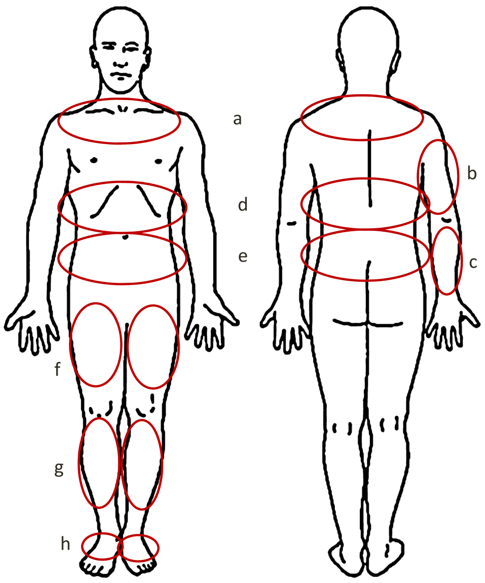


Figure 5.1: Areas most fitting for wearable sensors: (a) collar area, (b) rear of upper arm, (c) forearm, (d) rear, side and front ribcage, (e) waist and hips, (f) thigh, (g) shin, (h) top of the foot. Figure adapted from [51].

An accelerometer on a test subject can be used to get information about the amount of movement and body orientation. The quality and consistency of any body-worn sensor is very dependent on the test subject, as wearing the sensor is crucial to getting information from it.

For body-worn sensors, an important aspect is where to place them. A study on the topic[51] has determined the best placement for body-worn sensors, which is presented in Figure 5.1.

Pros and cons for different placements for an accelerometer on a person can be seen in Table 5.1.

Placement	**Pros**	**Cons**
Belt	Easy to fasten Well defined $\vec{g^b}$ Good mechanical connection	Ambiguous result from sitting and standing Does not catch direction of upper body
Pocket	Easy to wear	Varying what position is down, difficult to determine body position Does not catch direction of upper body Not so good mechanical connection
Sown into shirt	Good mechanical connection Catches direction of upper body Well defined $\vec{g^b}$	Same shirt needs to be worn every day Might be uncomfortable
Around ankle	Can compute stride length	Might be uncomfortable
Necklace	Can be worn regardless of clothing Well defined $\vec{g^b}$ if the necklace is not dangling Easy to wear	Not so good mechanical connection

Table 5.1: Our considerations on accelerometer placement. By $\vec{g^b}$ is meant direction of gravity with respect to the body-axis system which a sensor is assumed to represent.

5.2.2 Gait Speed

In a health aspect, measuring the gait speed of a subject can be of great interest, as a study[68] shows that gait speed can be associated with survival of older adults. This study states that deteriorating health leads to a declining gait speed, and measuring gait speed over time, can therefore give an indication of the health of a test subject.

5.2.3 Interaction with the Environment

Monitoring how a subject interacts with the environment can also be beneficial. Examples of such interaction could be use of kitchen appliances and washing machines, use of hot water for e.g. showering etc. Interaction with electrical appliances could for instance be detected by monitoring energy use.

5.3 Characteristics of Reported Location

Ambulant and stationary systems differ in the characteristics of their reported position. Stationary systems can only detect activity within a specific zone, but this with a high accuracy. They have a low precision in that they can't determine where in the zone the activity has taken place.

The probabilities for tracked objects being in a given location for the two types of systems are illustrated in Figure 5.2. Here, it can be seen that in a stationary system, the probability is uniform over the entire zone, while for an ambulant sytem, the probability is distributed around a point.

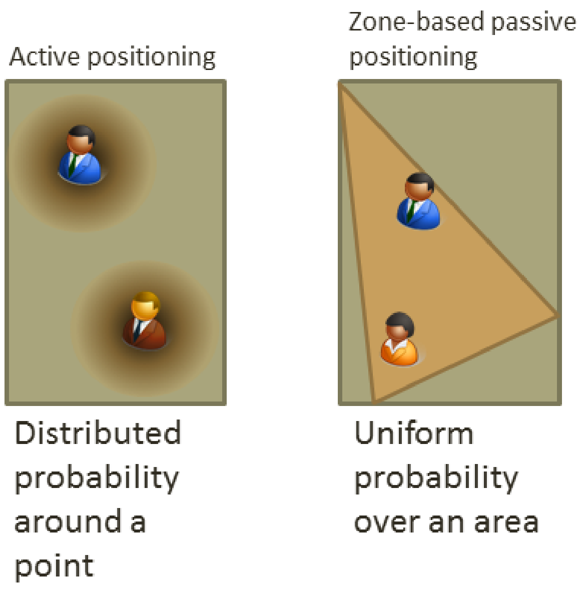

Active positioning
Zone-based passive positioning
Distributed probability around a point
Uniform probability over an area

Figure 5.2: Probability distribution for reported location

Ambulant systems can locate the activity more precisely, but may not be able to map the location directly to a zone. This is illustrated in Figure 5.3, where the subjects are located close to a wall. The subjects' positions can thereby wrongly be interpreted as being in the same room, or in the room which they are not in.

In a health aspect, the classification of activity into certain zones may be crucial, as whether a subject for instance is in the bedroom or in the bathroom (if these are located next to each other) can lead to different information being extracted.

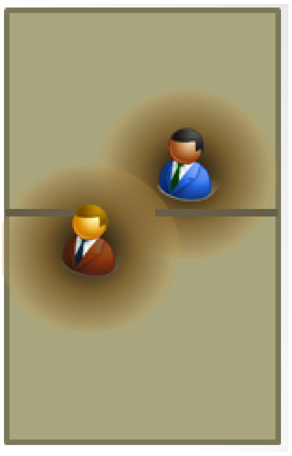


Figure 5.3: Possible problematic situation with active triangulation of position.

An example is if a location-based system interprets a user getting up in the middle of the night and going to the bathroom as all happening inside the bedroom. This could lead to prolonged immobility in the bathroom being incorrectly considered as the person being in bed instead of giving an alert of what might be a critical situation.

5.3.1 Precision and Accuracy of Sensors

We have here made a distiction borrowed from statistics between the precision and the accuracy of sensor measurements. By precision is understood the degree of detail provided by the sensor, as well as the degree to which repeated measurements show the same results. Accuracy is understood as whether the measurement reported is actually true.

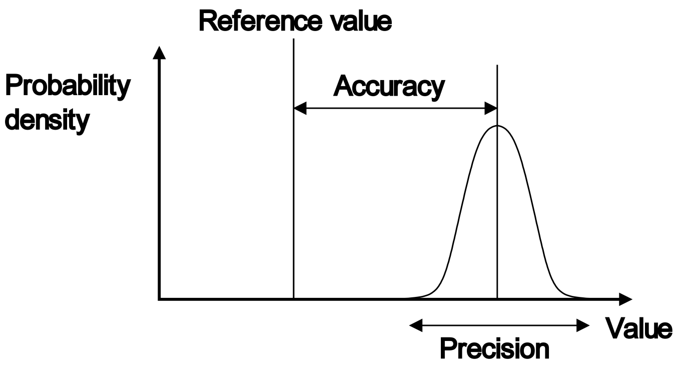


Figure 5.4: Relationship between real value and the accuracy and precision of measurements. (©Wikimedia Commons)

6

Context-Aware Processing

Raw sensor output doesn't give much information if it is not evaluated in a context. For instance, what is the scale of the data, what is the accuracy and precision, what does it imply?

In many classic control systems, an IO function maps these raw values explicitly according to the application. This quickly becomes unwieldy if we have several sensors of a different physical nature or from different vendors giving information on the same thing in different ways, as is often the case in so called Cyber-physical Systems.

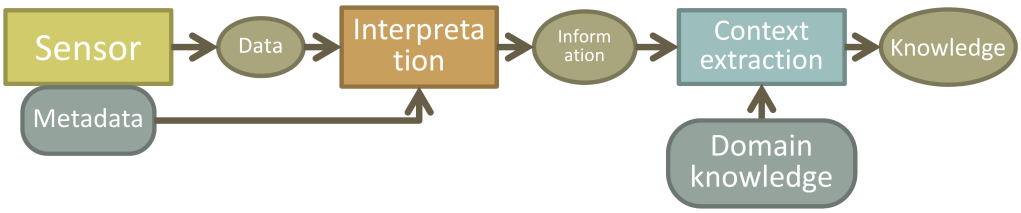


Figure 6.1: Overview of transformation of data to knowledge

A distinction can be made between data, information and knowledge[54], such that:

- Data is the lowest level of sensor output, and cannot be interpreted without knowledge of units, reference systems etc.
- Information can be understood by humans.
- Knowledge is information in combination with theories of the applicable system.

Thus, a knowledge based system is able to react to and understand input at a higher level of abstraction more aligned with human cognition. It is therefore desirable to abstract away sensor details from higher order processing. Figure 6.1 shows a generic data-path from sensor data via abstraction and processing to contextual knowledge.

Below, two real-world examples will be mapped to this model to better explain the meaning of the different parts.

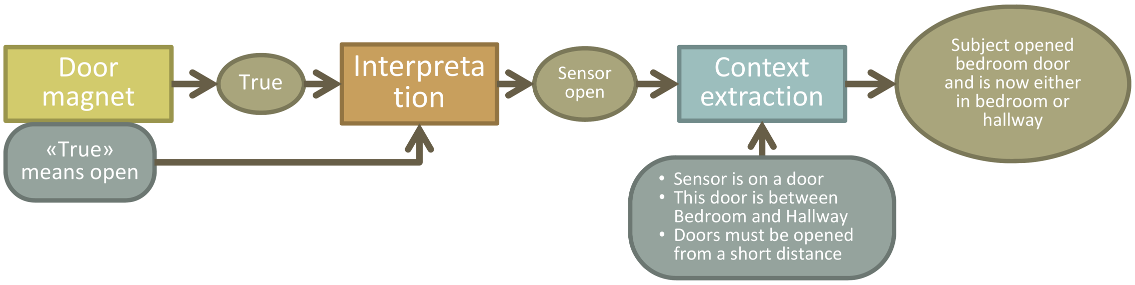

Door magnet
True
«True» means open
Interpretation
Sensor open
Context extraction
• Sensor is on a door
• This door is between Bedroom and Hallway
• Doors must be opened from a short distance
Subject opened bedroom door and is now either in bedroom or hallway

Figure 6.2: Processing magnet switch sensor data

Figure 6.2 shows how a door sensor event is translated to usable knowledge. In the first abstraction layer by knowing how to interpret the value from the sensor, and in the second by using knowledge of the sensor placement and some assumptions about the physical world.

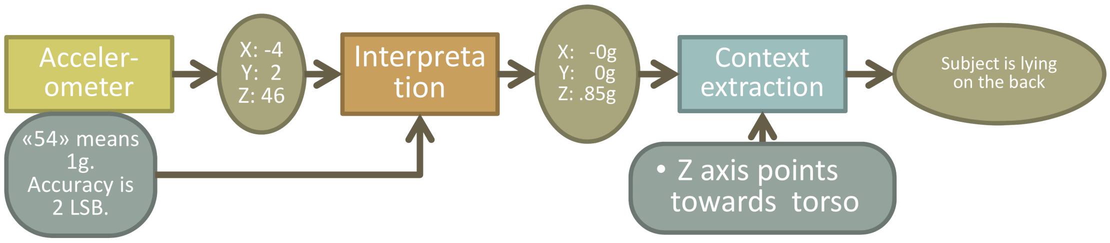

Accelerometer
X: -4
Y: 2
Z: 46
«54» means 1g. Accuracy is 2 LSB.
Interpretation
X: -0g
Y: 0g
Z: .85g
Context extraction
• Z axis points towards torso
Subject is lying on the back

Figure 6.3: Processing body-worn accelerometer data

Figure 6.3 shows a similar case where metadata is taken from a device datasheet in the first layer of abstraction, and knowledge of the body coordinate system and some thresholds is used to say something on a more narrative/cognitive level about the orientation of the subject.

6.1 Interpretation

The transformation of raw data to information can be as simple as scaling the raw data, but information can also result from a combination of raw data sources or some aggregation of raw data. Schmidt et. al[69] have presented something called the TEA[1] model.

[1]Technology for Enabled Awareness

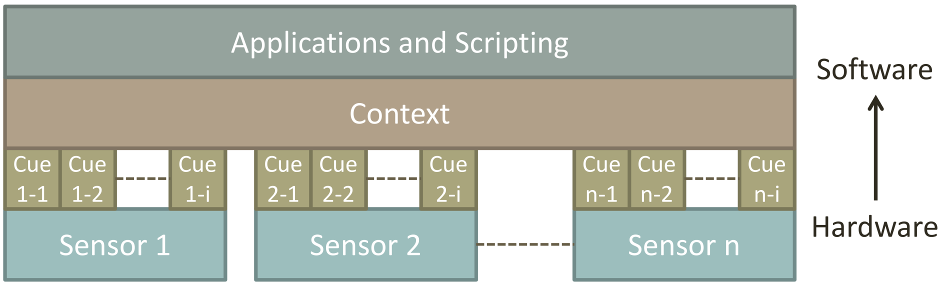


Figure 6.4: TEA Model. Figure adapted from Schmidt 1999.

In this model, a sensor is a "Smart Sensor" and can emit *cues* which can be anything from an event taking place to an average value or a more complex function such as machine learning classification.

A simple example is that of an accelerometer reporting only a sliding standard deviation as a measure of the activity level. An advantage of using aggregated cues in this fashion is a significant reduction of transmitted data - a desirable property for wireless sensor nodes.

6.2 Context Extraction

> "**Context**: any information that can be used to characterize the situation of entities (i.e., whether a person, place, or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity, and state of people, groups, and computational and physical objects." - Dey 2001

A few practical software models have been developed for gathering context information and presenting them to an application. Two of these, the "Context Toolkit" presented by Dey et. al. and a Java framework developed by Henricksen and Indulska will be briefly described to give a feeling for the work in this field.

6.2.1 Context Toolkit

In their 2001 paper, Dey et al[48] describe some requirements for a framework dealing with context. These are inspired by traditional GUI toolkits, and can be summarized as follows:

Separation of Concerns Sensor data acquisition methods should be hidden from higher layers.

Context Interpretation If a certain context needs several layers of interpretation, these layers should be transparent to the application.

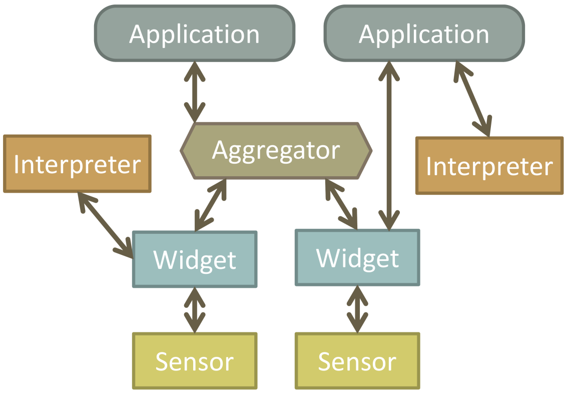


Figure 6.5: Overview of "Context Toolkit" as described by Dey[48]

Transparent, Distributed Communications Input should be able to come from any of several interconnected concentrator nodes.

Constant Availability of Context Information Widgets holding context information should be running constantly, as it is not known when this information is needed by an application.

Context Storage and History A repository of historical context states should be available independent of any application.

Resource Discovery Instead of hardcoding addresses of sensors, an application should be able to request what type of context is needed and be provided with appropriate component references.

Components of the "Context Toolkit"

Figure 6.5 shows the interconnection between the component classes of their framework:

Widget Provides the separation of concern mentioned in the requirements, hiding complexity of sensors and providing abstracted context information such as location. Widgets provide callback notification of context changes to an application and can be polled.

Interpreter Reasoning engine, responsible for raising the level of abstraction - for instance activity, noise and many occupants might suggest a meeting.

Aggregator Facilitates the requirement of distributed communication, collects and aggregates context cues that may be of a different nature, but relate to a single logical entity.

6.2.2 Henricksen/Indulska framework

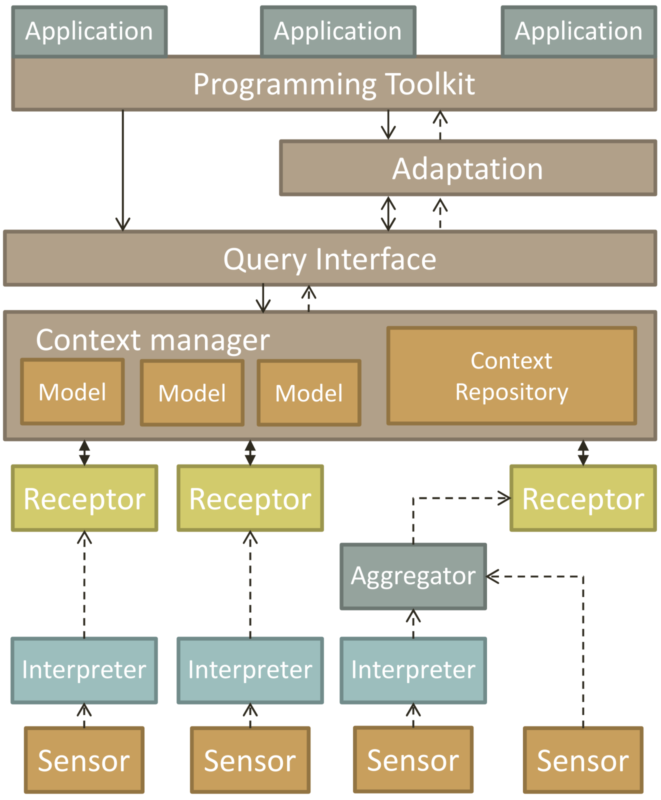


Figure 6.6: Java framework presented as an approach to developing context-aware applications.

Henricksen and Indulska point out that context can originate from a variety of sources resulting in heterogenity in terms of quality and persistence[59]. Further they point out that context may be:

- Sensed, i.e. highly dynamic and subject to noise.
- Derived, similar to aggregated / raised in Dey.
- User-supplied, initially reliable but degrades.
- Static.

It is also mentioned that context information is often imperfect because of sensing errors and noise, but it may also happen that sources of information report conflicting data or none at all, resulting in ambiguous or unknown situations.

They then argue that as a parameter describing the quality of extracted context information, it is not sufficient to supply only a *confidence* attribute, but that an additional ternary attribute of `True`, `Possibly true` and `False` is required to better represent ambiguity.

Framework Components

The components as shown in Figure 6.6 on the preceding page are designed as loosely coupled, and a content-based routing scheme is used. A loosely coupled structure increases failure tolerance and facilitates expansion.

Context Gathering Layer Sensor, Interpreter and Aggregator form the lowest layer, responsible for raw data mapping, interpretation and data fusion.

Context Reception Layer Receptors map queries from the context manager to appropriate components and translates lower layer inputs into fact-based representations.

Management Layer Maintains models and relationships between instances of these models. This layer is also responsible for handling and inserting static, profiled (through machine learning) or user-inserted context cues.

Higher Layers Relate to query routing and repositories of situations, event triggers and user preferences.

6.3 Considerations

Reviewed papers on the subject of machine intelligence and context extraction put a varying degree of emphasis on expressing rules and knowledge in terms of formal logic. For this thesis and in the preceding framework summaries we have ignored this aspect somewhat, as generalizing and reasoning about knowledge of the world by artificial intelligence is a field unto its own.

7 Automated Reasoning

7.1 Pattern and Behaviour

The motivation for using automatic reasoning is to establish models of the expected behavior of a human being. An approach to detect behavior patterns of occupants of a smart home is presented in Lühr[61]. This article proposes the use of intertransaction association rules (IAR) mining for behaviour analysis.

Classic association mining is a method of distinguishing sets of items that sequentially occur together in a database, whereas non-sequential relationships of events are allowed to be captured with IAR. Other ways of detecting patterns include episode mining, where partial ordering of items are found without considering time intervals.

IAR consider the associative relationships between items within certain time intervals, without focusing on the ordering of them. This makes sense as human behaviour is usually not sequentially predictable due to interruptions etc., but certain actions must nevertheless occur together.

IAR consider the associative relationships between items within certain time intervals, without focusing on the ordering of them. This makes sense as human behaviour is usually not sequentially predictable, but certain actions must nevertheless occur together.

For example, the rule $A_0, B_0, C_1, D_3 \Rightarrow E_3$ says that if A and B is encountered in the current transaction interval, C is encountered in the next interval and D in the third interval, this implies that E also will occur in the third interval. This means that When A, B, C and D has happened, a normal pattern would be that E should happen shortly after. An illustration of this could be that when the test subject is going to bed, doors will be closed and the bathroom will be used before going into the bedroom to go to sleep. The pattern may not be the same every evening, and this is accounted for with the IAR.

7.2 Anomaly detection

IAR can also be used to detect anomalities from the recongnized patterns. A method for detecting abnormal values in a discrete dataset on the other hand, is presented in Seem[49]. In this paper, the method is used for detecting abnormal energy consumption in buildings, but could be adapted for use in smart homes as well, by using the method on the amount of time spent in separate zones in a house. This would require an in advance analysis of the smart home, to decide upon which rooms should be analyzed.

This method is based upon sorting power consumption for days into groups with theoretically similar consumption, such as weekends and weekdays. It then finds abnormalities by outlier identification. Outliers are data points that appear as inconsistent with the rest of the data in a set. The procedure for identifying outliers can be seen in Figure 7.1. An upper bound (n_u) on the number of potential outliers must be set, and the flowchart is looped through the same number of times as possible outliers. The upper bound can be found from the inequality 7.1:

$$n_u \leq 0.5\,(n-1) \tag{7.1}$$

Where n is number of entries in the data set. Average value and standard deviation for the data set are calculated with Equation 7.2 and 7.3:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7.2}$$

$$\sigma = \sqrt{\frac{\sum\limits_{i=1}^{n}(x_i-\bar{x})^2}{n-1}} \tag{7.3}$$

If the standard deviation is zero, all values in the data set are the same, and no abnormality is detected. If not, the value furthest from the average value is found and removed from the data set. The extreme studentized deviate of the removed value is computed with Equation 7.4:

$$R_i = \frac{|x_{e,i}-\bar{x}|}{\sigma} \tag{7.4}$$

The critical value λ_i is then computed from Equation 7.6, developed by Rosner[67]:

$$\lambda_i = \frac{(n-i)t_{n-i-1,p}}{\sqrt{(n-i+1)(n-i-1+t_{n-i-1,p}^2)}} \tag{7.5}$$

Where $t_{n-i-1,p}$ is Student's t-distribution with (n-i-1) degrees of freedom, and probability p is determined from 7.6:

$$p = \frac{\alpha}{2(n-i+1)} \tag{7.6}$$

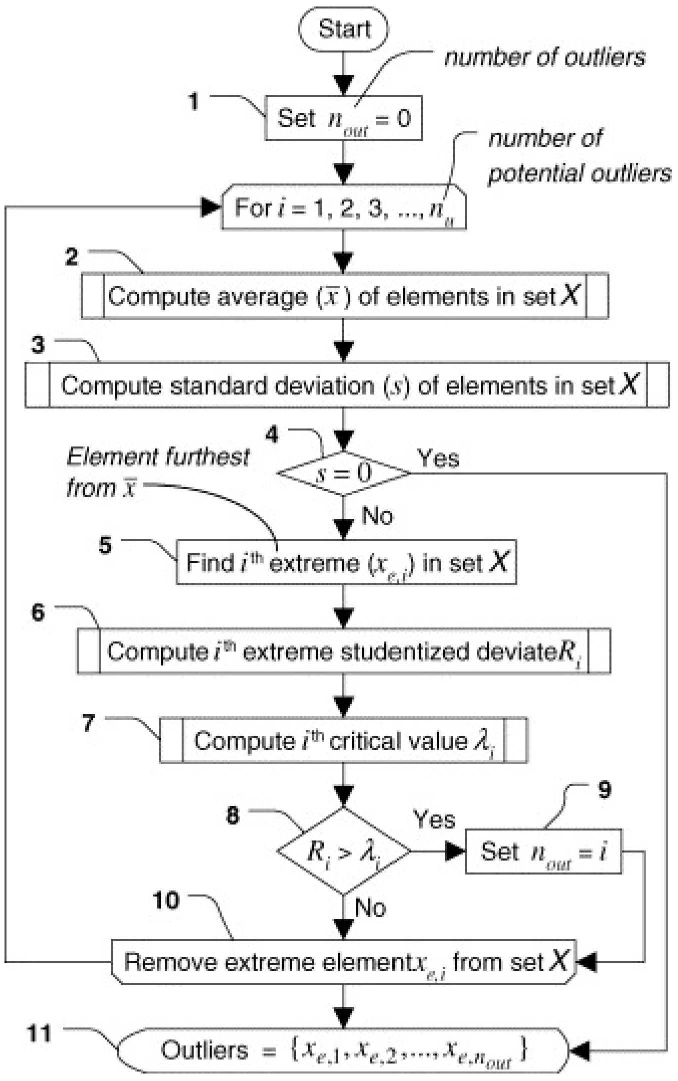

Start
number of outliers
1
Set $n_{out} = 0$
number of potential outliers
For $i = 1, 2, 3, ..., n_u$
2
Compute average ($\bar{x}$) of elements in set X
3
Compute standard deviation (s) of elements in set X
4
Element furthest from $\bar{x}$
$s = 0$
Yes
No
5
Find i^{th} extreme ($x_{e,i}$) in set X
6
Compute i^{th} extreme studentized deviate R_i
7
Compute i^{th} critical value λ_i
8
9
$R_i > \lambda_i$
Yes
Set $n_{out} = i$
10
No
Remove extreme element $x_{e,i}$ from set X
11
Outliers = $\{x_{e,1}, x_{e,2}, ..., x_{e,n_{out}}\}$

Figure 7.1: Flow chart for implementing many-outlier procedure. From [49]

α is provided by the user, and is the probability of incorrectly declaring outliers when none exists.

If the extreme studentized deviate is larger than the critical value, the data element is marked as an outlier, and in any case, it is removed from the data set, before the process is started all over. If i equals the number of potential outliers, the procedure is finished, and extreme values or outliers are found.

The method also proposes a way of using a modified z-score to determine the severity of a an outlier, which will not be covered here.

8 Mobile Development

The two dominating operating systems for smart phones and tablets as of 2012 are Android from Google Inc. and iOS from Apple Inc. Together, these two have 82% market share[4]. Several other operating systems are also available at the market, common platforms include BlackBerry OS from RIM, Windows Phone from Microsoft and Symbian OS from the Symbian Foundation.

An IDE[1] with an SDK[2] targeting the platform is normally used to develop applications for these operating systems. Tools for writing, testing and deploying applications for the platform are provided from different vendors. These often include an emulator or a simulator, to eliminate the need for a physical device at the beginning of the development process. After an application is developed and has been tested, it is possible to publish it in an online application store. The processes for doing this are varying, and depends on the platform.

In this chapter, iOS will be presented. For a presentation on Android, see Appendix B.

8.1 iOS

The information in this section is gathered from the books Programming iOS 4[65] and iOS 5 Programming Cookbook[64] as well as the iOS Developer Library[3].

iOS is Apple's mobile operating system, and runs on iPhone, iPod Touch and iPad. It's derived from Mac OS X, and is a Unix-like operating system.

Apple's IDE is called Xcode, and provides debugging and a simulator. The coding is carried out in Objective-C. It is possible for anyone to develop applications and run them on the simulator, but to run applications on an actual iOS device, an iOS Developer Account is needed. This costs 99$ a year. With a Developer Account, up 100 different devices can be used for testing applications.

[1]Integrated Development Environment

[2]Software Development Kit

[3]iOS Developer Library available at `http://developer.apple.com/library/ios/navigation/`

iOS supports a limited form of multitasking from iOS 4 onwards, which means that as a developer, you can let some applications run in the background. These background applications are limited to playing audible content, navigating, Voice over Internet Protocol or receiving regular updates from external accessories.

8.1.1 Objective-C

Objective-C is an object-oriented language layered upon C, which means that C code can be used in Objective-C applications. To invoke a method on an object, a message is sent to the object by using the Objective-C syntax with brackets:
`[object method:argument]`

As any object oriented language, Objective-C is modularize into classes. A basic example of a class header file can be seen in Listing 8.1.

Listing 8.1: Class interface

```
@interface class:superclass
- (return type) method:(argument type)argument;
+ (return type) method2;
@end
```

The class is declared between the `@interface` and `@end` compiler directives. Methods declared with a - sign are instance methods and can only be invoked on a particular instance of the class. Methods declared with a + sign are methods that can be called on the class itself, without any instance of it. Implementation of a class is done in a file with a .m extension between `@implementation` and `@end` compiler directives.

Properties

Objective-C allows for adding variables to classes which can be addressed via dot notation. Calling `class.property` does not really directly address the variable, but invokes getter and setter methods for the property. Properties are declared in the interface with the `@property` keyword, and in the implementation with `@synthesize`. `@synthesize` makes the compiler automatically generate getter and setters.

Properties can be defined with several attributes:

- `nonatomic` - For properties that are not meant to be accessed by several threads at the same time. Unless this attribute is specified, any property is accessed with locking to ensure that it is thread-safe.
- `strong` - This property will be valid until the end of its scope and then automatically released.
- `weak` - When the object this property points to gets deallocated, the property is set to `nil`.
- `unsafe_unretained` - Points one property variable to another, and if the other variable is released, this will point to a point in memory which might make the application crash if it is referenced.

- `getter=gname, setter=sname` - Can be used to give the getter and the setter methods of a property different names than the default names.
- `readonly, readwrite` - Readonly can be used to not create a setter for the propery, while readwrite is the default property.

ARC

Automatic Reference Counting (ARC) was introduced in Xcode 4.2[35]. ARC provides automatic memory management of Objective-C objects. Previously, all objects needed to be deallocated by the programmer. This is now taken care of by ARC, and calling dealloc on an object will result in a compiler error[41]. To make sure ARC is able to clean up correctly, all resources should be set to `nil` when a view is unloaded.

8.1.2 Application Life Cycle

The states an application may be in are shown in Figure 8.1. Any app that has not yet been launched or has been terminated, is in the not running state. When an app is being brought to the foreground or background, it transitions through the inactive state. In this state it is not receiving any events. In the active state, the application is in normal running mode and responds to events. If an application is getting suspended, or will run in the background, it will transition to the background state. It is also possible to launch an application directly into the background state. In the suspended state, the application is still in memory, but it is not executing any code. Suspended applications will be automatically terminated by the system[21].

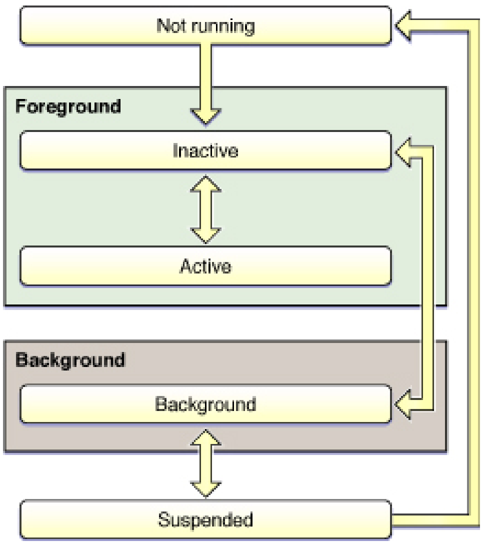


Figure 8.1: Life cycle states of an iOS app [26]

8.1.3 Intra-Application Communication

Protocols and Delegates

A protocol is a list of method declarations that can be implemented by any class[33]. The class declaring the protocol, can use this to invoke the methods in another class, adopting to the protocol. The class adopting the protocol is responsible for implementing the methods.

When a class is adopting a protocol, it is said to be a delegate of it. It must thus register itself as a delegate of the protocol-declaring class, which in turn can invoke methods by sending a message to its delegate(s). In this way, the class declaring the protocol won't need any knowledge of the delegate.

Declaring a protocol is done by using the `@protocol` compiler directive, and the protocol's methods can be marked as either optional or required by using the `@optional` or `@required` keywords. To adopt to a protocol, the adopting class must list the protocol in angle brackets after its superclass name:
`Class: Superclass ⟨Protocol⟩`

Notifications

`NSNotifications`[24] can be used to broadcast events with or without encapsulated information. Any object may register to the notifications and use it to take action. It is also possible for an object to register to notifications from outside its application, such as notifications for the music playing in the background etc.

