CHALMERS
UNIVERSITY OF TECHNOLOGY

IoT Communication Protocols in Healthcare

Master's thesis in Computer Science

ANDREAS SVANSTRÖM

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

IoT Communication Protocols in Healthcare

ANDREAS SVANSTRÖM

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
GOTHENBURG, SWEDEN 2016

IoT Communication Protocols in Healthcare
ANDREAS SVANSTRÖM

Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31 – 772 1000

Gothenburg, Sweden 2016

IoT Communication Protocols in Healthcare
ANDREAS SVANSTRÖM
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

This report is the result of a master's thesis with the goal of investigating which are the most commonly used Internet of Things protocols in healthcare and creating a working prototype for integration of at least one of them.

The investigation made it clear that the only serious attempt at standardising communication amongst medical devices in the IoT realm is the ISO/IEEE 11073 protocols family. Thus the integration is made to support this protocol family.

The definition, creation and testing of the integration are described in this report. Through the integration, Bluetooth enabled personal health devices using the ISO/IEEE 11073-20601 messaging protocol can communicate with web services, using the Open Data protocol.

The integration consists of an Android application functioning as a gateway for the personal health devices and an example web service that can receive measurements from pulse oximeters and store them in a database. The gateway application is built using an MVC pattern, to make it easy to modify and extend, and it relies on the Antidote library to decode the ISO/IEEE 11073-20601 messages. Because the web service uses the Open Data protocol, which includes an easy-to-use querying interface, it is easy to use it for further integration.

The system definition focuses on healthcare use and is trying to be as complete as possible for a project of this size, whereas the implemented gateway and web service software serve as a proof-of-concept. They show the possibility to integrate IoT pulse oximeters with a web service using an open protocol. This makes the documentation, in the form of scenarios, user stories and requirements, a good basis for developing a full-fledged integration, supporting all device specialisations, and the proof-of-concept a good base for this full integration.

Keywords: IoT, Internet of Things, Healthcare, Bluetooth, Communication, Connected Health, Pulse oximetry, OData, Open Data, ISO 11073, IEEE 11073, CEN 11073

Acknowledgements

This report is the result of a master's thesis work at the department of Computer Science and Engineering at Chalmers University of Technology performed in cooperation with Ascom Wireless Solutions in Gothenburg.

I would like to thank my advisors at Ascom, Mats Andreasen and Jan Bentzer, for their support and for giving me new points of view on the subject. I would also like to thank my supervisor at Chalmers, K.V.S. Prasad, for his support and constructive feedback.

Contents

1. Introduction

1.1 Motivation

Enabling the use of small, Internet of Things-enabled health sensors in healthcare would give several benefits for everyone involved. The patient would benefit from not having to be tied-up to big stationary sensors and thereby be more mobile and feel more comfortable. Patients may also be able to return home at an earlier stage, still being remotely monitored, which could heighten the perceived quality of life for the patient. The hospital would benefit from lower costs, both because these small devices are relatively cheap and also because hospitalisation is expensive, for example the average cost of an excess bed day in the British healthcare system in 2012-2013 was £273 (Department of Health, 2013).

The first attempt to standardise a communication protocol for such devices is the CEN ISO/IEEE 11073 standards family that was announced by ISO an IEEE in 2010 (Seo, Kim, Lee, & Kim, 2014). Reading the standard documents and articles regarding the standards gives a picture of a mature messaging protocol, but there also seems to be a gap in integrations including the full chain from personal health device to hospital information system or electronic health record, a gap which this thesis aims to fill.

1.2 Objectives

The aim of this thesis is to investigate the market of Internet of Things-enabled medical devices and find which communication protocols are common or seem promising. From there on, the objective is to create a working prototype of an integration of such a protocol.

1.3 Method

The method for achieving the objectives of this thesis can be divided into four different phases, namely the information phase, the design phase, the implementation and testing phase and the report phase.

During the information phase, information is first gathered by literature studies and market searches for available devices, this information will then be analysed to decide upon a protocol to use for the integration prototype.

The design phase consists of designing and defining the prototype in terms of scenarios, user stories, and acceptance criteria. The acceptance criteria will be compiled into requirements, from which a test plan will be written. Different ways to realise the prototype will be evaluated and a preferred one will be chosen.

In the implementation and testing phase the prototype will be developed and also tested, to see that it passes the tests written during the design phase, and thereby lives up to the requirements, which will be an indication that the prototype is actually what was intended during the design phase.

Lastly the report phase consists of reporting about the findings of the information phase, the decisions of the design phase and the results of the implementation and testing phase.

1.4 Skipping tags

Some chapters go much into detail and are not necessary to read for getting an overview of the project, but are intended for those readers interested in specific details. Such chapters are tagged with a tag looking like this:

[This chapter can be skipped on a first reading]

They regard the tagged chapter including any subchapters, meaning that if chapter X is tagged it includes all chapters X.*, or if chapter X.Y is tagged it includes all chapters X.Y.* but not for example X.X or X.Z.*. Those chapters are mainly slight modifications of documents written as part of the development process during the design phase.

2. Background

2.1 Ascom Wireless Solutions

Ascom Wireless Solutions is a company which develops on-site wireless communications solutions, its customers are located all over the world. Product types include purpose-built handsets, wireless voice and message transmission systems and customised alarm and positioning applications. Customers can be found in areas such as hospitals, elderly care, industry, retail sector, secure establishments and hotels (Ascom, 2015).

2.2 Internet of Things

The Internet of Things, often abbreviated IoT, is the notion of so called "things" being interconnected via the Internet, enabling them to exchange information. The function of IoT is thus to reduce the gap between real world objects and their virtual representations in information systems (Weber & Weber, 2010).

These things can be any type of object able to generate or consume information (Weber & Weber, 2010), an example is cars that can register slippery road parts and traffic jams and inform other cars or local authorities about these situations (Volvo Car Group, 2015). There are also examples of networks of sensors and actuators working together through a smart middleware or server software, controlling for example temperature or lighting (Castellani, et al., 2010).

In a forecast made by Cisco in 2013, they state that 8.7 billion devices were connected to the Internet in 2012 and that the number had exceeded 10 billion in 2013 already. They expect that the number of connected devices in 2020 will be 50 billion, and that 20 billion of those will get connected during the last three years of the period (Cisco, 2013). A more recent forecast by Ericsson expects 26 billion devices to be connected to the Internet in 2020 (Ericsson, 2015), though back in 2012 they also had a vision of 50 billion connected devices in 2020 (Höller & Arkko, 2012). In either way it seems like the amount of Internet connections are increasing at a high pace and that connecting things is a big part of this increase.

In the realm of healthcare, IoT devices are expected to change the overall way that healthcare works, by decentralising care. Chronic disease patients are expected to be remotely monitored by small sensor devices, making it possible to detect bad conditions before they get really bad and thereby avoid hospitalisation in many cases. Also doctoral consultations are anticipated to partly take place remotely in the future, and together these two changes to healthcare are expected to lower the costs of healthcare with more than 20% in the United States (Roman, et al., 2015).

2.3 Setting the scope

As the scope of the project proposal included searching the market for common and/or promising communication protocol standards, the first thing that was done was collecting information about available, commonly used and future communication protocols used by IoT health devices. The information gathered was used for setting the scope of the proof-of-concept project part.

The method used for finding out about possible existing protocols was a combination of database searches in different scientific databases provided by the Chalmers library, mostly Scopus because of its size and the fact that it offers literature from different fields, including e.g. technology and medicine, with google searches for existing connected medical or health devices, both on the consumer and corporate markets. Keywords used at the beginning included "IoT", "Internet of Things", "healthcare", "health", "connected" and "medicine". As more information was gathered, more specific keywords, such as "personal health device", "PHD", "Continua" and "IEEE 11073", came up.

While reading articles and looking at data sheets for different existing products, it became clear that proprietary protocols are very common, and sometimes even different models of the same type of device from the same manufacturer use different protocols for communication (Day, 2011).

What also came out of the information gathering was the fact that there is a message exchange protocol standard for communication between health devices intended for personal use, so called Personal Health Devices, or PHDs. This standard was announced in 2010 by IEEE and ISO, and is called ISO/IEEE 11073-20601 (Seo, Kim, Lee, & Kim, 2014). The standard only defines the message data structure and is transport-independent (The Institute of Electrical and Electronics Engineers, Inc., 2014). On top of the exchange protocol are device specialisations, i.e. descriptions of message structure for data originating from different types of PHD agents, such as blood pressure monitors, pedometers or weighing scales (The Institute of Electrical and Electronics Engineers, Inc., 2014).

Except being accepted by ISO, the 11073-20601 standard, and its specialisations, by IEEE has also been accepted by CEN (European Committee for Standardization) (CEN, 2016) and the Bluetooth Special Interest Group has chosen the ISO/IEEE 11073 standards family as the protocols to use when developing a Bluetooth device using their Health Device Profile (Bluetooth SIG, 2016).

When searching for devices using the ISO/IEEE 11073 protocol family for communication, a logo saying "Continua Certified" accompanied most of those products. Further investigation showed that Continua is an international non-profit industry group composed of 122 member companies, working for plug-and-play compatibility for personal connected health devices (Personal Connected Health Alliance, 2015). One requirement for getting a product certified is that it complies with the ISO/IEEE 11073 protocol family, and the Continua web page includes a list of certified products (http://www.continuaalliance.org/products/product-showcase). When checking this list, it's easy to state that most certified products (in the end of 2015 that is) are manager software programs for Microsoft Windows, i.e. programs intended for receiving data from health agents (sensors). There are however certified agent products as well, of multiple device types (e.g. blood pressure monitors, pulse oximeters and thermometers).

Considering that the ISO/IEEE 11073 protocol family seems to be an active set of standards and the only serious attempt on standardising communication between personal health devices, as well as it being backed by a large amount of companies,

including several leading companies in the healthcare industry, in the form of the Continua Alliance, it seems to be a good choice for a protocol standard to use in the proof-of concept. The fact that there already are products using this protocol on the market also speaks for choosing it for the proof-of-concept, as this makes the product testable.

Different solutions were discussed, and in the end it was decided that the proof-of-concept shall be an Android application functioning as a gateway between devices talking ISO/IEEE 11073-20601 over Bluetooth transport and the Ascom Unite messaging system. This decision was partially changed in the end of the development phase, after the realisation that the data generated is pure health data, and not alarms, which usually is the case when using Ascom Unite in healthcare. The receiving end was then changed into a standalone web service talking the Open Data protocol. This web service was written in C#/.NET, to enable easy integration into Ascom Unite later on, if that would be wanted.

2.4 The ISO/IEEE 11073 standards family

The ISO/IEEE 11073 standards family consists of a plethora of standards, including different transport profiles, nomenclatures, an overview and application profiles for example. Most of them are not of interest for completing this project however. The ones relevant for realising this project is the previously mentioned exchange protocol (ISO/IEEE 11073-20601) and the different device specialisations (ISO/IEEE 11073-104xx), more specifically the pulse oximeter specialisation, namely ISO/IEEE 11073-10404, as a device of this type was the only one used for testing the proof-of-concept implementation.

In the ISO/IEEE 11073-20601 standard, two types of devices are defined, namely managers and agents. An agent device is typically a sensor unit, e.g. a blood pressure monitor, a glucose meter or a pulse oximeter and generally it communicates only with a single manager at any arbitrary point in time. A manager device is a unit which typically receives data from agents and can communicate simultaneously with multiple agent devices at the same time (The Institute of Electrical and Electronics Engineers, Inc., 2014). Thus the gateway application created in this project is an ISO/IEEE 11073-20601 manager.

A connection between an agent and a manager is typically initiated by the agent when it has new data to send to the manager, though there are exceptions from this pattern, for example when the agent has a persistent metric store (often abbreviated PM-store). To read the stored data, the manager will send a get message to the agent (The Institute of Electrical and Electronics Engineers, Inc., 2014).

When it comes to security, the ISO/IEEE 11073-20601 standard fully relies on other layers to secure the communication, e.g. securing the transport channel (The Institute of Electrical and Electronics Engineers, Inc., 2014).

A neat feature of this standard is that if the manager doesn't already know the device configuration that the agent wants to use, it can request that the both enter the “Configuring state”, in which the agent will send a “Configuration event report”, containing a description of all objects included in the agent’s device configuration as

well as an identification number for the configuration (The Institute of Electrical and Electronics Engineers, Inc., 2014). Thus a correctly implemented manager is future compatible as long as it is updated when the ISO/IEEE 11073-20601 protocol is.

2.4.1 Available transport protocols

As mentioned before, the ISO/IEEE 11073-20601 protocol is transport independent, though there are some well-defined ways of doing the transport, such as the three transport profiles defined by IEEE: cable connected (ISO/IEEE 11073-30200), infrared (ISO/IEEE 11073-30300) and cabled Ethernet (ISO/IEEE 11073-30400). Also there is the Bluetooth Health Device Profile, which is used by Bluetooth enabled PHDs, and the ZigBee Health Care profile for ZigBee enabled PHDs.

There are already PHDs on the market communicating ISO/IEEE 11073 over USB, Bluetooth and ZigBee (Personal Connected Health Alliance, 2015), where USB connection is cabled and Bluetooth and ZigBee are known for being low-power wireless communication protocols. Wi-Fi connections are usually not the first choice for IoT implementations, as Wi-Fi chips traditionally have been expensive and not very low-power, and they also have a longer connection setup time than the competition, though this might change in future (Mathias, 2015). The Antidote library, an ISO/IEEE 11073-20601 implementation, which will be described more later on in the report, also includes a plug-in for communicating ISO/IEEE 11073-20601 over TCP/IP (Livio, et al., 2012).

The upcoming IEEE 802.11ah standard, expected to be finished in March 2016, will address several of the aforementioned disadvantages for Wi-Fi, e.g. by introducing sub 1 GHz channels, narrower channel bandwidths and longer sleep intervals (Larmo, 2015). Thus Wi-Fi enabled PHDs may turn up on the market during the coming years.

2.5 The Open Data protocol

The Open Data protocol, often abbreviated OData, is an open protocol that allows creating and consuming queryable APIs following the REST principles. It is standardised by OASIS and queries can be made in a database like manner (OData, 2015).

The choice to use OData for the receiving backend was made because it is easy to work with, commonly used and easy to integrate in other applications as there are OData libraries available for major platforms and languages like .NET, Java, C++ and JavaScript (OData, 2015). Major spreadsheet software like Microsoft Excel and LibreOffice Calc are also able to handle OData queries, thereby making the data directly usable by anyone knowing how to use regular office suite programs.

For example if one would like to look at all pulse oximetry measurements saved in the web service's database, with a heart rate value between 39-45 bpm exclusive, and sort them by patient, one would simply make a GET request to the web service, looking like this:

```
http://localhost/IoTREST/odata/PulseOximetryMeasurements/?
$filter=HeartRate gt 39 and HeartRate lt 45&$orderby=
PatientIdentification
```

Which very much resembles how an SQL query for the same information would look like:

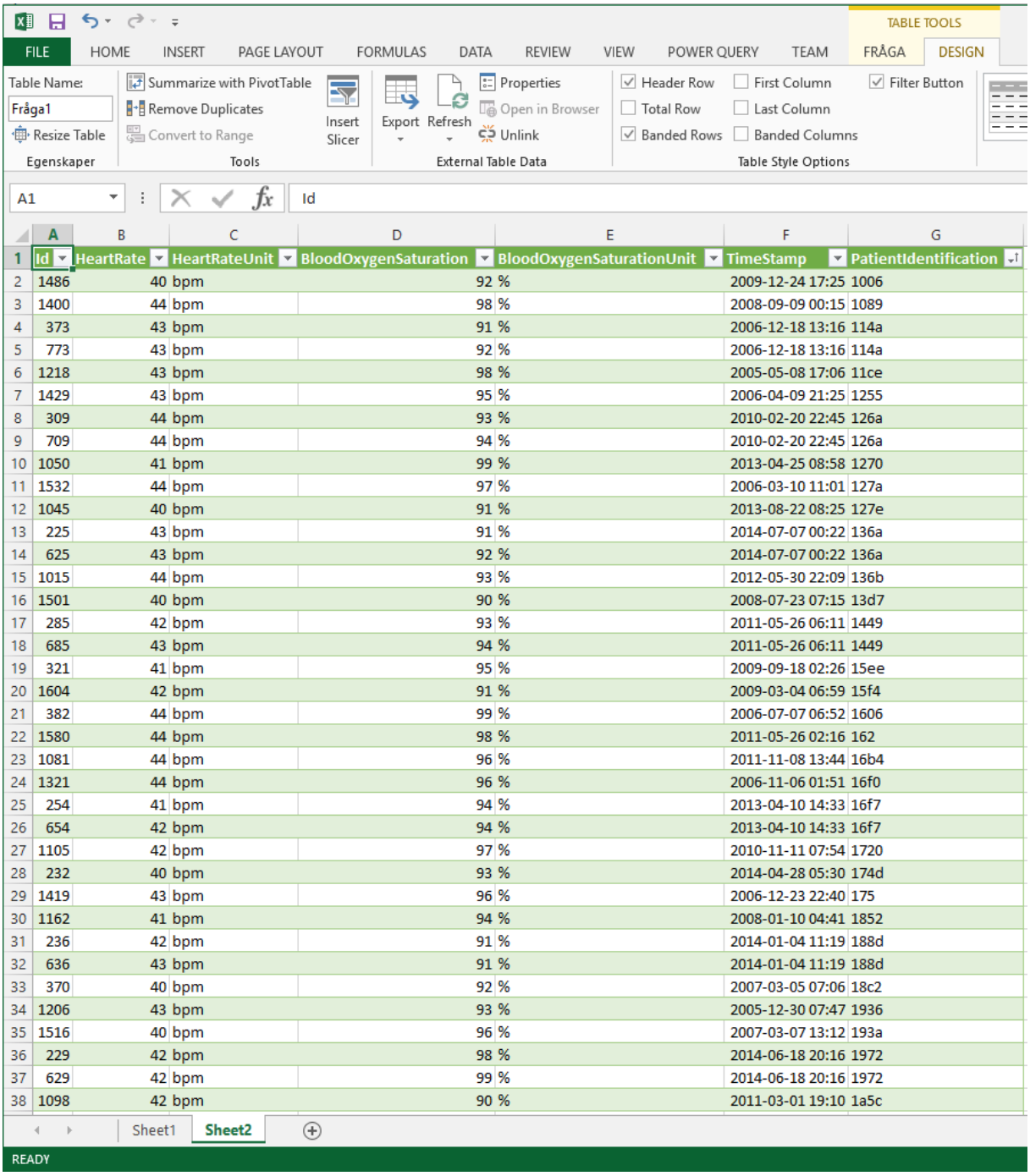
```
SELECT * FROM PulseOximetryMeasurements WHERE
HeartRate > 39 AND HeartRate < 45 ORDER BY
PatientIdentification;
```

Running this query in a browser results in a textual representation of a JSON object list with the corresponding rows from the database, for the actual query above, the beginning of it would look like this:


```
{
  "odata.metadata":"http://localhost/IoTREST/odata/$metadata#PulseOximetryMeasurements","value":[
    {
      "Id":1486,"HeartRate":40,"HeartRateUnit":"bpm","BloodOxygenSaturation":92,"BloodOxygenSaturationUnit":"%","TimeStamp":"2009-12-24T17:25:00","PatientIdentification":"1006"
    },{
      "Id":1400,"HeartRate":44,"HeartRateUnit":"bpm","BloodOxygenSaturation":98,"BloodOxygenSaturationUnit":"%","TimeStamp":"2008-09-09T00:15:00","PatientIdentification":"1089"
    },{
      "Id":373,"HeartRate":43,"HeartRateUnit":"bpm","BloodOxygenSaturation":91,"BloodOxygenSaturationUnit":"%","TimeStamp":"2006-12-18T13:16:00","PatientIdentification":"114a"
    },{
      "Id":573,"HeartRate":43,"HeartRateUnit":"bpm","BloodOxygenSaturation":91,"BloodOxygenSaturationUnit":"%","TimeStamp":"2006-12-18T13:16:00","PatientIdentification":"114a"
    },{
      "Id":773,"HeartRate":43.41555,"HeartRateUnit":"bpm","BloodOxygenSaturation":91.6492844,"BloodOxygenSaturationUnit":"%","TimeStamp":"2006-12-18T13:16:00","PatientIdentification":"114a"
    },{
      "Id":973,"HeartRate":43.41555,"HeartRateUnit":"bpm","BloodOxygenSaturation":91.6492844,"BloodOxygenSaturationUnit":"%","TimeStamp":"2006-12-18T13:16:00","PatientIdentification":"114a"
    },{
      "Id":1218,"HeartRate":43,"HeartRateUnit":"bpm","BloodOxygenSaturation":98,"BloodOxygenSaturationUnit":"%","TimeStamp":"2005-05-08T17:06:00","PatientIdentification":"11ce"
    },{
      "Id":1429,"HeartRate":43,"HeartRateUnit":"bpm","BloodOxygenSaturation":95,"BloodOxygenSaturationUnit":"%","TimeStamp":"2006-04-09T21:25:00","PatientIdentification":"1255"
    },{
      "Id":909,"HeartRate":44.2765732,"HeartRateUnit":"bpm","BloodOxygenSaturation":93.5415649,"BloodOxygenSaturationUnit":"%","TimeStamp":"2010-02-20T22:45:00","PatientIdentification":"126a"
    },{
      "Id":709,"HeartRate":44.2765732,"HeartRateUnit":"bpm","BloodOxygenSaturation":93.5415649,"BloodOxygenSaturationUnit":"%","TimeStamp":"2010-02-20T22:45:00","PatientIdentification":"126a"
    },{
      "Id":509,"HeartRate":44,"HeartRateUnit":"bpm","BloodOxygenSaturation":93,"BloodOxygenSaturationUnit":"%","TimeStamp":"2010-02-20T22:45:00","PatientIdentification":"126a"
    },{
      "Id":309,"HeartRate":44,"HeartRateUnit":"bpm","BloodOxygenSaturation":93,"BloodOxygenSaturationUnit":"%","TimeStamp":"2010-02-20T22:45:00","PatientIdentification":"126a"
    },{
      "Id":1050,"HeartRate":41,"HeartRateUnit":"bpm","BloodOxygenSaturation":99,"BloodOxygenSaturationUnit":"%","TimeStamp":"2013-04-25T08:58:00","PatientIdentification":"1270"
    },{
      "Id":1532,"HeartRate":44,"HeartRateUnit":"bpm","BloodOxygenSaturation":97,"BloodOxygenSaturationUnit":"%","TimeStamp":"2006-03-10T11:01:00","PatientIdentification":"127a"
    },{
      "Id":1045,"HeartRate":40,"HeartRateUnit":"bpm","BloodOxygenSaturation":91,"BloodOxygenSaturationUnit":"%","TimeStamp":"2013-08-22T08:25:00","PatientIdentification":"127e"
    },{
      "Id":825,"HeartRate":43.35913,"HeartRateUnit":"bpm","BloodOxygenSaturation":91.96587,"BloodOxygenSaturationUnit":"%","TimeStamp":"2014-07-07T00:22:00","PatientIdentification":"136a"
```


Figure 1 Part of return from querying the OData interface of the web service

Running the same query in Microsoft Excel's PowerQuery tool gives the following result:



Id	HeartRate	HeartRateUnit	BloodOxygenSaturation	BloodOxygenSaturationUnit	TimeStamp	PatientIdentification
1486	40	bpm	92	%	2009-12-24 17:25	1006
1400	44	bpm	98	%	2008-09-09 00:15	1089
373	43	bpm	91	%	2006-12-18 13:16	114a
773	43	bpm	92	%	2006-12-18 13:16	114a
1218	43	bpm	98	%	2005-05-08 17:06	11ce
1429	43	bpm	95	%	2006-04-09 21:25	1255
309	44	bpm	93	%	2010-02-20 22:45	126a
709	44	bpm	94	%	2010-02-20 22:45	126a
1050	41	bpm	99	%	2013-04-25 08:58	1270
1532	44	bpm	97	%	2006-03-10 11:01	127a
1045	40	bpm	91	%	2013-08-22 08:25	127e
225	43	bpm	91	%	2014-07-07 00:22	136a
625	43	bpm	92	%	2014-07-07 00:22	136a
1015	44	bpm	93	%	2012-05-30 22:09	136b
1501	40	bpm	90	%	2008-07-23 07:15	13d7
285	42	bpm	93	%	2011-05-26 06:11	1449
685	43	bpm	94	%	2011-05-26 06:11	1449
321	41	bpm	95	%	2009-09-18 02:26	15ee
1604	42	bpm	91	%	2009-03-04 06:59	15f4
382	44	bpm	99	%	2006-07-07 06:52	1606
1580	44	bpm	98	%	2011-05-26 02:16	162
1081	44	bpm	96	%	2011-11-08 13:44	16b4
1321	44	bpm	96	%	2006-11-06 01:51	16f0
254	41	bpm	94	%	2013-04-10 14:33	16f7
654	42	bpm	94	%	2013-04-10 14:33	16f7
1105	42	bpm	97	%	2010-11-11 07:54	1720
232	40	bpm	93	%	2014-04-28 05:30	174d
1419	43	bpm	96	%	2006-12-23 22:40	175
1162	41	bpm	94	%	2008-01-10 04:41	1852
236	42	bpm	91	%	2014-01-04 11:19	188d
636	43	bpm	91	%	2014-01-04 11:19	188d
370	40	bpm	92	%	2007-03-05 07:06	18c2
1206	43	bpm	93	%	2005-12-30 07:47	1936
1516	40	bpm	96	%	2007-03-07 13:12	193a
229	42	bpm	98	%	2014-06-18 20:16	1972
629	42	bpm	99	%	2014-06-18 20:16	1972
1098	42	bpm	90	%	2011-03-01 19:10	1a5c

Figure 2 Running a query against the OData interface of the web service in Excel

Lastly, if one would like to do something with the newly imported data, one could for example count occurrences of different blood oxygen saturation levels to get an idea of what the most common blood oxygen saturation level was for the measurement set, when the pulse was between 40 and 44.

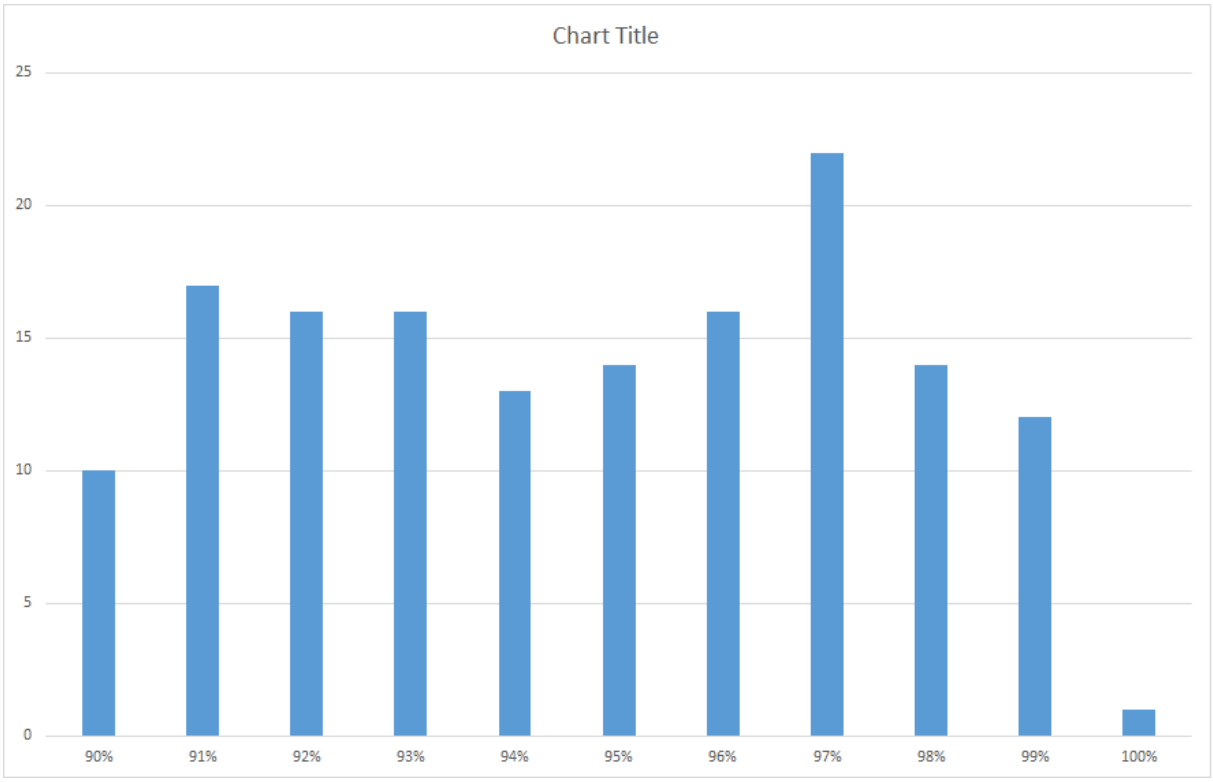


Figure 3 Frequency chart for blood oxygen saturation levels from a query against the web service's OData interface in Excel

This might not bring much direct value to this project, but it leaves openings for the future, for example enabling easy data set generation for big data and machine learning projects.

Please note that most of the data that was returned from the query used in the examples above had been pseudo-randomly generated by a feature in the gateway application, intended for generating data to test the gateway – web service connection. This means that the data seen in the figures does not represent real medical information.

2.6 Bluetooth

Bluetooth is a wireless data exchange technology created in 1994 as an alternative to data cables. The name is an English translation of the last name of the 10th century Viking king Harald Blåtand, who is famous for having united warring factions in what today are parts of Norway, Sweden and Denmark. This is a metaphor for Bluetooth technology being an open standard, enabling disparate products and industries to connect and collaborate (Bluetooth SIG, 2016).

Bluetooth devices are divided into three different power classes, depending on their maximum output power level at the antenna connector, the three maximum output power levels are 1, 2.5 and 100 mW (Bluetooth SIG, 2007), with corresponding transmission ranges of about 1, 10 and 100 metres (Bluetooth SIG, 2011).

To avoid interference and fading, Bluetooth transceivers use frequency hopping (Bluetooth SIG, 2007). Bluetooth has 79 different channels to hop between, up to a maximum of 1600 hops/s in connection state, and the hopping sequence is pseudo-

randomly generated with part of the device addresses as random seed (Bluetooth SIG, 2007).

For two Bluetooth devices to be able to communicate with each other, they first need to pair with each other (Bluetooth SIG, 2016). When two devices pair, they exchange cryptographic information in order to enable encrypted communication, making eavesdropping attacks harder. There are four different association models that can be used when pairing, three of them includes authentication through a six digit number, accepted or entered, to protect against man-in-the-middle attacks (Bluetooth SIG, 2007).

Since version 4.0 of Bluetooth, there is also another standard called Bluetooth LE (Low Energy) or Bluetooth Smart, developed for IoT applications specifically (Bluetooth SIG, 2016). This new standard does not have the same characteristics as described above, and it is also not covered in this report because the classic Bluetooth standard was the one used by the devices in this project.

2.7 Ascom Unite

Ascom Unite is a messaging system developed by Ascom, it consists of three main parts: Connect, Core and Axess. Connect is a collector name for all in-data interfaces, and in healthcare this corresponds to interfaces that receive alarms from patient monitors, patient alarm buttons etc. Core is a message broker that handles all the incoming messages, e.g. prioritises alarms and decides to whom a message should be sent. Axess is a collector name for all out-data interfaces that deliver messages to their recipients, such interfaces could send messages to e.g. cell phones, pagers or email addresses (Bentzer, 2016).

The proof-of-concept developed as part of this project doesn't make use of Unite in any way, but was intended to during the design phase, hence Unite and Unite Connect are referenced in chapter 3. Design.

3. Design

A reader who wants to get a good idea of the process but doesn't want to get all the details can read the overview and one of the detailed scenarios together with its user stories and acceptance criteria and skim through the requirements. A reader who isn't interested in the details at all can skip everything after the overview.

This chapter mainly consists of documentation written in order to define the proof-of-concept before it was implemented. At a first glance this documentation can seem informal or maybe even incomplete, but it was a thought-through decision to write this way, and it does include everything needed to understand how the application should work. The two biggest benefits coming from this way of writing is the short start-time and the easiness to understand the documentation. This way of writing is also a form of a simplified version of the method used at Ascom.

It is proposed in the book User Stories Applied that this kind of documentation is both more effective and efficient than traditional lengthy requirements documentation (Cohn, 2004). Also in the book Managing Software Requirements, it is suggested that many crucial software system requirements can be written in plain language, making them understandable for "ordinary" people, in the same way as blueprints for houses are drawn in a way that the house buyer can understand what they will get (Leffingwell & Widrig, 2000).

The documentation aims to describe a more developed product than the proof-of-concept, not necessarily a finished product, but one that has more functionality than the proof-of-concept.

When designing this software, a brainstorming session on where and when it could be used was held. This resulted in four scenarios, and a brainstorming session for what could be reasonable user stories for those scenarios was held as well. The input hence builds on own experiences, stories from friends who work in healthcare as well as market knowledge from the advisors at Ascom.

For each user story a number of acceptance criteria has been written, these have later on been reviewed and discussed, regarding their completeness and reasonableness.

Lastly all acceptance criteria have been compiled into more formal requirements, and all together this documentation describes how the software is intended to work.

As mentioned in the background chapter, a decision was made to switch the receiving backend part from Ascom's messaging system Unite to a simple web service, but the text in this chapter still refers to the receiving end as Unite, because that better reflects the intentions that were current during the design phase of the project. Likewise the not yet implemented collector unit is still documented, because it still is a part of thought-of functionality.

3.1 Scenarios overview

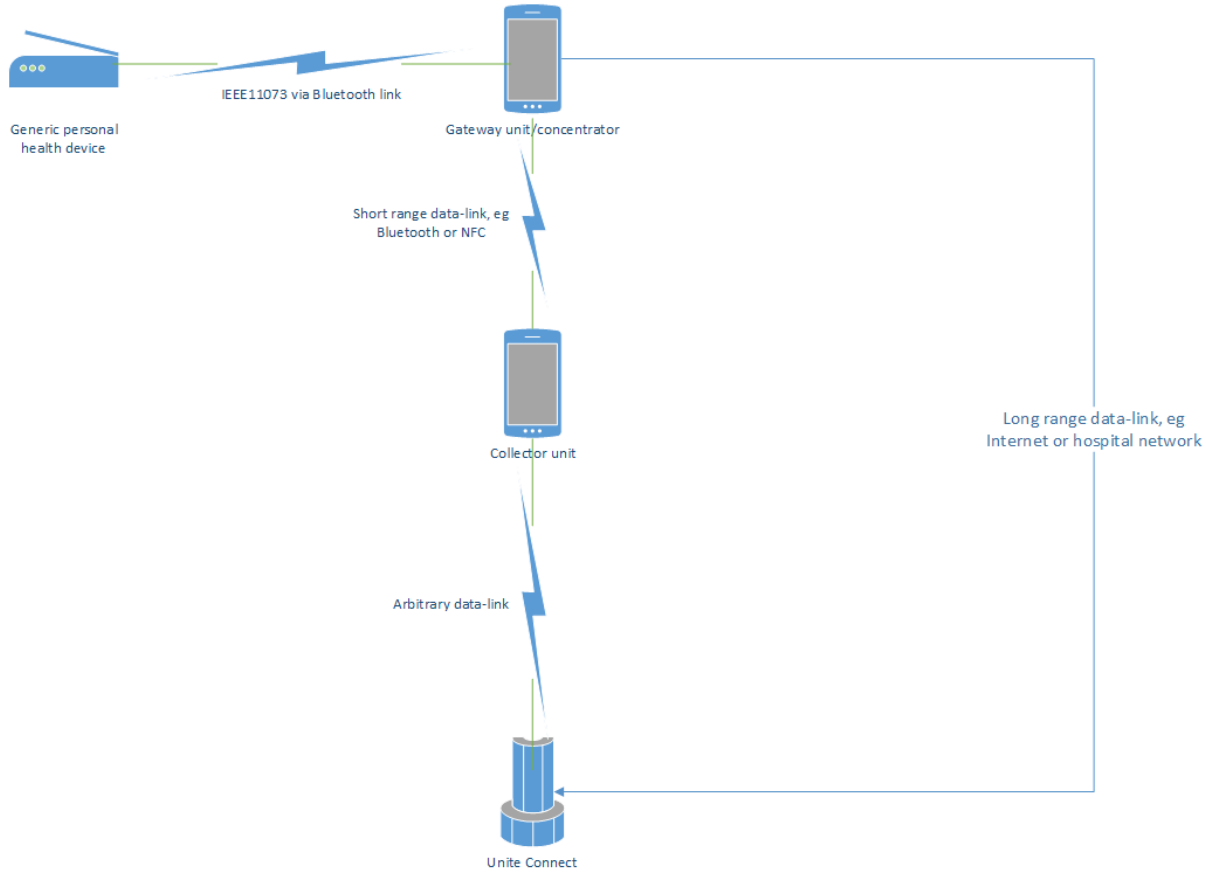


Figure 4 System setup overview

When defining the functionality of the integration, four different main scenarios were thought of mainly. There were more scenarios in the process, but to keep it simple, these four scenarios were chosen since they catch all the functionality discussed.