It is not possible to guarantee the delivery time of a notification, delays from a few milliseconds up to seconds may occur. Notifications should thus not be used in time critical applications.

Passing Data Between Scenes

By using an object called `UIStoryboardSegue`[28], it is possible to pass data from one scene to another. These objects are created by the storyboard runtime and are used to prepare for transitions between view controllers. They also contain information about the view controllers. This information can be used to call methods on the view controller that is about to appear, and thereby passing data to it. When a transition is about to happen, the `prepareForSegue` method is called, and this method must be implemented to call methods on the destination view controller.

8.1.4 Application Files

When a new project is created in Xcode, several files are also automatically created and added to the project. These include the application delegate, the storyboard and one or more view controllers. These will be more thoroughly explained in this section.

Application Delegate

The application delegate implements the `UIApplicationDelegate` protocol[27] which provides information about important events in the application's life cycle. Important methods to implement are:

- `didFinishLaunchingWithOptions` - Tells when the application has launched.
- `applicationDidEnterBackground` - Tell that an application that supports background exectution will enter background. This method should release shared resources, save user data and stop updating of user interface.
- `applicationWillEnterForeground` Tells that the application is about to move from the background state to inactive state.
- `applicationDidBecomeActive` - Tells that the application has moved from the incative to the active state.
- `applicationWillResignActive` - Tells that the application will move from the active to the inactive state. This could be due to an incoming phone call, or when the application is quitted by the user.
- applicationWillTerminate - The application will terminate. This method should release all shared resources as well as saving user data etc. If this method is not returned before five seconds have passed, the process may be killed by the system.

View Controller

A view controller manages a single screen, and all its subviews [31]. It implements how the program should respond to user interaction. Important functions of the view controller to be overridden when subclassing it are presented below:

- `viewDidLoad` - Called when the view is loaded into memory. Typically the first time in the application it is shown on the screen.
- `viewDidUnload` - Called when the view has been released, and where any final cleanup should happen.
- `viewWillAppear` - Called every time the view is about to appear on the screen.
- `viewWillDisappear` - Called every time the view is about to disappear from the screen.

Storyboard

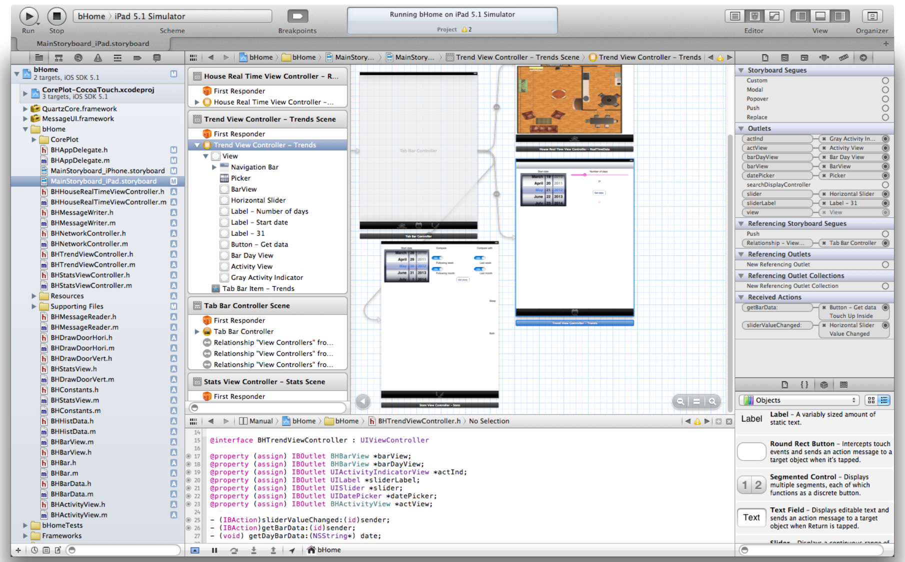
The storyboard lets the developer create the entire UI[4] of the application, as well as specify the connections between the different scenes. A scene is a single screen on the iOS device, and each scene is controlled by a view controller.

In the different scenes, it's possible to drag and drop different types of views, such as labels, buttons, images etc. to the screen. These can then be connected to

[4]User Interface

properties in the scene's view controller, which gives easy access to them programatically. It is also possible to connects events from e.g. buttons to methods that will be called when a user interacts with them.

A screenshot from Xcode with the storyboard builder can be seen in Figure 8.2. Here, a tabbed application with three scenes is shown. The transitions between the scenes is managed by a tab controller, and are marked with arrows. On the far right, all the outlets of a scene and the properties or methods they are connected to can be seen. The header file of the view controller controlling one of the screens and containing outlet properties is at the bottom.



```
@interface BHTrendViewController : UIViewController

@property (assign) IBOutlet BHBarView *barView;
@property (assign) IBOutlet BHBarView *barDayView;
@property (assign) IBOutlet UIActivityIndicatorView *actInd;
@property (assign) IBOutlet UILabel *sliderLabel;
@property (assign) IBOutlet UISlider *slider;
@property (assign) IBOutlet UIDatePicker *datePicker;
@property (assign) IBOutlet BHActivityView *actView;

- (IBAction)sliderValueChanged:(id)sender;
- (IBAction)getBarData:(id)sender;
- (void) getDayBarData:(NSString*) date;
```

Figure 8.2: Storyboard in Xcode

A storyboard is automatically created when creating a new project, with corresponding view controllers depending on what kind of template is used.

8.1.5 Bluetooth in iOS

Apple only provides use of Classic Bluetooth for developers via the Game Kit[22]. This is a framework ment to create social games which can be played by two players over a Bluetooth or WiFi connection. It can also be used for other applications to exchange data, but can only be used to connect between iOS devices. There are Bluetooth frameworks available online, but these require that the iOS device is jailbroken[5], and the application could then not be published in the Apple App

[5]Jailbreaking an iOS device means to gain root access to the operating system, thereby removing the limitations set by Apple and being able to install applications not authorized by Apple[43]

Store.

Newer versions of iOS products (such as the new iPad and iPhone 4s) supports BLE as well, which can be used in master-mode to connect to peripherals[17], such as pulse monitors, without any restriction.

8.1.6 Health in iOS

There are several applications related to health and medicine available for iOS. This includes first aid applications and symptom checkers as well as sleep cycle analyzers that wake the user up at the most appropriate time in their sleep. These are all available from the Health and Fitness section in Apple's AppStore.

Part III

Implementation

Overview of the total implemented system

The figure below shows an overview of the structure and data flow in the system. The leftmost box shows that in the house used as test-environment, there is a processor board running Linux which works as a concentrator for various wireless sensor data. This data is stored in a database and can be processed locally at the

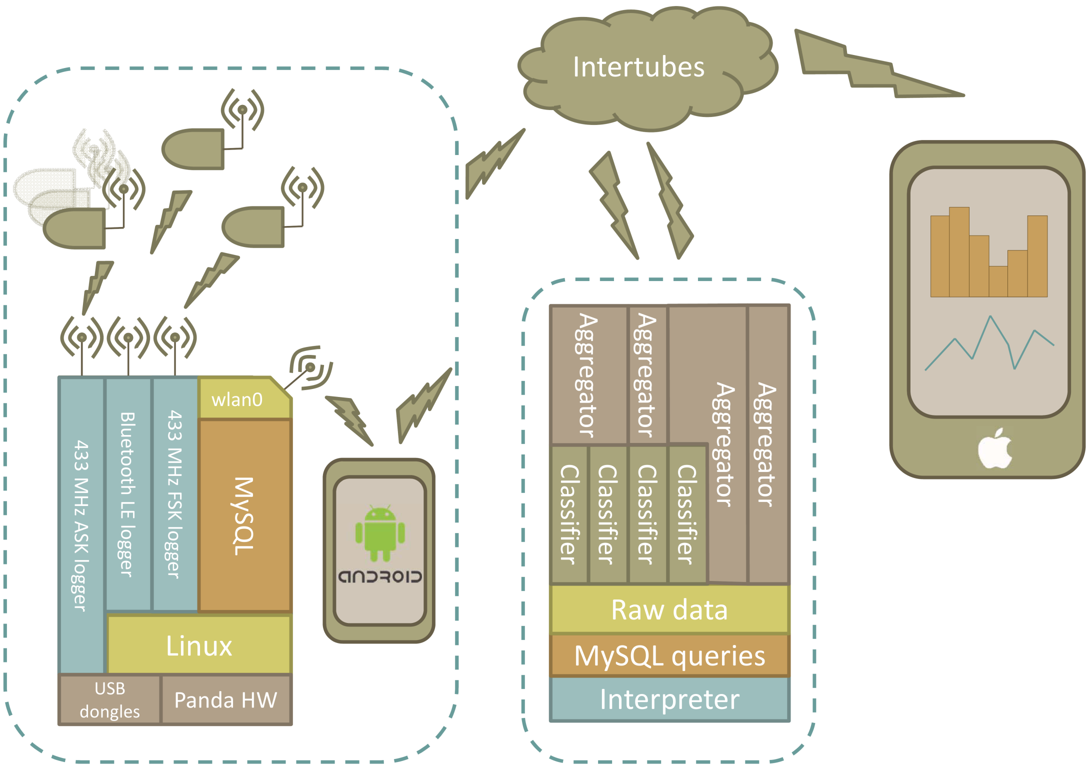


Figure 8.3: General structure of data flow from sensor nodes through a concentrator, through an internet gateway, through an interpreting mechanism to a presentation platform.

user's home, or can be sent via a gateway to an interpreting process running on a remote machine. These topics will be covered in Chapters 9 through 13.

The middle box gives a taste of how the further sensor interpretation is performed, taking advantage of concepts discussed in the background chapters. This interpretation framework is presented in Chapter 14.

Chapter 15, the last chapter of the implementation, will describe the presentation platform for the data, an Apple iOS application named bHome.

9

Sensor Types and Wireless Stack Selection

In a Wireless Sensor Network(WSN), sensors and wireless network are needed. The selection of sensor types and wireless solution for a node are however somewhat intertwined. Ideally, components like the antenna, radio, processor and sensor should be independently selectable. But using available off-the-shelf products while maintaining interoperability often means compromising either on the radio stack quality or on sensor type diversity within a given stack. Three possible roads to an implemented WSN can be classified:

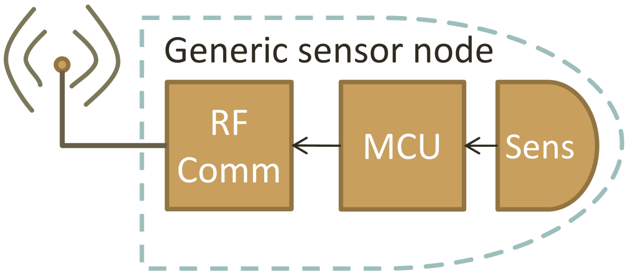


Figure 9.1: Generic Wireless Sensor Node

Fully Customized First of all it is eminently possible to make a fully customized node where the entire chain of components seen in fig 9.1 are chosen from a catalog of components and assembled.

Existing WSN Platform Secondly, one can buy a wireless board where host MCU, wireless communication protocol stack and antenna are alerady provided and tuned, then simply add various sensors.

Existing System Finally it is possible to use complete wireless sensor node systems and redirect the sensorial output into an application.

What we must consider then, are the following questions:

- What types of sensors give relevant context information
- What combinations of the above solutions exist and how do they compare in terms of:
  - Price
  - Reliability
  - Integration cost

9.1 Sensor Type Selection

Different sensor types are discussed in Chapter 5 on page 24, and the sensor types required for the implementation are given from what we would like to know about the subject's state, namely:

- Location — Because stays in different rooms have different significance.
- Activity level — To perhaps correlate with overall health.
- Activity — What the subject is doing and interacts with.

9.1.1 Location

For location, two main categories are defined in Chapter 5: Stationary and ambulant/body-worn.

Stationary

With regards to robustness, it is obvious that stationary systems are easier to implement and maintain, and has the advantage that it's impossible to forget wearing them. Stationary sensors are also the most common type commercially off the shelf, as they are widely used in alarm systems et cetera.

Body-Worn

Aside from the fact that body-worn sensors are more obtrusive and easy to forget, as will be discussed in the results chapter, they offer a couple of significant advantages:

- Higher spatial resolution — In that while the calculated location may be a bit fuzzy, it is distributed around a point. On the other hand, infrared sensors can only define zones equal to their field of view with uniform probability distribution.
- Scalability — If more than one person inhabits a domicile, it is impossible to know which person triggered an infrared-sensor event, but an arbitrary number of active tags may co-exist in close proximity.

9.1.2 Activity Level

The same distinction can be made between stationary and ambulant systems for detecting activity level, with the same reservations. Activity level can further be divided into locomotion or activity in-place.

Stationary

Stationary systems typically have low – e.g. infra-red – and/or discontinuous – e.g. pressure sensors and switches – spatial sensing capability. As such they are not well suited to differentiate between transitive and intransitive activity except for zone crossings or switch activation.

Body-Worn

With body-worn systems, one can either differentiate the location with respect to time to establish movement, or one may use an accelerometer or a vibration-sensing ball-switch to realize a pedometer. Both location and activity sensing can be implemented on the same device, thus offering superior data-quality compared with stationary systems.

9.1.3 Environment Interaction/Activity

Activity in this regard corresponds to interpreted context as described in Chapter 6, and may be derived from a variety of sources. Aspects of different sensors is described in Chapter 5. For an implementation it could be interesting to look at activities of daily life, such as:

- Refrigerator use.
- Kitchen appliance use.
- Washing machine use.
- Sitting or lying down.

It could be helpful to instrument all appliances to monitor their use, and for instance add pressure mats to furniture, but cost permits only a selection of activites to be monitored in this project.

9.2 Wireless Properties of the Stacks

From the discussion on radio- and protocol properties in chapter 4 it is clear that from a robustness and scalability perspective, two-way solutions with a proper media access layer are superior to a naïve transmitter-only solution such as one from Nexa.

Secondly it is clear that from a topology perspective, Bluetooth is less well suited for a wireless sensor network than are for instance Z-wave or Zigbee.

9.2.1 Diversity and Cost

There are currently no Bluetooth or Bluetooth Low Energy sensors usable for home-automation. Below is a rough price comparison between systems:

Component	Zigbee[a]	Z-Wave[b]	Nexa[c]	Efergy[d]
Motion sensor	25£	39€	229 NOK	
Magnet switch	25£	39€	161 NOK	
Powermeter	29£	49€		699 NOK
Lightswitch	20£/ 80$	40€	199 NOK	
Relay	25£	35€	70 NOK	
Interface dongle	Supplied	54€	450 NOK	

Table 9.1: Sample component costs from various vendors

With a setup with 9 motion sensors, 16 magnet switches, 4 light switches with relays and one powermeter, the cost including import taxes adds up to:

- ZigBee: 9,830 NOK
- Z-Wave: 12,600 NOK + 18,000 NOK for API documentation
- Nexa+efergy: 6,400 NOK

9.2.2 Availability and Ease of Integration

ZigBee

The ZigBee market is not as mature as for instance Z-Wave, and the components and vendors we found were few, with unique centrally managed solutions, developed separately by each vendor.

The ZigBee Home Automation standard states that certified products are interoperable, but we have found no instances in literature or on hobby forums where anyone have intercepted data from or combined sensors from different vendors.

In addition, most vendors have British or American sockets, including the vendor used in the example above which had the widest selection of products.

Z-Wave

There is a plethora of vendors using this protocol, including many with European wall plugs. There is also a hobbyist community dedicated to this technology, but the protocol is proprietary, the official software runs only on Windows and open-source attempts to reverse engineer the protocol are at best imperfect.

[a] http://www.alertme.com
[b] store.zwaveeurope.com
[c] clasohlson.no, elektroimportoren.no
[d] clasohlson.no

The computer-interface is simply serial communication with special packets. These packets are easily created with access to the API documentation and a developer kit which costs 3,000 USD.

Nexa/Efergy

While the simplicity of these protocols have a decidedly negative impact on reliability and scalability, it opens the door for a wide hobbyist community.

Open-source projects such as Teldus TellStick have implemented parsing of many commercial systems using OOK modulation, including Nexa, and private enterpreneurs like Jon Petter Skagmo offer similar bundles of radio receiver dongle and decoding software.

9.3 Final System

Price, sensor diversity, ease of integration and availabilty all considered, the system from Nexa is the preferred choice for stationary nodes in a pilot-study such as this thesis. Passive infrared activity sensors where bought at Clas Ohlson, and magnet switches for doors etc were bought on elektroimportoren.no.

For monitoring power consumption and detecting appliance use, an Efergy e2 powermeter was also bought at Clas Ohlson.

To look at the performance of body-worn sensors we have used a demonstration kit from Texas Instruments which uses Bluetooth Low Energy, runs on a coin-cell battery, has an on-board accelerometer and comes with a transceiver USB-dongle.

For reception of Nexa sensor data, we looked at TellStick[1], but they were sold out at the time of implementation, so the MultiTRX USB-dongle produced by Jon Petter Skagmo was used for data reception. MultiTRX will be described further in Chapter 13 on page 71.

The actual test-setup will be described in Chapter 11 on page 59, while the price of the entire system can be seen in Table C.1 in the Appendix.

9.3.1 Discussion

As already mentioned, most wireless solutions outperform the ones used by Nexa and Efergy. However, pricing weighed heavily in our decision. It was also important to get a system working within the time constraints of the thesis, and knowing that at least Nexa is widely used in hobby projects gave us a measure of confidence. It was incorrectly assumed that Efergy used the same modulation and frequency as Nexa, and this will be discussed in Chapter 13.

Because "System Nexa" devices don't have any media access control, we decided against using wireless power switches from this system for monitoring e.g. use of lighting because of the almost certainty of collision. Even though they would give interesting information on the subject's behaviour, the interference between sensor

[1]www.telldus.se, an USB interface to Nexa and other OOK-based systems

transmissions and user appliance control would at some point frustrate the user and hinder daily activites.

10 KeyFob "Smart Sensor"

To get a better understanding for how body-worn sensors perform compared to stationary sensors and to attempt an implementation of a smart sensor, a combined orientation sensor and pedometer was realized. A Texas Instrument CC2540 KeyFob evaluation board was used, as the previously developed sensor platform mentioned in the introduction was unfortunately lost in an administrative oversight after evaluation.

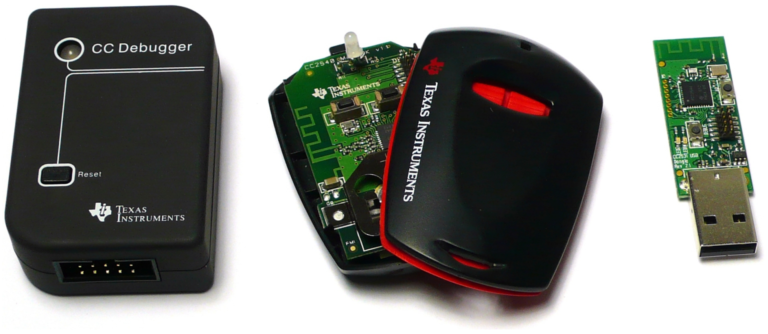


Figure 10.1: Texas Instruments' Bluetooth LE evaluation kit. Displayed here: Debugger, KeyFob and USB transceiver dongle. Picture from ti.com.

The details of the Bluetooth LE stack, device profiles and programming with the TI framework are covered in Section 4.5 and in Normann/Skjønsfjell[66]. This chapter will describe:

- TI's example project "KeyfobDemo" which this sensor is based on
- Modifications and additions to the TI's project
- Developed pedometer GATT profiles and characteristic values
- Step detection algorithm

10.1 KeyfobDemo Project

To the outside, this application exposes GATT profiles which a Bluetooth LE master device can use to read the accelerometer status and turn the accelerometer on and off:

UUID[a]	Description	Type
0xFFA1	Enable/disable Accelerometer	Read/write
0xFFA3	X-axis value	Notification
0xFFA4	Y-axis value	Notification
0xFFA5	Z-axis value	Notification

Table 10.1: KeyfobDemo GATT profile parameters relevant to accelerometer

After enabling the accelerometer, the master device must then register for notifications. When this is done, the KeyFob will transmit accelerometer values every connection event[1] if the values have changed by more than a defined threshold.

Internally in `KeyfobDemo`, the accelerometer is polled over SPI every 50ms by an OSAL[2] timer callback. The timer event callback calls a function `accelRead` that reads the accelerometer's registers and decides whether or not to post notifications to the GATT layer in the framework.

10.2 Modifications and Additions

A new GATT profile was implemented (`movement.h`, `movement.c` on the CD) to handle input and output between Bluetooth master devices and our smart sensor implementation. Table 10.2 shows the GATT UUIDs which act as gateways to the program.

The accelerometer GATT profile was changed to let the X-, Y- and Z-axis values be of Readable- instead of Notification type. In addition, `accelRead` was changed to update the characteristic values associated with each UUID instead of sending notifications.

At the end of `accelRead`, a step detection algorithm was added which then is run every time new data is received from the accelerometer.

10.2.1 “Movement” GATT Profile

Table 10.2 on the facing page shows the values the `Movement` profile exposes to a host device. The “acceleration magnitude” thresholds affect the performance of the step detection algorithm, while the “Steps taken” value is incremented on each detected step and reset when read by a master.

[a]Universally Unique Identifier

[1]Connetion events take place every 100ms by default even if no data is scheduled for transfer. This is used by some applications for proximity detection.

[2]Operating System Abstaction Layer

UUID	Description	Type
0xFFF1	Step magnitude low threshold	R/W
0xFFF2	Step magnitude high threshold	R/W
0xFFF3	Steps taken	R
0xFFF4	Acceleration magnitude	R

Table 10.2: Values available in the developed Movement GATT profile

"Acceleration magnitude" is included for debugging purposes, as the length

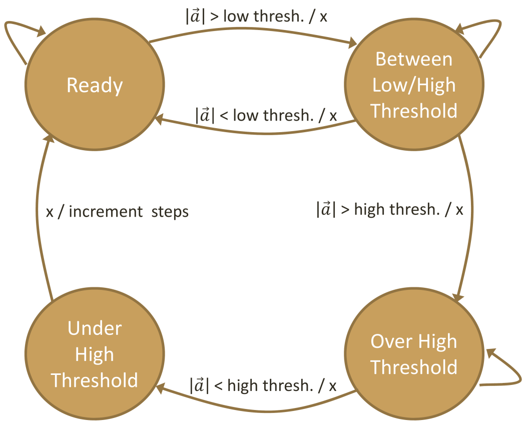
$$|\vec{a}| = \sqrt{\mathrm{acc}_x{}^2 + \mathrm{acc}_y{}^2 + \mathrm{acc}_z{}^2} \tag{10.1}$$

should according to the accelerometer datasheet be approximately 54, corresponding to 1 g when the KeyFob is at rest.

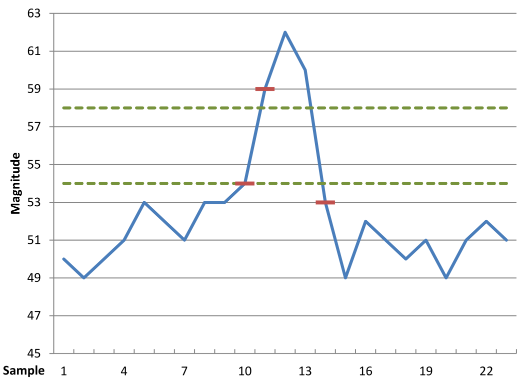
10.2.2 Movement Detection

In the specialization project[66] every accelerometer sample was sent to a host device via Bluetooth LE. Hatlevoll[46] showed in his master's thesis that radio transmissions take a big toll on battery life.

To mediate this and to conform to the "Smart Sensor" ideal briefly described in Chapter 6 – sometimes called *In-network processing* – a peak detector was implemented to count the number of sharp movements experienced by the sensor.



(a) Movement detection states



(b) Illustration of magnitude over time

Figure 10.2: Design of step detection

This approach has several advantages:

- Data is not lost if a connection is broken, because the receiving party is not dependent on a continuous stream of samples.

- As a corollary, recipients can poll data when they are ready for it.
- Emulating the cue-principle in chapter 6, much complexity is transparent to the recipient.
- Battery life is improved because processor time is much cheaper than transmission time.

Figure 10.2a on the previous page shows the structure of the peak detector. A state machine structure was used to be able to emit only one signal every time the magnitude of the acceleration had risen above and fallen below some set thresholds.

Figure 10.2b on the preceding page shows an illustration of the magnitude over time crossing the thresholds, the first sample causing a state change marked in red.

Fewer states would also have been a workable solution, but we need to at least know if we are on the way up or the way down, and more states have the effect of working like a low pass filter that helps ensure us that the signal rises and falls "intentionally", if not entirely monotonically.

Peaks are only calculated on the projection of $\vec{a}$ onto a calculated $\vec{g}^b$ representing the assumed up-direction of the body-frame with respect to gravity. This is done to ignore sideways jerks. The formula used to get the acceleration only in the up/down direction with respect to the gravity field is then:

$$a_{\hat{g}} = \frac{\vec{a}\vec{\hat{g}}}{\vec{\hat{g}}\vec{\hat{g}}} \tag{10.2}$$

where $\vec{\hat{g}}$ is made by averaging the received samples over some seconds, and also give us a stable measure of the subject's orientation in space.

10.3 Discussion

The intention with this node is, as mentioned, primarily testing the actual use of a body-worn sensor on a subject. Some other measure of activity than steps, such as simply summing the difference between instantaneous and low-passed acceleration, could have been better overall. Step detection was chosen as a measure of activity because it is something everyone is familiar with, and orientation sensing comes naturally from the use of an accelerometer.

As we had only an accelerometer available on the board and not, for instance, a mechanical vibration sensor, we had to power the accelerometer continuously for accurate data. Low battery life considerably limits the usefulness of this particular sensor, but nevertheless gives insight into the type and quality of data available.

11 Test-Case Setup

11.1 Test Subject

The system was installed at the house of the test subject. The test subject is an 86 year old male, living alone in his own house. The test subject is in good health, and is able to do everyday chores by himself. He is the father of our supervisor, and volunteered to let his house be used for this pilot-study.

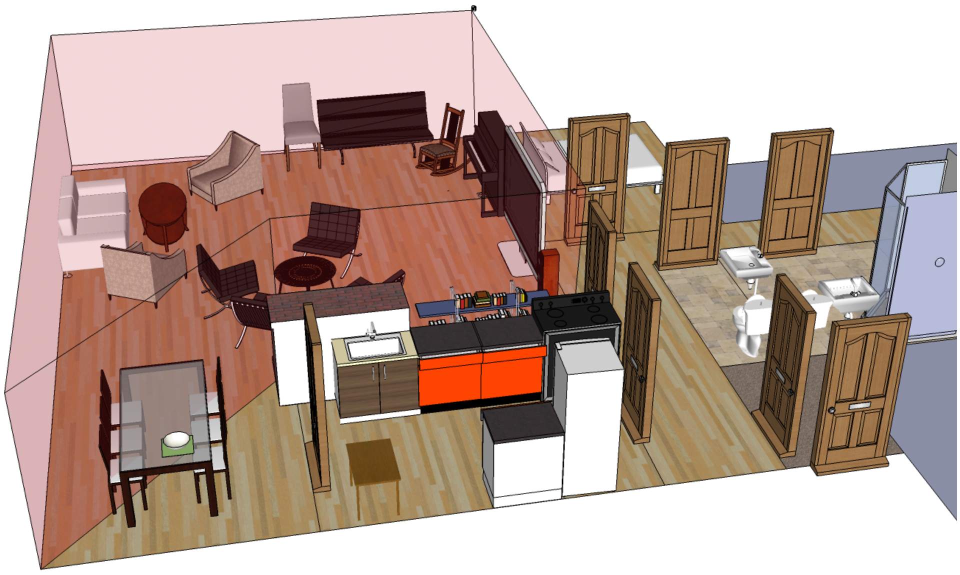


Figure 11.1: 3D model of test subject's house, showing the field of view of one of the livingroom sensors

11.1.1 Description of the House

A 3D model of the house was created using Google Sketchup and can be seen in Figure 11.1. This figure also illustrates the coverage area of a motion detector. A not in scale floor plan was also created, using an online floor plan tool[1] and can be seen in Figure 11.2. Here, all the main rooms of the house are numbered according to their `zone_id` in the database, and Table 11.1 sums up the names of the rooms. The rooms not numbered are extra bedrooms and storage, which are not in frequent use.

#	**Room**
1	Hallway
2	Living room
3	Bathroom
4	Small toilet
5	Bedroom
6	Washing room
7	Kitchen
8	Airlock
9	Outside
10	Freezer room
11	Basement staircase

Table 11.1: Room description

11.2 Placement of Sensors

The sensors chosen to be placed in the house were motion detectors and magnet switches from System Nexa, as well as an electricity monitor from Efergy. To investigate where the different sensors should be placed, the 3D model of the house was examined and the following considerations were made:

- Decide what doors and rooms should be monitored.
- Find a suitable place for the concentrator to be unobtrusive, but in range of all sensors.
- Place sensors to avoid collision between sensor packets.

It was decided upon placing magnet switches on all the most commonly used doors. This excludes the doors to the unused bedrooms and storage room. Motion detectors were chosen to be placed in the same rooms as the magnet switches, apart from in the basement staircase, airlock, washing room and freezer room as well as

[1]Autodesk Homestyler, can be found at`http://www.homestyler.com/designer`

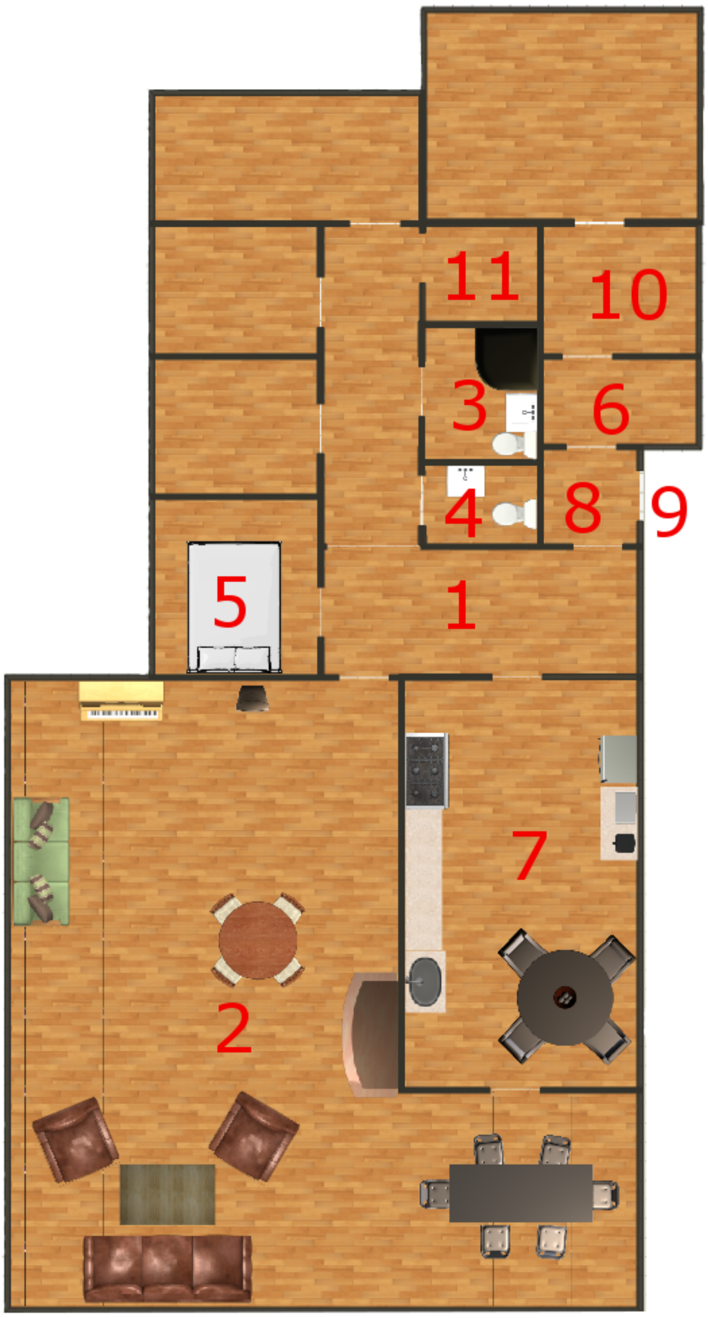


Figure 11.2: Floor plan of test subject's house

outside. These rooms were omitted as it was considered that they were either so rarely used that the cost/use ratio was not defendable, or that the normal stays in these rooms would be of such transient nature that the transmission from other sensors would be compromised.

Placement of the motion detectors also try to make sure that their detection area does not overlap with another motion detector. This is done to reduce the amount of interfence from the sensors. But as the motion detectors always send a "False" message a few seconds after detecting action, some interference is inevitable. An exception has been done for the living room, which includes two sensors, as the shape of the room makes it hard for one sensor to cover the entire zone.

The range of the signals sent from the sensors where tested in the hallway at the university before placement, and it was discovered that the range was approximately 16 meters. Packets were still received from a larger distance, but with a very limited and undeterministic amount of packets. From this, it was decided that the concentrator should be placed as close to the center of the house as possible.

A figure of the house marked with sensor placement can be seen in Fig 11.2. Note that the upper part of the house where no sensors are placed have been removed to scale the image better. The red lines are doors equipped with magnet switches and the red dots are motion detectors with their approximate direction. The pink dot is the Efergy energy monitor.

The blue dot is the concentrator, which is placed behind the television in the living room. This location was chosen as it is close to the center of the house in addition to be hidden from view.

11.2.1 Motion Detectors

Eight motion detectors were placed in the house. They were mounted on the wall either by screwing them directly onto the surface, or by screwing the detector onto a piece of wood and glue this on by using double-sided adhesive tape. The motion detector mounted in the bathroom, together with its approximate detection area is shown in Figure 11.4

During the installation, two motion detectors were placed in the bathroom. One covering the toilet, and one covering the shower. After initial testing, it became evident that the sensor covering the shower was affected by the opening of the door, and that it had problems detecting movement inside the shower. This could be due to the fact that the shower is covered inn glass, or that it was placed too high. However, it was decided to remove this sensor, as two of the other sensor's bought was not functioning, so this replaced one of them.

11.2.2 Magnet Switches

13 magnet switches were mounted on the doors by using the double-sided adhesive tape that came with the sensor. A magnet switch placed on the bedroom door can be seen in Figure 11.5.

Magnet switches are placed on most doors, as well as on the refrigerator door and washing machine lid. It was also made an effort to place a magnet switch on

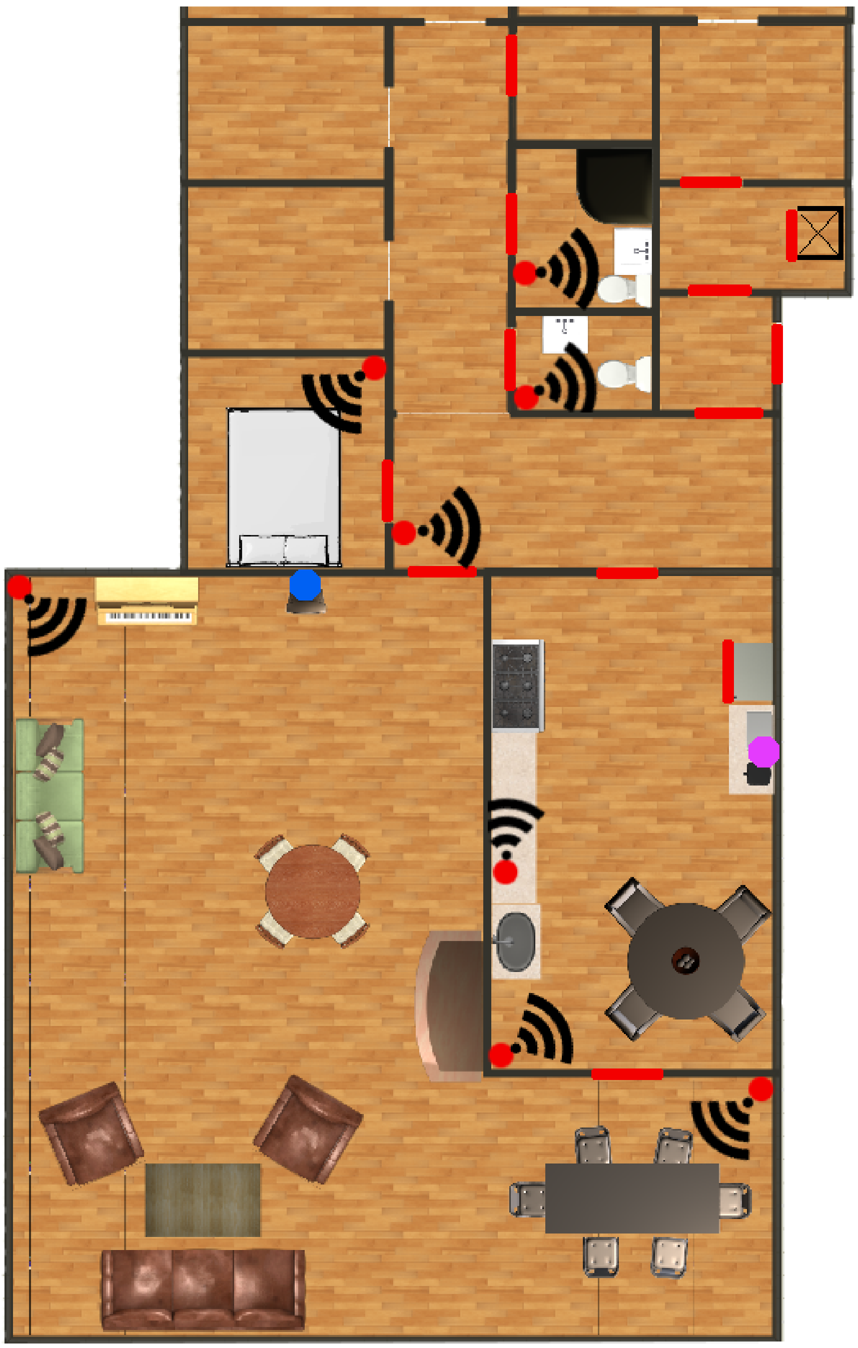
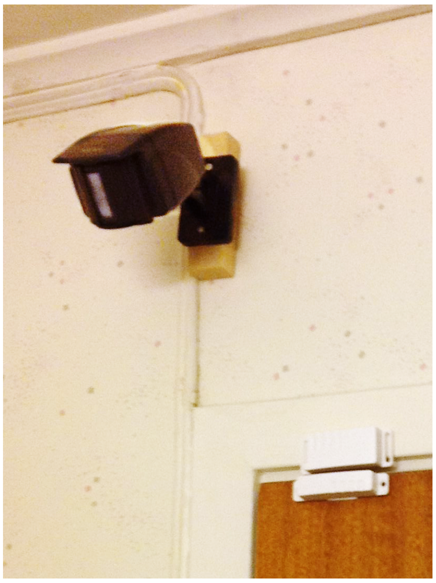


Figure 11.3: Floor plan of house with sensors


(a) Motion detector on wall

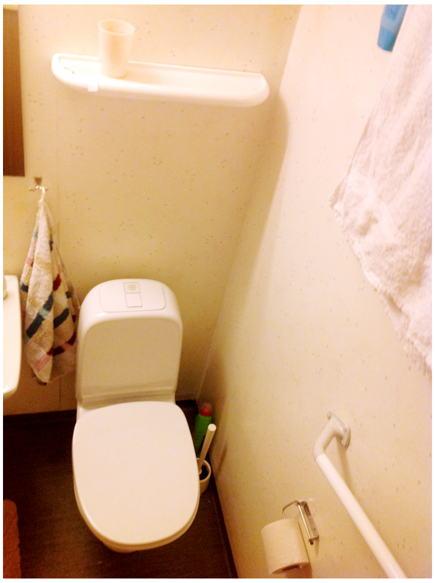

(b) Coverage area

Figure 11.4: Motion detector in bathroom

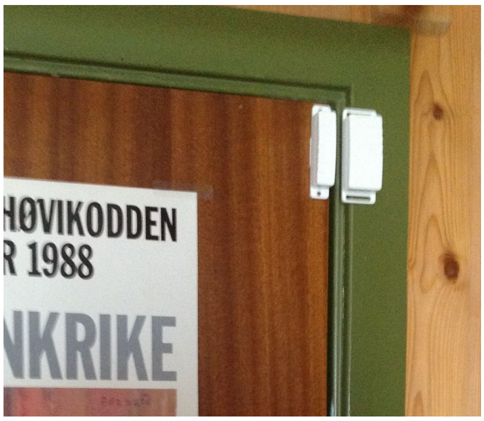


Figure 11.5: Magnet switch on bedroom door

the the microwave oven lid, but this did not attach and is therefore not included in the analysis.

11.2.3 Efergy

The Efergy energy monitor was attached to the extension cord the microwave oven, coffee boiler and water heater was plugged into. The attachment of the transformer clamp can be seen in Figure 11.2.3. The plastic isolation of the cord is opened, and the clamp if fastened around one of the conductors inside. This is done because the transformer measures the current induced by the magnetic field, and if the clamp is fastened around the entire cable, the opposite polarities of the conductors inside cancel out the magnetic field, leaving no magnetic field to measure.

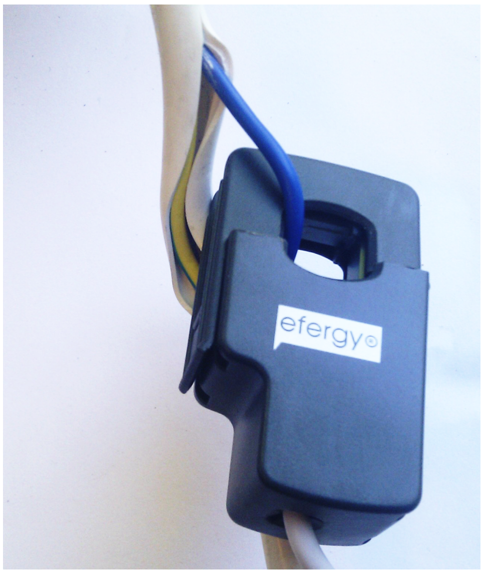


Figure 11.6: Efergy transformer clamp attached to extension cord

11.3 Discussion

To get a more general coverage of the house, more sensors could be beneficial. But more sensors might leed to more packets being dropped and thereby resulting in a less accurate set of data. It was therefore regarded as better to get more accurate readings from fewer sensors and work with this data.

12 Local Concentrator and Communication Node

In a deployed sensor system, some device needs to concentrate and store the various sensor data. This could essentially be any device with an interface to a storage medium and wireless communication. For wide adoptation, it is not cost-effective to use a fully fledged personal computer, and developing a custom device e.g. using an 8-bit architecture would take time and be error prone, even though many OS frameworks are available for free.

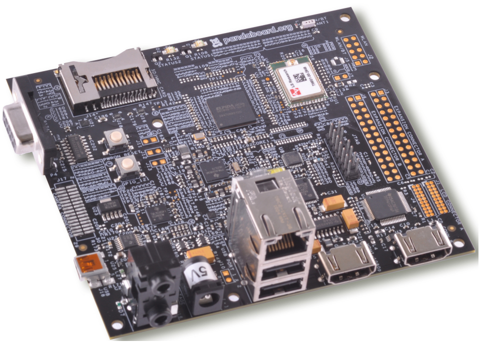


Figure 12.1: PandaBoard: 11cm x 10cm includes dual-core 1 GHz ARM Cortex A9, Ethernet, USB, 1GB RAM, SD-card slot, 1080p HDMI out, runs OS from SD-card.

Luckily, the last couple of years have seen much development on the mobile device front, with some of the effort spilling over into the embedded computing field. Two examples of off the shelf Linux-capable devices are Raspberry Pi[1]; a 25$ device with HDMI, Ethernet, USB, 256 MB RAM and a Broadcom SoC based on ARM, and PandaBoard[2]; a 175$ device with more RAM, more connectors, a wireless IEEE 802.11 interface and a dual-core ARM SoC from Texas Instruments

[1] http://www.raspberrypi.org
[2] http://pandaboard.org

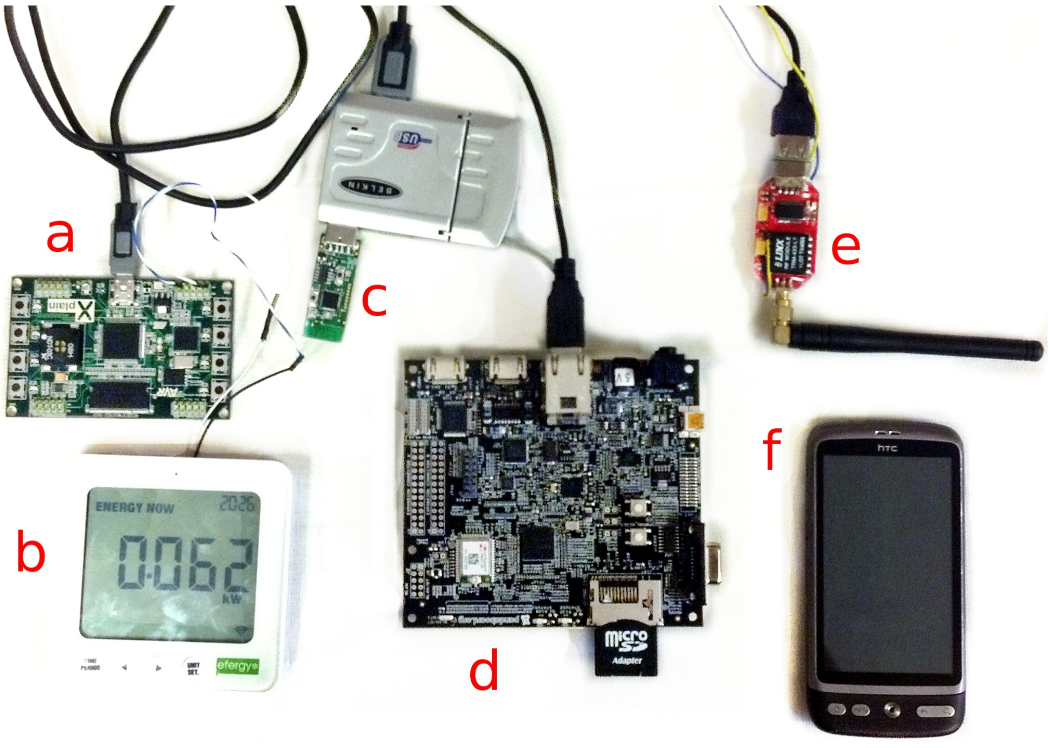


Figure 12.2: The concentrator as deployed. (a) Xplain board connected to a test pad on the (b) Efergy display module, (c) BLE dongle, (d) PandaBoard, (e) MultiTRX transceiver, (f) Mobile Wifi-Hotspot

originally developed for Android tablets.

A PandaBoard was made available to us by the Department of Engineering Cybernetics. During development, a serial cable was attached to the D-sub connector which offers terminal output during and after boot of the device. A monitor was connected via DVI, and a USB keyboard and mouse were also attached. For network access, an TP CAT5-cable was inserted until the wireless interface was configured and working.

During deployment, only the receptor dongles were attached to the board and communication was done via Wi-Fi to a portable gateway. The deployed setup can be seen in Figure 12.2.

12.1 Getting Started

There are guides for getting started on the PandaBoard website which describe how a kernel image is written to a SD-card starting at a known offset. When inserted in the PandaBoard, the bootloader starts executing from there.

12.1.1 Linux Distributions

The kernel image may be any executable code, but is usually a Linux kernel. Many Linux distributions have been ported to the ARM architecture, including Angstrom, Ubuntu and Android. Angstrom is a very minimal distribution with a following in the scientific community, whereas Ubuntu can be configured as anything from a bare-bones server to a desktop workstation.

Both Angstrom and Ubuntu were tested, but at the time of development, Ubuntu had the widest repository of software packages and is currently the Linux distribution with the biggest user base and support forums. Ubuntu was therefore used as a base for our concentrator.

Canonical, the maintainers of Ubuntu, have made an OMAP[3]-specific version of Ubuntu available on their website, but it was discovered that it did not work correctly with some of the modules on the PandaBoard, most notably the Wi-Fi and Bluetooth module. A developer called Robert C. Nelson[4] maintains a fork of the Ubuntu kernel specifically targeting the PandaBoard.

Using his patches we were able to take advantage of the WiFi-module, and through collaboration with us he was able to implement a patch to get the Bluetooth module working. It turned out, however, that the PandaBoard did not support Bluetooth Low Energy as was initially assumed. For BLE support, a newer version of the board called PandaBoard ES is available.

12.1.2 Software Packages

On top of the base installation, some additional packages were installed from the Ubuntu Universe repository, most notably:

- `openssh` – Secure Shell, encrypted terminal server and client that communicates over TCP/IP, allowing remote access to the device.
- `python` with third party serial and mysql packages for logger execution.
- `samba` – Windows file-sharing service, allowing drag and drop file exchange with our development workstations.
- `mysqld` – MySQL server deamon, used for database access.
- `ntpd` – Network Time Protocol daemon, a service which synchronizes the device's real-time clock with official time servers.
- `autossh` – A software package designed to keep ssh-connections alive, and reestablish them if necessary.

12.2 Concentration and Storage

Each of the three different wireless dongles (MultiTRX, Xplain-Efergy bridge and TI Bluetooth LE) appear as character block device nodes under `/dev/` in the file system. As will be described later, each of these are accessed separately by

[3]TI's brand of mobile ARM SoCs

[4]https://github.com/RobertCNelson/stable-kernel

individual Python programs that parse the serial data and store the output in the MySQL server via local TCP/IP connections.

12.3 Communication

During development, the PandaBoard was connected to the internet via an ethernet drop point at NTNU, and communication with the device was done via SSH, SCP[5] and Microsoft Windows file sharing.

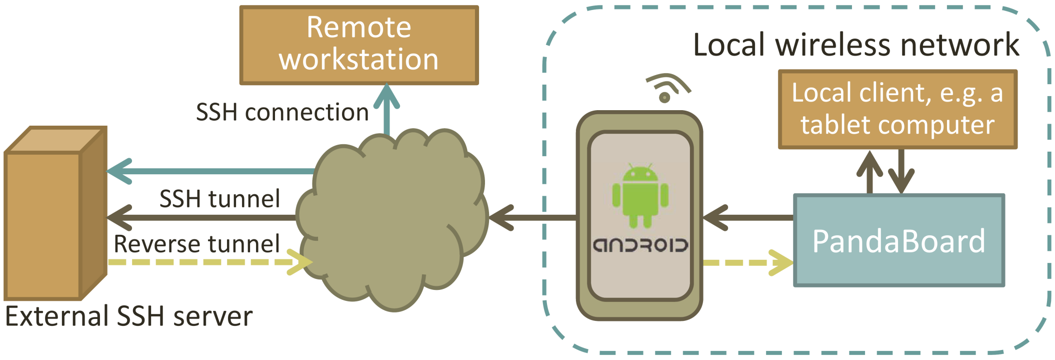


Figure 12.3: Connecting to the PandaBoard externally. Android Wi-Fi hotspot only allows outgoing connections, and an external server acting as bridge must be used.

However, our test-subject did not have internet service in his house, so during deployment a mobile 3G-device running Android 2.3 was used as a Wi-Fi hotspot. Figure 12.3 shows the local network, and how connections can be made through the firewall in the mobile hotspot.

The main obstacle in connecting remotely to the PandaBoard is that the hotspot service on Android doesn't allow incoming connections to clients from the internet. Instead, a reverse tunneling approach must be used where the device on the local network connects to a predetermined host and creates a TCP socket on the host that allows reverse connection.

```
ssh -f -N -R 19022:localhost:22 tunnel@imbesil.ed.ntnu.no
```

Shell Code 12.1: Arguments force background mode and creates a reverse tunnel socket on the remote host.

Shell Code 12.1 shows the command that must be executed on the PandaBoard to establish a reverse tunnel. The arguments are `<remote port>:<local endpoint:port>` and `<remote user>@<remote host>`, meaning that `localhost:22` on the PandaBoard appears as `localhost:19022` on the remote machine. The reverse connection is routed through the connection initiated by the PandaBoard, indicated by dotted lines in Figure 12.3.

[5]Secure Copy, an encrypted file transfer protocol

This has two implications: A host machine with a fixed address must be available, and most importantly, the PandaBoard must autonomously connect and if necessary reconnect to the remote host.

12.3.1 Maintaining a Persistent Connection

Several layers of the communication stack must be configured on the PandaBoard for the WiFi to remain connected. Two separate mechanisms were used to ensure a connection.

On the lowest level, a shell script was made which runs at boot-time and at regular intervals to set up the `wlan0` wireless base interface, configure and run the `wpa_supplicant` service which is needed for encrypted Wireless LAN connections. Finally it establishes a reverse tunnel as needed. For redundance, the `autossh` package, which automagically monitors an established tunnel was used on a secondary socket.

For security, outside network access was made by a designated user, `tunnel`, with restricted access privileges, relying on the operating system to prevent escalation of privilege attacks.

13 Sensor Data Reception

As discussed in Chapter 9, a PIR sensor of the Nexa code-wheel type, a magnet switch of the Nexa self-learning type, an Efergy clamp-type amperemeter and a Bluetooth LE "KeyFob" demo-kit from Texas was chosen for instrumenting the home in our test-case.

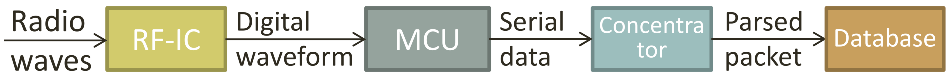


Figure 13.1: General signal path for the intercepted OOK and FSK sensor node data

In this chapter we will consider the reception, packet parsing and storage for these four sensor types, along with their specific software and firmware implementations. Figure 13.1 shows the general structure used in each case.

13.1 Nexa PIR and Magnet Using MultiTRX

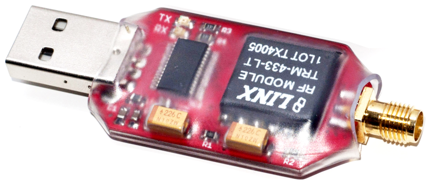
Interception of packets from the Passive Infra-Red (PIR) motion detector and wireless magnet switch is accomplished using an USB-dongle called MultiTRX, produced and sold by Jon Petter Skagmo, currently a Master's student in Electronics at NTNU.

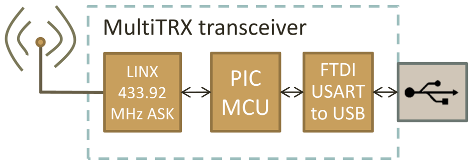
This device corresponds to the first two blocks in Figure 13.1.

13.1.1 MultiTRX

An overview of the device is shown below in Figure 13.2. The RF-IC performs an envelope detection similar to Figure 3.3a on page 12 and outputs the resulting waveform and an analog voltage representing the Received Signal Strength Indicator (RSSI) to the PIC microcontroller on the MultiTRX board.

The firmware already includes parsing for Nexa self-learning code used by the chosen magnet switch, along with several other protocols.


(a) MultiTRX dongle



(b) Simplified diagram

Figure 13.2: MultiTRX wireless 433.92 MHz OOK Tranceiver

The firmware-code is proprietary, but was graciously made available to us for this thesis. Programming is done in C, using MPLAB IDE which is freely available from Microchip Technology Inc.

Aside from device, the firmware consists of a loop which monitors pin change from the Linx chip seen in Figure 13.2b and sends the duration of the previous high or low period to all the included parser routines, which in turn call a serial transmit function if a packet has been correctly received.

13.1.2 PIR

A parser for this type of device was not included with MultiTRX and had to be developed. The PIR sensor uses two user-changeable code-wheels each with 16 values to determine the identification the sensor will send out.

Development consisted of:

1. Adding continuous RSSI readings during the program loop.
2. Sending the RSSI value as an argument to the parsing routines.
3. Making a skeletal parsing routine that sent every high/low duration with a high RSSI value (to exclude noise) serially to a Python program.
4. Visually identifying the waveform patterns.
5. Sketching and testing waverform parsing in Python.
6. Creating a firmware parser for the protocol.

Waveform

Looking at the code for other protocols supported by MultiTRX and observing the captured waveform, it was likely that this one used two periods for each bit, the periods differentiated by having a low or a high duty-cycle.

Figure 13.3 on the next page shows the identified bit-patterns sent by the PIR sensor. Figure 13.4 on the facing page shows how a byte-nibble is constructed from the bits.

Each packet is separated by a delay much larger than the bit-lengths. When all bits from a packet are combined, they form a message of 12 bits, as seen in figure 13.6 on page 74.

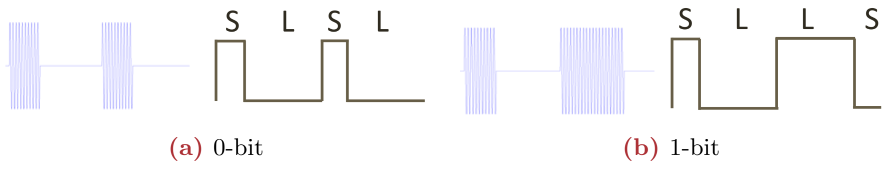


Figure 13.3: Nexa Code Wheel bit-patterns. RF-signal simulated with Matlab. Value is arbitrarily assigned.

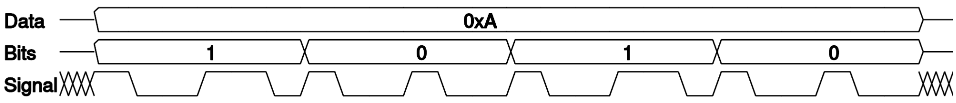


Figure 13.4: A bitstream translated to hexadecimal 0xA.

Parsing

The bit- and packet parsing is implemented in C as shown in figure 13.5.

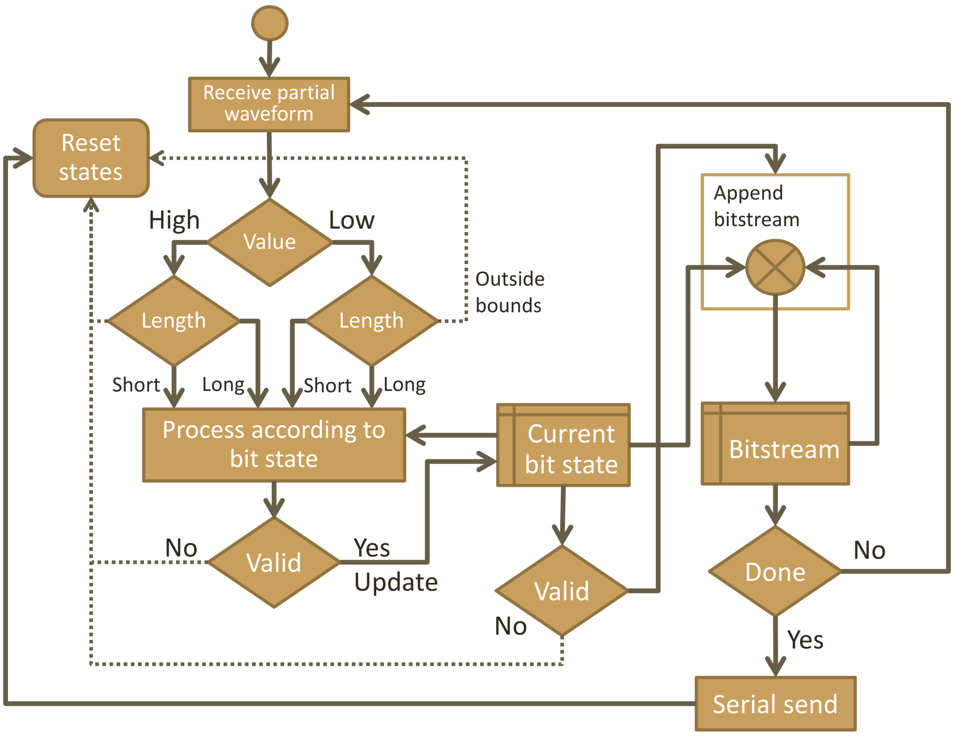


Figure 13.5: Flowchart describing the parsing routine for Nexa Code Wheel.

The C-code is available in on the enclosed CD as `nexa_codewheel.c` in the MultiTRX folder. Through observation it became evident that the first two nibbles correspond to the house and device code selectable on the sensor device, and that the last byte contains the sensor value.

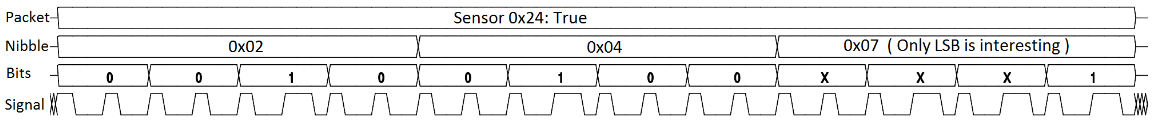


Figure 13.6: A complete packet as received from the sensor.

During testing, only the LSB of the data nibble changed in response to forced sensor input. The remaining bits were constant, and may be battery life indication or something completely different.

13.1.3 Magnet

The waveform parsing for all Nexa self-learning code products is already provided and implemented by Jon Petter Skagmo. For self-learning products, each sensor transmits a longer 28-bit factory-set identification that receiving products must "learn" by going into a special mode of operation where it will associate its behaviour with the next received packet.

13.1.4 Concentration and Storage

The MultiTRX uses a chip from FTDI to create a USB CDC[1] end-point that appears as a virtual serial port on host devices.

A receiver program was developed in Python (`serialserver.py` on the enclosed CD) with the following components:

- TCP server to allow outside communication with the MultiTRX dongle
- Serial communication thread that routes received data to listening classes and writes queued outgoing data.
- A packet parser responsible for checking and handling self-learning and code-wheel packets.
- A database wrapper responsible for inserting records into a MySQL database.

The data received from the MultiTRX has the following format:

```
                iiiiiiigsp                 iip
Self-learning: $N1986132091  Code Wheel: $C247
```

Message Format 13.1: Nexa message formats. i: identifier, p: power on/off, g: group mode, s: subgroup

[1]Communication Device Class

Where $ denotes the start of a message, N and C signify a self-learning and codewheel packet respectively. The g,s,p entries are actually sigle bits extracted from the message frame signifying group mode on, group sub-id and power on/off.

For Code Wheel-device packets the message is transmitted as received from the RF-chip. It should be noted that although the PIR and Magnet sensors transmit data differently, they both employ a blind spray and pray method for media access, transmitting around 50 and 30 packets per event respectively.

13.2 Efergy e2 Wireless Electricity Monitor

The e2 was purchased on the assumption that it transmitted using OOK on the 433.92 MHz frequency and could be used with the MultiTRX directly.

This turned out to be wrong, as it transmits using FSK on a different frequency. An attempt was made to intercept this data with another tranceiver before finally listening to the output of the data-pin on the wireless display unit that comes in the e2 package.

13.2.1 Attempted RF Interception

Identifying the transmitter used in the e2 power meter as an A7201A from Amicom, the configuration of the chip on the PCB suggests according to the datasheet that it is set up in the default mode without SPI enabled.

This means it will transmit using FSK on the 434 MHz band with center frequency $F_{rf} = F_{crystal} \times \frac{1023}{32}$ and the default IF[2] bandwidth of 200khz[3]. With the crystal being 13.457 Mhz, $F_{RF} = 430.2034$ MHz.

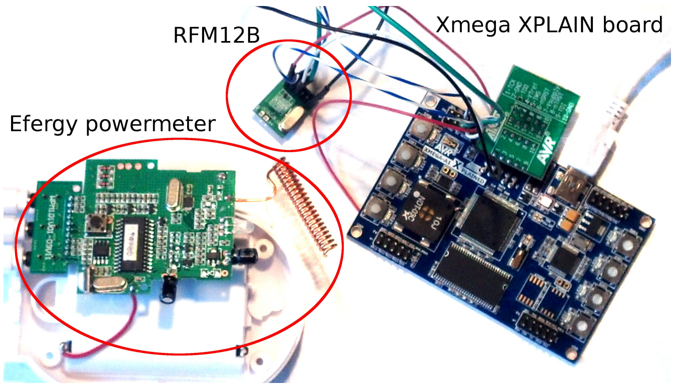


Figure 13.7: e2 Power Meter and RFM12B tranceiver connected via SPI to an Xmega128A1 Xplained board used for attempted interception of power meter data.