The first scenario is that a hospital wants to send a recovering patient to their home, for the benefit of both parts; the patient will feel more convenient getting home earlier and the hospital will get more space and resources for other patients.

The second scenario is using the gateway in a hospital ward together with cheap and small off the shelf health sensor devices, partly to cut costs for the hospitals and partly to make hospitalisation more convenient for patients by using smaller devices and thereby reduce the patients' immobilisation.

The third scenario is that a caretaker in home care gets a sensor device or some sensor devices, to measure medical data in-between visits from the district nurse. This could add accuracy to diagnoses by generating data that otherwise wouldn't be available, or in another case letting a caretaker stay at home while being monitored instead of being hospitalised, thereby making the situation for the caretaker more convenient and at the same time reducing costs for the hospital.

The fourth and last scenario is a district nurse in home care, who brings a set of easy-to-carry sensor devices, to be able to easily take measurements that directly get recorded into an electronic health record system.

In all four scenarios ease of use will be of importance, so that nurses easily can configure the generic gateway device and don't have to waste their time on non-healthcare activities. It's also important that it's easy for nurses, or in scenario one and three caretakers, to start and stop measuring sessions.

For scenario three it's also important that the gateway device has the capability to store data in-between visits.

For all scenarios the possibility to use the device with a battery and also being able to use it while charging is important; the battery is important for mobility and the possibility to use it while charging is important because otherwise bothersome waiting times will occur and the devices might not be able to be used when they should.

The possible solution depicted in Figure 4 is a full system solution, which depicts the capabilities needed for all four scenarios. It consists of a gateway/concentrator unit that is able to communicate with off-the-shelf personal health devices that make use of the standard ISO/IEEE 11073 protocol over a Bluetooth link and make the data identifiable, in a way that the recorded data is associated with the right caretaker/patient when entering Unite. This unit could either be connected to Unite directly, via the Internet or a local network for example (depending on whether it's used on- or off-site), or it could work as an offline unit that just collects and stores data. This data would then be collected via some kind of short-range communication to a collector unit. The collector unit could then be used to transfer the data to a Unite system, or possibly directly into a medical records system.

As the system will handle sensitive personal data, security and compliance with personal data handling laws will be crucial for a deployed system.

3.2 Detailed scenarios and user stories

[This chapter can be skipped on a first reading]

In this subchapter a system setup overview and a use case diagram for each scenario will be shown together with its related user stories and acceptance criteria

3.2.1 Scenario A: permission from hospital

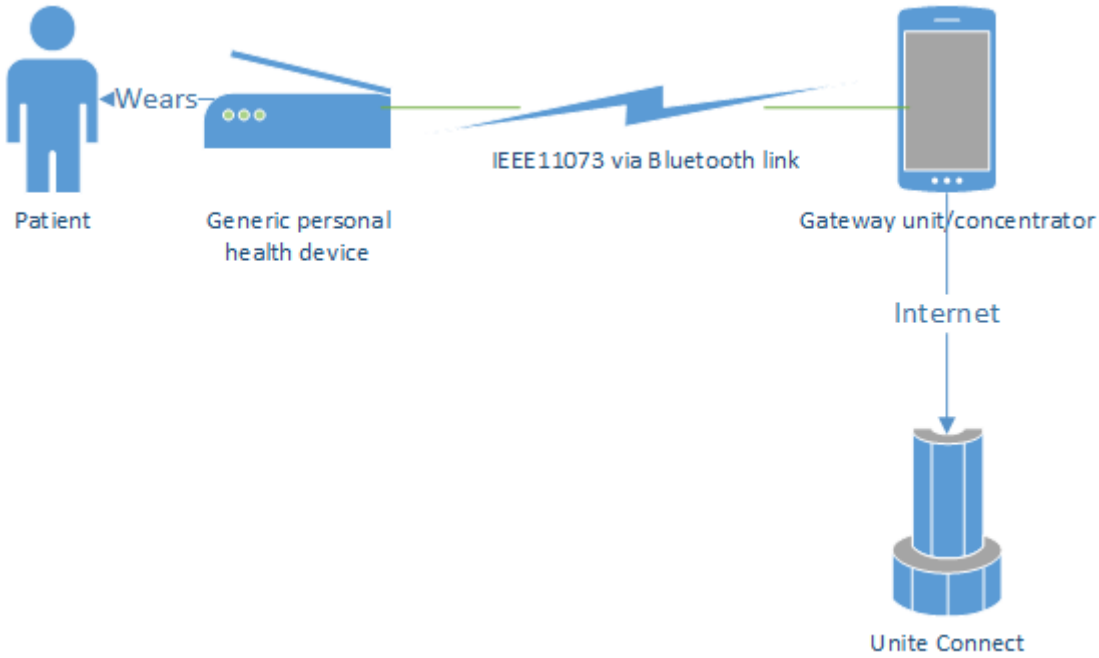


Figure 5 System setup for the permission from hospital scenario

The system setup in the above picture consists of a patient in their home, wearing a PHD, which communicates with a gateway unit either placed close enough to the patient or worn by the patient (e.g. in a pocket). The gateway unit communicates with the Unite system at the hospital via the Internet.

Story A1

As a ***nurse at a hospital*** I want ***to be able to send recovering patients home while still monitoring them*** so that ***there can be available spots for new patients, the patients can come home sooner and to be able to monitor patients during their whole recovery periods.***

Acceptance criteria for story A1

Functional acceptance criteria for story A1

A1.1. Gateway unit shall be able to communicate with sensor units

A1.2. Gateway unit shall be able to send monitoring (sensor) data over the Internet to the hospital in real-time

A1.3. Gateway unit shall notify the patient if a sensor unit falls off or stops transferring data

A1.4. Server shall be able to receive all data sent by the gateway unit

A1.5. Server shall have the data presentable to nurses in real-time

A1.6. Server shall generate a notification in case of sensor data out of accepted ranges or not received sensor data at defined data reception times

Non-functional acceptance criteria for story A1

A1.7. Gateway unit shall be easy to configure

Story A2

As a ***patient sent on permission*** I want to ***have an easy to use and accurate monitoring system that is reliable and doesn't cause me too many inconveniences*** so that ***I can feel safe being at home while recovering and can deal with my everyday life without too much extra hassle.***

Acceptance criteria for story A2

Functional acceptance criteria for story A2

A2.1. Gateway unit shall collect all sensor data sent by the sensor units

A2.2. Gateway unit shall be able to send recorded data in real-time over the Internet to the hospital

A2.3. Gateway unit shall notify the patient and the hospital if it is unable to collect or transmit data

Non-functional acceptance criteria for story A2

A2.4. Gateway unit shall be easy to use for the patient

A2.5. Sensor units and gateway unit shall not make the patient too immobile

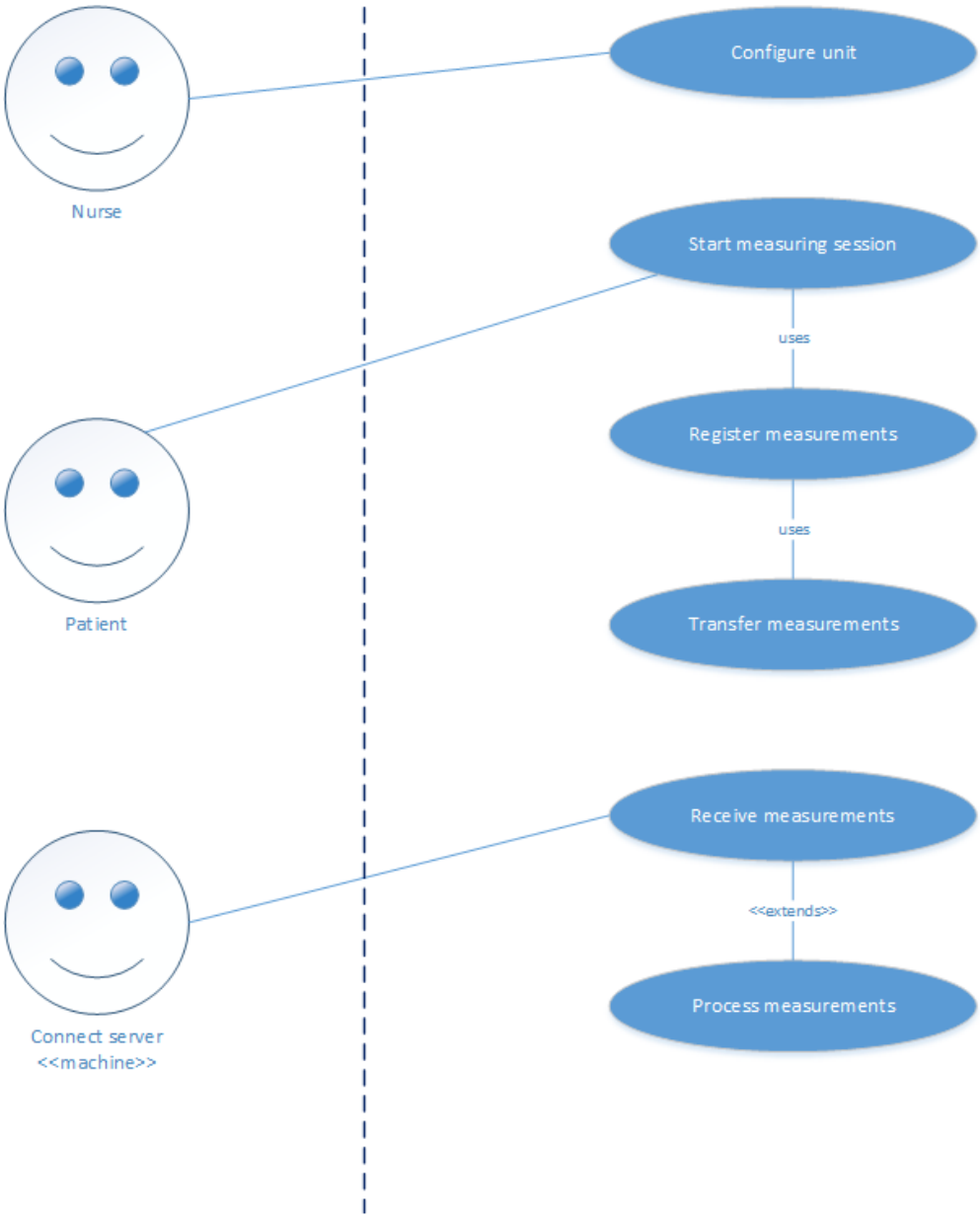


Figure 6 Use case diagram for the permission scenario

3.2.2 Scenario B: home care visit (using continuous measurements and data dumping)

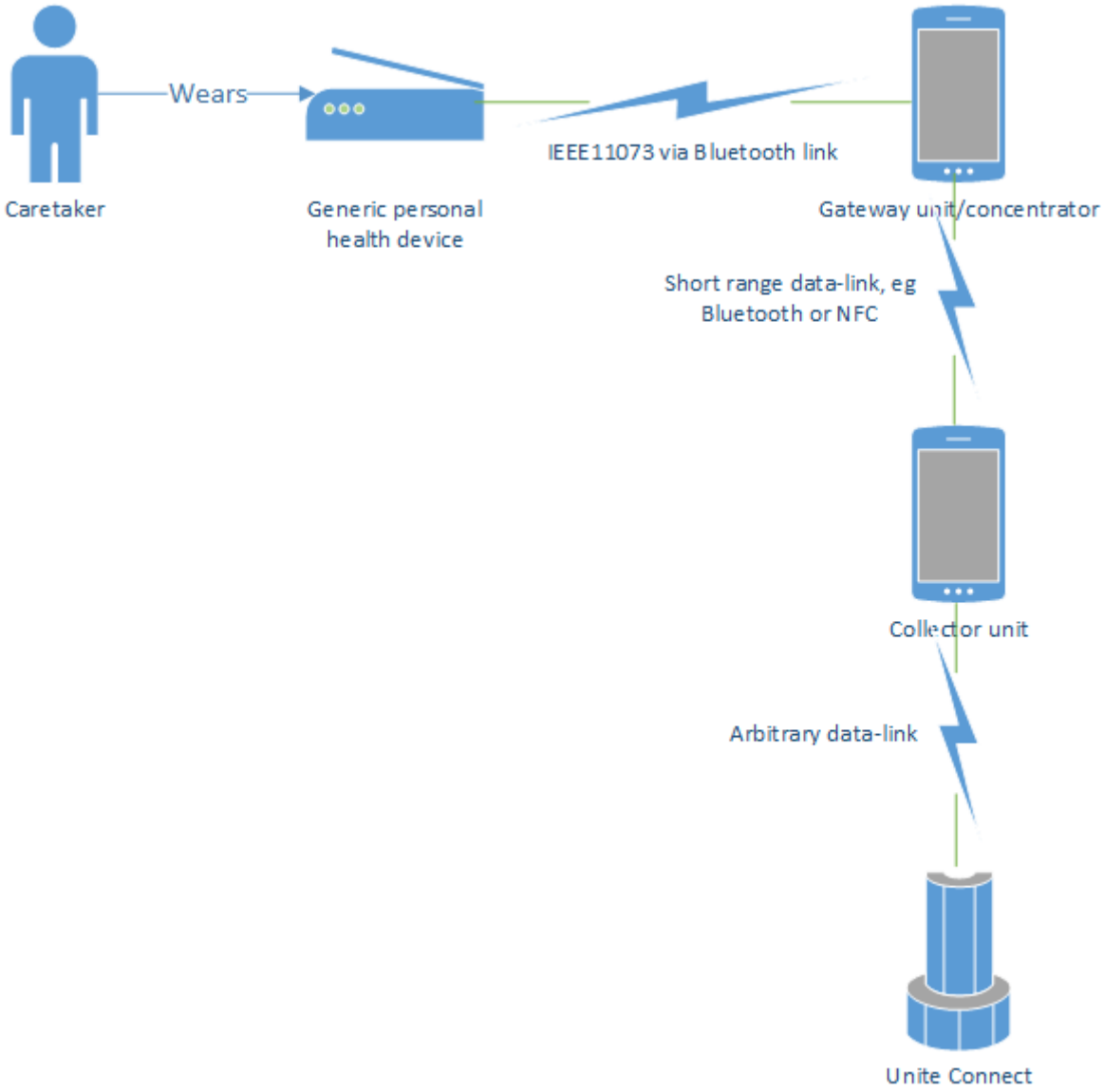


Figure 7 System setup for the home care visit (using continuous measurements and data dumping) scenario

The system setup for this scenario consists of a caretaker in their home, wearing a PHD that communicates via Bluetooth to a concentrator unit that records medical data. When the district nurse come visiting, they bring the collector unit to collect the recorded data, which in turn will either send the data to the care centre directly via the Internet or when the district nurse gets back there, via e.g. a local network or Bluetooth.

Story B1

As a ***nurse working with home care*** I want ***to be able to monitor caretakers in-between my visits*** so that ***I can give them more accurate diagnoses.***

Acceptance criteria for story B1

Functional acceptance criteria for story B1

B1.1. Gateway unit shall be able to store recorded sensor data between two visits

B1.2. Gateway unit shall be able to communicate with sensor units

Non-functional acceptance criteria for story B1

B1.3. Gateway unit shall be easy and fast to configure

Story B2

As a ***caretaker in home care getting to use monitoring equipment in-between the nurse's visits*** I want ***to know that I have started up all the devices correctly*** so that ***I can feel re-assured that the nurse will get my medical data next time they're visiting me.***

Acceptance criteria for story B2

Functional acceptance criteria for story B2

B2.1. Gateway unit should remind caretaker of measuring sessions, in case they haven't been started within a given time period after the scheduled time

Non-functional acceptance criteria for story B2

B2.2. Gateway unit shall be easy to use

B2.3. Gateway unit should, in an easily comprehensible way, show that everything is up and running

B2.4. Gateway unit shall tell the caretaker in an easy-to-understand language what is wrong / how to fix it if something doesn't work.