An attempt was then made to calibrate an RFM12B module produced by Hope RF to this frequency over SPI from an Xplain board. Reading back the configu-

[2]Intermediate Frequency, how much the carrier is offset

ration registers on the RFM12B module indicated that it was operating, but no discernible signal could be observed over the noise on the module output pin.

The cause could not be determined, but is assumed to be because the RFM couldn't be configured to the correct frequency as it uses a (more standard) 13.56 MHz oscillator, and the selectable steps don't quite match the calculated frequency of the e2 meter. The attempt was then abandoned for lack of time.

13.2.2 Implemented Interception

Inside the display unit in the Efergy e2 package, a fairly large test-pad is laid out that is connected to the output from the RF chip.

A wire was soldered to the test-pad, and then attached to an analog comparator pin on an Xmega128A1 MCU as shown in the figure below.

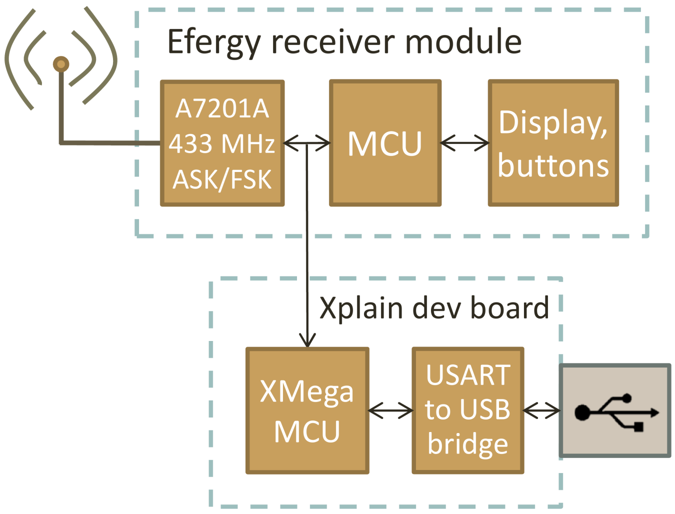


Figure 13.8: Diagram of the connection between the Efergy e2 display-unit and an Xplain board used for listening to the receiver output.

The Xplain board was programmed in C using AVR Studio. The program uses a hardware timer and analog interrupt to detect pin state transitions, then transmits the last state and duration serially to an attached host device.

Waveform

As with the Nexa waveform decoding, this was read into a Python program. The Xmega uses a 16 bit integer to store each timer value, shifts this left by one and adds the pin state as the LSB of each high or low duration. This is sent serially as hexadecimal ASCII characters[3] to a Python program. The value is then converted to a tuple of duration and state in Python for further parsing.

[3]ASCII characters were used to make debugging easier.

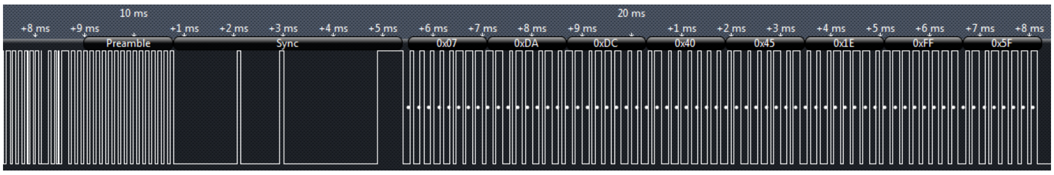


Figure 13.9: Output from the Efergy receiver IC displayed by Saleae Logic. Seen here one complete transmission of a 62W reading comprised of 1. Noise before the gain control is tuned to the preamble, 2. Preamble, 3. Sync sequence and 4. Data.

Through observation, it became apparent that each bit is sent as one period with long or short duty cycle. The short duty cycle periods were interpreted as 0-bits, as this resulted in a value of zero in what was believed to be the value-part of the message frame when no power was measured.

After the system was deployed, a custom parser plugin was written in C for the Saleae Logic Analyzer[4] (`SuperAnalyzer` under Saleae on the enclosed CD). The parsed result is identical to the Python implementation, and can be seen with the captured waveforms in Figures 13.9 and 13.10, showing the output of the developed Saleae Logic parser plugin.

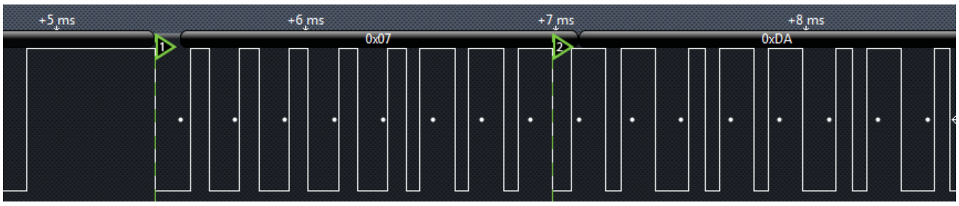


Figure 13.10: End of sync sequence and two first bytes. Low duty cycle is around 34-38% and high is around 70-74%

Parsing

After manipulating the input to the sensor and reading the value of the accompanying display unit, a picture of how the message frames are constructed emerged.

In figure 13.2 we can see that the value-part of the message from byte 4 to 6 behaves like a floating point number. Note that the byte designated `e` changes slower than the bytes comprising `f`. Between 50W and 62W we see that the fraction wraps around its maximum value of $+2^{15}$ and the exponent increases.

As the fraction (or mantissa) is 16 bits, and the exponent seems to be stored in two's complement form we can convert both to signed integers and try to calculate:

$$\text{Value} = \frac{\text{Fraction}}{2^{15}} * 2^{\text{Exponent}} \tag{13.1}$$

[4]A cheap logic analyzer privately purchased for another project, which allows and encourages third party parsing plugins, available at http://www.saleae.com/logic

```
    byte# 0 1 2 3 4 5 6 7
          iiiiiippffffeecc
    0.00W  07DADC40000000FD, ID: 0x07DADC, period 6s
   29.00W  07DADC40428CFEC9 - f: 17036, e: -2 (frac and exp as int)
   36.79W  07DADC4051E8FE34 - f: 20968, e: -2
   50.60W  07DADC4070A0FE0B - f: 28832, e: -2
   62.10W  07DADC40451EFF5F - f: 17694, e: -1
2760.00W  07DADC4060000461 - f: 24576, e:  4
```

Message Format 13.2: i: node identifier(assumed), p: transmit period (6, 12 or 18 seconds), f: fraction, e: exponent, c: checksum

Which does not give us the average wattage, but the average amperage during the previous measuring period. Multiplying this by 230 (which was the setting on the display unit) we get our final value which matches up perfectly with the value displayed by the Efergy device.

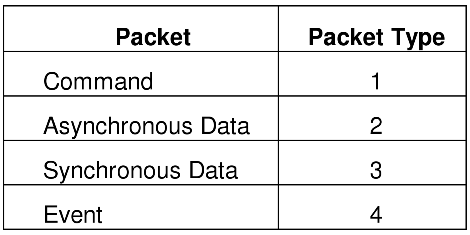
$$\text{Wattage[W]} = 230[V] \times \frac{\text{Fraction}}{2^{15}} * 2^{\text{Exponent}}[A] \tag{13.2}$$

13.3 Bluetooth LE Dongle

As previously mentioned, the version of the PandaBoard used to run the code was unfortunately not equipped for Bluetooth LE. Therefore, a BLE dongle from Texas Instruments is used to connect to the body-worn KeyFob.

A Python program (`bledongle.py`) was developed to communicate with the dongle via a serial interface. The program sends commands to the dongle for connecting to and requesting data from the KeyFob, as well as adding the received data to the database.

This program will initialize the dongle, connect to the KeyFob and turn on the on-board accelerometer. Thresholds for the pedometer application on the KeyFob are set, and data is then periodically requested from the KeyFob and placed in the database. The source code for `bledongle.py` can be found on the enclosed CD.



Packet	Packet Type
Command	1
Asynchronous Data	2
Synchronous Data	3
Event	4

Figure 13.11: HCI packet types, image from [40]

Commands for operating the dongle was found in the TI BLE Vendor Specific HCI Reference Guide[40]. TI's PC application for testing BLE applications, BTool, was also used to examine the commands sent back and forth, especially in error situations, which were not discussed very thoroughly in the reference guide. Using BTool for testing purposes made it possible to make the program more robust by helping interpretation of error situations, allowing the developed program to take appropriate measures to maintain the connection to the KeyFob.

There are four types of packets supported by the HCI layer as seen in Figure 13.11; Command Packet, Asynchronous Data Packet, Synchronous Data Packet, and Event Packet. The packet type is determined by a byte value preceding the packet.

The packets sent to the dongle are command packets. These include an Opcode, length of the following parameters and the parameters themselves as seen in Figure 13.12.

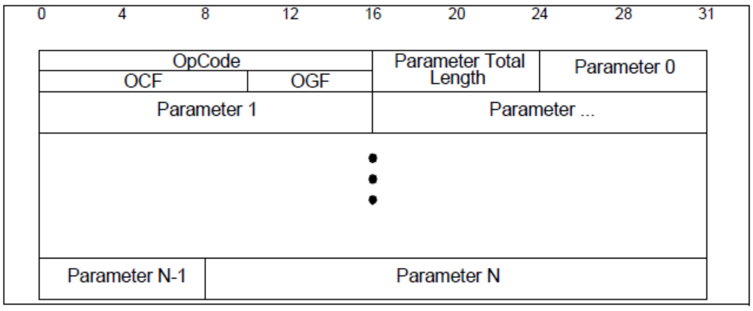


Figure 13.12: Command packed, image from [40]

Packets from the BLE dongle are event packets. These are sent from the dongle as acknowledgements and replies to command packets such as read requests. The event packet format can be seen in Figure 13.13 and consists of an event code, length of the following event parameters and the parameters.

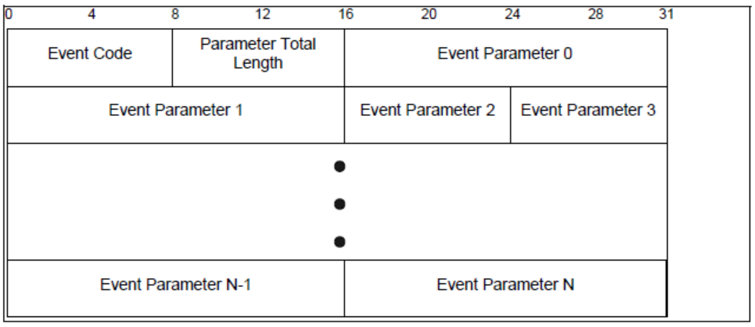


Figure 13.13: Event packet, image from [40]

With this knowledge, it's possible to distinguish the different messages received from the dongle and handle them appropriately.

Both receipt of messages through the serial port and handling of the messages are implemented as simple state machines, and the receipt of messages is done in a separate thread. The dongle replies with an ACK event for each command sent to it, and these are used to make sure a transition to the next state is safe. If no ACKs are received, the process that failed is aborted, and initialization of the dongle or the connection to the KeyFob is restarted as necessary, depending on which state the system is in. The state machine for handling messages from the dongle is seen in Figure 13.14 and the state machine for receiving messages is shown in Figure 13.15

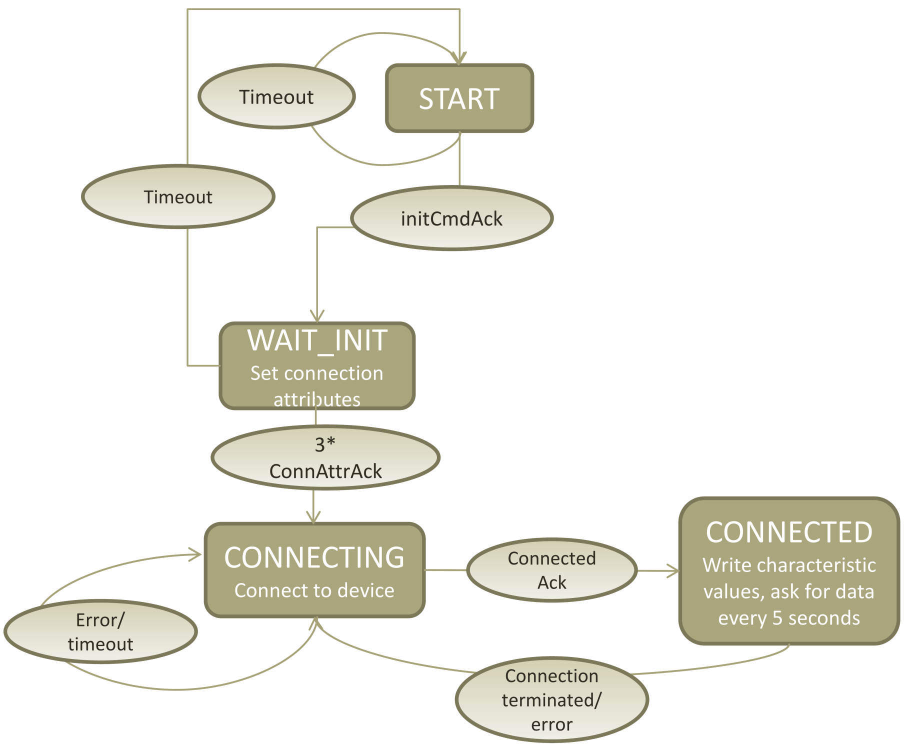


Figure 13.14: Main state machine of bledongle.py

In Listing 13.1 the initialization process of the dongle is shown. Before entering the while-loop containing the state machine, the `initDevice` function is called. If the acknowledgement for the init command is not received within 40 seconds, or an error message is sent from the dongle, the initialization is restarted. When the init acknowledgement is received, the state is changed to `WAIT_INIT`, where a message from the dongle signaling that the initializing is done as expected. If this is not received within 40 seconds, the initialization is restarted. When the expected message is received, connection attributes such as connection interval for the dongle are set and when the dongle has acknowledged the setting of all of these,

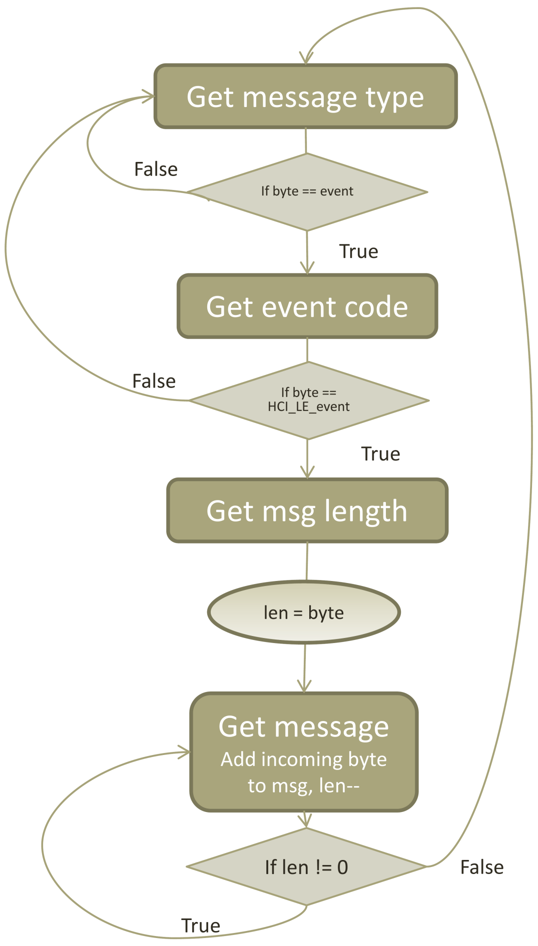


Figure 13.15: Message receiving state machine of bledongle.py

the state is changed to `CONNECTING`, and a connect request message is sent to the dongle to connect to the KeyFob.

Listing 13.1: Initialization of BLE dongle

```
if state is States.START:
        if msg == const.initCmdAck:
                state = States.WAIT_INIT
        else:
                ble.initDevice()
elif state is States.WAIT_INIT:
        if msg[0:3] == const.deviceInitDone:
                #Dongle initialized, set connection attributes
                ble.setMinConInt(const.ms500)
                ble.setMaxConInt(const.ms500)
                ble.setSlaveLatency(const.latency5)
                #Number of connection attributes acked:
                connAttr = 0

        elif msg == const.gapSetConnAttrAck:
                #Connection attribute ack, wait for three
                connAttr += 1
                if connAttr == 3:
                        state = States.CONNECTING
                        ble.deviceConnect()
        else:
                ble.initDevice()
                state = States.START
```

The dongle connects directly to the KeyFob worn by the test subject, but functionality for scanning for and connecting to an arbitrary device is added to the program, and can easily be reimplemented.

When the dongle is connected to the KeyFob, it will ask for set up the Movement profile, enable the accelerometer, and request accelerometer, battery, pedometer and rssi values every 5 seconds and insert this into the database. Requesting this data is done by sending the byte sequence: 0x01, 0x8E, 0xFD, 0x0C, 0x00, 0x00, 0x39, 0x00, 0x3D, 0x00, 0x41, 0x00, 0x51, 0x00, 0x2F, 0x00. The meaning of the different bytes is presented in Table 13.1.

	Description
0x01	Message type: Command
0xFD8E	Opcode: GATT_ReadMultiCharValues
0x0A	Data length: 10 bytes
0x0000	Connection Handle: 0
0x3900	Accelerometer X characteristic value handle
0x3D00	Accelerometer Y characteristic value handle
0x4900	Accelerometer Z characteristic value handle
0x5100	Pedometer characteristic value handle
0x2F00	Battery characteristic value handle

Table 13.1: Byte sequence sent to BLE dongle to read characteristic values

The handles for the different characteristic values can be found by sending a “Discover characteristics by UUID” command to the KeyFob. The UUIDs are determined by the software on the KeyFob as described in Chapter 10 on page 55.

If the KeyFob is disconnected from the dongle for any reason, such as being out of reach, the dongle will retry to connect until it is reconnected.

14 Interpretation of Sensor Data

This chapter first describes the framework developed for routing sensor events to classifiers and aggregators, then each classifier and aggregator is described in detail.

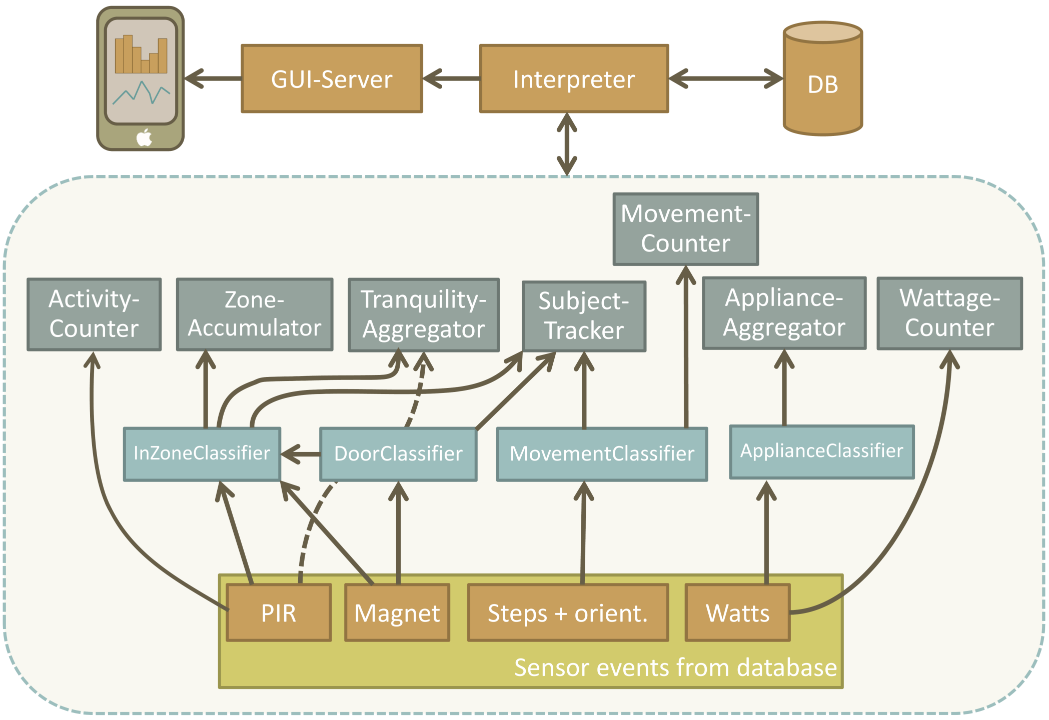


Figure 14.1: Overview of framework class interconnections

14.1 Framework

The glue that binds the different parts together consists of three main ingredients, the `Interpreter` class, written in Python, the `logdb` relational database which contains the sensor log and sensor- and room metadata, and a heavy reliance on the `Observer pattern`, a classic design pattern that facilitates weak coupling between class-instances.

In Figure 14.1 on the preceding page we see how information passes through each of the developed classes, roughly divided into layers. On the top we see the `Interpreter` class which is responsible for database communication and instantiating and polling the context-acquiring classes.

The GUI Server, further described in Chapter 15 on page 99, polls the aggregators to present data, and even subscribes to the `SubjectTracker` class for a live-feed of the action.

14.1.1 Observer Pattern

Most classes which consume or emit data inherit from the `Subscriber` and/or `Publisher` class defined in `observerpattern.py`, available on the enclosed CD.

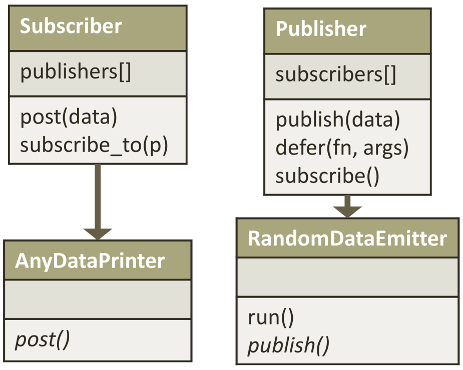


Figure 14.2: Observer pattern class inheritance

Using the pattern defined by `ObserverPattern` allows for a separation of concern between producers and consumers of information, because they do not need to know about each other and are not dependent on other facilites than `post` and `publish` to communicate.

Listing 14.1 on the next page shows a minimal use-case:

Listing 14.1: Example usage of ObserverPattern.py

```python
from ObserverPattern import Publisher, Subscriber

class RandomDataEmitter(Publisher, Thread):
	def __init__(self):
		Publisher.__init__(self)
	def __run__(self):
		while 1:
			time.sleep(4)
			self.publish(random.randint(0,100))

class AnyDataPrinter(Subscriber):
	def __init__(self):
		Subscriber.__init__(self)
	def post(self, sender, data):
		print "Got this data:", data

>>> # Some agent has to instantiate and tie the classes together. Here we show it in the
  REPL prompt.
>>> em = RandomDataEmitter(); em.start()
>>> # ....... Nothing happens
>>> AnyDataPrinter(em)
4 # Value emitted by RandomDataEmitter and printed by AnyDataPrinter.
46
```

In this manner a class instance responsible for detecting occupance of a room will subscribe to sensor events, and a class responsible for keeping track of e.g. sleeping habits will subscribe to classified context cues, conforming to the requirements laid out by Dey et. al. described in Section 6.2.1.

This pattern makes the interpretation system event-driven, in that the objects do not themselves seek information, but can act only on received events.

14.1.2 Database

MySQL was used for database storage as it is free, mature and platform independent. The database can roughly be divided into **sensor logs**, **metadata** and **parsed data**, and is shown in Figure 14.3 on the facing page.

Sensor Logs

- `events` – Stores data received from the PIR and Magnet sensors used for location tracking.
- `wattage` – Stores instantaneous power consumption as recorded by the e2 meter. In case of lost packets, delta_t is increased appropriately.
- `movements` – Stores orientation and movement data from the wearable accelerometer.

Metadata

Data about which sensors are placed where, and how the house is structured is kept in these tables.

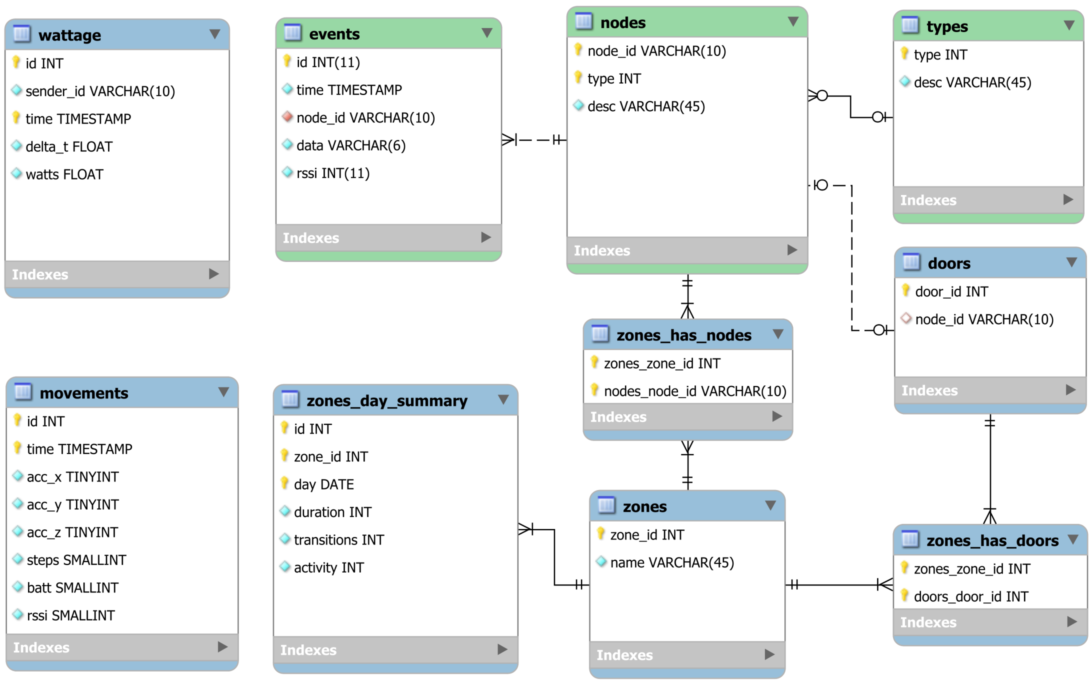


Figure 14.3: Database table overview

- `nodes` – Keeps track of all sensor nodes and their type. Each node is associated with one Zone.
- `type` – Type of sensor, here either PIR or magnet.
- `door` – Logical entity for a door. Is associated with two zones and a sensor-node.
- `zone` – Denotes a logical room or part of a room. Is connected to other rooms via doors.
- `a_has_b` – Relationships / graph edges between a and b

Parsed Data

Is only populated by `zones_day_summary` which stores accumulated time spent in each zone for each day. Ideally all aggregated data should be stored in the database, but during development it didn't remain constant, and this was therefore not done.

14.2 Classifiers

Classifier should here be understood as any object that consumes sensor events and emits some interpreted context information without storing history. In Figure 14.1 on page 84 these are shown in light blue. In a sense, ActivityCounter could also be seen as a classifier, but as it contains a history it's marked as an aggregator.

The Classifier classes roughly correspond to Dey's Widgets and Henricksen/Indulska's Interpreters from Chapter 6, and are available in `classifiers.py` on the enclosed CD.

14.2.1 Location Classifiers

Location data is provided by the passive infrared and magnet sensors located throughout the house. Chapter 11 shows a map of sensor placements, and Figure 14.4 shows a graph of all the rooms and doors that is generated by `Interpreter` from the metadata in the database.

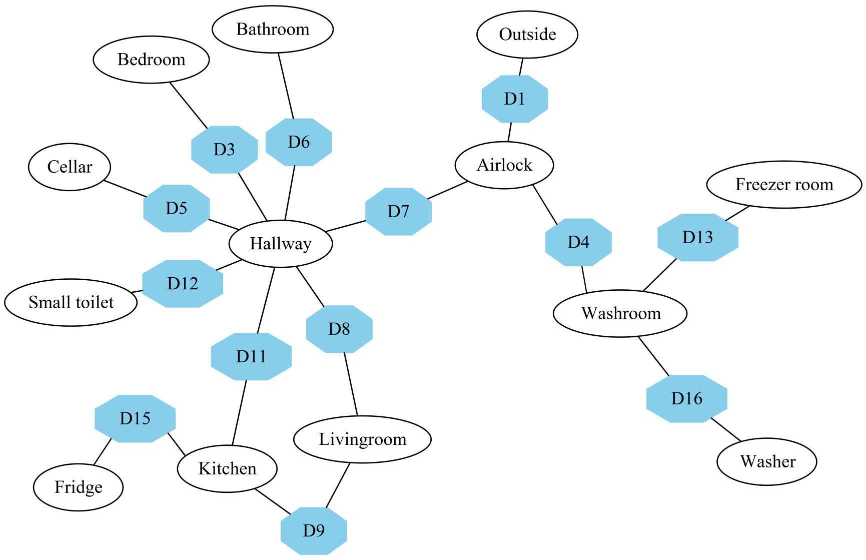


Figure 14.4: Graph of rooms and doors automatically generated from available metadata, rendered with GraphViz.

This graph is a representation of `Zone` and `Door` objects as they are represented in memory. Each room and door object is further associated with a classifier to interpret raw data and an aggregator for keeping track of the history.

Characteristics of the Sensor Sata and Placement

As discussed in Chapter 5 and 13, the data that is received show some characteristics which must be taken into account:

- Packets may be lost due to interference/collisions.
- PIR-sensors transmit both True and False, but never send False if a True was not immediately preceding.

- True-packets from the PIR sensor are not ambiguous, but False packets are sent after movement has stopped, and may not imply zone presence e.g. after changing rooms.
- Door events give us somewhat precise location, but doesn't tell us on which side the subject is.
- Door events may tell us something about the intention of the subject.
- Not all rooms have infrared sensors, so we must rely on assumptions and degrees of certainty.

Interpreting Location Data

Figure 14.5 shows the interconnection betweeen sensor data, rooms, doors and higher layers.

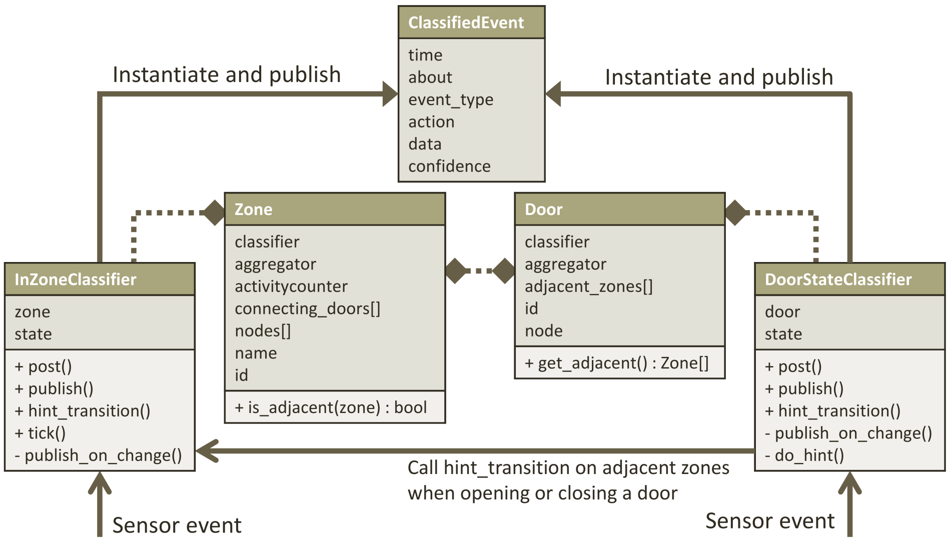


Figure 14.5: Communication and connection between Zone- (location) and door classifiers. Diamond arrows denote strong references.

Each room is represented by a `InZoneClassifier` class which accepts sensor events through its `post` method and emits a `ClassifiedEvent` to interested parties.

The `ClassifiedEvent` data type is used by all classifiers, and the `data` property takes on different meanings for each. For zones and doors `data` contains the classifier's state.