Story B3

As a ***nurse working with home care*** I want ***to be able to easily get measurement data from a caretaker who has been using monitoring devices since my last visit*** so that ***I don't have to waste patient time on device configuration***

Acceptance criteria for story B3

Functional acceptance criteria for story B3

B3.1. Gateway unit shall transfer saved measurement data to collector unit in an automated or mostly automated fashion

B3.2. Gateway unit shall give a notification that all data has been transferred

B3.3. Collector unit shall give a notification that valid data has been received

B3.4. Collector unit shall give a notification that data has been stored into the medical record for the right caretaker

Story B4

As a ***nurse in home care*** I want ***to carry around as few appliances as possible*** so that ***I don't have to have my hands full while walking up to someone's doorstep in the dark.***

Acceptance criteria for story B4

Functional acceptance criteria for story B4

B4.1. Collector unit shall be part of an already existing device, e.g. Myco or work phone

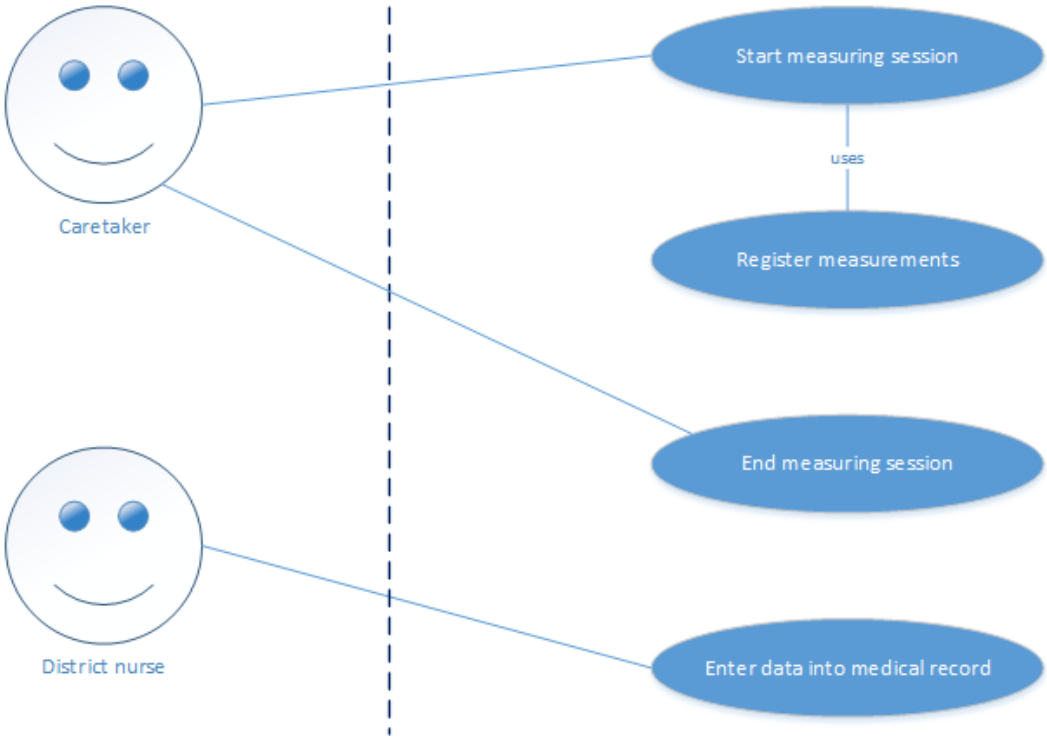


Figure 8 Use case diagram for the home care visit (using continuous measurements and data dumping) scenario

3.2.3 Scenario C: monitoring in a ward

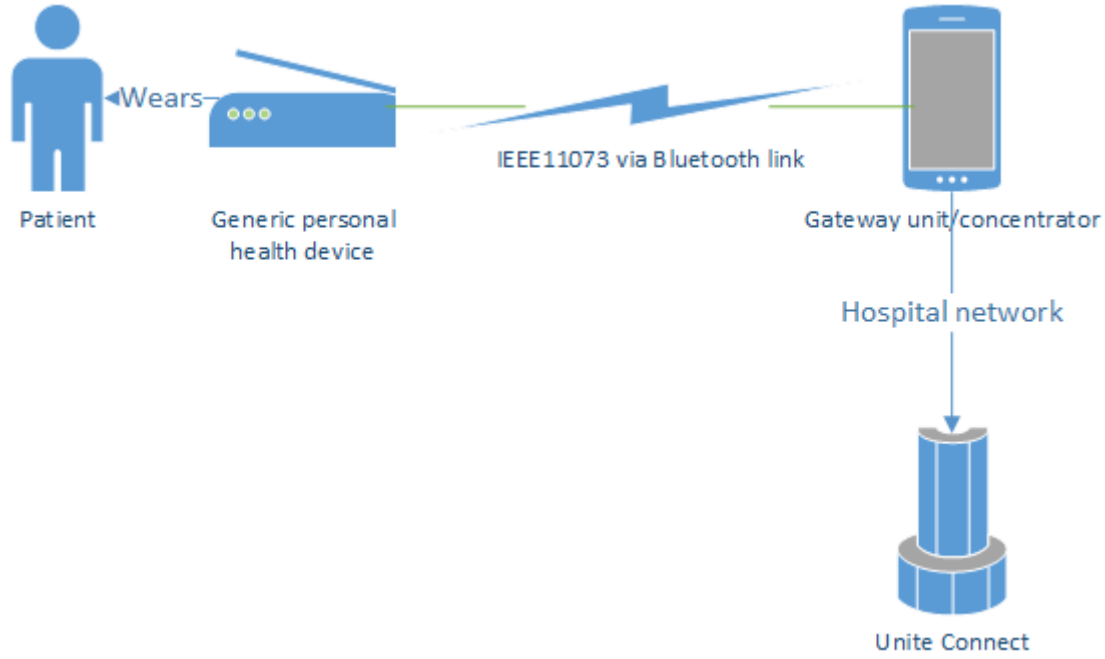


Figure 9 System setup for the monitoring in a ward scenario

This scenario's system setup consists of a patient at a hospital, wearing a PHD and a gateway unit, the PHD communicates with the gateway unit via Bluetooth and the gateway unit in turn communicates with Unite via the Hospital's local network.

Story C1

As a ***nurse at a hospital*** I want ***to be able to start medical monitoring of a patient easily and quickly*** so that ***I don't have to waste patient time on issues not directly connected to care and don't have to break my workflow.***

Acceptance criteria for story C1

Functional acceptance criteria for story C1

C1.1. Gateway unit shall be able to communicate with sensor units

C1.2. Gateway unit should give feedback on its working state (i.e. tell if everything is up and running or what's not and how to fix it)

C1.3. Gateway unit shall be able to send sensor data to the hospital's server in real-time

C1.4. Server shall be able to receive all data sent by the gateway unit

C1.5. Server shall process whatever data that has to be processed upon receiving it

C1.6. Server shall generate a notification in case of sensor data out of accepted ranges or data reception failure

Non-functional acceptance criteria for story C1

C1.7. Gateway unit shall be easy to configure

C1.8. Gateway unit shall be easy to use

Story C2

As a ***patient getting to use new small PHD technology*** I want ***to be mobile*** so that ***I can go to e.g. the toilet or the kiosk without help (provided I'm allowed to do this)***

Acceptance criteria for story C2

Functional acceptance criteria for story C2

C2.1. Gateway unit shall be usable without any cords attached for at least an hour

C2.2. Gateway unit shall be able to continue transmitting without user interaction when changing access points

Non-functional acceptance criteria for story C2

C2.3. Gateway unit shall be mobile

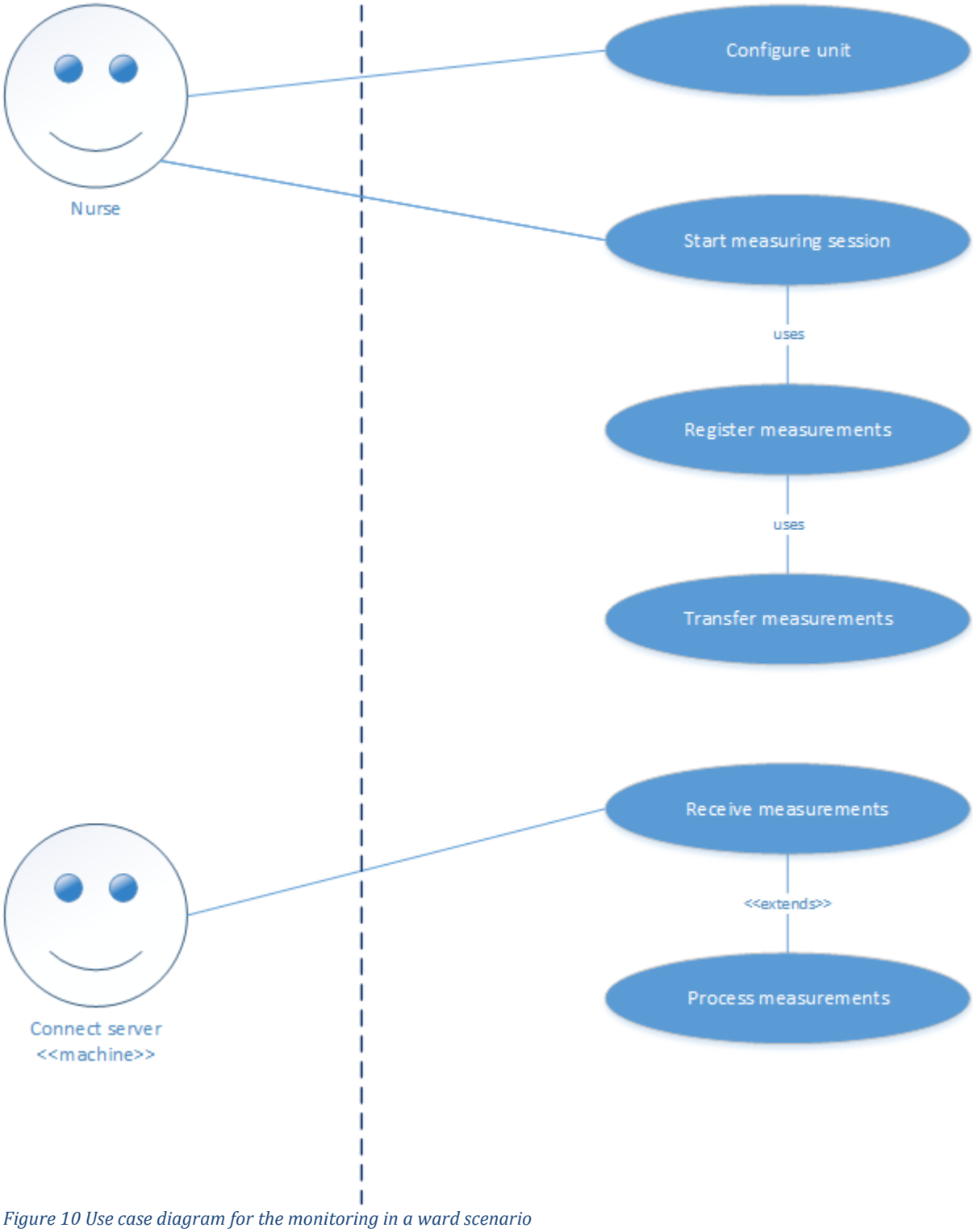


Figure 10 Use case diagram for the monitoring in a ward scenario

3.2.4 Scenario D: home care visit (point measurement)

Figure 11 System setup for the home care visit (point measurement) scenario

This last example setup is the smallest one and consists only of a PHD and a concentrator communicating via Bluetooth. In this scenario the district nurse brings the equipment with them when visiting a caretaker they want to measure some kind of medical data for. It is possible that the concentrator will be connected on the other end as well, but that functionality is already covered by other scenarios.

Story D1

As a ***nurse in home care*** I want ***to be able to easily perform various medical check-ups when I visit my caretakers*** so that ***I can get information quicker and don't have to send caretakers to a care centre.***

Acceptance criteria for story D1

Functional acceptance criteria for story D1

D1.1. Gateway unit shall be able to communicate with sensor units

D1.2. Gateway unit shall make a clear notification if the communication with the sensor(s) doesn't work

D1.3. Gateway unit shall make a clear notification if the sensor unit doesn't send any real data (e.g. if the sensor isn't attached properly)

D1.4. Gateway unit shall show a notification if the measurements are out of pre-defined accepted boundaries

D1.5. Gateway unit shall make a clear notification that data has been stored into the medical record for the right caretaker

Non-functional acceptance criteria for story D1

D1.6. Gateway unit shall be easy to use

D1.7. Gateway unit shall be able to either construct good medical record data or be able to send data that can be used by a unit later on in the chain to construct good medical record data

Story D2

As a ***nurse in home care*** I want ***to carry around as few appliances as possible*** so that ***I don't have to have my hands full while walking up to someone's doorstep in the dark.***

Acceptance criteria for story D2

Functional acceptance criteria for story D2

D2.1. Gateway unit shall be part of an already existing device, e.g. Myco or work phone.

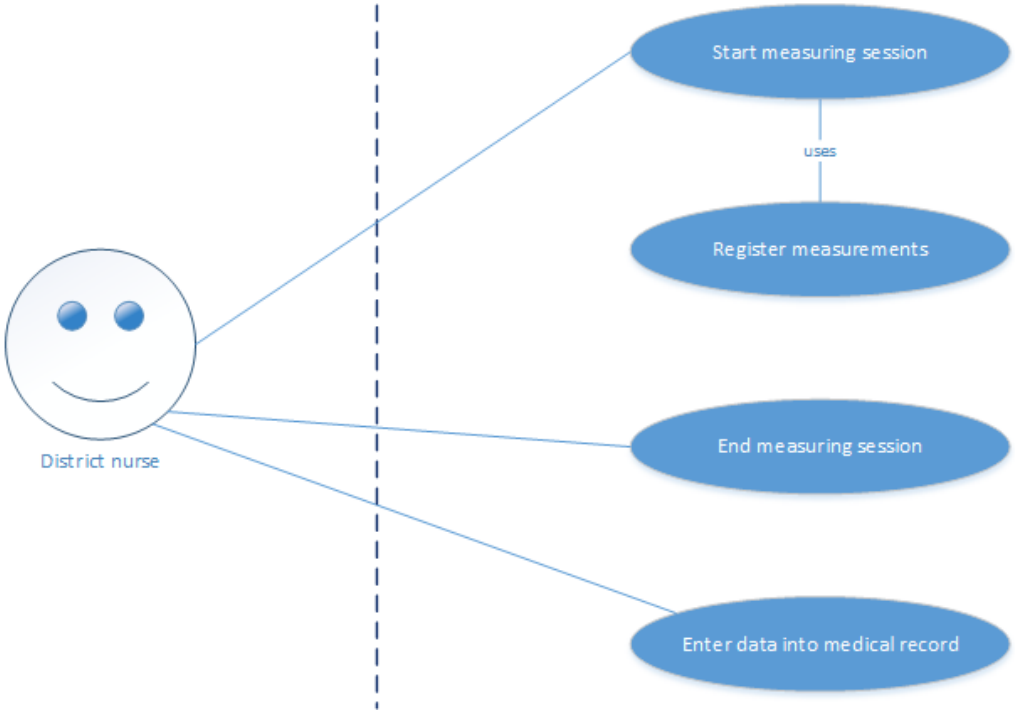


Figure 12 Use case diagram for the home care visit (point measurement) scenario

3.3 Requirements

[This chapter can be skipped on a first reading]

This subchapter lists the identified requirements for the gateway (and collector) Android application created as the major part of the proof-of-concept in this project. The requirements are divided into functional and non-functional requirements, and each requirement states from which acceptance criterion/criteria it originates.

Functional requirements

R001. Gateway unit shall be able to communicate with sensor units
origin: A1.1, B1.2, C1.1, D1.1

R002. Gateway unit shall be able to send recorded monitoring (sensor) data over the Internet to the hospital in real-time
origin: A1.2, A2.2, C1.3

R003. Gateway unit shall notify the patient if a sensor unit falls off or stops transferring data
origin: A1.3, D1.3

R004. Server shall be able to receive all data sent by the gateway unit
origin: A1.4, C1.4

R005. Server shall have the data presentable to nurses in real-time
origin: A1.5

R006. Server shall generate a notification in case of sensor data out of accepted ranges, not received sensor data at defined data reception times or data reception failure
origin: A1.6, C1.6

R007. Gateway unit shall collect all sensor data sent by the sensor units
origin: A2.1

R008. Gateway unit shall notify the patient and the hospital if it is unable to collect or transmit data
origin: A2.3, D1.2

R009. Gateway unit shall be able to store recorded data
origin: B1.1

R010. Gateway unit should remind caretaker of measuring sessions, in case they haven't been started within a given time period after the scheduled time
origin: B2.1

R011. Gateway unit shall transfer saved measurement data to collector unit in an automated or mostly automated fashion
origin: B3.1

R012. Gateway unit shall indicate when all data has been transferred to collector unit
origin: B3.2

R013. Collector unit shall give a notification that valid data has been received
origin: B3.3

R014. Gateway/Collector unit shall give a notification when data has been stored into the medical record, for the right caretaker
origin: B3.4, D1.5

R015. Collector unit shall be part of an already existing device, e.g. Myco or work phone
origin: B4.1, D2.1

R016. Gateway unit should give feedback on its working state
origin: C1.2

R017. Server shall process patients' medical data, upon receiving it
origin: C1.5

R018. Gateway unit shall be usable without any cords attached for at least an hour
origin: C2.1

R019. Gateway unit shall be able to continue transmitting sensor data without user interaction when switching between access points
origin: C2.2

R020. Gateway unit shall show a notification if the measurements are out of accepted ranges
origin: D1.4

Non-functional requirements

N001. Gateway unit shall be easy and fast to configure
origin: A1.7, B1.3, B2.2, C1.7

N002. Gateway unit shall be intuitive and easy to use
origin: A2.4, C1.8, D1.6

N003. Sensor units and gateway unit shall not make the patient too immobile
origin: A2.5

N004. Gateway unit should, in an easily comprehensible way, show that everything is up and running
origin: B2.3

N005. Gateway unit shall tell the caretaker in an easy-to-understand language what is wrong / how to fix it, if something doesn't work
origin: B2.4

N006. Gateway unit shall be portable
origin: C2.3

N007. Gateway unit shall be able to either construct good medical record data, or be able to send data that can be used by a unit later on in the chain to construct good medical record data
origin: D1.7

3.4 Platform choice

In the beginning of the design phase, different hardware and software platforms were considered before finally settling upon smartphone Android/Java. The two other main contestants in hardware were single-board computers and standalone smartwatches, but the smartphone platform was chosen because it was deemed to be the easiest platform for testing, as they are already packaged with everything needed and there were small examples written for Android, using a library called Antidote. For further reasoning about the different platforms, see appendix 10.2 Gateway/Concentrator unit reasoning and definition document.

After settling upon the smartphone platform for hardware, a software platform was to be chosen, and the three main contestants were Android/Java, iOS/Objective-C and Windows phone/.NET. The iOS platform was ruled out quickly because of it being too closed, not letting the developer access Bluetooth as necessary (Snyder, 2015). Comparing the two remaining platforms, Android is much more wide-spread and offers many more devices to choose from, so the only benefit of choosing Windows phone would be that the full project could have been developed on the .NET platform, and as this benefit wasn't seen as very important, Android was chosen in the end.

There was also a thought on using the Xamarin platform to enable the use of the .NET/C# and developing for all three aforementioned platforms at the same time. It seemed competent enough when reading through its own documentation, but looking around at different developers' forums at the time gave a picture of a too immature product (lacking features, regular crashes, files disappearing, native functionality binds that either don't exist or are outdated, the need of a Mac to build apps for iOS, outdated tutorials etc.). Also the pricing for using it with Visual Studio integration starts at $999 (Xamarin Inc., 2016), which made the decision simpler.

4. Implementation

This chapter discusses the actual software design implemented in the proof-of-concept, meaning this is a picture of what really is implemented. It starts out with a subchapter about the IDEs and tools used to develop the proof-of-concept.

4.1 Development environments

4.1.1 Android Studio

The official IDE for developing Android applications is called Android Studio and is an IDE based on IntelliJ IDEA. On top of the standard functionality in IDEA it has a few extra features such as ADB integration for easy testing on emulators or real Android devices, code templates for common Android features and a set of lint tools (Android Open Source project, 2016).