Because of the varying quality of data, as discussed in Chapter 6 it is helpful to introduce a variable which represents how certain we can be of a value. To implement this, the state is one of the `ConfidenceState` enumerator values:

- `DEFINETIVELY` – If data quality is certain.
- `PROBABLY` – Highest level of confidence when guessing.

- MAYBE – Initial state when making a positive guess.
- MAYBE_NOT – Initial state when making a negative guess.
- PROBABLY_NOT – Highest level of confidence when guessing.
- DEFINITIVELY_NOT – If data quality is certain.

InZoneClassifier

Figure 14.6 shows the states a Zone classifier may have regarding whether the subject is present in a given zone. Outer transitions are a result of observations we

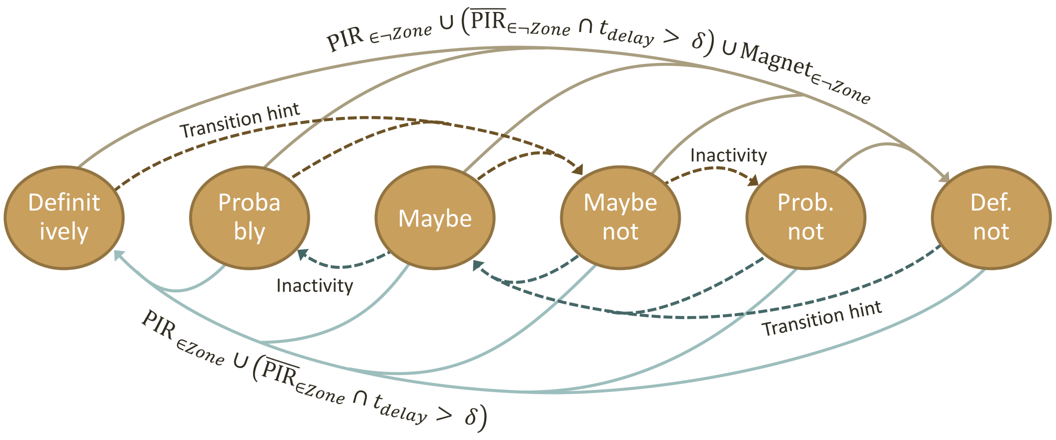

$PIR_{\in \neg Zone} \cup (\overline{PIR}_{\in \neg Zone} \cap t_{delay} > \delta) \cup Magnet_{\in \neg Zone}$
Transition hint
Inactivity
Definit ively
Proba bly
Maybe
Maybe not
Prob. not
Def. not
Inactivity
Transition hint
$PIR_{\in Zone} \cup (\overline{PIR}_{\in Zone} \cap t_{delay} > \delta)$

Figure 14.6: States a Zone-classifier may take. Changes to state or confidence result in an event being published. Note that this takes place independently and in parallel for each classifier.

consider certain, while the inner dashed transitions are initiated by the DoorClassifier and sent to its adjacent zones on door events. This algorithm will be described later.

Considering the sensor and physical characteristics we define the InZoneClassifier such that

- A `true` PIR sensor event from a sensor associated with the zone means presence.
- A `false` event is interpreted as presence, because it must necessarily have been preceded by a `true` event, but `false` is ignored until a delay has passed after a previous event.
- Events from sensors not in the classifier's zone are treated as non-presence, except that:
  - Door events connecting to a zone are ignored by that zone's classifier.
  - It may receive hints from other objects that presence is likely.

The reason that `false` events are not considered as presence immediately is that they are automatically emitted by the sensor after it has retuned its feedback gain. Therefore, a `false` event may be emitted from a zone after the subject has left it.

It could be possible for a `ZoneClassifier` to consider whether another `ZoneClassifier` has recently reported presence and thereby discard a `false` from its own sensor. However, the fact that transition hints may change the current state makes the determination of whether the subject is entering or leaving dependent on knowledge of the hinting mechanism, which we wanted to avoid to preserve separation of concerns between classes.

Tick-events are periodically emitted by the `Interpreter` object, and allows us to gradually increase confidence of presence in zones when no other activity is sensed. Most notably this behaviour is used for detecting stays in the `Outside` and `Bedroom` zones as `Bedroom` did not initially have a working PIR-sensor, and `Outside` does not have a PIR-sensor.

DoorClassifier

Aside from translating "True" and "False" to Open and Closed, the door classifier is responsible for interpreting the intention of the subject when interacting with doors.

Initally, a stateless approach was tried, where opening a door automatically transitioned the subject to the other side, and a subsequent close-event was ignored. However, through testing it became evident that this failed on some occasions where a door was opened and closed an even number of times before entering, for instance leaving the subject apparently standing around in the hallway for hours.

Listing 14.2: Hinting algorithm

```
 1  def do_hint(self, sender, tid, state):
 2      def activity_in_direction(zone, entrydoor, visited_zones=[], period=1 * 60,
           count=0):
 3        count += zone.aggregator.sum_time_between(tid - period, tid)
 4        for d in [d for d in zone.doors if d is not entrydoor]:
 5          visit_zone = d.get_adjacent(zone)
 6          if not visit_zone in visited_zones:
 7            count += activity_in_direction(visit_zone, d, visited_zones + [
                 visit_zone], period, count)
 8        return count
 9
10      def discourage_zones(zone, entrydoor, visited_zones=[]):
11        # (...)
12        # Traverse the graph, hinting zones that may not be occupied
13
14            # On opening and closing, assume that we go to the zone away from
                 recent activity
15            activity, likely_zone = min([(activity_in_direction(z, self.door, [
                 self.door.get_adjacent(z)]), z) for z in self.door.get_adjacent
                 ()])
16      # To ensure concurrency, defer hinting until all zones have received this door-
           event.
17      sender.defer_post_publish(likely_zone.classifier.hint_transition, tid, True)
18      # Tell the zone opposite the likely zone that it might have been left.
19      unlikely_zone = self.door.get_adjacent(likely_zone)
20      discourage_zones(unlikely_zone, self.door, [likely_zone])
```

The implemented solution in listing 14.2 has the `DoorClassifier` traverse the room graph (fig 14.4) in both directions from the respective door node (line 15), analyze where time has been spent the last minutes (`activity_in_direction`, line 3), and hint that a transition may be imminent in the direction of lowest occupance, making the respective zones change state to `MAYBE` and `MAYBE_NOT` and continue independently.

This effectively builds a one-dimentional gradient through the graph via the door, allowing us to correctly handle rooms without PIR-sensors or which have had the door open.

Discussion

We have aimed for as simple a solution as possible that's still effective. The guesses are not always correct, but always hinting towards lowest occupance density will ensure that leaf-nodes such as Bedroom and Outside are preferred, and relies on either PIR-sensor events or other door-events to correct the guess.

It is clear that walking through closed doors is impossible, so a definitive transition between zones implies a door must have been opened even though the sensor-message may have been lost. An implementation of this logic was developed, but was abandoned because it didn't work correctly in some instances, and was considered unimportant in a health-perspective.

The results will show that lost sensor events sometimes lead to seemingly irrational behaviour from the subject, and could even falsely indicate serious medical problems. If the subject leaves the home, but seems to remain still in a room, the system could either report the facts as recorded, albeit incorrectly, or could try to guess that the subject has left the house.

As this is intended for medical surveillance, it was decided to not hazard any guesses based on the *absence* of sensor data, but this could be a topic for debate.

14.2.2 Appliance Classifier

This classifier receives events when the power meter transmits data. For simplicity, we assume that only one appliance is used at a time. The connected appliances are a microwave, a coffee maker and a water boiler.

Below some logged sample-series are plotted against time and have been identified manually.

The coffee maker can in some cases seem to turn off, but with five-minute intervals turn back on to re-heat the contents (not shown), and the microwave switches between maximum and low power with a duty-cycle depending on the chosen power-level.

Identifying appliances by looking at individual powermeter samples is easy for the water boiler as its power consumption is much higher than the two others. For the microwave and coffemaker, individual samples may cross a chosen separation threshold.

Instead of looking at individual samples, it is assumed that when an appliance is turned on, the next samples in the series belongs to the same individual appliance.

With this, we can then average each series for more reliable classification, and use the noise caused by the power-characteristics of each appliance measured by

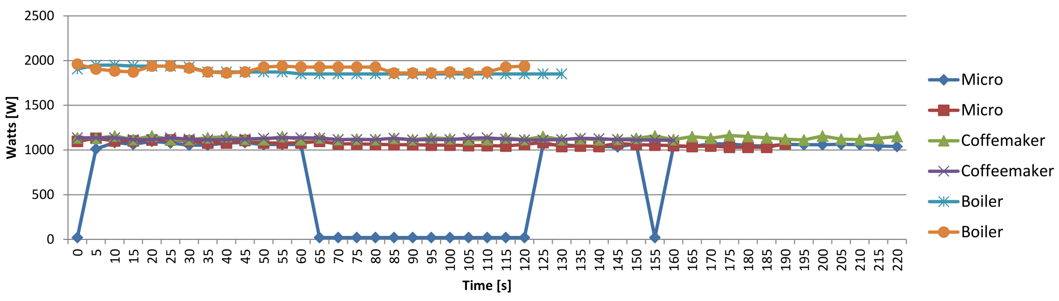


Figure 14.7: Two sample runs of each appliance. One microwave-series shows a lower-than maximum power setting realized by a reduced duty-cycle, continuing off-screen.

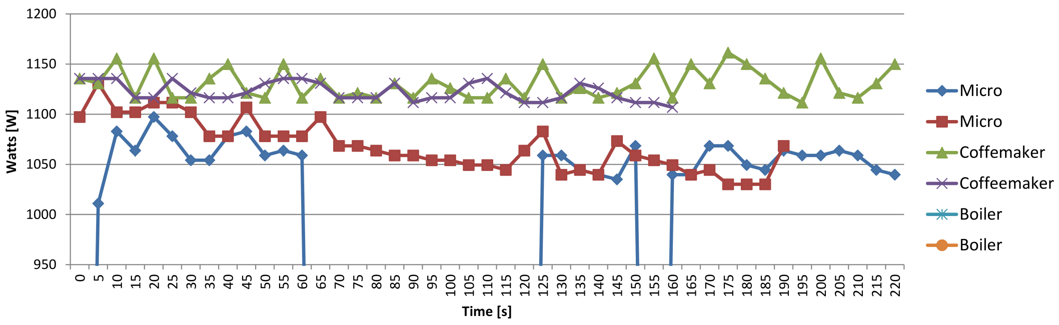


Figure 14.8: Vertically zoomed view of only microwave and coffe-brewer.

standard-deviations and distance between maximum and minimum power to further differentiate each appliance.

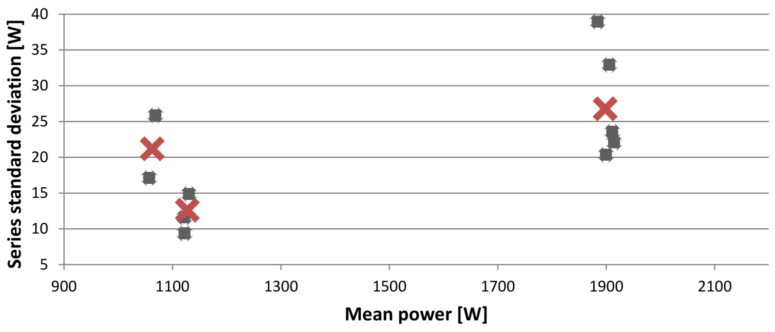


Figure 14.9: Series characteristics; average power vs standard deviation. Cluster centroids from table 14.1 on the following page marked in red.

Figure 14.9 shows a few series with average power plotted against series standard deviation. The centroids are found via a clustering algorithm called K-means. For the microwave, the low duty-cycle periods are ignored, as we do not have enough sample series to classify each power-level.

Clustering

The K-means algorithm[56] is implemented by a Python package called `scipy`, and is a relatively simple iterative algorithm that finds the centroids which most differentiate a given set of clusters.

Using this algorithm, samples are read from the database and split into series, and the mean average, standard deviation and distance between maximum and minimum outliers for each series are calculated and stored as features in a feature-vector.

This feature-vector is then passed to the K-means algorithm along with the desired number of clusters, and a set of centroids is returned. The result of such a classification can be seen in Table 14.1, showing the values used in the `ApplianceClassifier` class.

Appliance	Centroid			N series
	Power	Standard deviation	Outlier difference	
Boiler	1898.62	28.38	88.03	26
Coffee-brewer	1125.27	13.15	82.66	5
Microwave	1032.14	21.15	93.44	4

Table 14.1: Appliance cluster centroids found during offline analysis.

Classifying Sample Series

On reception of powermeter readings, the classifier makes a new `ApplianceEvent` if one is not active and stores the samples in an array.

When no power has been consumed within a given time, the event is ended, the features mentioned above are calculated, and is passed to a reverse-lookup method `scipy.vq` (Vector Quantization). The closest matching centroid index and a distortion score is returned, the index is mapped to an appliance and the event is published inside a `ClassifiedEvent` to higher layers.

Discussion

Assuming that only one appliance is used at a time is a severe limitation. It should be possible with this implementation to create a cluster centroid for each combination of appliances. Considering the variation seen in the samples here, it seems like a difficult limitation to overcome however, as a low-powered appliance could easily drown in the combined variance from high-powered appliances.

A better, or at least easier way to classify appliance use on a large scale might be using one power-meter per appliance or inserting power-buttons with radio transmitters.

For detecting abnormal power consumption patterns however, it is not necessary to differentiate appliances and this is described in Chapter 7.2.

14.2.3 Movement Classifier

Steps As the classification has already been done on the sensor, step count is simply wrapped in a `ClassifiedEvent` and published. Steps represent a count of peaks on a high-passed series of accelerometer values. The bottom graph in Figure 14.10 from Normann/Skjønsfjell[66] shows approximately what the step counter on the KeyFob sees, although it is not the same brand of accelerometer.

Orientation The top three graphs in Figure 14.10 show how $\vec{g}$ affects the X-, Y- and Z-acceleration measured when a subject moved around in different positions.

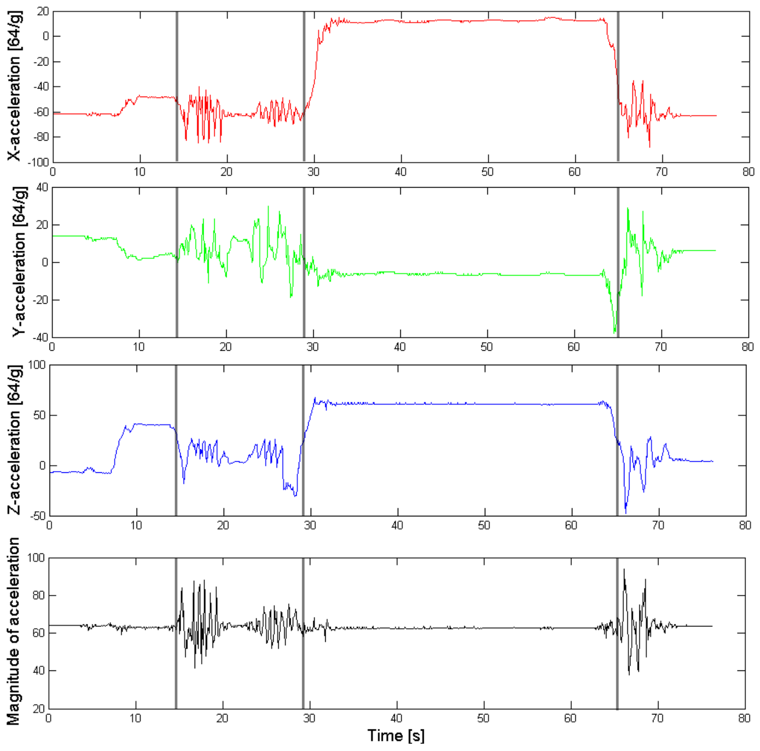


Figure 14.10: Chest-worn sensor: 1) Sitting, then leaning back, 2) Standing up, walking, standing, walking, 3) Lying down, 4) Standing up, walking, sitting down

An attempt at interpreting the physical orientation of our test-subject was made using X-, Y- and Z-value thresholds and assuming a constant orientation of the sensor with respect to the body. The results will show that this assumption did not hold, as the sensor appears to dislocate over time and the distinction between orientations was therefore ambiguous at times.

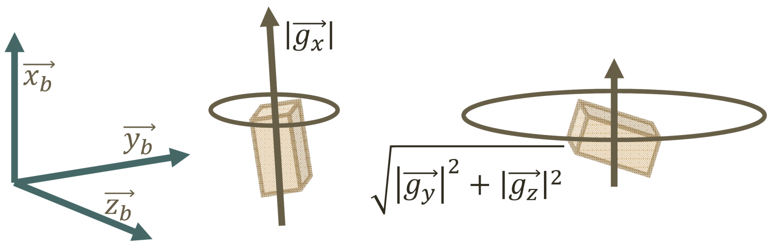


Figure 14.11: Using the influence of $\vec{g}$ on accelerometer values to determine tilt. Axis are fixed on the body-frame. Y and Z are combined to ignore direction of tilt.

Location The sensor is configured with a connection timeout of 20 seconds. If no sensor data is logged within this time, it is assumed the connection has been lost and an `ActionTypes.OUT_OF_RANGE` message is published. If the sensor comes back into range, `IN_RANGE` is published.

14.3 Aggregators

The aggregator classes (seen in Figure 14.1 on page 84) as implemented can be divided into two categories: Those that only log events and provide historical context to the application, and those that combine multiple lower level events. The code is available in `aggregators.py` on the CD.

14.3.1 Context History

As mentioned, all `Zone` objects have reference to a classifier and an aggregator. In this way we satisfy the requirement of persistent historical access described by Dey et. al.

WattAccumulator Logs power-consumption; is identical to the `wattage` table in the database. Used mainly to avoid bottlenecks in the SQL server during testing.

ActivityCounter Sums total PIR activation events into periods of variable size. Serves as a cognitive wrapper for the database contents of the `events` table.

MovementCounter Identical to `ActivityCounter`, but stores steps/high-passed accelerometer activity.

ApplianceAggregator Stores the `ApplianceEvent` objects published by `ApplianceClassifier`.

ZoneAccumulator Stores all published state changes pertaining to a zone. When polled, iterates through the log and calculates time spent in a zone based on level of confidence desired.

14.3.2 Combination Aggregators

SubjectTracker

This class is referenced and managed by `Interpreter`, and listens to all classified events. Every time the subject enters a new zone, a `Stay` object is instantiated and a reference is stored in a log.

The intent of `SubjectTracker` is to organize the subject's actions in a meaningful way, i.e. to say that when the subject was in *that* zone for *that* long, these other actions were taken and are therefore both spatially and temporally associated.

As can be seen from Figure 14.6 on page 90, the state changes emitted from zone classifiers are not always symmetrical. It can and does happen for instance when walking through a zone without a PIR sensor that one zone is certainly not occupied, whereas another zone has received a hint that it may be occupied.

`SubjectTracker` resolves the ambiguity by only acting on positive occupance of a certain confidence level. If and when a client such as a GUI-server serving an iPad is connected and has registered for real-time data, `SubjectTracker` will publish these events for display on the client.

TranquilityAggregator

This class is mainly intended for sleep pattern analysis, in that it only stores `TranquilStays`, defined as stays lasting more than 20 minutes with breaks no longer than 10 minutes. If a break has lasted more than 10 minutes the stay is ended.

Only activity occuring in the designated zone in the time-frame of the "tranquil stay" will be logged, and so the output from this class will give us some measure of the restfulness of sleep both from the activity level and from the number and duration of nightly errands.

14.4 Discussion

In translating sensor data to knowledge, the concept of which is shown in Figure 6.1 on page 30, metadata and domain knowledge is required. In Henricksen[59], Guha[52] and other papers on the subject of context, much effort is devoted to formally expressing knowledge, building repositories of knowledge and automatically abstracting and lifting[1] contexts.

This implementation does not keep a separate repository for what can be called domain knowledge, but rather has this knowledge implicit in the programming of classifiers and aggregators. The reason for this is that if a such a repository, that is not simply ad-hoc, were to be used, it would require much more work in a direction that was deemed as less important for the problem at hand.

It has not been a primary concern to conform to specifics in the "Context Toolkit" and "Java framework" described in Chapter 6, but they have been helpful

[1]Mapping specific information to generalized patterns, identifying variable parameters

in providing guidelines for an implementation. Looking back to Dey et al's requirements however, we claim that they are for the most part satisified as understood by the candidates.

During testing and development, extracted information was not committed to persistent storage. Once the quality and format of the classifiers is established, this should be done to avoid re-interpreting the whole database if for instance power is lost. Once this is done we have the option of erasing detailed logs and only keeping aggregated information to provide more privacy for users.

15 iOS Application - bHome

An application to graphically show current and past behavior of the test subject was developed for an iPad and will be presented in this chapter. An iPad running the bHome application can be seen in Figure 15.1, and was chosen due to availability, the size of its screen as well as a desire to learn more about programming for iOS. Source code for the project can be found on the enclosed CD.

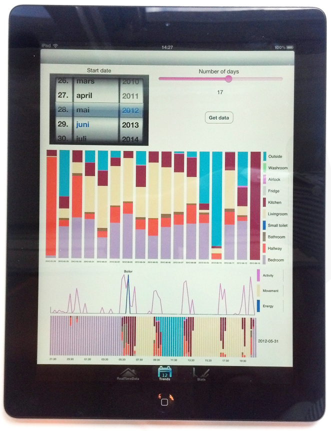


Figure 15.1: iPad running the bHome application

A Mac with the Xcode IDE was provided by the Department of Engineering Cybernetics and was used for the development of the application. The application is called `bHome`, which is a wordplay on "be home" to reflect the motivation of allowing the aging population live at home as long as possible.

An Apple Developer account was acquired by our supervisor, which made it possible to deploy the application both on the simulator and on an actual iPad. All the application screenshots in this chapter are taken from the iPad. The application is based on a Tabbed Application template, which provides a new project with a storyboard that includes a Tab Bar controller for easy navigation between the different tabs. The template can then be modified to include the number of tabs needed.

The application is intended to be used by health personnel or relatives of the person under observation. If the iPad is placed in the home of the observed person, it could also be used for recreational purposes, such as talking to family members via Skype, which could be beneficial for mental health. This is one of the reasons why the size of the screen was of particular interest, as a large screen is easier for elderly to interact with. The fact that a lot of information was to be presented on the screen also favored a tablet over a smart phone.

Most of the implemented classes and their application are presented in Table 15.1, and the means of communication between the main classes are shown in Figure 15.2 on the facing page.

Class	**Description**
BHAppDelegate	Global state of the application
BHHouseRealTimeViewController	Responsible for the real time tab
BHTrendViewController	Responsible for history tab
BHStatsViewController	Responsible for the stats tab
BHNetworkController	Handles the connection to the server, as well as sending and receiving messages
BHMessageWriter	Converts message data to the same format as the server
BHMessageReader	Extracts message data
BHBarView	A view that contains a bar chart and a color coded list
BHBar	A view that contains one bar
BHActivityView	A view that contains a line chart
BHStatsView	A view that contains a line chart with statistical values
BHDrawDoorHori	Handles drawing of the horizontal doors
BHDrawDoorVert	Handles drawing of the vertical doors
BHConstants	Includes string constants for the rooms

Table 15.1: Some of the main classes comprising the bHome Application

The rest of this chapter will introduce the different parts of the application as well as the GUI server followed by a brief discussion.

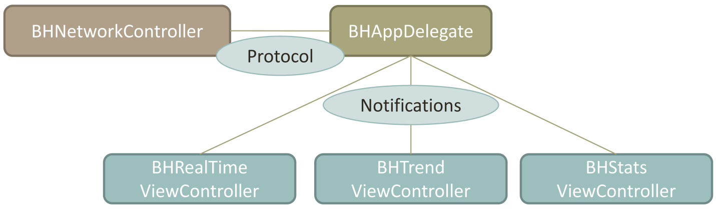


Figure 15.2: Communication methods between the main classes

15.1 Network Connection

The iPad connects to a GUI server used to communicate with the framework discussed in the last chapter via a TCP/IP connection. The connection and message sending to the server was based on a tutorial for making a multiplayer game[37]. From this, a network controller (`BHNetworkController`) that handles setup of the connection and receipt and sending of messages was implemented. The network controller declares a protocol that is implemented by the `BHAppdelegate` and used for the controller to communicate with the app delegate.

Classes for reading and writing the messages to the GUI server (`BHMessageWriter` and `BHMessageReader`) were created directly from the tutorial. Messages between the server and the application are sent as message strings together with a byte to determine the message type. The packets sent are seen in Figure 15.3 and consist of an integer, describing how long the message is followed by a byte that identifies the type of message, then another integer, telling length of the following string, and finally the message string. This information is used by the network controller to distinguish one packet from another and separate the message string and the byte identifier. The byte identifier is then used to call the appropriate protocol function in the app delegate.

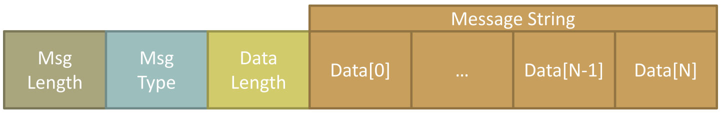


Figure 15.3: Structure of packets

15.2 BHAppDelegate

The app delegate is responsible for initializing the network controller, as well as communicating with it and the different view controllers. The `BHAppDelegate` communicates with `BHNetworkController` via method calls, while the other way around is done by adopting the `BHNetworkControllerDelegate` protocol. Initialization of the `BHNetworkController` and declaring the `BHAppDelegate` as a `BHNetworkControllerDelegate` is shown in listing 15.1.

Listing 15.1: Code snippet from didFinishLaunchingWithOptions

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(
     NSDictionary *)launchOptions
{
    networkController = [[BHNetworkController alloc] init];
    [networkController setDelegate:self];
    return YES;
}
```

Communication to and from the view controllers are done via notifications. Registration for and posting of a notification when a connect message is received from the server is shown in Listing 15.2.

Listing 15.2: Registration for and posting of notifications after connect message is received

```
- (void) connectMsgReceived:(NSString *) string {
    [[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(
         sendRTDataRequest) name:@"rtDataReq" object:nil];
        [[NSNotificationCenter defaultCenter] postNotificationName:@"connectMsgRecv"
             object:nil];
}
```

15.3 Real Time Tab

The Real Time Tab is the first tab shown to the user when the application starts, and features an image of the floor plan of the house. When the application is connected to the server all doors will be drawn in their current state (opened or closed). Whenever a message indicating movement in the house is received, the corresponding door will be updated or a room will be highlighted. This can be seen in Figure 15.4 on the next page. It is also possible for the user to long press on any room to get a list of the last ten visits in that room with the date, time and duration of the stay as seen in Figure 15.5 on page 104. Each stay in a room corresponds to a `Stay` object in the `SubjectTracker` as presented in Chapter 14.

The `BHRealTimeViewController` is responsible for this scene. When it receives a notification from the BHAppDelegate that the connection to the server is established, it sends a notification back to request real time data from the server. Whenever a real time data message is received in the `BHNetworkController`, it is passed on to the view controller which in turn extracts the name of the door or the room and updates the view according to the state of it.

All rooms are rectangular `UIViews`[30], placed in the appropriate place on top of the floor plan image. These are then hidden behind the floor plan, and brought to front when they are active. When a room is active, its view is filled with a transparent color to highlight it. When the room is no longer active, but was the last room active, it is made more transparent to make it easier to track the movement of the subject.

The doors are instances of the `BHDrawDoor` classes, one for horizontal doors and one for vertical doors. These classes subclass the `UIView` class, and implement the drawing a colored line with direction determined by whether a door is open

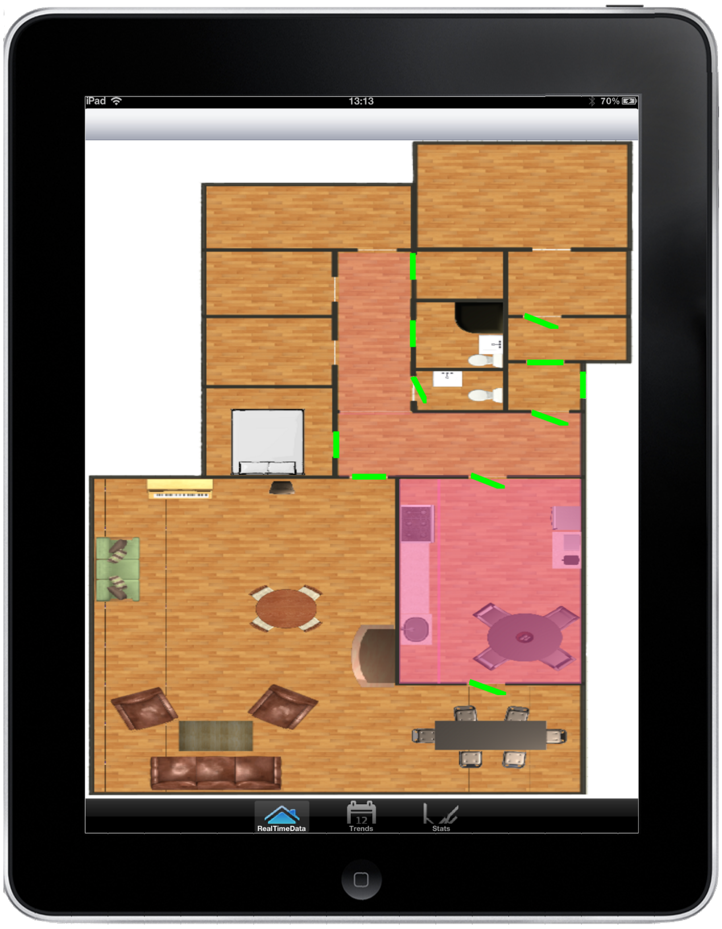
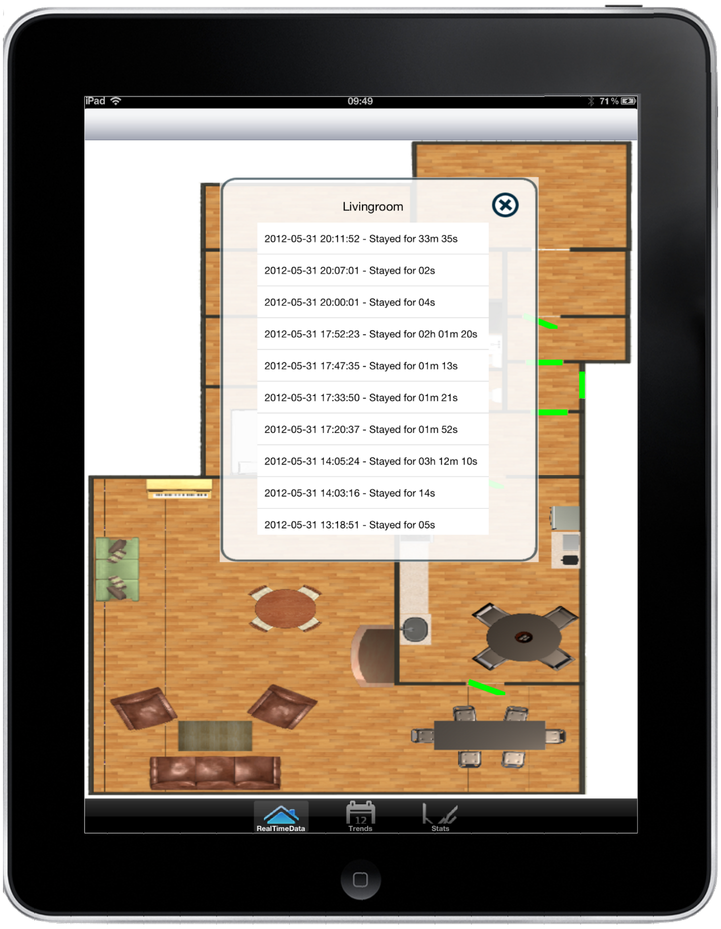


Figure 15.4: Screenshot from Real Time Tab

or closed. Any `UIView` will redraw itself when the `setNeedsDisplay` method is invoked. How a door view is updated after receipt of a string from the server can be seen in Listing 15.3. The string from the server is on the form:
`Door:DoorState:DoorNumber`
Where `DoorState` is either True (open) or False (closed) and `DoorNumber` is the id of the door in the database.



Livingroom

2012-05-31 20:11:52 - Stayed for 33m 35s
2012-05-31 20:07:01 - Stayed for 02s
2012-05-31 20:00:01 - Stayed for 04s
2012-05-31 17:52:23 - Stayed for 02h 01m 20s
2012-05-31 17:47:35 - Stayed for 01m 13s
2012-05-31 17:33:50 - Stayed for 01m 21s
2012-05-31 17:20:37 - Stayed for 01m 52s
2012-05-31 14:05:24 - Stayed for 03h 12m 10s
2012-05-31 14:03:16 - Stayed for 14s
2012-05-31 13:18:51 - Stayed for 05s

Figure 15.5: Screenshot from real time tab with historical data from a room

Listing 15.3: Updating a door view

```
if (doorNum == DoorBathroom) {
           if ([value isEqualToString:@"True:"]) {
               bathroomDoor.color = green;
               bathroomDoor.open = TRUE;
               [bathroomDoor setNeedsDisplay];
           } else {
               bathroomDoor.color = green;
               bathroomDoor.open = FALSE;
               [bathroomDoor setNeedsDisplay];
           }
}
```

When the user presses a room on the screen, a touch event is registered and a `performSelector`[25] method is called, which will run a specified method after a delay of one second, thereby implementing the long press function as seen in Listing 15.4. If the press is ended before the delay, the `performSelector` method is cancelled and nothing happens.

Listing 15.4: React to touches on screen

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {

    UITouch *touch = [touches anyObject];
    [self performSelector:@selector(fireLongPress:) withObject:touch afterDelay:1.0];
}
```

The `fireLongPress` method compares the touch location with the location of the different room rectangles, and if the touch is inside any of these, a request to get historical data for that room is sent to the server. When the data is received, the data is added to a `UITableView`[29] which in turn is added to a background view that contains a close button and a label stating which room the information is for. This is then brought to the top of the view hierarchy which shows it to the user. When the close button is pressed, the view is hidden again.

15.4 History Tab

The second tab is the History Tab, which will give the user information of the amount of time each day the user has spent in the different rooms in a bar chart. The screen has a date picker where the user can select the start date, a slider bar to select the number of days to display and a button to request the data. The screen also consists of two empty `BHBarViews`, referenced in the code as `barView` and `dayBarView`, as well as an empty `BHActivityView` referenced as `actView`. This tab is controlled by the `BHTrendViewController`. How the different views are organized inside each other and inside the main view of the view controller can be seen in Figure 15.6.

When the user presses the "Get Data" button, a request for information about the selected dates are sent to the server. When data is received, it is added to the `NSMutableArray`[23] `data` property of the `BHBarView` named `barView` which will draw the bar chart. The screen with the bar chart displaying information about 9 days is shown in Figure 15.7.

After the bar chart is drawn, it is possible for the user to long press on any bar, to get a detailed overview of the day divided into 15 minute intervals together with a line chart of movement, sensor activity and energy use of kitchen appliances as shown in Figure 15.8.

When a touch event is registered, the view controller checks if it inside the `barView` rectangle. If it is, the touch location is mapped to the coordinate system of the `barView` as seen in Listing 15.5 and passed on to the subview.

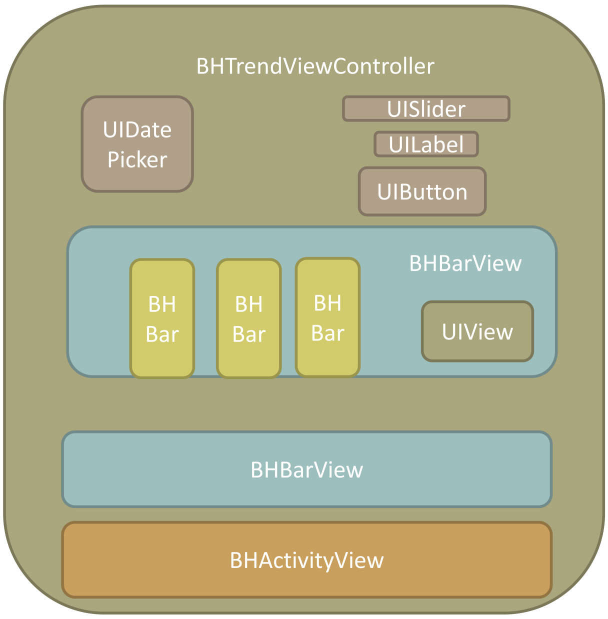


Figure 15.6: Views inside the `BHTrendViewController`

Listing 15.5: Detecting a touch event in the `barView`

```
if (CGRectContainsPoint(barViewRect, touchLocation)) {
        touchLocation.x -= barViewRect.origin.x;
        touchLocation.y -= barViewRect.origin.y;
        [barView handleTouch:touchLocation];
    }
```

`barView` will compare the touch location with its bars to find the date of the pressed bar, and send a notification containing the date back to the view controller, which in turn forwards the request to the server.

Upon receipt of the data, depending on whether it is for the line chart or the bar chart, the data is added to the `data` property of the `BHActivityView actView` or the `BHBarView barDayView`, and both views are drawn when all the data has arrived.

15.4.1 BHBarView

The `BHBarView` class is a subclass of `UIView` and draws bars to the screen. This is done by making a list of all the different time intervals that are to be presented as bars and a list of all the rooms with a separate color assigned. Depending on the number time intervals, the width of each bar and the offset between them is calculated.

For each time interval, a label with and a `BHBar` element is created at the

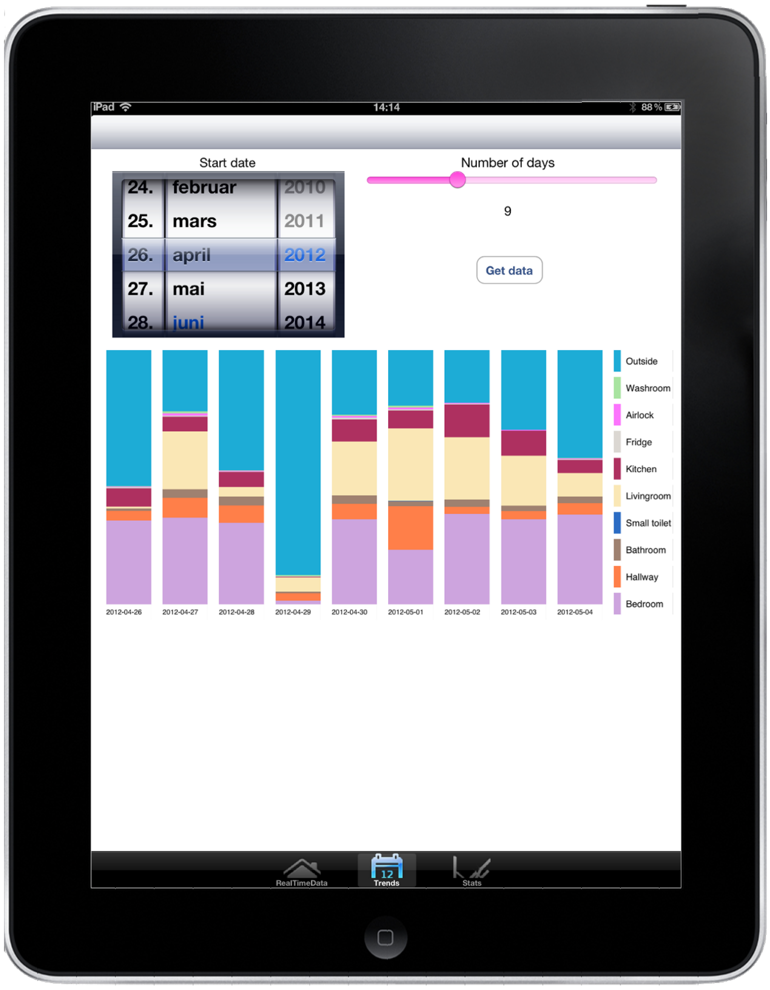


Figure 15.7: Screenshot from the history tab a presentation of 9 days

appropriate location, and amount of time spent in each room and the corresponding color is added to the `NSMutableArray data` property of the `BHBar`. The fact that each bar is created runtime, lets the `BHBarView` display an arbitrary amount of bars and rooms.

After creating the bars, a view with a list of the rooms together with a box of the corresponding color is added to the far right of the `BHBarView`.

This class also has a `boolean` property called `showRoomList`, which decides whether the list of rooms should appear or not. If this is set to false, as in `barDayView`, the bars will take up all the space inside the view, and labels for bars will only be made at every 8th bar to accommodate to a larger set of bars.

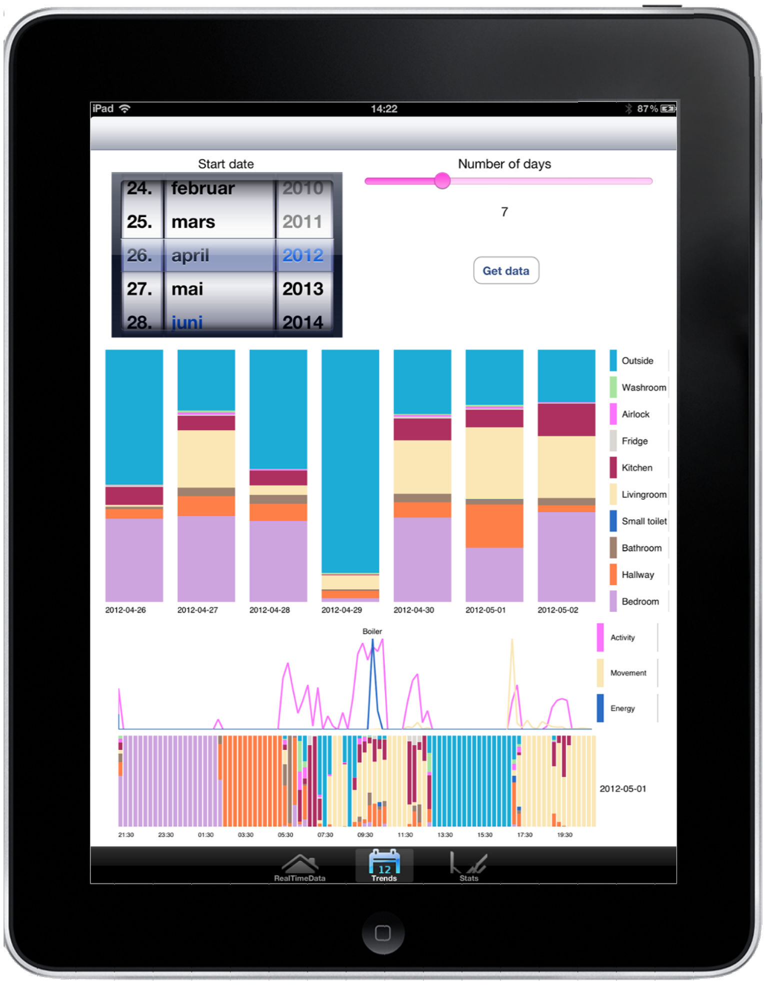


Figure 15.8: Screenshot from history tab with extended info about a day

BHBar The `BHBar` class is also subclassing `UIView`, and when drawn, each bar item first loops through its array of amount of time spent in each room and calculates the total number of seconds. This is used to scale the size of each color box so that the entire bar is filled, as well as to make the class flexible to present any amount of time in a bar.

After calculating the total amount, the amount for each room is drawn under the next as a color-filled subview with the specified color and the correct height. The creation of a new bar can be seen in Listing 15.6. Here, the scaling of the height of the view, initalizing the view in the right position and adding of color can be seen.

Listing 15.6: Drawing of a BHBar

```
for (int i = 0; i < data.count; i++) {
        height = (((BHBarData *)[data objectAtIndex:i]).duration) * self.frame.size.height/
              seconds;

        [barItems addObject:[[UIView alloc] initWithFrame: CGRectMake(x, y, width, height)
              ]];
        itemColor = ((BHBarData *)[data objectAtIndex:i]).color;
        [((UIView *)[barItems objectAtIndex:i]) setBackgroundColor:itemColor];

        [self addSubview:((UIView *)[barItems objectAtIndex:i])];
        [self bringSubviewToFront:((UIView *)[barItems objectAtIndex:i])];
        y += height;
}
```

All the allocated view objects are added to arrays, to be able to remove them when the view is to be updated.

15.4.2 BHActivityView

This class is also subclassing `UIView` and draws a line chart to the screen. It has several properties that can be set by the view controller, such as an array of data points, number of days, maximum value of the incoming data as well as labels to place in the chart. The activity view starts by creating a list of the points on the x-axis (e.g. 15 minute intervals), and calculates the width between each data point depending on the number of points to draw. It then creates any labels that has been added by the view controller at the appropriate place on the x-axis found by the name of the point it's correlated with.

The view goes through all the items in the data array and makes a list of the different lines that should be drawn. In this application, this is the amount of activity from the sensors in the house, movements from the body-worn sensor as well as activity from the kitchen appliances. The name of each line is also mapped with a color. A simplified code listing of the drawing of each line can be seen in Listing 15.7. In reality, the x and y-values for each line are accessed from an array, to accommodate that these are normally different for each line as the data is not sorted. The height of each line is scaled according to its maximum value to improve readability, as the values of the different items may be very different.

Listing 15.7: Drawing of lines in BHActivityView

```
// Scale amount
amount = actData.duration/maxVal;

CGContextRef context = UIGraphicsGetCurrentContext();
CGContextSetStrokeColorWithColor(context, color.CGColor);

CGContextSetLineWidth(context, 2.0);
// Start point of drawing
CGContextMoveToPoint(context, x, self.frame.size.height - y);
x += width;
y = amount*height;
// End point of drawing
CGContextAddLineToPoint(context, x, self.frame.size.height - y);
CGContextStrokePath(context);
```

Lastly, the BHActivityView creates a new subview with a list of the items represented by the different lines shown with color codes in the same way as in the `BHBarView`.

15.5 Stats Tab

The third tab is the Stats Tab, which gives the user information on how the amount of times spent sleeping, in the bathroom or outside for a selected number of days deviates from the last month, week or year. At the moment, one can only compare with last month or last week because of the limited amount of data available to the system. This screen also consists of a date picker where the user can select the start date, as well as slider buttons to select what to compare and what to compare with and a button to request the information from the server. The class `BHStatsViewController` is in control of this tab. The third tab is the Stats Tab, which gives the user information on how the amount of times spent sleeping, in the bathroom or outside for a selected number of days deviates from the last month, week or year. At the moment, one can only compare with last month or last week because of the limited amount of data available to the system. This screen also consists of a date picker where the user can select the start date, as well as slider buttons to select what to compare and what to compare with and a button to request the information from the server. The class `BHStatsViewController` is in control of this tab.

The screen also has three empty `BHStatsViews`, referred to programmatically as `sleepView`, `bathView` and `outView`. Whenever a message is received, the view controller checks the type of message (data, maximum value of data, average value or standard deviation), extracts the data according to type and then adds the incoming data to the appropriate subview. Adding data values to the right view can be seen in Listing 15.8. After all the data has been received, `setNeedsDisplay` is called on the `BHStatsViews` to make them redraw themselves.

This tab is intended to let the user compare data with the mean and standard deviations to see if anything abnormal has occurred. There can also be displayed exclamation marks at data that are rated as outliers from the server.

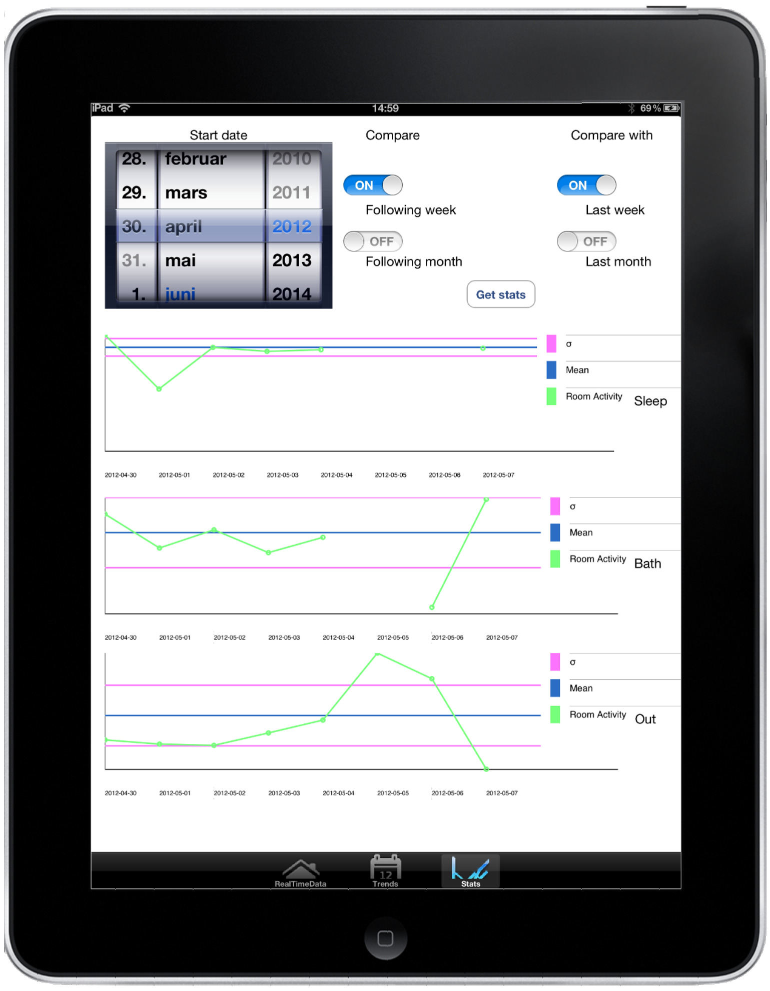


Figure 15.9: Screenshot from stats tab

Listing 15.8: Mapping a message to the right view in BHStatsViewController

```
if ([valueType isEqualToString:@"Stats"]) {

        range = [msg rangeOfString:@"!"];
        NSString *date = [msg substringToIndex:NSMaxRange(range)-1];
        msg = [msg substringFromIndex:NSMaxRange(range)];
        float value = 0;
        [[NSScanner scannerWithString:msg] scanFloat:&value];
        BHBarViewData *data = [[BHBarViewData alloc] initWithData:date room:@"" duration:
             value];

        if ([room isEqualToString:roomBedroom]) {
                [sleepView.data addObject:data];
        } else if ([room isEqualToString:roomBathroom]) {
                [bathView.data addObject:data];
        } else if ([room isEqualToString:roomOutside]) {
                [outView.data addObject:data];
        }
}
```

BHStatsView This view draws itself in a similar manner to the `BHActivityView`, with the main difference being that it also draws lines for the mean value of the selected compare time and the standard deviations above and below the mean value. In contrast to the `BHActivityView`, it does not draw lines between all data points, only the ones not going to and from zero. A small circle is also added to each point, for better visibility. Exclamation marks inside the corresponding data point can also be displayed on top of detected outliers to mark these.

15.6 Python GUI Server

The Python server, `server.py`, starts an instance of the `Interpreter` class which is descriped in Section 14.1 on page 85. An instance of the `Stats` class and the `SocketListener` class are also started. These classes will be presented later in this section. An overview of the classes used by the server can be seen in figure 15.10.

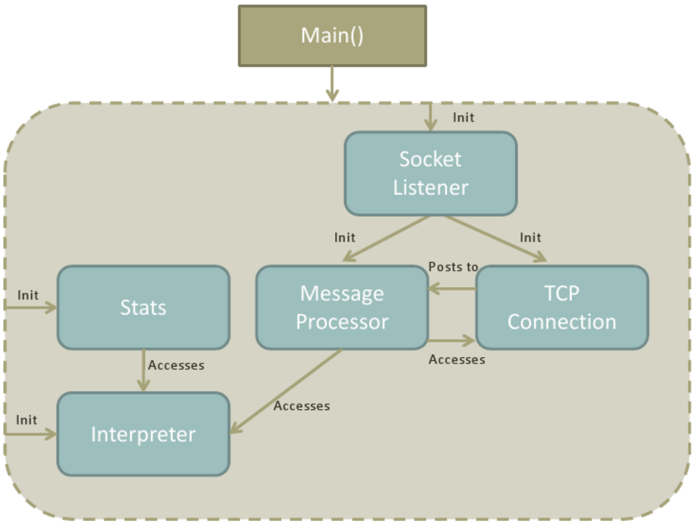


Figure 15.10: Overview of classes in the server

15.6.1 SocketListener

The `SocketListener` listens for connections at a specified port. Whenever a new connection in accepted, it creates a `TCPConnection` with a corresponding `MessageProcessor`, and adds these to its lists of connections and message processors. It also adds the `MessageProcessor` to the `TCPConnection`'s list of subscribers, and starts the `TCPConnection` thread.

15.6.2 TCPConnection

The TCPConnection has a thread that handles sending and receiving of data in the connection. The message format is as described in Section 15.1 on page 101, and a MessageReader and MessageWriter was implemented for Python as well based on the tutorial[37].

In Listing 15.9, the process extracting the incoming message can be seen. When a message of the specified length has been received, the processMessage function is called, which reads the type byte and the message string from the message object, and calls the corresponding function in the MessageProcessor with the string as a parameter.

Listing 15.9: Receipt of messages in TCPConnection

```python
def dataReceived(self, data):
    self.inBuffer = self.inBuffer + data
    while(True):
                if (len(self.inBuffer) < 4):
                        return;
                msgLen = unpack('!I', self.inBuffer[:4])[0]
                if (len(self.inBuffer)) < msgLen:
                        return;
                messageString = self.inBuffer[4:msgLen+4]
                self.inBuffer = self.inBuffer[msgLen+4:]
                message = MessageReader(messageString)
                self.processMessage(message)
```

Messages are appended with the message length and added to the send queue by any object calling the connection's post function, and are sent from the thread. If an exception is thrown while sending, the bytes that were not sent are resent as seen in Listing 15.10. This ensures that the correct message is received at the other end.

Listing 15.10: Sending a message

```python
try:
        bytes = c.send(out)
except:
        self.outq.appendleft(out[bytes:len(out)])
```

15.6.3 MessageProcessor

The message processor is responsible for handling the requests from the iPad application, and to create the data responses containing data from the Interpreter. It consists of functions for handling all the different messages types and is called from the TCPConnection when the message type has been determined.

Real Time Data

In Listing 15.11 the function called when the bHome application requests real time data can be seen.

Listing 15.11: Real time request received

```
def rtMsg(self, data):
        doors = inter.get_all_door_states()
        for d in doors:
                door, conf_state = d
                if conf_state == ConfStates.DEFINITIVELY or conf_state == ConfStates.
                      PROBABLY or conf_state == ConfStates.MAYBE:
                        msg = MessageWriter()
                        string = "Door:True:" + str(door.id)
                        msg.writeString(string, 0x01)
                        self.connection.post(msg)

                elif conf_state == ConfStates.DEFINITIVELY_NOT or conf_state == ConfStates.
                      PROBABLY_NOT or conf_state ==   ConfStates.MAYBE_NOT:
                        msg = MessageWriter()
                        string = "Door:False:" + str(door.id)
        self.inter.tracker.add_client(self)
```

The states of all the doors in the house is fetched from the `Interpreter`, and sent to the application. The message starts with “Door:”, a “True” or “False” to indicate whether the door is open or closed and finally the database id of the door. The colons are used to make it simple to create substrings to distinguish the different parts of the command. Each door state is sent as a separate message by using the post function in the `TCPConnection`.

After sending all the door states, the `MessageProcessor` is added as a client to the `SubjectTracker` in the `Interpreter`. Whenever the tracker registers movements in the house, it posts to the `MessageProcessor`, which forwards the information to the bHome application as seen in Listing 15.12. The sending of door messages has been removed from the listing for readability, as this is handled much in the same manner as in Listing 15.11.

Listing 15.12: Sending of real time data to application

```
def post(self, sender, data):
        msg = MessageWriter()
        ev_type, ev_data = data
        if ev_type is EventTypes.ZONE:
                stay = ev_data
                string = stay.zone.name + ":True"
                msg.writeString(string, 0x01)
                self.connection.post(msg)
```

History Data

When the bHome application requests history data for a set of days, the `barMsg` function is called. This can be seen in Listing 15.13.

Listing 15.13: Sending of history data to application

```python
def barMsg(self, data):
        index = data.find(':')
        starttime = int(data[0:index])
        endtime = int(data[index+1:len(data)])
        msg = MessageWriter()
        roomsAdded = False;
        dateAdded = False;
        i = 0

        for row in self.inter.get_sumtime(starttime, endtime):
                if not roomsAdded:
                        roomlist = row
                        roomsAdded = True
                else:
                        dateAdded = False
                        for entry in row:
                                if not dateAdded:
                                        realdate = isotime(time2mktime(entry) + (60*60*24))
                                        date = str(realdate[0:10])
                                        dateAdded = True
                                        i = 0
                                else:
                                        string = date + "!" + roomlist[i] + "!" + str(entry)
                                        msg.writeString(string, 0x04)
                                        self.connection.post(msg)
                                        msg.reset()
                                        i+=1

        msg.writeString("Finished", 0x04)
        self.connection.post(msg)
```

First, the start- and end time are extracted from the incoming message. Then the `Interpreter` function `get_sumtime` is called to return a list of all the zone names, the dates and the number of seconds spent in each zone that day. The `barMsg` function goes through each row, mapping the number of seconds reported to the correct zone on the given day, and sends the message. 24 hours is added to the date, as each day is presumed to start at 21.30 instead of 00.00. The date is stripped of hours, minutes and seconds, to get a readable date to present at the iPad.

Stats Data

Stats data is sent to the iPad from the `statsDataMsg` function in three steps:

- Standard deviation and average values are calculated and sent.
- Values for each day and zone are retrieved from the database and sent, as well as their maximum values.
- Outliers for the period is calculated for each room and sent.

These steps rely on the `Stats` class, which is described below.

15.6.4 Stats

The `Stats` class consists of a running thread as well as functions to retrieve standard deviations and average values of a data set. The running thread adds all stays of a day to the `zone_day_summary` table of the database. When the simulation is running, the thread starts by adding all previous days up to the simulation point to the database, it then waits for a new day, before this is added. When not running simulation, the thread waits for a new day, and adds this to the database.

The duration values for the zones `Small toilet` and `Bathroom` are summed together and added to the zone `ToiletVisits`, to be able to detect all toilet activity.

The algorithm for detecting abnormalities as presented in Section 7.2 was implemented and can be seen in Listing 15.14.

Listing 15.14: Implementations of outlier algorithm

```python
def get_outliers(self, alpha, tstart, tstop, zone):
        outliers = []
        dataset = []
        dates = []
        dateoutliers = []
        zone_id = self.inter.get_zone_by_name(zone).id
        for d in self.inter.dbr.fetch_dur_day_not_zero_stats(zone_id, tstart[0:10], tstop
             [0:10]):
                dataset.append(int(d[0]))
                dates.append(str(d[1]))

        num = int((0.5) * (len(dataset) - 1))
        print "max number of outliers", num, len(dataset), zone
        exval = dataset[0]
        print dataset, zone
        for i in range(0, num):
            n = len(dataset)
            stdev = float(std(dataset, None))
            aveg = float(mean(dataset, None))
            avgex = 0
            if int(stdev) is 0:
                return (dateoutliers, outliers)
            for d in range(0, n):
                if max(abs(dataset[d] - aveg), abs(avgex)) == abs((dataset[d] - aveg)):
                    exval = dataset[d]
                    outindex = d
                    avgex = exval - aveg
            r = abs((exval - aveg)) / stdev
            p = self.probability(alpha, n, i)
            critval = self.critical_value(n, i, p)
            if r > critval:
                        dateoutliers.append(dates[outindex])
                        outliers.append(exval)
            # Remove exval from dataset
                    del dates[outindex]
                    del dataset[outindex]
        return (dateoutliers, outliers)
```

Average values and standard deviations are calculated from the dataset. For the bedroom and bathroom zones, values that are zero are omitted, and for the outdoors zone, values that are 24 hours are omitted from the calculation. This is done as these are values for days most likely to not be spent at home, and should thus not be regarded as outliers.

The t-distribution values in the `critical_value` function are calculated with the `stats.t.ppt` function from the scipy package.

15.7 Discussion

Another proposed solution for connecting to the GUI server would have been to connect via Bluetooth, but as the iPad would have to be jailbroken as discussed in Chapter 8, this was disregarded. However, if Bluetooth framworks should become available from Apple at a later time, it would be possible to change the connection type in the application without having to do any big changes.

When the user requests data from the server, it takes several seconds before the screen is updated. The amount of time depends on how much information has been requested from the server. There are two time consuming processes that causes this. Firstly, it's the sending of the data from the server, which is depending on the internet connections of both of the server and the iPad. Secondly, it is the `draw_rect` function of the views that takes a long time. These functions loop through all the data several times, to determine the size and placement of the views to draw, as well as what data to map to each view. A way to speed things up could be to do more of the processing of the data on the server, and send data in the order it is shown on the screen.

No means have been taken to make the connection secure. If the WLAN option is to be used in a commercial setting, the data would obviously need to be encrypted. Providing a user name and password from the application to the server should also be considered. A Bluetooth option would provide more privacy, as this means that the iPad could only display the personal data when in the near vicinity of the server, which should be placed at the subject's home. For testing purposes, the WLAN option was the most obvious and achievable solution, but in further development, security should be taken into account.

To make the application universal to any house implemented with the system, the application could be modified to let the user do the set up of the floor plan and the rooms. This could be done by letting the user upload their own floor plan, and giving the user a list of all the zone names from the database. This could be used to let the user create the different zones by dragging and dropping, as well as sizing views, to the right place in the floor plan.

The history tab is already unaware of what rooms the house consists of, and uses only information from the server to draw its views. The stats tab on the other hand relies on getting information about the amount of time spent sleeping, outside or in the bathroom. This could be an appropriate solution, but further investigation into what data have signinficance for detecting deteroration of health must be conducted.

15.7.1 Detecting Outliers

The outlier algorithm presented in Chapter 6 on page 30 proved to not be directly transferable without a better classification of the days. There is a large variety in the data, and after removing outliers, the average value of the new data set is

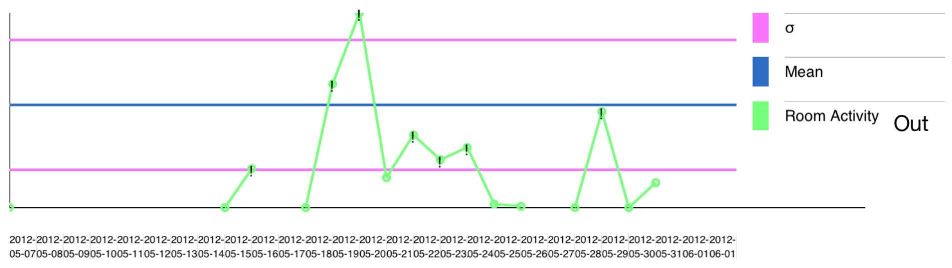


Figure 15.11: Outside zone outliers

changed rather drastically. This phenomenon can be seen in Figure 15.11 where a lot of zero (not shown with green points) and low values were present. When the most extreme value is removed, the average will fall, and the next extreme values will lower the average value even more. In the end, this leads to almost all days not being very close to zero being identified as outliers, when they in reality are normal variations in outside movement for the test subject.

It seems apparent that the days need to be classified to achieve a better result from the outliers. This could be by amount of time spent outdoors for example.

This also indicates that for the zones with a high variety in the amount of activity, comparing with the average and standard deviation of last week, month or year may not give a clear indication of something being wrong. For the sleep cycle on the other hand, which is much more stable, comparing with the last month, week or year could detect changes that have happened in the sleep cycle.

Part IV

Results

16 Results

In this chapter, the findings of the developed system will be presented. Graphs and tables showing the output of the system will be presented both overall for the test-period and for a selection of intervals where appropriate. Results will be explained and discussed in the same section, as the findings are of a varied nature and we considered it the easiest form of presentation. All timestamps are in Coordinated Universal Time (UTC, local time -2 hours) unless otherwise stated.