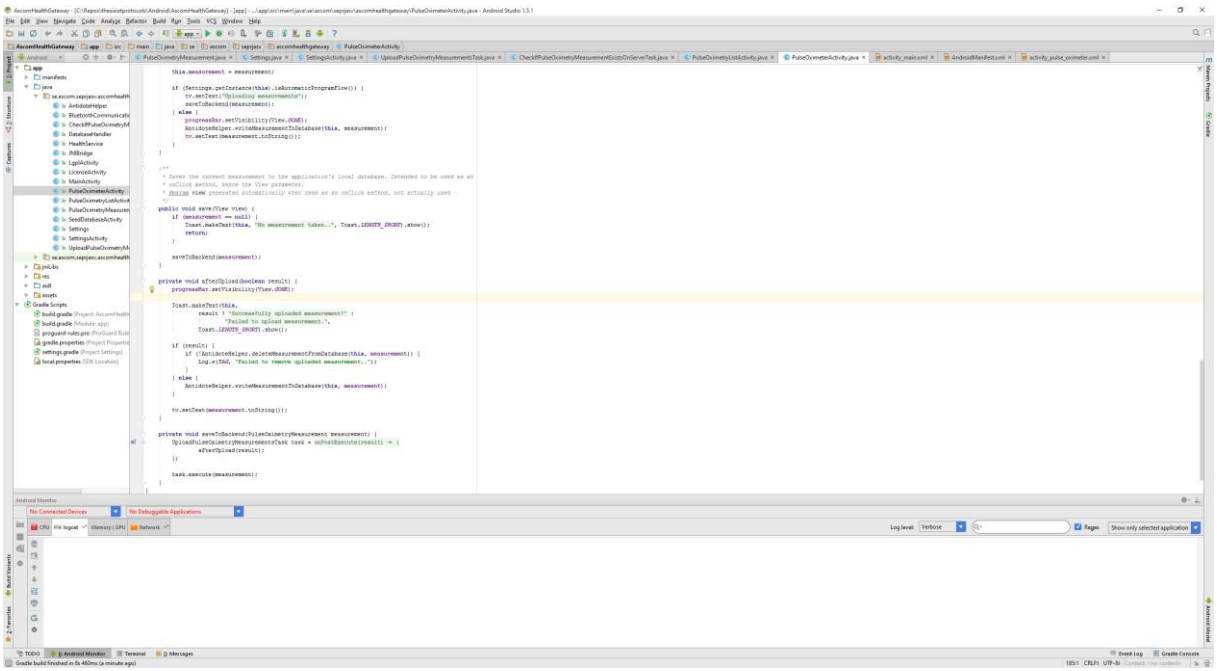


Figure 13 Android Studio having the Gateway application project open

4.1.2 Android NDK

The Android Native Development Kit is a set of tools that enables implementing parts of an application in native-code languages, e.g. C and C++. Its recommended uses are CPU-intensive applications, physics simulation and signal processing (Android Open Source project, 2016).

Android NDK is used in this project to compile the Antidote library, which is written in C.

4.1.3 Visual Studio

Visual Studio is an IDE from Microsoft, it supports many platforms and programming languages and has a powerful text-editor with code completion and real-time lint functionality. In this project it was used for developing the web service.

4.2 Gateway application design

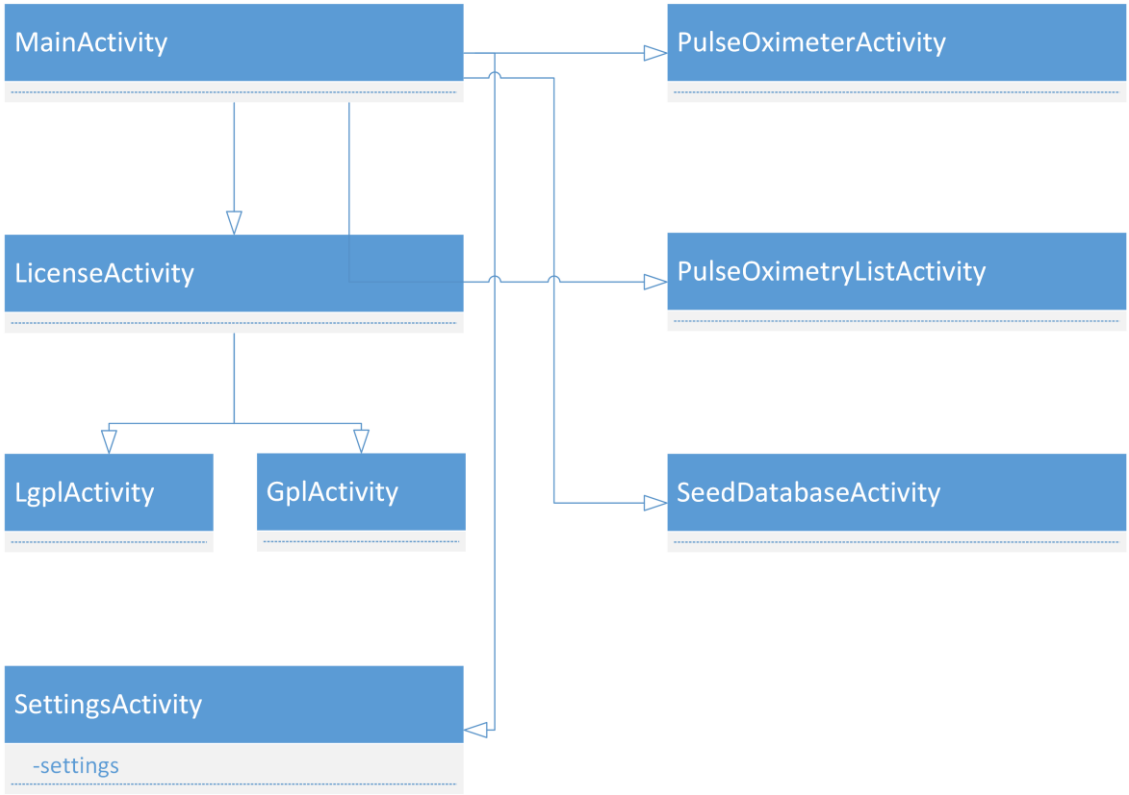


Figure 14 Class diagram showing the flow between Activity (View) classes in the Gateway application

This subchapter is dedicated to explaining how the application that realises the gateway part of the system is designed. Firstly, the application is an Android application written in Java, using a third-party library called Antidote for handling the ISO/IEEE 11073 protocol. Antidote is written in C and compiles directly with the Android NDK (although the names of the methods exposed to Java have to be changed to match the package names of the project in question) on recent Linux distributions.

The application is built in accordance with the MVC pattern, which often is the case with Android applications, because of Android's architecture. Looking at the UML class diagram above, all classes' names end with "Activity" and are hence subclasses of Activity and therefore belong to the View part of the pattern.

Using an MVC pattern makes the code reusable and easily extendable, which are two desired properties for a proof-of-concept project. As the model is already there, one could also think the other way: it would be easy to add new views and controllers, e.g. for visualising measurement data that is stored at some sort of backend server with which the application can communicate.

4.2.1 Basic application flow

This subchapter describes the basic flow of the gateway application when using it for receiving a measurement. If the reader doesn't want to go into detail and just get an overview of how the application works, they can read this subchapter and skip the coming four subchapters. This is the main flow:

- `MainActivity` starts and presents the user with a list of paired medical devices to choose from.
- The user chooses which device they want to receive measurements from.
- The right view, according to the type of the chosen device, is started, the only supported one in the proof-of-concept is the `PulseOximeterActivity`.
- The started view starts up `HealthService`, which in turn uses `JNIBridge` to start up Antidote's `healthd` service, as well as starts up `BluetoothCommunicationService`.
- `BluetoothCommunicationService` registers the application as a manager medical device, thus making the unit communicate via Bluetooth that it is a medical device capable of speaking ISO/IEEE 11073. It starts up its private `ReadThread` class, which listens for incoming Bluetooth connections from medical device agents that speak ISO/IEEE 11073.
- An agent initiates communication with the application, all received messages are handled by the private `IncomingHandler` class.
- The `PulseOximeterActivity` class (or in a full product another responsible view class) gets notified of the incoming connection and notifies the user by showing a progress bar and printing that it's receiving a measurement.
- When the measurement is received, the `AntidoteHelper` class is used to parse the incoming data and a `PulseOximetryMeasurement` object is instantiated and thereafter presented in the `PulseOximeterActivity` and a task is started to upload the measurement to the web service.
- `PulseOximeterActivity` notifies the user on whether the upload was successful or not, if it failed, the measurement is written to the application's database with the `AntidoteHelper`.
- As the `DatabaseHandler` class is a subclass of `SQLiteOpenHelper`, it is used by the `AntidoteHelper` class to get references to the database object.

Also, to get full detail on how the application is implemented, the reader can check out the full source code of the project from the GitHub link in appendix 10.1 GitHub repository.

4.2.2 View

[This chapter can be skipped on a first reading]

The arrows used in the diagram in Figure 14 are for depicting program flow, i.e. when the application is started, the user is presented with the view of `MainActivity`. This `Activity` presents the user with a list of paired medical devices to choose from as well as a button to start using the chosen device. When the button is clicked, a method is called which determines the device type of the chosen device and starts the corresponding `Activity` class, as of now only pulse oximeters are supported, meaning a pulse oximeter would yield the start of `PulseOximeterActivity`, and any other type of medical device would yield a message telling the user that the specific device type is not yet supported.

A screenshot of the view is depicted in Figure 15, and as can be seen it also features a floating button with an envelope as well as a menu. Clicking the floating button switches

the view to that of *PulseOximetryListActivity*. Figure 16 shows *MainActivity* with its menu open.

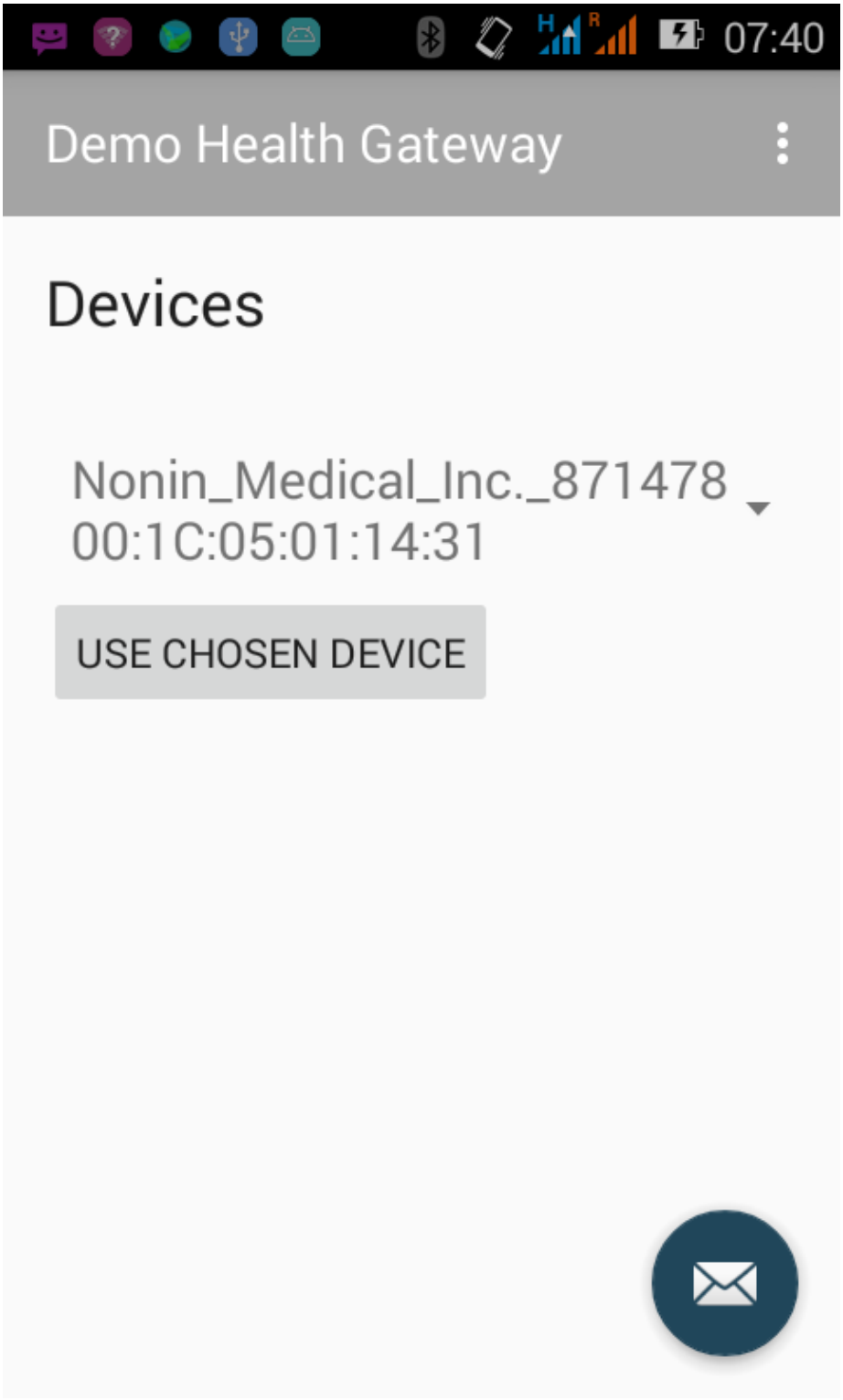


Figure 15 MainActivity view

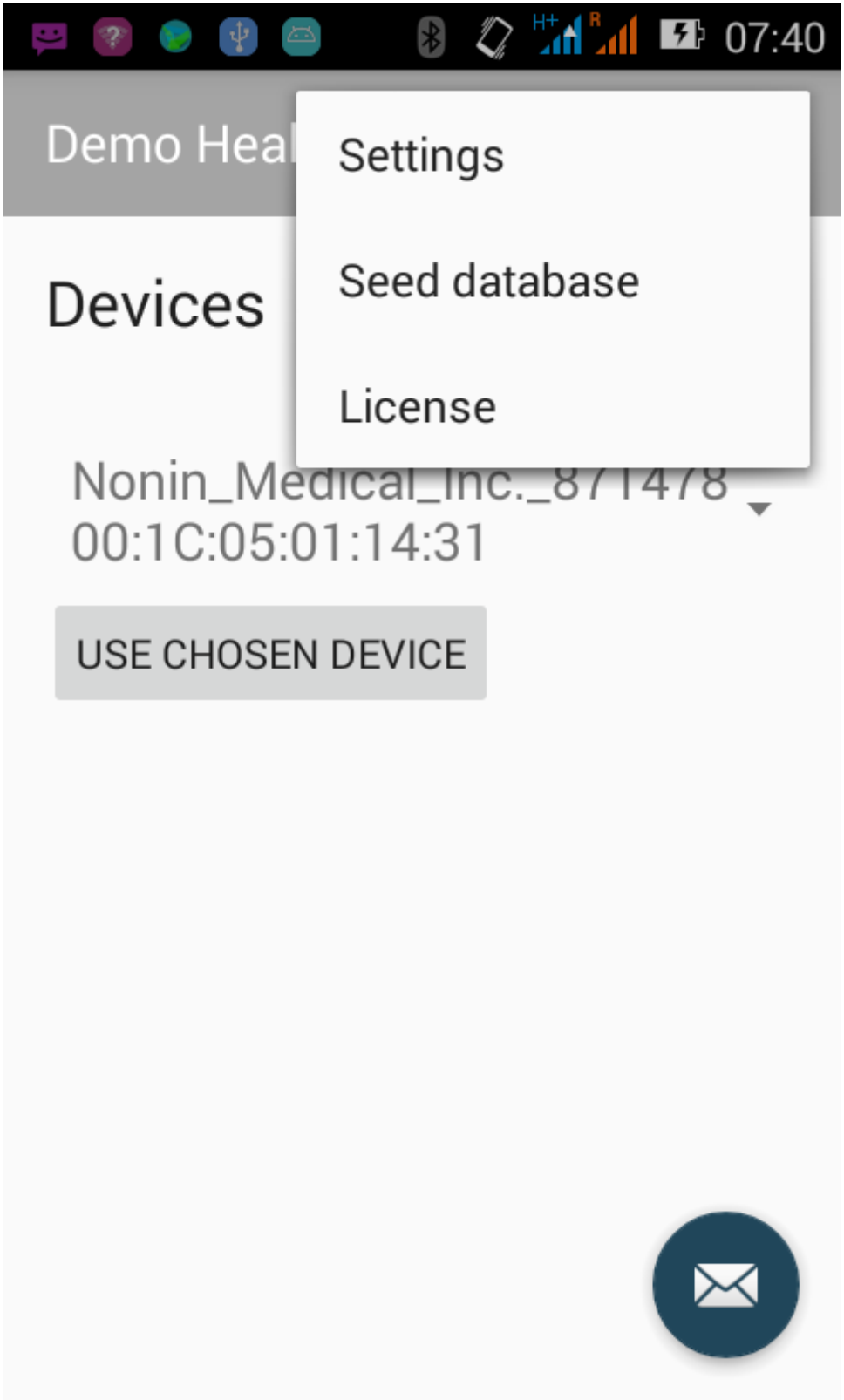


Figure 16 MainActivity with its menu open

Choosing Settings in the menu switches to the `SettingsActivity` view, choosing Seed Database switches to `SeedDatabaseActivity` and choosing License switches to `LicenseActivity`.

`LicenseActivity` simply tells the user that the application uses the Antidote library, copyright information about it and that it is licensed under the GNU lesser general public license, and offers to show this license by clicking a button. Clicking this button switches to `LgplActivity`, which shows the GNU LGPL license (version 2.1) in its entirety. Presenting this license information to the user is a requirement for using a library or software part licensed under GNU LGPL (Free Software Foundation, 1999).