16.1 Produced Code

A variety of languages were used in this thesis. Table 16.1 shows a summary for each language. We use a free utility called CLOC to calculate lines of code, and if

Language	files	blank	comment	code	scale	3rd gen. equiv
Python	16	751	738	2940	x 4.20 =	12348.00
Objective C	19	484	253	2231	x 2.96 =	6603.76
C	5	411	773	1287	x 0.77 =	990.99
C/C++ Header	24	149	220	383	x 1.00 =	383.00
C++	3	76	6	273	x 1.51 =	412.23
MATLAB	2	13	3	47	x 4.00 =	188.00
Bourne Shell	2	5	3	38	x 3.81 =	144.78
SUM:	71	1889	1996	7199	x 2.93 =	21070.76

Table 16.1: Produced code lines calculated by a free utility called CLOC. Languages are scaled to third generation (C/++, Java, Fortran) equivalents to show the expressiveness of e.g. Python.

we use CLOC's estimation of how many lines of code an equivalent implementation in a third generation language would take[1] as a guideline, we can see that using Python as the main implementation language may have saved considerable effort with regards to coding. The fact that the produced Python code executed without modification both under Linux and Windows also made development easier.

16.2 Wireless Network Performance

Each of the three wireless systems used will be evaluated in the following subsections as far as this is possible from recorded data.

16.2.1 Nexa

In the period between April 24th and June 1st a total of 37,877 entries were logged as originating from a Nexa sensor. Of these, 35,791 entries correspond to a known sensor identificator. Table 16.2 shows an overview and tables 16.3 and 16.4 split up events and packets received on a per-sensor basis.

	Identified	Unknown	% unknown
Log entries	35,791	2,086	5.83
Packets	1,250,173	380	0.03

Table 16.2: Logged events, total packets received and their origin.

As can be seen from the database diagram in figure 14.3 on page 87, each sensor event row consists of time, node id, data and number of packets in the transmission. Considering that the sensors must by nature transmit alternating `True` and `False` messages we can estimate the packet loss by counting consecutive events with the same content.

We can see from Table 16.2 that spurious packets, i.e. packets which have been falsely generated or falsely parsed, constitute only 0.03% of the total. Curiously, 1,355 (65%) of the events with unknown origin exactly match a PIR sensor address with the most significant bit cleared. The cause of this is not known.

The event loss seen in Table 16.3 on the following page can in most cases be explained by the sensors' placement in relation to each other, i.e. that their transmissions overlap and drown each other out, as for example the door between the hallway and the living room which is directly in view of an infrared sensor.

An instructive observation can be made by comparing Table 16.3 on the next page with Table 16.4 on page 123. We can see that while the door between the hallway and the airlock had 41% more events than the entrance door, the entrance door successfully sent 4% more packets. When seen against the estimated event loss for the airlock door, the number of events receiving less than five packets and the fact that the hallway PIR sensor is directly in view of the airlock door, we can conclude that a significant amount of collision has occurred between them.

[1]CLOC uses numbers from http://softwareestimator.com/IndustryData2.htm

Node	Where	Events	Wrong data	% loss
D 3	Bedroom	596	79	13.26
D 7	Airlock	567	81	14.29
D 1	Entrance	400	23	5.75
D 6	Bathroom	393	29	7.38
D 4	Laundry rooom	309	24	7.77
D 15	Fridge	234	17	7.26
D 8	Hall to Livingroom	56	7	12.50
D 13	Freezerroom	35	2	5.71
D 5	Cellar	24	1	4.17
D 9	Kitchen, Livingroom	17	6	35.29
D 12	Small toilet	11	1	9.09
D 16	Washing machine	9	0	0.00
D 17	Microwave	7	3	42.86
D 11	Kitchen	0	0	0.00
Z 7a	Kitchen by table	11,735	382	3.26
Z 1	Hallway	9,498	425	4.47
Z 3a	Bathroom	3,488	173	4.96
Z 2a	Livingroom by piano	3,291	141	4.28
Z 5	Bedroom	2,598	84	3.23
Z 2b	Dining table and sofa	1,323	70	5.29
Z 7b	Kitchen unit	985	32	3.25
Z 4	Small toilet	137	6	4.38
Z 3b	Shower	56	7	12.50
Overall		35,791	1,593	4.45

Table 16.3: Events logged per Nexa-sensor. Doors are marked with Dx, Zones with Zx.

Similarly we can assume that the high event loss and low packet throughput for the bedroom door has a similar explanation.

However, it was found during installation that one of the PIR sensors didn't transmit at all, and another (shower) only intermittently, so it may well be that there are individual differences in the behaviour of the sensors.

Of all correctly received events, 6.83% had less than five received packets to confirm it. One correct packet is enough, but this can be seen as a measure of robustness, and is distributed very unevenly across sensors as can be seen in Table 16.4. As the sensors were not tested in a controlled environment, it's hard to tell whether poor perfomance is caused by poor placement or poor production.

16.2.2 Efergy

Between installation on May 1st and June 1st, 27,387 samples were received, with 929 samples measuring other than zero watts. From May 2nd, zero-samples were no longer logged to conserve space, but they are included here to increase statistical

Node	Where	Packets	Median	Max	<5 (%)
D 3	Bedroom	10,576	18.0	62	16.11
D 1	Entrance	10,042	29.5	62	5.00
D 7	Airlock	9,630	17.0	60	17.81
D 6	Bathroom	8,669	26.0	52	7.89
D 4	Laundry room	8,058	30.0	62	7.77
D 15	Fridge	4,469	18.0	78	8.55
D 8	Hall to Livingroom	1,123	23.5	31	14.29
D 13	Freezerroom	931	30.0	46	2.86
D 5	Cellar	817	31.0	62	0.00
D 9	Kitchen, Livingroom	423	28.0	49	0.00
D 12	Small toilet	261	30.0	31	0.00
D 16	Washing machine	230	30.0	31	0.00
D 17	Microwave	146	25.0	30	0.00
D 11	Kitchen	0	0.0	-	-
Z 7a	Kitchen by table	359,515	34.0	139	9.50
Z 1	Hallway	281,470	30.0	170	15.17
Z 5	Bedroom	177,464	60.0	624	0.77
Z 2a	Livingroom by piano	139,565	43.0	574	4.59
Z 3a	Bathroom	134,614	35.0	409	13.04
Z 2b	Dining table and sofa	47,714	31.0	406	15.95
Z 7b	Kitchen unit	45,140	50.0	247	3.65
Z 4	Small toilet	8,185	55.0	295	3.65
Z 3b	Shower	809	7.5	61	32.14

Table 16.4: Packets received from each Nexa-sensor, median per event, maximum received for one event and percentage of events with less than five packets received.

confidence.

Of samples received, 2,282 had a longer inter-sample period than the expected 6 seconds, indicating a dropped packet. Of these 1,957 indicate one missed sample period, 266 two periods, and 59 three or more periods. Combined, this results in approximately 9.74% packet loss.

16.2.3 Keyfob

The accelerometer was consistently connected to the concentrator with breaks of no more than five minutes for very nearly five days. Two breaks larger than one hour were recorded, and four between half an hour and an hour, all coinciding with presence in the Outside zone.

The logger polled the device every five seconds. Of 82,456 transmissions, 68.27% were received within 5 seconds and 99.93% were received within 10 seconds of the previous transmission. It is not known if any data was lost, but as Bluetooth LE features reception acknowledgement loss is unlikely to have been high.

16.3 Information from Accelerometer vs PIR

Figure 16.1 on the next page shows the activity level recorded from the accelerometer (steps) and passive infrared sensors against the subject's location in the house.

Looking at the figure, we can see that the subject got up once during the night, got up again around 6:15, moved the accelerometer attached to the cell-phone holder twice before attaching it to the belt at around 7:45. During this time, the PIR sensors may have told us about uneasy sleep. Around 8:45, the cellphone casing is detached and laid down as the subject goes outside. It can be seen from the log that the accelerometer is lying flat during the time the subject is outside.

Around 13:40 the subject reattaches the accelerometer and is inactive for about two hours in the living room. The orientation log shows the subject was lying partially sideways, indicating rest. Around 17:00 there is a discrepancy between accelerometer-data and PIR values. This could be because the sensitivity selected for the movement detection was set too high, or considering the implementation of the pedometer, that a rocking chair or something like it was used. Regardless, it's informative that the accelerometer picks up on movement too small for the passive sensors.

Note that during activity high enoguh to be picked up by passive sensors, such as zone transitions, the correlation between sensor types seems very high. Overall for this day, the correlation between PIR activity and reported steps as given by covariance over the product of the standard deviations is calculated to 0.44. For the period after the subject starts wearing the accelerometer, the correlation coeffecient is 0.52.

16.4 Appliance Classification

None of the recorded series of appliance use show multiple appliances used at the same time. The appliance-feature centroids found early on also held when classifying later use, with the total recorded instances shown in Table 16.5.

The iPad user interface in Figure 16.2 would have been a natural place to also display refrigerator and washing machine use as well as other activities if classifiers had been implemented.

	Uses	Avg. time	Std. dev
Boiler	34	2:21	0:27
Coffee maker	12	10:30	7:27
Microwave	9	4:39	3:59

Table 16.5: Identified appliance uses. Two series were too short (<10s) to identify.

Boiler use was most prevalent during the morning (61%) and could perhaps be used as an indication that the subject has gotten out of bed for the day. One particular instance at 1:45am is around five hours earlier than normal and coincides

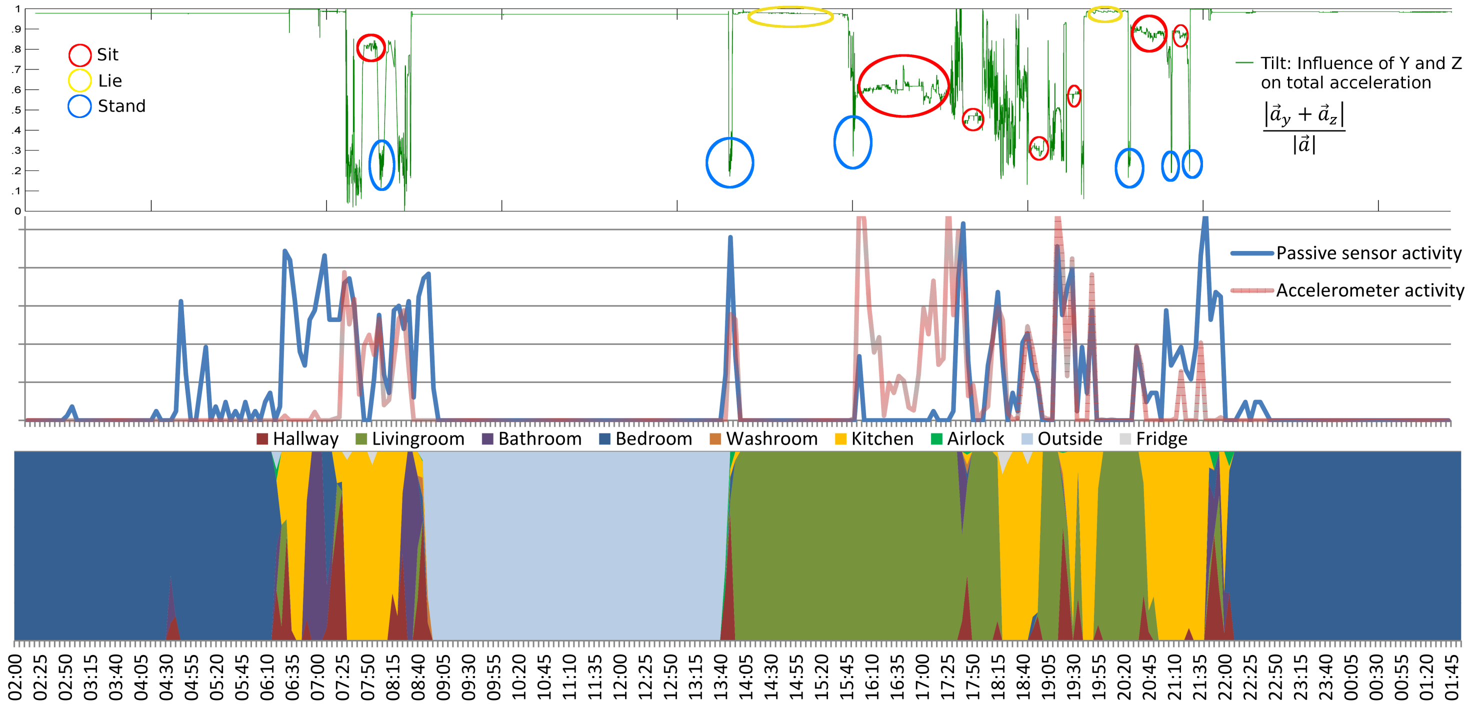


Figure 16.1: Activity logged on May 2nd 2012. Top: Amount of tilt along the horizontal plane, i.e. a value of 1 means lying on the side or back. Middle: Activity count for PIR and pedometer. Bottom: Location in the house.

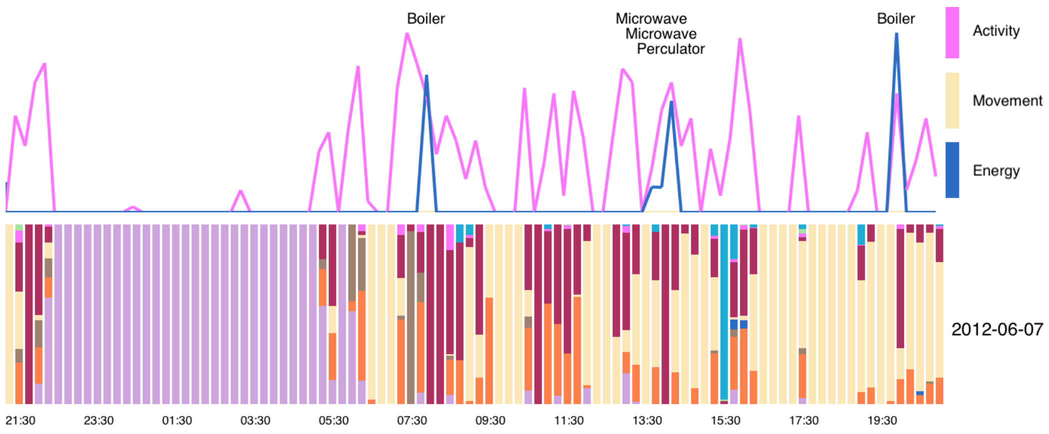


Figure 16.2: (iPad) June 7th shows use of all three appliances during the course of the day. Bottom bars show room occupance, color legend in Figure 16.3 on the next page. It's likely there are visitors as rooms change rapidly.

with a period of illness reported by the subject. Coffee maker on-time variance is due to the re-heat function whenever the appliance was not turned off immediately.

16.5 Zone Occupance

From Figure 16.3 on the next page we can see how the location classification performed. Some glaring errors are obvious and are listed in Table 16.6. In general all lost events lead to some deviance from the truth, but as the zones with PIR sensors will recieve events throughout a stay, some loss can be tolerated while maintaining an adequate degree accuracy. The same is not true for lost door events for doors leading to zones without PIR sensors, as they are responsible for most of the obvious errors shown in Table 16.6.

Date	Error	Result
25.04	Lost bedroom door event	Sleeping in hallway
01.05	Lost bedroom door event	Sleeping in hallway
07.05	Lost entrance door event	Vacation in laundry room
15.05	Pandaboard reset	One day extra vacation
16.05	Logger hangup	Long stay in kitchen
25.05	Lost entrance door event	Hallway instead of outside
26.05	Lost entrance door event	Hallway instead of outside
30.05	Lost entrance door event	Hallway instead of outside

Table 16.6: Obvious location report errors seen in Figure 16.3.

On the other hand, it can be seen for Outside – and Bedroom before May 2nd when a working PIR sensor was installed – that the implemented hinting algorithm

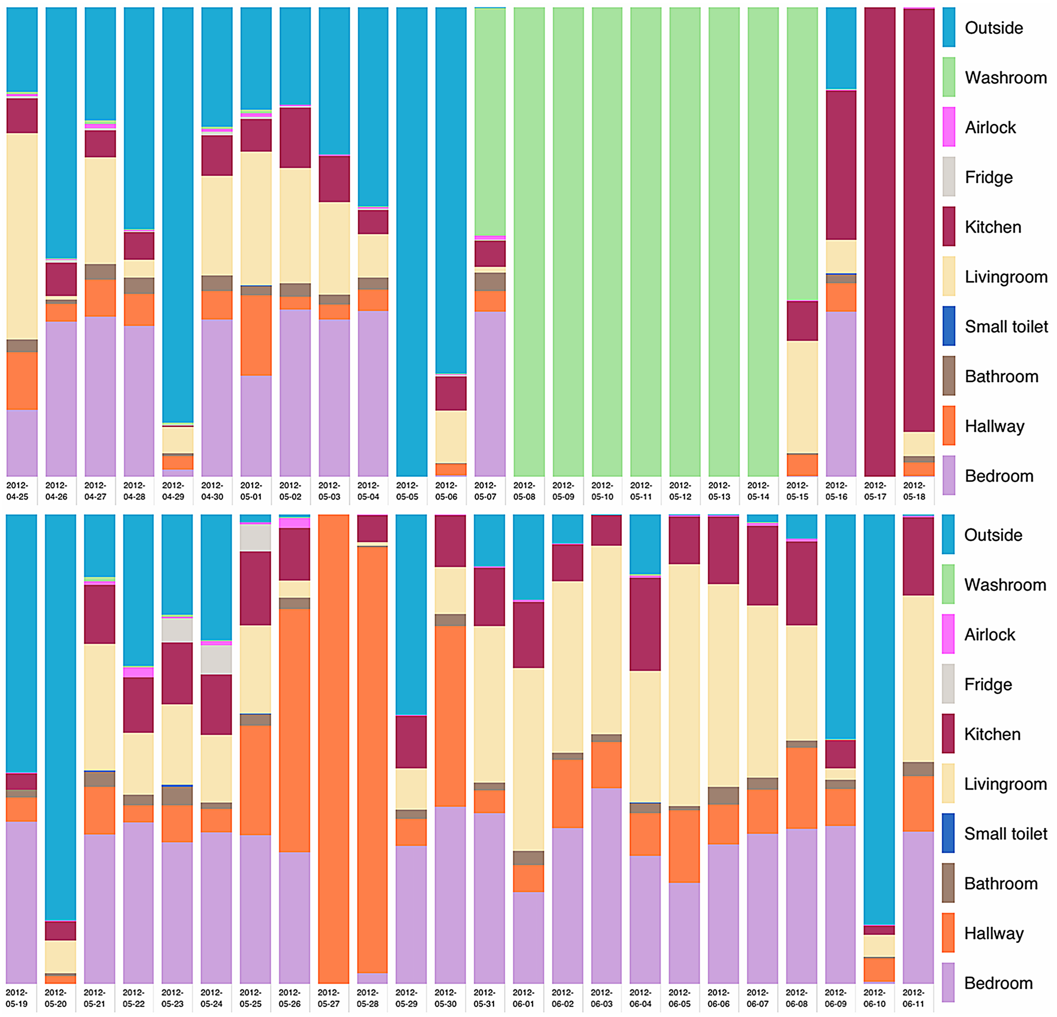


Figure 16.3: iPad screenshots. Location summary per day for May 25th until June 11th.

used when doors are opened or closed works as intended as long as the necessary events reach the interpreter.

16.6 Orientation and Pedometer Data

Figure 16.1 on page 125 shows a plot of the orientation and pedometer count for May 2nd. Before the subject goes outside in the morning it's quite clear when the subject is sitting or standing up. We can also see that the subject reattaches the sensor and lies down on the side or back.

If we look at stable periods with medium tilt (red rings), it's safe to assume that the subject is sitting down, because a belt is naturally tilted at an angle when sitting, and an absence of activity can often be observed. However, we note that the degree of tilt varies wildly between the assumed sitting periods.

Whether this is a result of varied sitting positions or a shift in sensor position is difficult to determine. The subject has a wide variety of chair and sofa types, and there are an infinite number of ways to sit, so a combination is likely.

16.7 Presentation

As seen from the screenshots in the previous chapter and the excerpts from screenshots in this chapter, the iPad application gives a fairly clear and intuitive overview of the subject's movements and history. For health personnel, this can be used to manually get an indication of past and present activity of a subject.

16.7.1 Statistics

Looking at the amount of time spent in the bedroom, outside and in the bathroom it is clear that the time spent outside, as well as time spent in the bathrooms varies unpredictably and are somewhat inversely correlated. Figure 16.6 on page 130 shows the relationship between time spent outside, and total activity inside the house per day. The time spent asleep on the other hand is much more stable, and here it is easier to see and draw conclusions from deviations from the norm.

Figure 16.4 shows graphs from the application for May 28th till June 6th. In this time period, the test subject came down with pneumonia, and had to get medical treatment. In the graph, a few days of interest are marked. June 1st is marked with red circles, and on this day, it can be seen that the test subject slept less than the days before and after. In Figure 16.5, the specifics of this day is shown. We see that the test subject went to bed before 21:30, and that he had a short bathroom break around 00:15. Then, the test subject got up again some time close to 01:45, and this time he turns on the boiler and starts his day.

On June 4th and 5th, marked with blue circles, it can be seen that the test subject slept for a very limited amount of time. This is not completely accurate, as examining the days more carefully, showed that there were breaks in the bedroom stays both nights for approximately 30 minutes. The sleep classifier as implemented considers any break for more 10 minutes as a sign of the sleep being finished, and any stays in the bedroom after this are regarded normal day activity and thus not added to the sleep. This means that the blue circles don't necessarily marks days with very little sleep, just a sleep that is more disturbed.

The orange circles mark June 9th and 10th. On the 9th, the test subject went on a trip, and didn't get home before the evening on the 10th. This can be seen as there is no sleep the 10th, and that the amount of time spent outside was rather large on the 9th, and very large on the 10th.

In Figure 16.6 the ratio of sensor activity is plotted against time spent outside. The main trend is that the more time spent outside, the less activity from the indoor sensors. This is to be expected, but it can also be seen that the activity varies a lot,

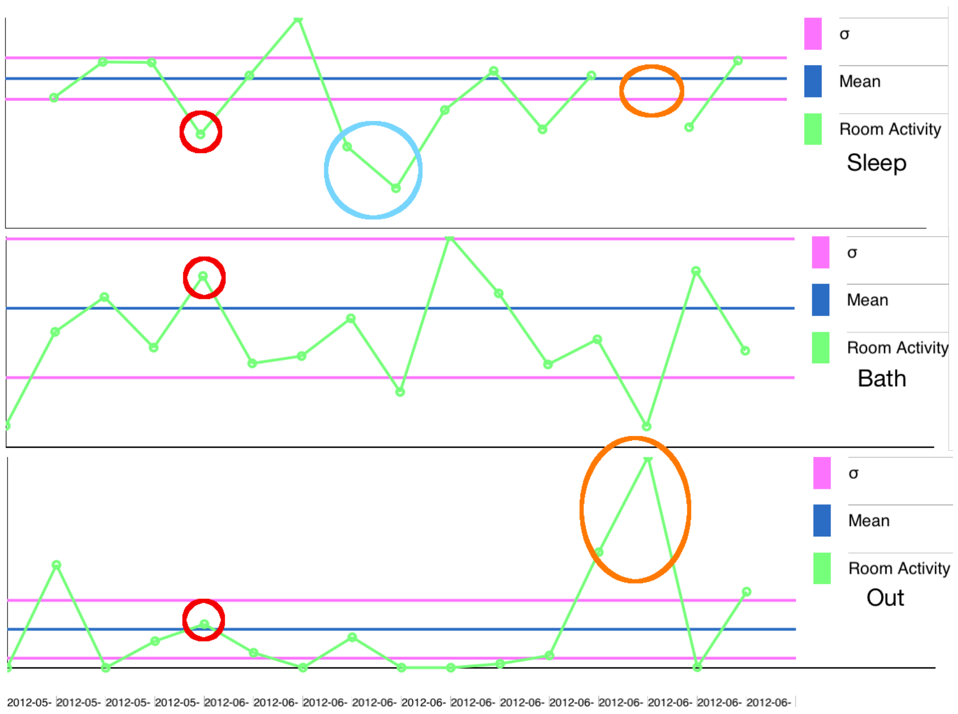


Figure 16.4: Activity from May 28th to June 6th. Circles marks days of special interest

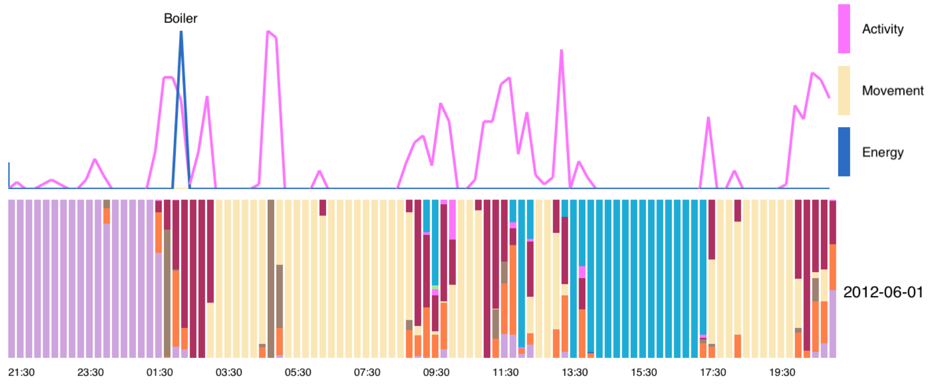


Figure 16.5: Detailed overview of June 1st

and that it does not follow the predicted (red line) value very consistently when much time is spent inside. This could well be a result of visitors in the house.

Assuming that the relationship is really linear may not hold, as there could be a set of discrete day types instead of a linear dependence. However, normalizing the

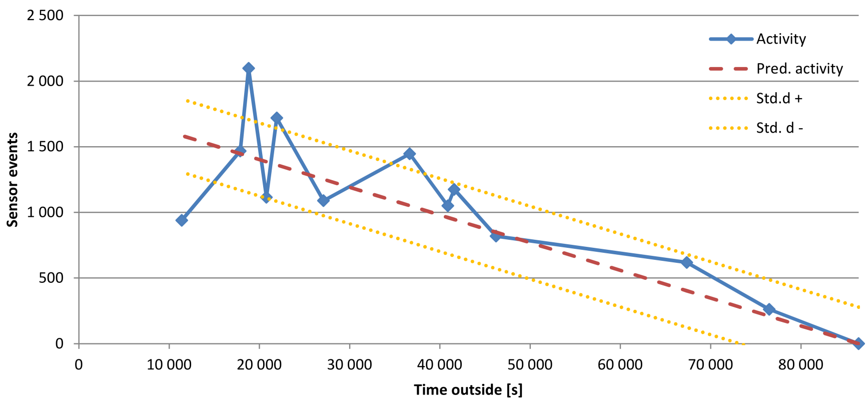


Figure 16.6: Sensor event count vs time spent outside for April 25th to May 12th. Red line denotes prediction from linear regression.

measured activity level against the amount of time spent in the sensors' field of view by some mechanism makes sense intuitively, and a deviance from the prediction could give us another indication of a subject's general condition. For activity sensed with infra-red sensors, this may hold only if activity goes down because activity from visitors can mask inactivity on the part of the subject under observation.

16.8 Experience of the Test Subject

The test subject was asked a few questions about his experience and how he perceived the monitoring.

He claimed that he did not feel uncomfortable being monitored, and that his acitivity level had not been reduced or increased due to the sensors in the house. We did particularly ask about how he felt regarding the sensors in private areas such as bathroom and bedroom, and he said that he had no hesitations to these, as he was well aware that the sensors only detected movement. He did mention that others might be more sceptical to this, as the sensors can remind a little of cameras.

When it came to the sensors, he said that he didn't mind them being in the house, and that the only time he paid any special interest to them was when some of them fell down from time to time. The sensors were not preceived as unsigthly either, and he got used to them pretty quickly. On the other hand, this could also be because he knew they were only there for a limited period of time.

The accelerometer, he tried to use as much as possible and it was not bothering to wear. He did think that in time bringing it with him it could become a part of the daily routine such as wearing his hearing aid and taking his cell phone with him.

If a health care worker would be monitoring the behavior, he thought it might increase the feeling of security at home in case something happened, such as feeling indisposed with his safety alarm out of reach.

Part V

Discussion, Further Work and Conclusion

17 Discussion

In this thesis we have tried to give an overview of the current state of the art in consumer-grade wireless sensor technologies, their possible application in health monitoring, and to implement techniques for handling, combining and presenting sensor data from a variety of sources typical, in our opinion, of the smart homes of the near future.

In hindsight, the scope of the thesis could have been narrowed down somewhat, giving more attention to only one or a few parts of the chain from real-world fact through observation to presentation. However, as this to our knowledge is one of the first ventures for the Department of Engineering Cybernetics into the world of domestic surveillance, we felt it pertinent to stake out the area in its entirety to the best of our abilities, shining a light wherever we could, and see how far we could take it. This in the hope that where we have tread wrong, others might learn from our experience and tread with more confidence.

17.1 Wireless Solution

Choices we have made in this regard in the implementation are, as previously discussed, not technically superior. The choices reflect a need for compromise between technical optimality on the one hand and concerns with regards to risk and cost on the other, with the goal being to get enough nodes of enough types running soon enough to evaluate the entirety of the solution.

Our results and experience lead us to recommend more robust wireless stacks for further work. As stated earlier, while there can be a significant amount of slack with regards to timing and precision without degrading the quality of the data, the data reported to health personnel or caregivers should not be inaccurate, i.e. reporting someting that is completely false such as an observand staying for days in a laundry room.

17.2 Data Concentration

As the results show, the local concentrator node had a high percentage of uptime. One unexplained reset occured, but remote access was reestablished automatically. It is therefore in our opinion a good choice to rely on tried and tested platforms such as Linux running on relatively cheap hardware is for future implementations and even full scale deployment.

Using a database server for persistent storage made attaching different logger-processes storing data simultaneously a walk in the park, while at the same time giving an arbitrary number of processes access to historical data. Creating a secondary channel for sending real-time sensor data directly to interpreting and displaying processes is a possible improvement, but considering the relatively slow dynamics of human behaviour this was not considered worth the increased complexity.

17.3 Privacy

As has been observed in the public debate surrounding electronic patient journals, privacy and security are paramount in gaining and keeping the trust of users. We believe that the use of restricted local access and public-key encrypted end-to-end remote connections as has been used in this implementation is sufficient regardless of the security of the local network.

However, depending on the dedication of an attacker it has been noted that wireless sensor signals are easily intercepted – indeed this has been the premise of our implementation and should therefore be considered a well proven point. Should such a system ever be widely deployed, this aspect must be taken into consideration.

Technically, securing stored data and restricting access is easily done. The challenges in this regard are more of an administrative nature, and has not been a focus in this thesis.

17.4 Event Routing and Interpretation Framework

This is perhaps the most important design decision to make except for how data is presented to the user, as it will greatly affect development and maintainance cost, as well as runtime performance. We studied some previous work, and as discussed used concepts and recommendations from these in our implementation. Our experience is that an emphasis on weak coupling between components and layers and a separation of concerns made it easy to develop and test each component separately. The effect of abstracting sensor data as context cues made it easy to develop higher order interpreting routines, but this effect was perhaps not fully utilized as time constraints required the development to stop before more classifiers could be implemented.

Although letting each context classifier process data independently means it takes some minutes to classify a month of raw data on a modern computer, we find the performance acceptable considering the reduction in complexity. The interpretation of sensor data is intended to run continuously and to persistently store generated information, making the performance more than adequate in a deployed system.

17.5 Presentation

As already mentioned in Chapter 15, a deeper evaluation is needed to decide which rooms should be presented as graphs with statistics. The amount of sleep, time spent in the bathroom and time spent outdoors all seem as good indicators of a subjects health condition, but need to be classified into groups of similar days to give a clearer view of anomalities.

With a larger data set, it would be interesting to compare a recent time interval with the same time interval last year, as this can give a better indication of e.g. sleep routines having changed over the last 12 months. Human behavior is as previously mentioned unpredictable, and this is clearly seen from the results. This is why intelligent systems need to be implemented to detect abnormalities, as simple statistics isn't necessarily enough.

17.6 Usefulness in a Health Context

When it comes to the usefulness of the implemented system in a health context, a more thorough examination in cooperation with health personnel or relatives is required. As discussed in the introduction, insomnia and appetite loss can be a symptom of deterorating health, and having a system for seeing what a patient has done since the last visit from health personnel might help detect this at an early stage.

Incipient dementia or similar diseases could also be captured by health personnel by seing small signs of a patient being disorientated e.g. by getting up in the middle of the night, cooking at unseemly times or wandering outside. These are symptoms that may not normally be seen before the disease is more developed, but by picking these up at an earlier stage, the medical treatment of the patient could be accomodated to allow the patient to stay safely at home as long as possible.

The sensor types we have looked at in this experiment record parameters related to behaviour and activity. While this can be interesting both for emergencies and as a diagnostic tool, more targeted sensor types such as for instance blood sugar measurements or lung capacity could also be integrated to perhaps avoid hospital visits simply for routine monitoring.

18 Further work

Suggestion: Improve or perfect one aspect of the chain data must pass through, be it a specific type of information from a sensor, or a way to present data clearly and usefully.
Rationale: A useful monitoring system must necessarily rely on accurate and precise reports, interpreted intelligently and presented in a useful manner. Focusing on just one aspect will make it more likely to achieve solid results.

Suggestion: Consider focusing on data from body-worn sensors, both activity level, orientation, location or specific sensor parameters like pulse etc.
Rationale: Body-worn sensors uniquely identify the individual wearer, unlike zone-based systems, thus scaling better, allowing cohabitation and delivering cleaner data on a subject.

Suggestion: Gather longitudinal data on a significant number of individuals. Try to determine a correlation between features extracted from the data and independently measured health parameters.
Rationale: As a step to wide scale deployment, it is necessary to assess the usefulness and quality of gathered data.

Suggestion: Interview practitioners and research the current state of out-patient programs. Determine what stops some patient groups from living at home and what can be done.
Rationale: Defining clear problems makes it easier to develop a targeted solution.

Suggestion: Develop a user interface in collaboration with practitioners, find out what they would need to identify problems and evaluate how they use the presented information.
Rationale: The target audience should be included at an early stage to evaluate the usefulness of data and clarity of presentation.

19 Conclusion

In this thesis, a platform for monitoring elderly in a domestic environment has been researched and developed. A full system has been implemented, presented and evaluated, from choosing and placing sensors, to receiving, interpreting and presenting the data to an end-user.

The system presented is directly reusable in any home environment by modifying metadata, is built upon cited design concepts and can serve as a firm foundation for further work in various directions.

Relevant background theory has been studied and presented, and different areas of applicaton of remote monitoring in a health context have been proven or made plausible.

It has been shown that is it possible to develop such a system with limited funds and simple sensors, and that useful information can be extracted from the system and presented intelligently to users.

We believe the presented platform as-is could benefit health personnel and caregivers in a diagnostic capacity as a tool for assessing objective data. With further work aimed at identifying abnormal behaviour or targeting specific health parameters, such a system could increase comfort and security and potentially save lives by assisting detection of critical situations or deteroration of health at an early stage.

Bibliography

[1] AlertMe - Smart Monitoring. `www.alertme.com/`. Retrieved February 16th, 2012.

[2] Ambient Assistant Living Joint Programme. `www.aal-europe.eu`. Retrieved March 3rd, 2012.

[3] Amicom a7201a datasheet. `http://electrohome.pbworks.com/f/A7201+Datasheet+v0.5+(1).pdf`. Retrieved March 20th, 2012.

[4] Android & iOS: 82% Market Share in 1Q 2012. `http://www.gpsbusinessnews.com/Android-iOS-82-Market-Share-in-1Q-2012_a3658.html`. Retrieved June 2nd, 2012.

[5] Android developers: Activity lifecycle figure. `http://developer.android.com/images/activity_lifecycle.png`. Retrieved December 11th, 2011.

[6] Android developers guide: Activities. `http://developer.android.com/guide/topics/fundamentals/activities.html`. Retrieved October 22nd, 2011.

[7] Android developers guide: Content providers. `http://developer.android.com/guide/topics/providers/content-providers.html`. Retrieved October 22nd, 2011.

[8] Android developers guide: Intents and intent filters. `http://developer.android.com/guide/topics/intents/intents-filters.html`. Retrieved October 22nd, 2011.

[9] Android developers guide: Services. `http://developer.android.com/guide/topics/fundamentals/services.html`. Retrieved October 22nd, 2011.

[10] Android developers reference: BroadcastReceiver class. `http://developer.android.com/reference/android/content/BroadcastReceiver.html`. Retrieved October 22nd, 2011.

[11] Android developers reference: Context class. `http://developer.android.com/reference/android/content/Context.html`. Retrieved October 22nd, 2011.

[12] Bluetooth core specifications. `https://www.bluetooth.org/Technical/Specifications/adopted.htm`. Retrieved October 25th, 2011.

[13] Bluetooth fast facts. `http://www.bluetooth.com/Pages/Fast-Facts.aspx`. Retrieved October 25th, 2011.

[14] CARE - Safe Private Homes for Elderly Persons. `http://care-aal.eu/`. Retrieved March 3rd, 2012.

[15] COGNITA FALLOFON, fall sensor with GPS. `http://www.hjelpemiddeldatabasen.no/r11x.asp?linkinfo=21482/`. Retrieved June 1st, 2012.

[16] A comparison between ook/ask and fsk modulation techniques for radio links. `http://cirronetinc.com/products/apnotes/ookvsfsk.pdf`. Retrieved March 15th, 2012.

[17] Core Bluetooth Framework Reference. `https://developer.apple.com/library/ios/documentation/CoreBluetooth/Reference/CoreBluetooth_Framework/CoreBluetooth_Framework.pdf`. Retrieved May 13th, 2012.

[18] For 2012-01-19 nr 77: Forskrift om generelle tillatelser til bruk av frekvenser (fribruksforskriften). `http://www.lovdata.no/cgi-wift/ldles?doc=/sf/sf/sf-20120119-0077.html`. Retrieved February 25th, 2012.

[19] H@H - Health at Home. `www.health-at-home.eu/`. Retrieved March 3rd, 2012.

[20] HOPE - Smart Home for Elderly People. `http://www.hope-project.eu/`. Retrieved March 3rd, 2012.

[21] iOS Developer Library: App States and Multitasking. `https://developer.apple.com/library/ios/#documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/ManagingYourApplicationsFlow/ManagingYourApplicationsFlow.html#//apple_ref/doc/uid/TP40007072-CH4-SW3`. Retrieved April 26th, 2012.

[22] iOS Developer Library: Game Kit. `https://developer.apple.com/library/ios/#documentation/NetworkingInternet/Conceptual/GameKit_Guide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40008304-CH1-SW1`. Retrieved May 13th, 2012.

[23] iOS Developer Library: NSMutableArray Class Reference. https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSMutableArray_Class/Reference/Reference.html. Retrieved May 4th, 2012.

[24] iOS Developer Library: NSNotification Class Reference. https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/NSNotification_Class/Reference/Reference.html. Retrieved May 4th, 2012.

[25] iOS Developer Library: NSObject Class Reference. https://developer.apple.com/library/mac/#documentation/Cocoa/Reference/Foundation/Classes/nsobject_Class/Reference/Reference.html. Retrieved May 4th, 2012.

[26] iOS Developer Library: State changes in an iOS app. https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Art/high_level_flow.jpg. Retrieved April 26th, 2012.

[27] iOS Developer Library: UIApplicationDelegate Protocol Reference. https://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIApplicationDelegate_Protocol/Reference/Reference.html. Retrieved April 26th, 2012.

[28] iOS Developer Library: UIStoryboardSegue Class Reference. https://developer.apple.com/library/ios/#documentation/UIKit/Reference/UIStoryboardSegue_Class/Reference/Reference.html. Retrieved April 26th, 2012.

[29] iOS Developer Library: UITableView Class Reference. http://developer.apple.com/library/ios/#documentation/uikit/reference/UITableView_Class/Reference/Reference.html. Retrieved May 4th, 2012.

[30] iOS Developer Library: UIView Class Reference. http://developer.apple.com/library/ios/#documentation/uikit/reference/uiview_class/uiview/uiview.html. Retrieved April 26th, 2012.

[31] iOS Developer Library: UIViewController Class Reference. http://developer.apple.com/library/ios/#DOCUMENTATION/UIKit/Reference/UIViewController_Class/Reference/Reference.html. Retrieved April 26th, 2012.

[32] Lov 2000-04-14 nr 31: Lov om behandling av personopplysninger (personopplysningsloven). http://www.lovdata.no/all/nl-20000414-031.html/. Retrieved March 3rd, 2012.

[33] Mac OS X Developer Library: Protocols. https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/ObjectiveC/Chapters/ocProtocols.html. Retrieved April 26th, 2012.

[34] Maxim application note 4439. http://www.maxim-ic.com/app-notes/index.mvp/id/4439. Retrieved April 14th, 2012.

[35] New features in Xcode 4.2. http://developer.apple.com/library/mac/#releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011226. Retrieved April 16th, 2012.

[36] Pressure mats for the elderly. http://www.pressuremat.com/. Retrieved May 19th, 2012.

[37] Ray Wenderlich: How to make a simple multiplayer game with game center tutorial. http://www.raywenderlich.com/3276/how-to-make-a-simple-multiplayer-game-with-game-center-tutorial-part-12. Retrieved April 13th, 2012.

[38] Studenter fikk ulovlig tilgang p pasientjournaler. http://www.vg.no/helse/artikkel.php?artid=560054. Retrieved March 14th, 2012.

[39] Texas Instruments CC2540 Bluetooth Low Energy Software Developers Guide v1.1. http://www.ti.com/lit/ug/swru271a/swru271a.pdf. Retrieved Sept 28th, 2011.

[40] TI BLE Vendor Specific HCI Reference Guide. http://www.ti.com/tool/ble-stack. Retrieved April 13th, 2012.

[41] Transitioning to ARC release notes. http://developer.apple.com/library/mac/#releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html#//apple_ref/doc/uid/TP40011226. Retrieved April 16th, 2012.

[42] Velferdsteknologi for hjemmeboende - Bærum kommune. https://www.baerum.kommune.no/Organisasjonen/Pleie--og-omsorg/Velferdsteknologi/. Retrieved June 1st, 2012.

[43] Wikipedia: iOS jailbreaking. http://en.wikipedia.org/wiki/IOS_jailbreaking. Retrieved May 18th, 2012.

[44] Z-Wave Protocol Overview. http://www.eilhk.com/en/product/Datasheet/Zensys/SDS10243-2%20-%20Z-Wave%20Protocol%20Overview.pdf. Retrieved February 16th, 2012.

[45] Hjemme hos fru Paulsen. *NHO magasinet*, 2011.

[46] Daniel Berkvam Hatlevoll. Using bluetooth low energy in sensor devices. Master's thesis, Norwegian University of Science and Technology, 2011.

[47] Shane Conder and Lauren Darcey. *Android Wireless Application Development*. Addison-Wesley, 2nd edition, 2010.

[48] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *HUMAN-COMPUTER INTERACTION*, 16, 2001.

[49] John E. Seem. Using intelligent data analysis to detect abnormal energy consumption in buildings. *Energy and Buildings*, 39(01), 2007.

[50] Marko Gargenta. *Learning Android.* O'Reilly, 1st edition, 2011.

[51] Francine Gemperle, Chris Kasabach, John Stivoric, Malcolm Bauer, and Richard Martin. Design for Wearbility. In *IEEE International Symposium on Wearable Computers*, 1999.

[52] Guha, R. and McCarthy, J. Varieties of contexts. *Modeling and Using Context, pages 164177*, 2003.

[53] H. Karl und A. Willig. *Protocols and Architectures for Wireless Sensor Networks.* Wiley and Sons, 2005.

[54] Hedda Schmidke. Introduction, context sensitive systems (lecture notes). `http://www.teco.edu/lehre/ctx/lecture/slides/CSS-1_Introduction.pdf`. Retrieved Feb 8th, 2012.

[55] Aina Beate Indreiten. Robotselen Paro kjenner igjen Sverre. `http://www.nrk.no/nyheter/distrikt/ostafjells/vestfold/1.7769505`, 2011. Retrieved June 1st 2012.

[56] J. B. MacQueen. Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, (1), 1967.

[57] J. F. Kurose and K. W. Ross. *Computer Networking: A Top-Down Approach.* Pearson Education, 5th edition, 2009.

[58] Kamol Kaemarungsi. *Design of Indoor Posistioning System Based on Location Fingerprinting Technique.* PhD thesis, University of Pittsburgh, 2005.

[59] Karen Henricksen, Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2, 2006.

[60] Patrick Kinney. Zigbee technology: Wireless control that simply works. `http://www.zigbee.org/imwp/idms/popups/pop_download.asp?contentID=5162`. Retrieved September 8th, 2011.

[61] Sebastian Lhr. Recognition of emergent human behaviour in a smart home: A data mining approach. *Pervasive and Mobile Computing*, 03(02), 2007.

[62] Hui Liu, Houshang Darabi, Pat Banerjee, and Jing Liu. Survey of Wireless Indoor Positioning Techniques and Systems. *IEEE Transactions on systems, man, and Cybernetics*, 37(6), 2007.

[63] Magnus Glendrange, Kristian Hove, Espen Hvideberg. Decoding gsm. Master's thesis, Norwegian University of Science and Technology, 2010.

[64] Vandad Nahavandipoor. *iOS 5 Programming Cookbook*. O'Reilly, 1st edition, 2012.

[65] Matt Neuburg. *Programming iOS 4*. O'Reilly, 1st edition, 2011.

[66] Aslak Ringvoll Normann and Marte Elisabeth Bakken Skjønsfjell. An Approach to Networked Welfare Sensing, December 2011.

[67] Bernard Rosner. Percentage Points for a Generalized ESD Many-Outlier Procedure. *Technometrics*, 25(02), 1983.

[68] Studenski S, Perera S, and Patel K et al. Gait Speed and Survival in Older Adults. *Jama*, 305(01), 2011.

[69] Schmidt et. al. Advanced interaction in context. *HUC, pages 89101*, 1999.

[70] Claire Weeks and Andrew Butterworth. *Measuring and Auditing Broiler Welfare*. CABI Publishing, 2004.

Part VI

Appendix

A Enclosed CD

A.1 Contents

- Code — Includes all the code used and developed in this project:
  - Concentrator
  - Interpreter
  - bHome
- Background — Includes the candidates specialization project, as well as datasheets and reference guides.

A.2 Deploying the Code

How to deploy as well as what to change in the different parts of the code is described in the section below.

A.2.1 bHome

In the bHome/bHome.xcodeproj folder, the Xcode project for the application can be found. This can be opened and edited in Xcode and be deployed on an iPad or the simulater. Note that to deploy the application, a Developer Account is required.

For connectivity, the server.py must be running on a computer, and the address for this computer must be modified in the `connect` method in BHNetworkController.m file on line 100.

A.2.2 Concentrator

Users may log in locally on the pandaboard with the user `root` and password `Ekstremt Brun Saus`. When the device is powered, it will initiate a connection to `imbesil.ed.ntnu` and set up a reverse tunnel on port 19022. This can be changed by modifying files in the `/home/tunnel` folder, which are run at startup. Remotely, the user `tunnel` can be used with the password `meget brun saus`.

Python programs which receive wireless data are in `/home/logger`, and `logger` has the password `tomatsuppe`. The relevant programs are

- `serialserver.pyw` — started with `/dev/multitrx` as argument
- `decode_efergy.py` — `/dev/efergy`
- `bledongle.py` — `/dev/ACM0` or `1`

The `/dev/multitrx` and `efergy` node are aliases created by rules in /etc/rules.d/. Normally these programs were run inside of a `screen` session. The MySQL database starts on power-up and can be accessed with the command `mysql -u root -p"brun saus"`.

A.2.3 Interpreter

`server.py` will run the GUI server as well as the framework. Modifications need to be done in `databasereader.py`, line 20, where the connection parameters of the database are set.

`interpreter.py` can be run independently by invoking it on the command line, but this functionality is used mainly for development and debugging and will probably not produce anything useful as presented on the CD.

A.2.4 KeyFob

The KeyFob device may be programmed using IAR Embedded Workbench. The project file for the firmware used in this thesis is found on the CD under `Code/ BLE-CC254x-1.2/ Projects/ ble/KeyFob/ CC2540DB/KeyFobDemo.eww`. The Movement profile is similarly under `Projects/ble/Profiles/MovementProfile` and is included in the above project.

B Android

This section was first featured in Normann/Skjønsfjell[66].

Most of the information in this section is gathered from Android Wireless Application Development[47] and Learning Android[50].

For developing Android applications, a Java IDE, such as Eclipse, is often used. Android Develoment Tools (ADT) is a plug-in for Eclipse to facilitate development and includes the Android SDK , which provides a lot of APIs[1].

Android applications are written in Java, but a Native Development Kit (NDK) can be used to build performance-critical portions of apps in native code languages such as C or C++.

ADT together with Eclipse provides tools for debugging the application, either with a physical device connected to the computer, or by using an Android Virtual Device (AVD) on the provided emulator (seen in Figure B.1).

When the Android application is written, the Android SDK compiles the code and data and resource files into an Android package with the .apk suffix. One single package is one application and can be deployed to an Android device.

Android Platform

The operating system is based on a Linux 2.6 kernel which handles core system services and acts as a HAL[2] between hardware and the Android software stack. Each application has it's own virtual machine (Dalvik[3]), runs in a separate process and has its own unique user ID. Several instances of the Dalvik virtual machine can run concurrently on a device. This can be seen in Figure B.2.

The fact that each application has its own user ID provides security in the Android system. This is because each application can only access its own files, so any user names and passwords stored by one application are not accessible by other applications, much like iOS' sandbox.

[1]Application Programming Interfaces
[2]Hardware Abstraction Layer
[3]Dalvik is the process virtual machine in Android.

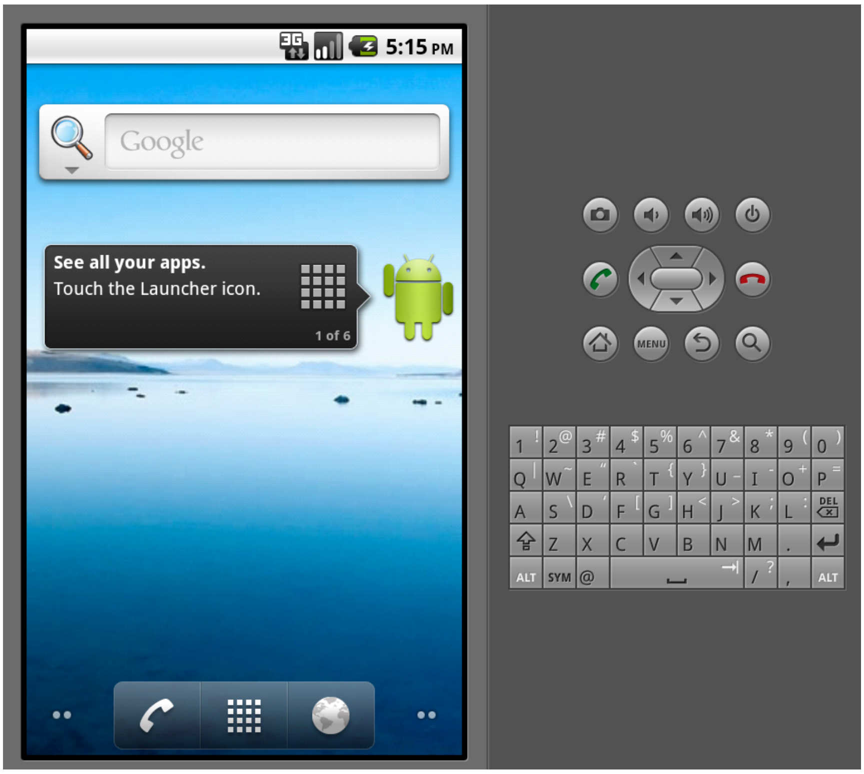


Figure B.1: Emulator

Application Building Blocks

Android has its own terminology for the main building blocks of an application:

- Context: This is the interface to global information about an application environment and it allows access to application-specific functionality such as data and resources[11].

- Activity: A single focused task in an application is called an Activity, and an application is usually a collection of activities. This can be seen as a single screen that the user can see and/or interact with[6].

- Intent: This is Android's asynchronous messaging system and may be used to launch activities and communicate with background services[8].

- Service: Tasks that do not interact with the user or tasks that supply functionality for other applications may be encapsulated into a service. However, services are not separate processes, but a part of an application, and run in the background without any user interface[9].

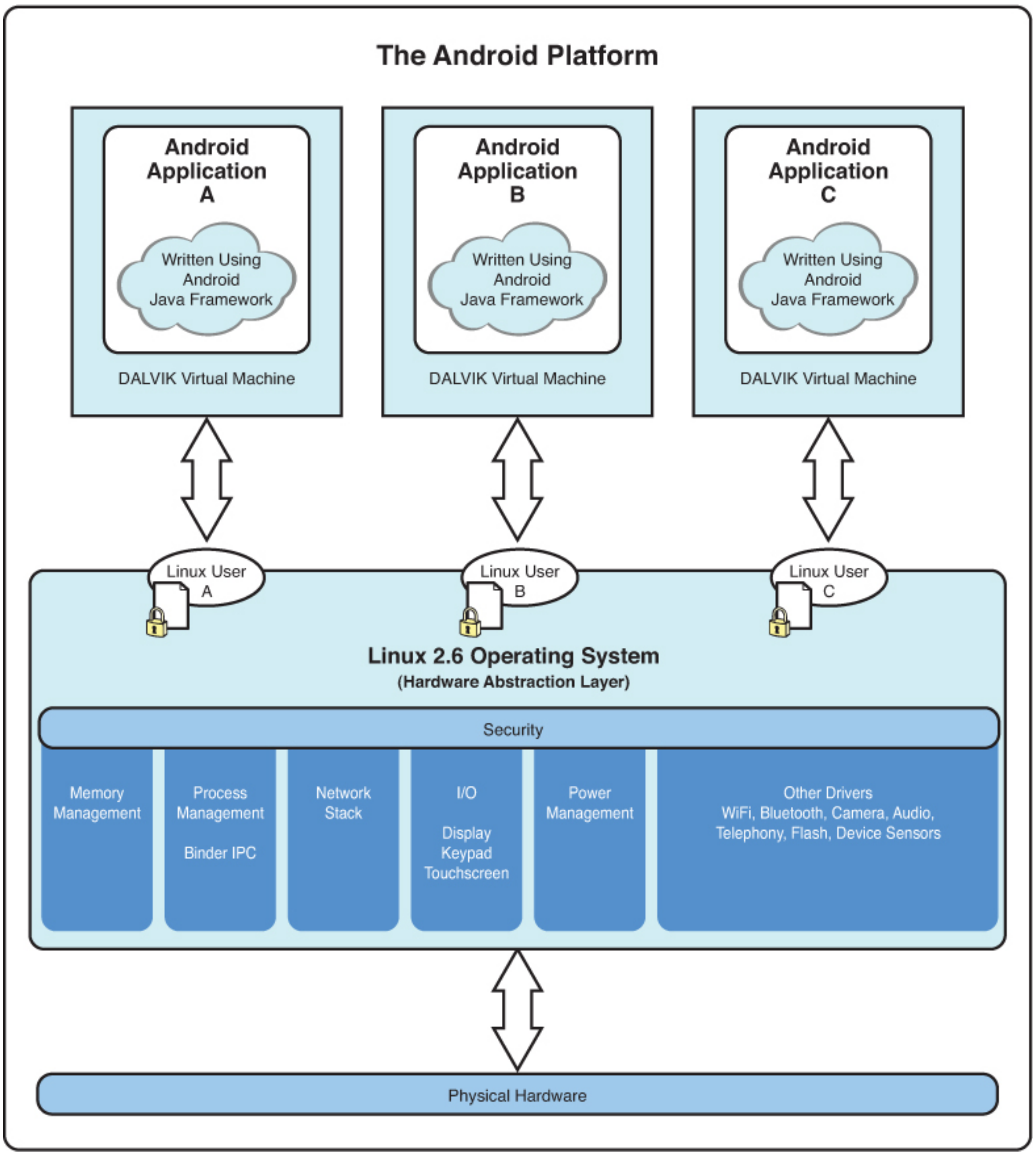


Figure B.2: The Android Platform. From Android Wireless Application Development[47]

- Broadcast Receiver: This is the system wide publisher/subscriber mechanism. A broadcast receiver will receive any broadcast intents it has subscribed to, and react in a predefined manner, such as starting an activity or notifying the user[10].

- Content Provider: To share data between applications, a Content Provider must be used[7]. Severeral Content Providers for common data types such as audio, video and images are provided in Android.

Activity Lifecycle

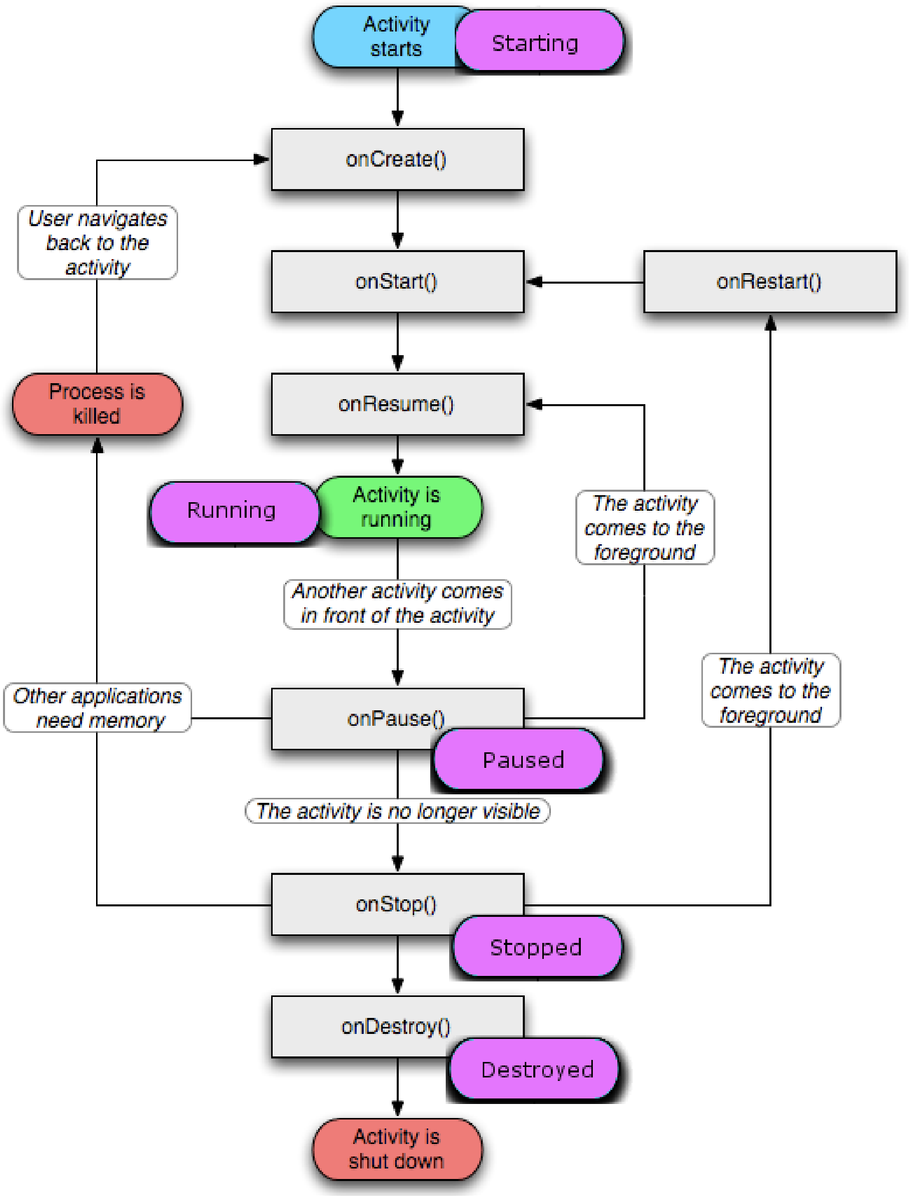
The lifecycle of an Android activity consists of five states which are shown in Figure B.3:

- Starting: When an activity does not exist in memory it goes into the starting state and performs several callback methods going into the running state. This transistion is expensive in terms of computing time, and therefore activities are not destroyed from memory as a rule when they are no longer shown to the user.
- Running: Here the activity is in focus (which means that the user can interact with it) on the screen and is given priority of memory and resources. Only one activity at a time can be in the running state.
- Paused: The activity is still visible on the screen, but is not in focus. All activities must go through this state to be stopped.
- Stopped: Here the activity is not visible, but it is still in the memory, and could be restored to running again.
- Destroyed: When an activity is destroyed, it is no longer in memory.

It is the system that decides when an activity should be destroyed, and this is usually when the user has not interacted with a task for a long time, or when the system needs to free more memory for other more recent tasks.

There are several callback methods that are always called whenever an activity changes states as seen in Figure B.3. These methods must be implemented by the programmer:

- onCreate(): Where the activity is initialized, and where static activity data should be set up.
- onRestart(): Called if the activity is being displayed to the user from the Stopped state.
- onStart(): When the activity is becoming visible to the user, this method is called.
- onResume(): Called when the activity is entering the Running state. This is where retrieval of instances to resources should happen.
- onPause(): Called when the activity is placed in the background. This is a good place to release and save activity resources, as well as stopping CPU intensive processes, such as threads, audio playing etc. The more resources released here, the less likely the activity is to be killed while in the background.


Activity starts
Starting
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Running
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
Paused
The activity is no longer visible
onStop()
Stopped
onDestroy()
Destroyed
Activity is shut down

[a]Figure modified from [5]

Figure B.3: Activity life cycle [a]

- onStop(): Called when the activity is no longer visible to the user.

- onDestroy(): Called when the activity is being destroyed, either because it has completed its lifecycle, or it's being killed by the system to free memory. Here, all static activity data should be released.

Both the onPause(), onStop() and onDestroy() methods are killable, which means that the system could kill the hosting process after any of these methods have returned, without executing the remainding methods. Therefore, all saving of user data must be done on the onPause() method to be sure this is done.

Resources

Android resources consist of everything that is not code. This could be images, audio and xml files.

Layout The layout of a screen is specified in a layout xml file.

Drawables In Eclipse, images can simply be added to the res/drawable directory. Preferred file format is PNG (Portable Network Graphics). Graphics that are indepentent of screen density can be but into this folder, but there are three other folders to add graphics to:

- /res/drawable-hdpi: For high-density screen devices
- /res/drawable-mdpi: For medium-density screen devices
- /res/drawable-ldpi: For low-density screen devices

Values Constant values to be used in the application can be added in xml files located in a res/values directory in the application project. This could be strings, integers, colors etc.

Providing strings as resources, in contrast to hardcoding them, makes it easier to extend the application to different languages for instance. This can be done by adding another string.xml file to a folder named values- and then a two-letter ISO 639-1 language code[4]. If the file is located in a res/values-fr/ folder, this file will be used if the device language is set to French. If the device language is something else, the default values from res/values will be used instead.

Menu Menus can be defined in a menu resource to separate the content for the menu from application code. The different types of menus that can be defined are

- Context menu: A floating list of menu items that inflates when an item is long-clicked
- Options menu: A menu that inflates when the menu button is pushed
- Submenu: A menu that inflates when a menu item that contains a nested menu is selected

[4]ISO 639-1 language codes can be found here: `http://www.loc.gov/standards/iso639-2/php/code_list.php`

R.java The R.java file is an index into all the resources defined in the xml files and allows the resources to be accessed programatically. This file is automatically generated by Eclipse whenever a resource xml file is updated, or a drawable is added.

Manifest Each Android application needs a manifest. This is an xml configuration file that glues the application together. It presents essential information about the application to the Android system and includes application name and components (such as activities, services, broadcast receivers and content providers), permissions the application requires, libraries to be linked against, hardware required, as well as the minimum level of the Android API required by the app.

The information in the manifest is also used by the Android Market to select which applications to display to different devices based on the minimum level of Android API required and hardware requirements.

C

Cost of the System

Product	**Price**	**Amount**
Nexa PIR motion detector	229,-	9
Nexa magnet switch	161,-	13
Efergy Energy monitor	600,-	1
MultiTRX	450,-	1
Total	5204,-	

Table C.1: Price of the implemented system