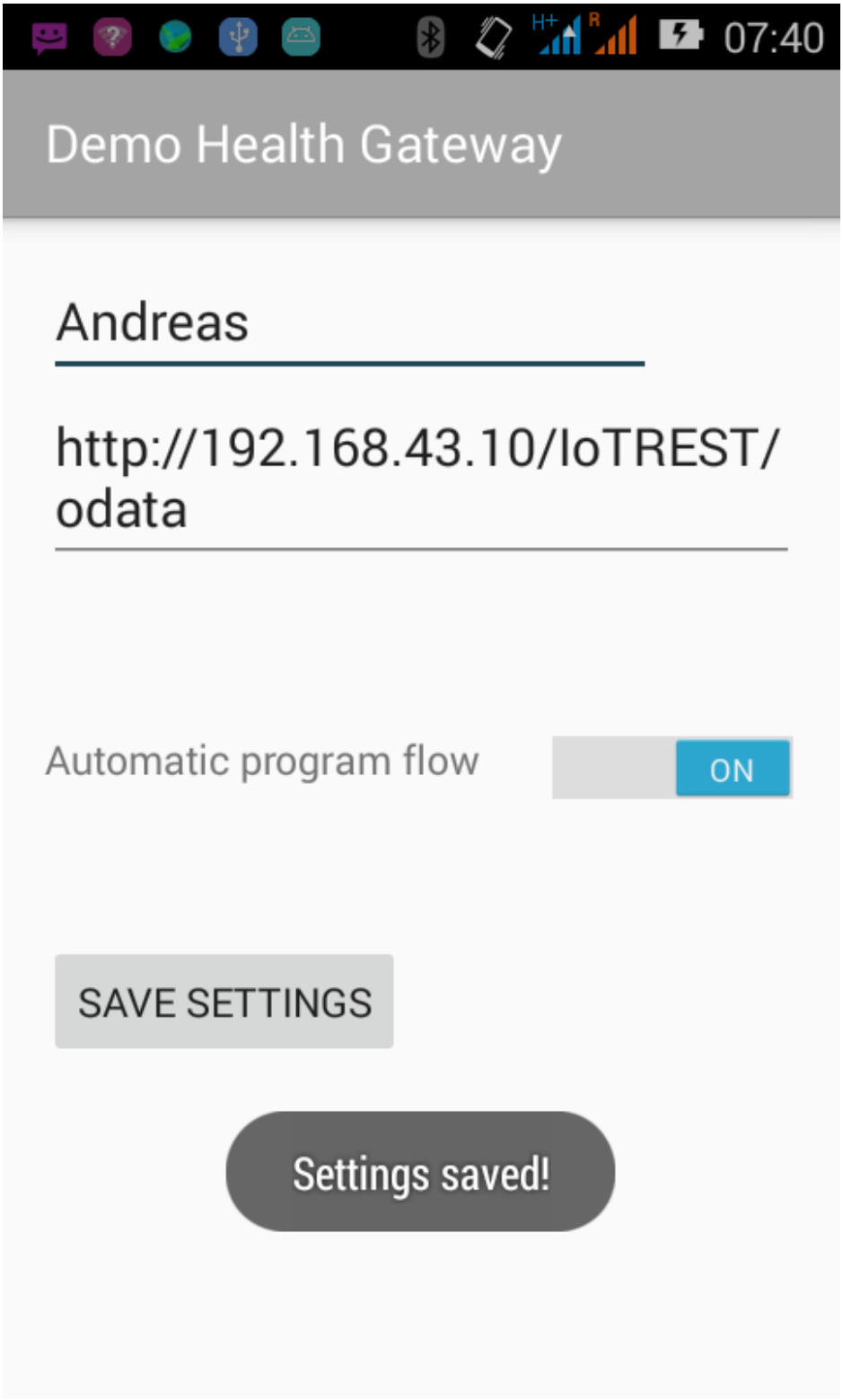


Figure 17 SettingsActivity, notification about the settings having been saved

Figure 17 shows the view of `SettingsActivity`, in which the user can set the patient identification string to be used for measurements, the web service URL and whether the flow of the application shall be automatic or manual.

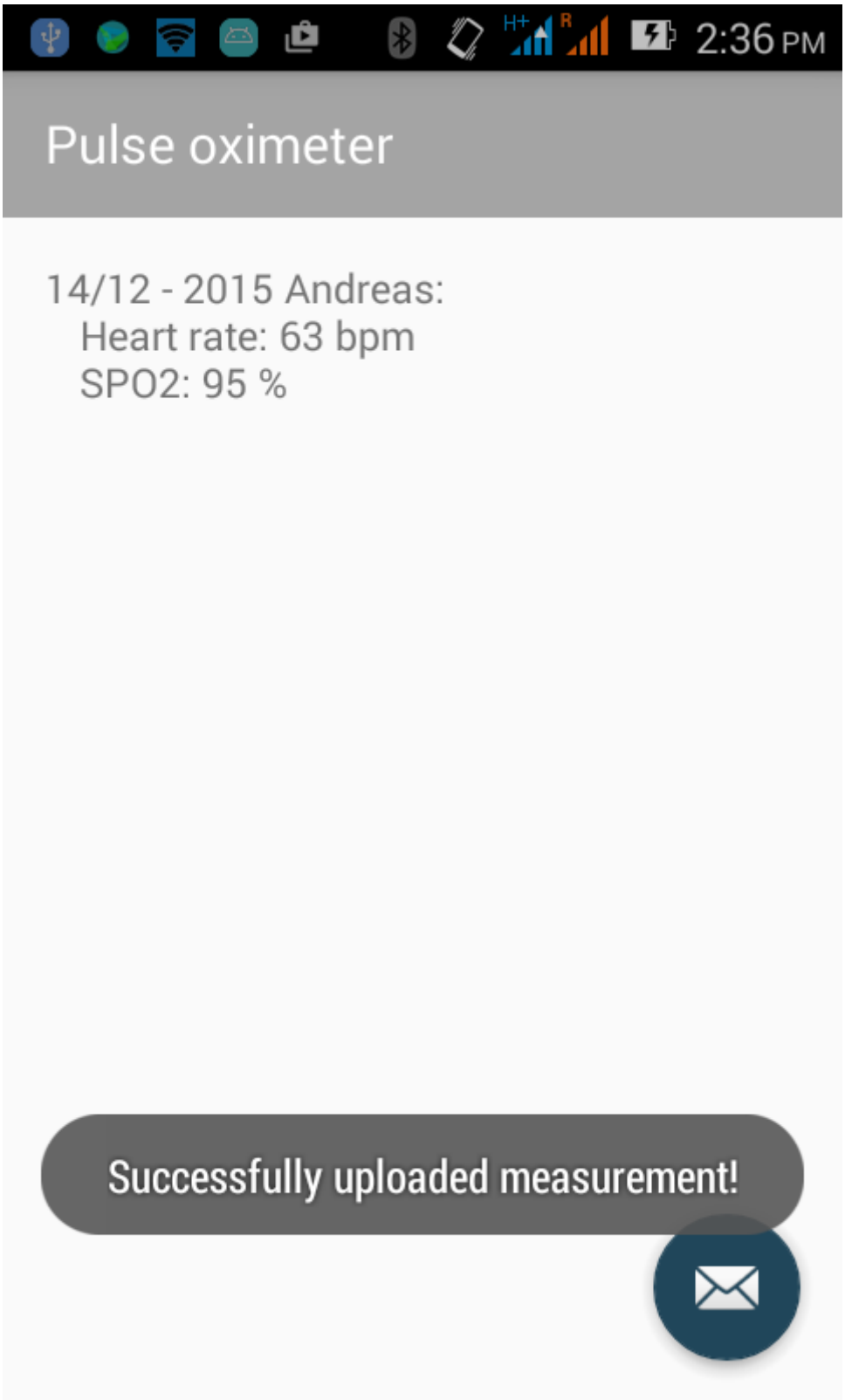


Figure 18 PulseOximeterActivity, a measurement was just uploaded to the web service

Figure 18 shows the view of `PulseOximeterActivity` after it has received a measurement from a pulse oximeter. As can be seen, a `String` representation of the measurement is shown, and in the bottom there's a floating button with an envelope. This floating button is to be used together with manual program flow, during which it initiates a task for uploading the shown measurement to the web service. When running the application with automatic program flow, this task is initiated upon full reception of a measurement instead. After the task is finished a message is shown, telling the user whether the upload was successful or not.

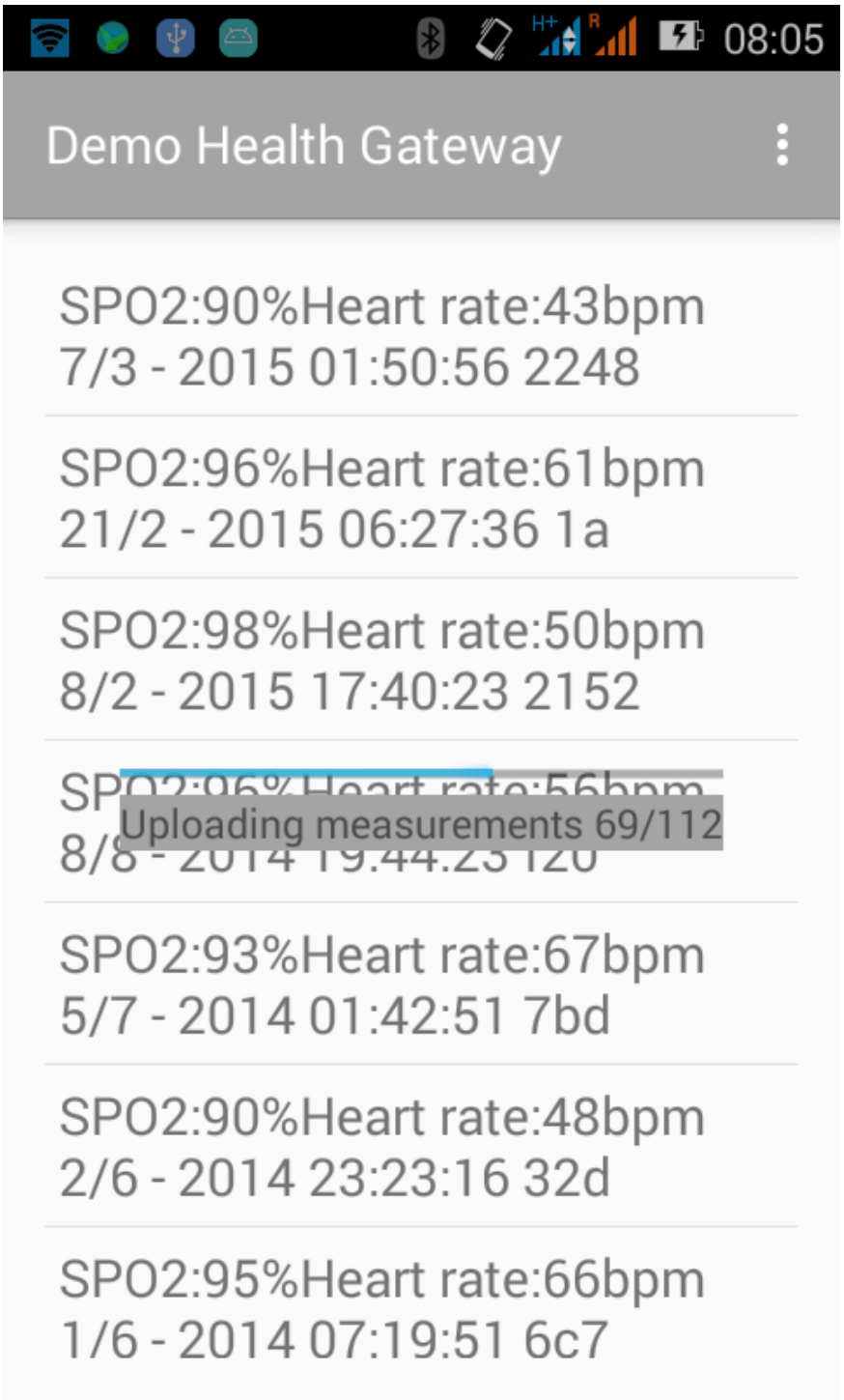


Figure 19 PulseOximetryListActivity, uploading locally stored measurements

In Figure 19 the view of `PulseOximetryListActivity` can be seen. The purpose of this view is to show measurements that couldn't be uploaded to the web service for some reason, and thus are stored in the application's local database. Due to this nature it features a menu choice to try uploading all locally stored measurements, afterwards removing all successfully uploaded measurements.

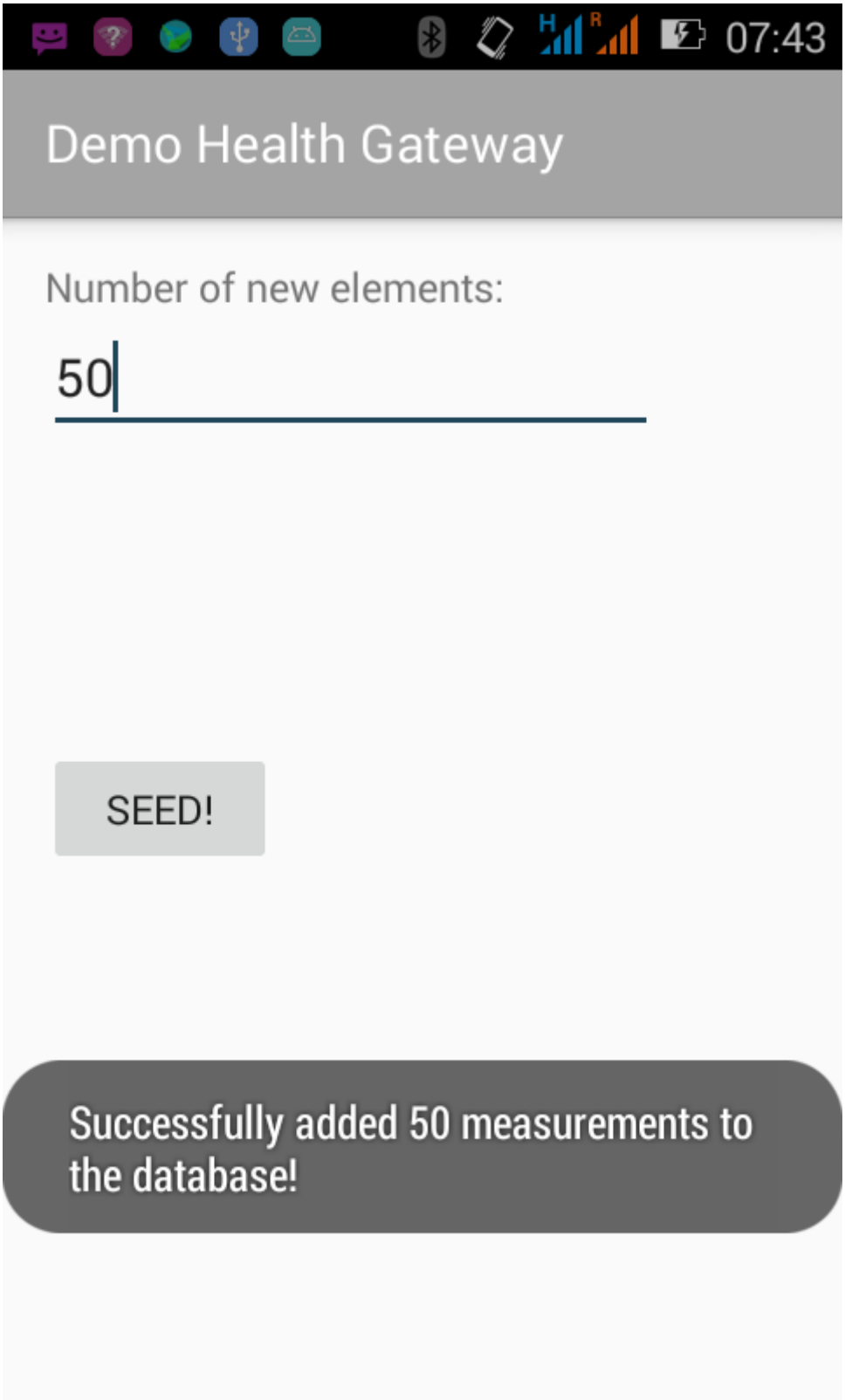


Figure 20 SeedDatabaseActivity, just created 50 random measurement objects

The last *Activity* class, *SeedDatabaseActivity*, is depicted in Figure 20. It was added solely for testing purposes, and allows the user to insert an arbitrary amount of pseudo random measurement data into the application's local database. This can be a useful feature for testing the upload functionality with big amounts of data, as it would take a considerable amount of time to enter hundreds of measurements into the database using a PHD.

4.2.3 Controller

[This chapter can be skipped on a first reading]

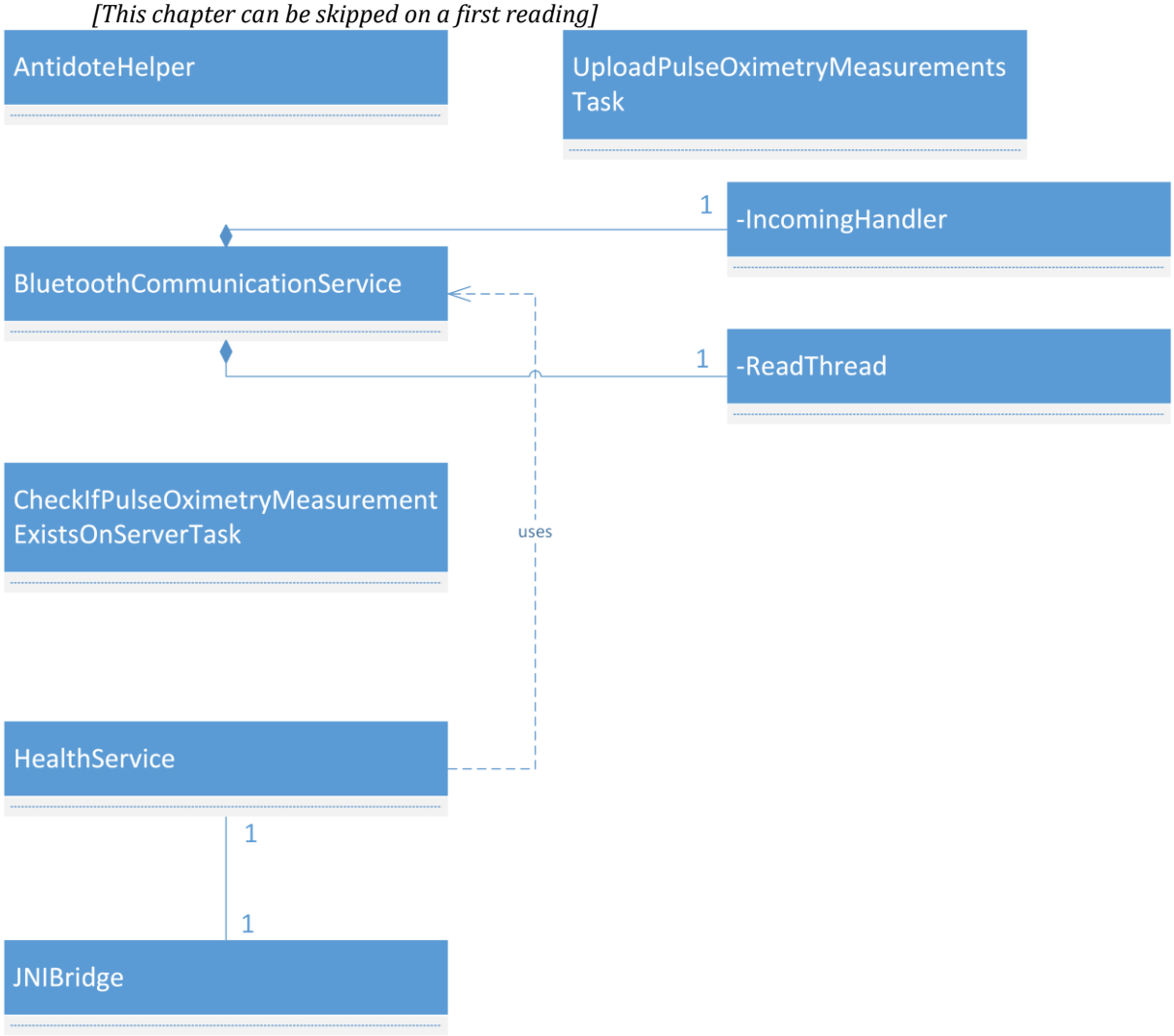


Figure 21 Overview of the controller classes

The controller classes of the gateway application could be divided into three groups, more or less: PHD communication classes (*BluetoothCommunicationService*, *IncomingHandler*, *ReadThread*, *HealthService* and JNIBridge), gateway logic classes (AntidoteHelper) and web service communication classes (*UploadPulseOximetry MeasurementsTask* and *CheckIfPulseOximetryMeasurementExistsOnServerTask*).

The PHD communication classes are responsible for Bluetooth communication with personal health devices and decoding ISO/IEEE 11073-20601 data streams into ISO/IEEE 11073-20601 xml in the form of *Strings*.

`HealthService` is started by the `onCreate` method in `PulseOximeterActivity`, it then initialises `BluetoothCommunicationService`, which handles the actual Bluetooth communication. Its private `IncomingHandler` class is used for communication with `HealthService`, and its private `ReadThread` class lets the Bluetooth communication live in its own thread, thus separating it from the `Service`.

`JNIBridge` is the bridge between the Java world and the native world, where the Antidote library resides, because of it being compiled with the Android NDK. `HealthService` calls functions exposed by `JNIBridge` to do the actual ISO/IEEE 11073 decoding, and likewise Antidote calls functions in `JNIBridge` that forwards communication to `HealthService`, which in turn sends these messages on to `BluetoothCommunicationService`, from where they are finally sent to the personal health device, with which the gateway is communicating.

The `AntidoteHelper` class is a typical helper class with many static methods used by the rest of the application, for example methods for turning xml in `String` format into xml `Document` objects, reading and writing from/to the application's database and converting time stamps between different formats.

The web service communication classes handle the http communication with the web service. Both classes in this category are subclasses of `AsyncTask`, which is the recommended way to handle network connections when developing Android applications, because of the unpredictable delays that network operations can involve (Android Open Source project, 2016). This way these delays don't influence the user interface.

The `UploadPulseOximetryMeasurementsTask` takes one or more `PulseOximetryMeasurement` objects (using the Java … pattern for parameters), tries to upload them one by one to the `URL` given in the settings and returns `true` if all uploads succeeded. This `Boolean` value is used to know whether or not to remove local database copies of the measurements. Only having one `Boolean` value to represent how multiple uploads went is not very precise, but using the … pattern for input is required when extending the `AsyncTask` class, and to compensate for this, the other class in this category, `CheckIfPulseOximetryMeasurementExistsOnServerTask`, was created.

`CheckIfPulseOximetryMeasurementExistsOnServerTask` does take one or more `PulseOximetryMeasurement` objects as input parameters, just like the previous `Task`, though it only checks for the first one in case there are multiple objects given. This way the local copies of uploaded measurements can be deleted when there were both failing and working uploads performed, and this task doesn't have to be run when all measurements were successfully uploaded.

4.2.4 Model

[This chapter can be skipped on a first reading]

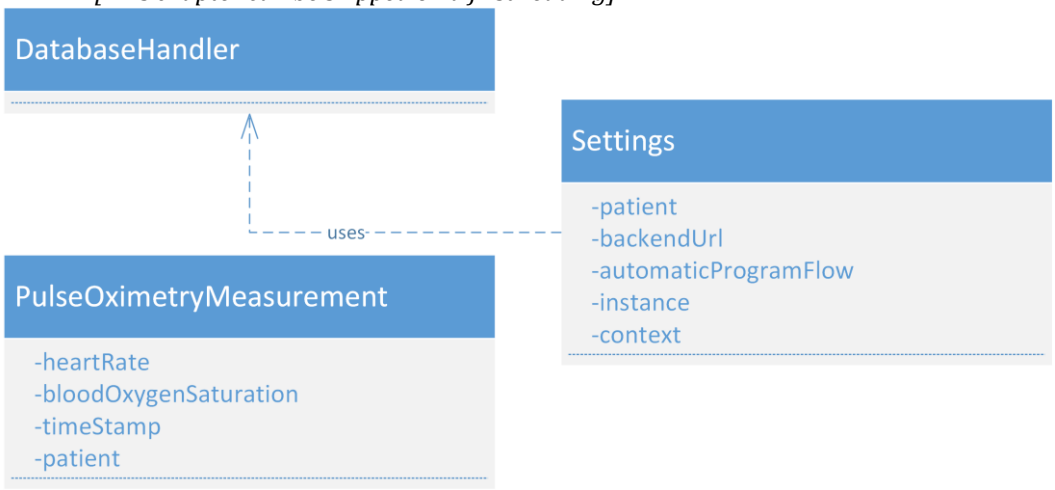


Figure 22 Overview of the model classes

Figure 22 depicts the model classes of the gateway application, as can be seen this is by far the smallest part of the MVC pattern. The model is neither fat nor anaemic, but rather somewhere in-between.

It is not entirely clear whether the `DatabaseHandler` should belong to the model or the controller, its name even implies that it handles something, and hence should be part of the controller. The actual methods implemented in `DatabaseHandler` are responsible for creating and updating the tables in the application's database, meaning they set up the database model. In this setting the `DatabaseHandler` is only used by the internal system, which calls `onCreate` or `onUpgrade` when appropriate. Each time a `DatabaseHandler` object is instantiated by the application, it is to enable fetching of database objects for reading from or writing to the database, meaning that also here it is used more as a model class than a controller class.

The `Settings` class is a singleton class, simply because there really shouldn't be multiple instances of the application's settings. It allows setting the identification string for the patient to take measurements for, setting the url `String` of the backend to upload measurements to and also setting a `Boolean` value on whether the program flow of the application should be automatic or manual. When the `Settings` class is instantiated, it reads the settings from the settings table in the application's database, to make the application's settings persistent, and every time a setting is changed, this change is also written to the database.

Lastly there is the `PulseOximetryMeasurement` class, which is intended to represent a measurement taken with a pulse oximeter following the ISO/IEEE 11073-10404 device specialisation standard. Using one of the standard device configurations for PHD pulse oximeters implies measuring blood oxygen saturation in percent and heart rate in beats per minute, both of them with the datatype `Basic-Nu-Observed-Value` (The Institute of Electrical and Electronics Engineers, Inc., 2009), which is a two byte numerical datatype (The Institute of Electrical and Electronics Engineers, Inc., 2014). In this implementation the units are both represented by `Strings`, to support extended configurations and the values are both represented by the `float` datatype, which actually is a 32-bit datatype, but also the smallest floating point primitive datatype in

Java (Oracle, 2015). Also the xml `Strings` returned by Antidote, representing measurements, use the `float` type for blood oxygen saturation and heart rate values. The timestamp in the ISO/IEEE 11073-20601 standard has 1/100 second precision (The Institute of Electrical and Electronics Engineers, Inc., 2014), in the `PulseOximetryMeasurement` class it is represented by the `GregorianCalendar` class. The patient identifier is not a part of the ISO/IEEE 11073 standards family, but is added in this implementation to enable measurements for multiple patients, it is represented by the `String` datatype.

`PulseOximetryMeasurement` has constructors using individual values, with or without timestamp, and in the latter case it sets the current time as timestamp, a constructor taking a `JSONObject` as paramater as well as a static method converting an xml `Document` object into a `PulseOximetryMeasurement`. Lastly it has an empty constructor, which creates a pseudo random measurement object, and is used together with the seed database functionality described in 4.2.2 View. The generated measurement objects will have a timestamp between 2005-01-01T00:00:00 – [previous year]-12-31T23:59:59 inclusive, a heart rate between 40 – 69 bpm inclusive, a blood oxygen saturation between 90 – 99 % inclusive and a hexadecimal number between 0 – 270F inclusive as patient identifier. Both numerical values will be generated as integers, because the pulse oximeter used to test the application only presents integer level accuracy.

4.2.5 Application flow design

[This chapter can be skipped on a first reading]

To visualise the idea of the application flow, two IDEF0 diagrams were made, one that sees the gateway device as a black box and a second that separates the gateway into different function boxes.

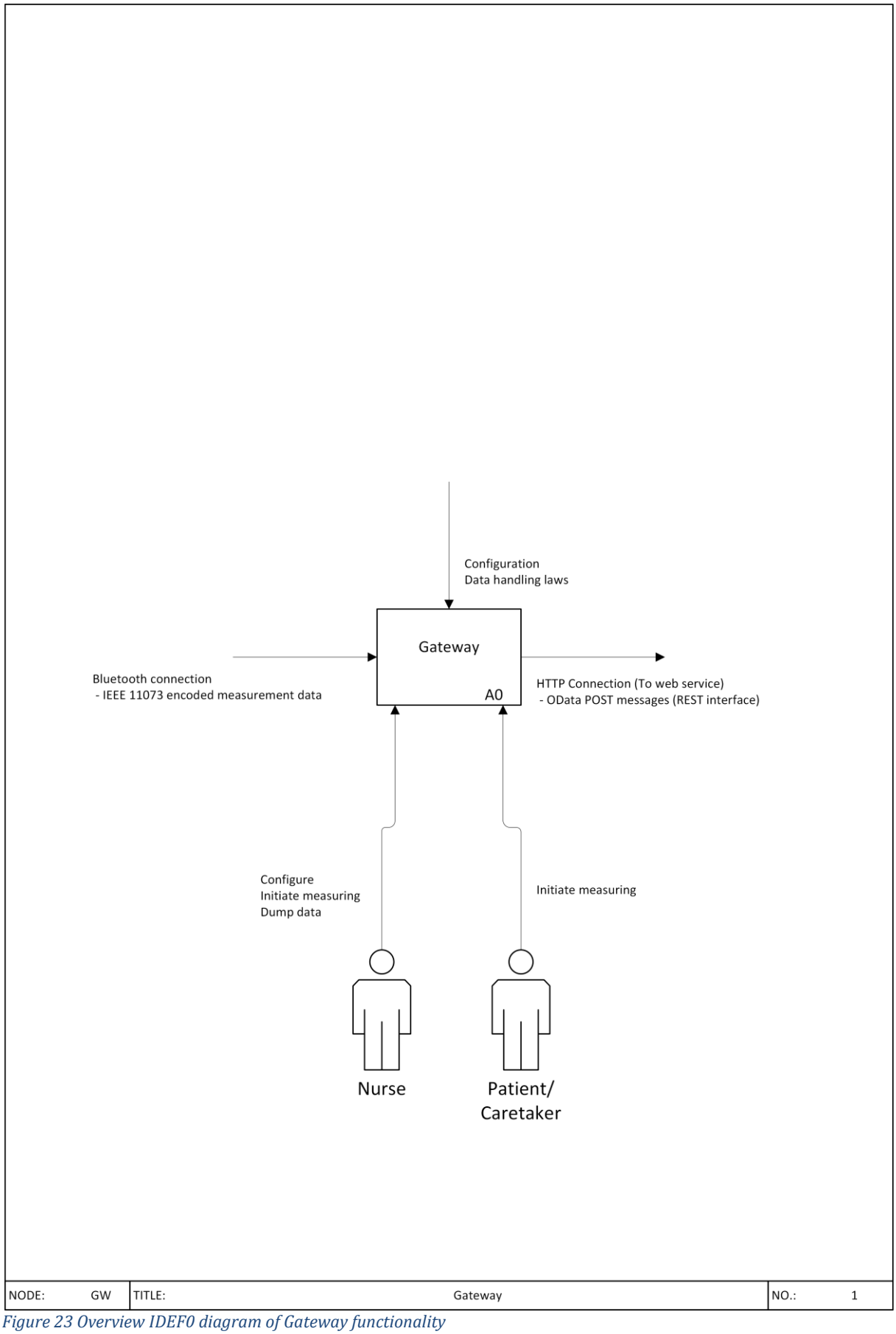


Figure 23 Overview IDEF0 diagram of Gateway functionality

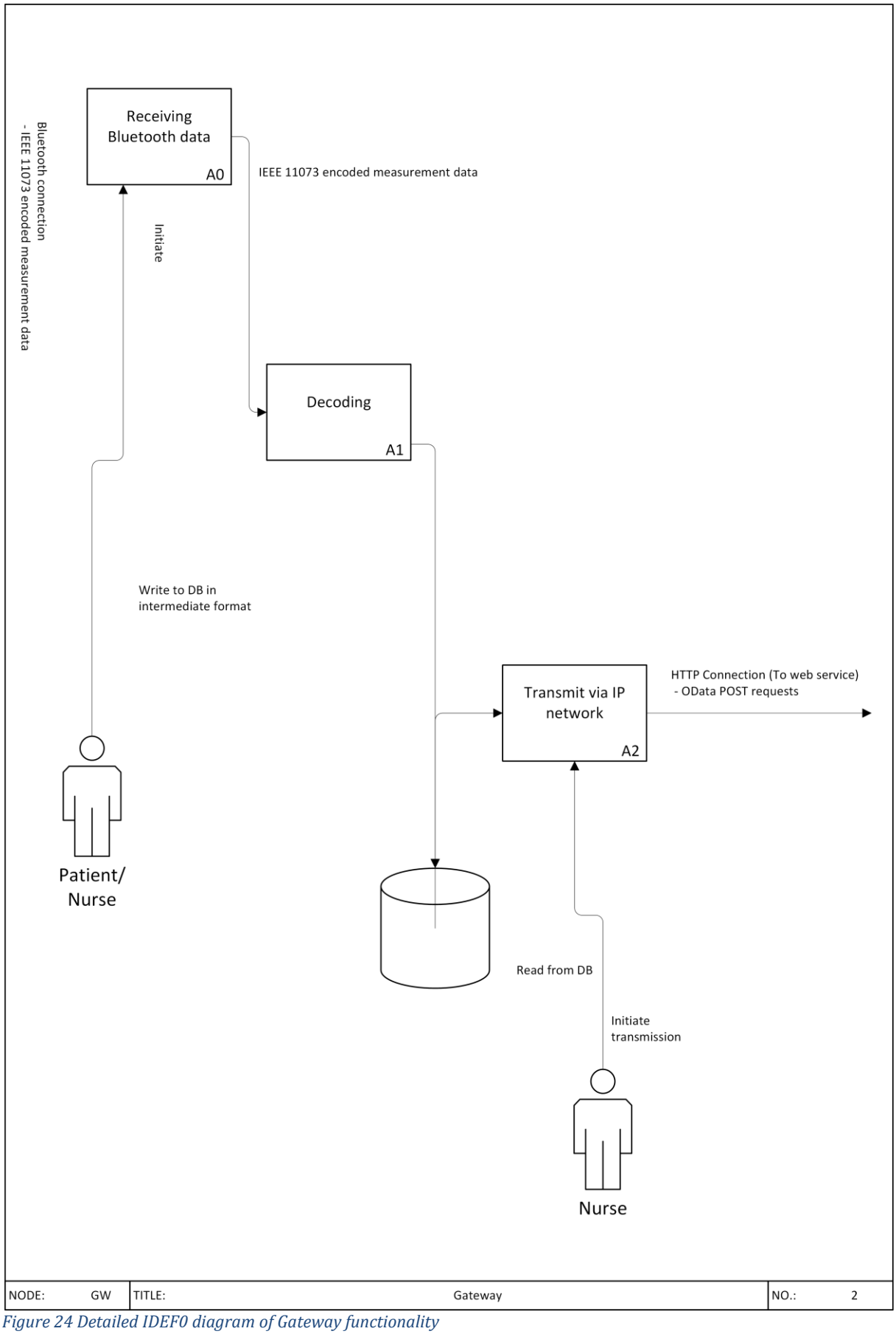


Figure 24 Detailed IDEF0 diagram of Gateway functionality

The Configure action in the first diagram is realised through the *SettingsActivity* and *Settings* classes, as described in 4.2.2 View and 4.2.4 Model. The Initiate measuring action is performed by the agent (i.e. the PHD), as this is how the ISO/IEEE 11073 protocol is designed (The Institute of Electrical and Electronics Engineers, Inc., 2014), and the application is listening for this, as described above, using the classes *PulseOximeterActivity*, *HealthService*, *JNIBridge*, *BluetoothCommunicationService* and *ReadThread*. The received data is then handled by *IncomingHandler* and *AntidoteHelper*, the measurement is identified with the patient returned from the *Settings* object and stored into the database which *DatabaseHandler* returns a reference to.

Going into detail on the second diagram, the Bluetooth data is received by the *ReadThread* and the *BluetoothCommunicationService*. The data is sent via message passing to *HealthService*, which uses *JNIBridge* to let Antidote decode the data into xml *Strings*, which in turn are sent by message passing to *PulseOximeterActivity*. From there *AntidoteHelper*'s *parseXml* method is used to convert the strings into xml *Document* objects, which are then used to create *PulseOximetryMeasurement* objects with the *fromXml* method in *PulseOximetryMeasurement*. These objects are then stored to the application's database when *PulseOximeterActivity* calls the method *writeMeasurementToDatabase* in *AntidoteHelper*, if the program flow is set to manual. Otherwise these objects are uploaded to the web service using *UploadPulseOximetryMeasurementsTask*, and if this fails they are stored in the database in the same way as when running the application with manual program flow.

If the application is run with manual program flow, the upload described in the end of the previous paragraph is performed upon user initiation (as can be seen in the diagram), with the difference that on success, the measurement will be removed from the database, using the *deleteMeasurementFromDatabase* method in *AntidoteHelper*.

4.3 Web service design

This report doesn't go as much into detail on the web service implementation as on the gateway application. Mainly because the biggest focus of the project was on the gateway application, but also because implementing the web service was very straightforward and the fact that it mostly consists of automatically generated classes. The full source code of both the web service and the gateway application is available on GitHub, and a link can be found in appendix 10.1 GitHub repository.

Just like the gateway application, the web service was developed using an MVC pattern, though it was implemented in .NET/C# instead of Android/Java, because it is intended to run on Microsoft Windows.

To realise the web service's OData interface, a library called Entity Data Model (EDM), written by Microsoft, is used. When using this library, OData GET request methods are generated automatically, and defining a method called *Post*, which takes a *PulseOximetryMeasurement* object as parameter automatically converts POST requests into *PulseOximetryMeasurement* objects as long as all constructor parameters are set in the request.

To handle the database communication, Microsoft's ORM Entity Framework is used

5. Testing

5.1 Testing the project

In the end of the design phase of the project a test plan was written, using the acceptance criteria of the user stories in the "Permission from hospital" scenario as input. The criteria were all valued as either needed or not needed, depending on whether they were part of base functionality or not. Thereafter three tests were written that together tests all needed acceptance criteria. An example, the first test, is shown here:

Tests criteria:

A1.1Gateway unit shall be able to communicate with sensor units

Course of action:

- The tester starts the gateway unit
- The tester starts the PHD
- The tester initiates measurement

Success criteria:

- Gateway unit receives a correct measurement from the PHD

The full test plan is available in appendix 10.3 Test plan, and when replacing all instances of Ascom Unite (Connect) with the web service, all tests pass, meaning the proof-of-concept is a success to some extent.

During the end of the design phase a risk and consequence analysis was conducted as well, and this analysis contains so called fall-back points that define how many percent of the full product design each part represent, thus giving a way to measure how much of the project was actually finished, programmatically speaking.

To summarise these fall-back points, 94% of functionality was achieved, 100% of robustness and 82.5% of usability. The lacking functionality is the possibility to configure PHDs from the gateway application, and the reason that this feature wasn't added is that it was discovered quite late during the project that the Antidote library didn't support choice of configuration, instead it just accepts the first configuration suggested by the PHD. Had it been discovered at an earlier point, it would have been possible to extend the Antidote library with this functionality, but as it was discovered in the end of the implementation phase, it was simply deemed non-vital for the proof-of-concept.

What lacks in usability is the possibility to configure PHDs through the user interface, which, given that there's no possibility to configure them at all through the application, isn't possible, and also the user interface lacks an activity log. The activity log was meant to be a collection of all notifications that showed up on the device, with filtering possibilities, so that a user could check this list, had they missed a notification. This function was never implemented, because during implementation and testing, there were never any thoughts that this functionality was missing, which probably is because

it's always clear if a measurement has been successfully uploaded or not, because of the listing of locally saved measurements.

When extending the proof-of-concept to something closer to a real product, it would probably be a good idea to implement an activity log though, especially to file errors when accessing the local database, as those would not be so easy to detect otherwise. This would also imply saving the log as a file rather than in the database, as saving it in the database wouldn't show database errors if the database was corrupt or in any other way inaccessible. Though on the other hand, an error message in the activity log view telling the user about the database not being accessible could probably make up for this.

To see the full risk and consequence analysis with all fall-back points, go to appendix 10.4 Risk analysis.

5.2 Checking a general device for compatibility

As health device manufacturers generally don't mention what protocol their products use to communicate and considering the fact that the Bluetooth SIG has chosen the ISO/IEEE 11073 protocol family for their health device profile (Bluetooth SIG, 2016), a blood pressure monitor from Withings was obtained to investigate if it did actually use ISO/IEEE 11073. The device in question is medically approved by the FDA in the USA and complies with European medical device regulations (Withings, 2016).

The device didn't use the Bluetooth health device profile. It identified itself as a headset (specifically major class 1024, AUDIO_VIDEO, and device class 1028, AUDIO_VIDEO_WEARABLE_HEADSET) and wanted to send its measurements to a specific application developed by Withings. To further investigate the communication between the blood pressure monitor and the smartphone, Bluetooth HCI snoop logging was enabled in the smartphone's Developer options and the logs were later examined in WireShark. The investigation showed that the actual data transfer between the devices was performed using the Bluetooth Serial Port Profile, which is meant to be used as a replacement for traditional serial cable connections (Bluetooth SIG, 2012). The messages sent were encoded in some not human readable format, and because investigating the protocol further wouldn't have given any direct value to the proof-of-concept, the investigation was put on ice.

When contacting Withings Support and asking about the protocol used, they referred to their public API for fetching medical data from their own web service (Withings, 2015). However it would be problematic not to access the device correctly, both because it would be a very bad solution to start another application and then fetch data from a web service to then present it and upload it to another web service, and also because it's not possible to give any guarantees about data handling when the measurements pass through a third party.

6. Results

6.1 Preconditions

The preconditions for this project were quite open, as the scope of the thesis description from Ascom was to explore how future products can implement communication protocols needed to retrieve data from connected IoT devices in healthcare, and the required results were an investigation regarding the most commonly used IoT protocols in healthcare and at least one working prototype (Ascom, 2015).

A general project plan was included with the thesis description, consisting of:

- Information gathering – define the problem area and read up on used technologies
- Analyse – analyse gathered data on Ascom specific needs and requirements
- Evaluation – evaluate different ways of solving the problem and select a preferred solution
- Design – create a test application prototype
- Validate – test the application
- Report – present the findings to Ascom Wireless Solutions

It boiled down to a more specific workflow plan, which also includes a time plan, which can be seen in Figure 25.

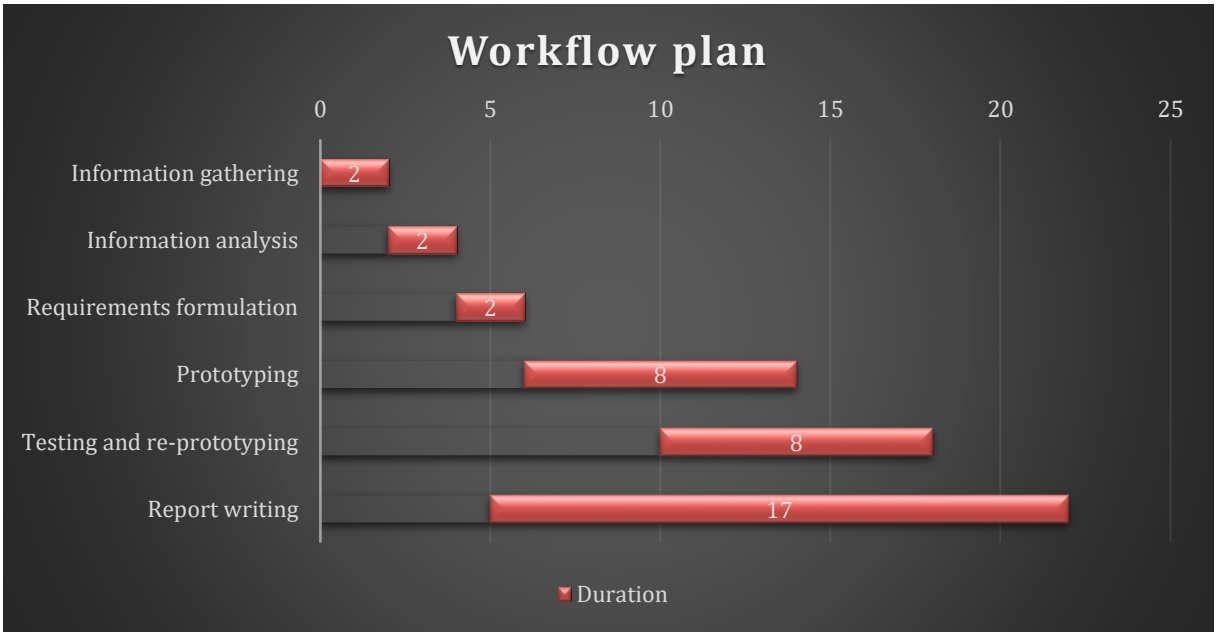


Figure 25 Original project time plan, showing the intended number of weeks to be spent on each part of the project.

The original time plan was actually kept to, to a large extent, though prototyping and testing and re-prototyping more or less merged together.

6.2 Findings

There are many different proprietary IoT protocols used by medical devices, sometimes even the same vendor uses different proprietary protocols for their different products (Day, 2011). There is however an attempt to standardise PHD communication, in the

form of the IEEE 11073 standards family (The Institute of Electrical and Electronics Engineers, Inc., 2014).
There has been projects suggesting solutions to how PHDs can be connected to the Internet, for example by letting PHDs communicate ISO/IEEE 11073 over CoAP on wireless network connections or letting them use IPv6 over Bluetooth LE (Martins, Santos, Perkuisch, & Almeida, 2014).

There has also been projects which have created adapters for health sensors using proprietary protocols to ISO/IEEE 11073, such as a USB connected weighing scale (Seo, Kim, Lee, & Kim, 2014) and even the Nintendo Wii Balance Board (Park, Lim, Jung, & Park, 2010).

A master's thesis has been written with a similar focus, though it didn't have the focus on general device compatibility, but rather two different proprietary systems. It focuses mainly on creating a reliable way of communicating over UDP and creating a good user interface in the receiving web service, but was also thought to have hospital integration and two-way communication. Also it didn't actually implement sensor connections, but only simulates data, and here this thesis fills a large gap (de Gouveia, 2013).

There are many started open source projects with the object of creating an ISO/IEEE 11073 library, but none seem to be finished, though Antidote features enough functionality to create working implementations.

6.3 Results

Figure 26 The gateway application receiving a measurement from a pulse oximeter

This thesis project has resulted in a working prototype for personal health device integrations, consisting of an Android application and a .NET web service. The Android application can receive measurement data from pulse oximeters using the ISO/IEEE 11073-20601 protocol over Bluetooth, tag measurements with a patient identifier, transfer measurements to the web service and store measurements in a local SQLite

database in case of lacking connectivity with the web service. The web service is a REST interface using the OData protocol, able to store measurements in an MSSQL database and fetch stored measurements according to OData queries.

The picture seen in the beginning of this subchapter, Figure 26, shows the gateway application running on a low-end Android phone, receiving a measurement from a pulse oximeter. To get a better picture of what the application looks like, see chapter 4.2.2 View for screenshots. To get an idea on how output from the web service can look and be used, see chapter 2.5 The Open Data protocol.

Both the findings and the results act as valuable input for Ascom in their future work.

7. Conclusion and future work

7.1 Conclusion

It is fully possible to create an integration of personal health devices using the ISO/IEEE 11073 protocol family and a web service and a plausible way of doing this is to use a gateway device as an adapter to connect the two end parts.

Also the ISO/IEEE 11073 protocol family seems to be the future standard for personal health devices, though this is not totally clear, as there still are many more manager devices than agent devices certified by Continua, six years after the announcement of the ISO/IEEE 11073-20601 protocol (Personal Connected Health Alliance, 2015). Though there isn't any need to get a product certified by Continua just because it uses the ISO/IEEE 11073 protocol, and an investigation should be done on how common it is to get such products certified by Continua, or rather how common it is not to, in order to be able to say more about the extent to which ISO/IEEE 11073 is used in available medical sensor devices.

7.2 Future work

When thinking of possible future work, it's a good idea to start out with what would need to be done to turn the proof-of-concept created with this thesis into a real product. Then there's of course a more scientific point of view, namely, what else than the intended product could be developed from this proof-of-concept, and what additions could be made for it to generate data useful for other research.

7.2.1 Creating a full integration

To create a full integration with either a messaging system or an electronic health record system, a link between the backend and this other system would have to be created. It would probably not be so much work finishing this, but still some considerations would have to be made, for example if this last part of the integration should read from the web service's database or send the received measurement objects on to the next system directly. As the integration in this proof-of-concept uses the Open Data standard for sending messages, it would also be possible to recreate the backend from scratch quite fast on almost any platform, for example if one doesn't use a Windows/.NET environment.

What is more important when creating a full product is that it would most certainly be deemed a medical device and would therefore have to comply with patient data handling laws as well as personal data handling laws. Furthermore it would have to be approved by the appropriate authorities for the intended market(s).

Another issue could be licensing, for example if a company would like to release an integration under their own proprietary license. This is possible to do when using the Antidote library, as it is licensed under the GNU Lesser General Public License, which allows other works to use the library without licensing this new work under any specific license, though it depends on how the new work is distributed. For the work to be considered a "work that uses the Library", it has to link dynamically to the library and be distributed separately. If the work is linking statically to the library or is distributed as a single executable package, it's considered a "work based on the Library" (Free Software

Foundation, 1999). It is of course possible to distribute the gateway part of the integration as two applications on e.g. Google Play, which is the case with some other applications already, but it creates hassle for the user and is certainly not something to wish for.

The other option would be to implement another ISO/IEEE 11073 library, which might not be a bad idea as it for example could be implemented in the same language as the rest of the application and thereby make coupling and debugging easier. The developers of Antidote claim that portability was their most important goal when developing the library, hence the usage of C and its standard library (Livio, et al., 2012). There is however a problem with the portability, since the C POSIX library is the one seen as the C standard library in Antidote and headers specific to this superset of the ANSI C standard library (such as strings.h) are used. This implies quite much work for getting core Antidote to just compile on Microsoft Windows, which still happens to be a quite common operating system with about 80% of the desktop OS market (StatCounter GlobalStats, 2015).

For a real product to be successful on today's market, it would probably have to implement support for more protocols than only ISO/IEEE 11073 since there still doesn't seem to be very many agent devices on the market using this standard. If using Antidote, there is a plugin system, and so plugins could be written for more protocols. This is a good design pattern since it is modular and thereby enables re-usage of code, hence it would be a good idea for a new ISO/IEEE 11073 protocol implementation to also support plugins.

When defining the software design for a new implementation, it would be a good idea to rewrite the user stories from real interviews, as those of now are mainly based on qualified guesses. Some things, such as ease-of-use and high automation, are obviously going to end up in the final definition, but other less general statements (from a software engineering point of view) are not necessarily going to be as guessed.

7.2.2 Security

The proof-of-concept has focused on security to a certain extent, but it sure is not good enough for a real product. For example the communication between the gateway and the backend should use https, which there is a branch for in the repository, but it isn't tested, since testing was only done on an internal IP network without DNS and TLS needs DNS addresses to verify authenticity. Without verified authenticity an exception is thrown and no connection is set up.

Another security issue is authentication of the gateway devices or their users; as it is now, the backend is wide open for reading and writing for anyone. A way to do this could be user authentication with certificates. The gateway application could have an encrypted key-value store for storing the certificates and in the case of an Android device with a fingerprint reader use fingerprints as keys. This would be a really easy to use type of authentication, but further research would have to investigate whether it is secure enough. Otherwise more traditional user authentication with usernames and passwords could of course be used.

As of now, patient identification uses a string datatype, the thought here is to keep the implementation open for future choices about how to tie measurement data to specific persons. What type of identifier to actually use in a real product implementation is subject of future research, but to comply with patient data handling laws and data handling laws, as well as to protect the privacy of the patients for moral and security reasons, it would be a good idea not to use anything directly identifying a specific person. For example, the US patient data handling law HIPAA lists name, address, birth data and social security number as identifiers. It also states that health information without such identifiers is not considered to be Protected Health Information (U.S. Department of Health & Human Services, 2013). It is a good idea to try avoid handling PHI, as this comes with several requirements (U.S. Department of Health & Human Services, 2013).

One possibility to get around this would be to simply use the database id from the hospital's internal patient database, which would probably be an integer and not tell anyone without access to that database anything about whom the measurement data belongs to. If using a database id as in the suggested solution above, one would have to think about security when transferring this id to the gateway application, i.e. how to transfer it.

One way to get the id to the gateway would be to have an interface in the backend that would return this database id when requested with a real patient identifier. This would however introduce a few attack vectors. For example, if an attacker were able to eavesdrop on the communication, they could tie the transferred database IDs to real persons and by that identify health information that they get hold of. Another possible attack would be if an attacker got hold of an authorised nurse's authentication details, then they could query the backend to make a list of identifiers with their database IDs. Also exposing a part of the internal database to the Internet might, with the help of security flaws, expose the full database to the Internet.

Another way of getting IDs into the gateway would be to use schedules and/or positioning (mostly useful for home care, but indoor positioning systems are in the making). As there can be delays to schedules and as positioning services are not always offering the level of preciseness needed or wanted, there would have to be some way of verifying, and if needed correcting, the identity inserted when collecting measurement data. When using schedules, this could be realised by showing which schedule position the gateway application thinks is the current one, and letting the application's user press it, get the schedule presented to them and choose the right one. If one would only use positioning, a map would probably be the best way to verify and correct for whom the data is measured, as using addresses would insert identifiers into the application.

The database ID could also be stored in a tag which the patient in question wears, e.g. as a bracelet. The tag could for example consist of a barcode or an NFC tag. Most modern Android devices have both a camera good enough for reading bar codes of different formats as well as an NFC transceiver. Using NFC tags or QR codes would enable the use of advanced cryptography with extra random data, as QR codes can hold almost 3 kB of data (Denso Wave Inc., 2015) and the slowest NFC data rate would transfer a little more than 3 kB in a quarter of a second (NFC Forum, 2013). There wouldn't necessarily be any need for encrypting information contained in a barcode worn by the patient, as they

themselves can control to whom they show it, but encrypting data on an NFC tag is probably a good idea, since there is a possibility of eavesdropping on NFC communication, possibly on distances of about 1 metre when using a regular NFC tag, which sends in passive mode (Haselsteiner & Breitfuß, 2006).

7.2.3 Notifications

The design documentation mentions a number of notifications that should be presented on both ends of the integration. Those telling about the program flow in the gateway application are already implemented, but other ones, such as notifications when there are gaps in the measurement series, are not yet implemented. Such gaps can of course occur for natural reasons, for example if the patient had to go to the toilet or got delayed with something else. In any case, if there is a schedule for when measurements should be taken, the gateway device should notify its holder when a certain time has passed after a predicted measurement hasn't been received. If the measurements are of vital importance for the patient's life or health status, notifications should also be given from the backend to the responsible nurse at the hospital, so that they can contact the patient in question and ask about the delay reason.

Notifications about sensor conditions should also be implemented, and the ISO/IEEE 11073 protocol already includes sensor status codes, making it a question of reading and handling those codes. They include for example the sensor being disconnected, malfunctioning, displaced or off (The Institute of Electrical and Electronics Engineers, Inc., 2009). Possibly instructional images can be added to these notifications.

7.2.4 Uses in other fields of science

A system of this type would gather a lot of medical data, which could be used for research in the field of medicine, specifically big data and machine learning implementations, but also generally. To make the data match this purpose, one could generate public databases with measurement data bundled with age group, gender and other available health data that can't serve as a patient identifier.

Further investigation should also be done to see if there actually are any benefits in using a system like this. For example, does it save administration time for nurses, so that more time can be spent with patients? Does it heighten the perceived quality of life or quality of care for patients who can use PHDs instead of large stationary devices? Would cheaper and more mobile measuring devices increase the number of measurements taken or the number of patients getting measurements taken? If so, would the increased data result in more accurate or earlier diagnoses, or would it cause a data flood which doesn't come with any real benefit? Would massive amounts of measurement data analysed as proposed in the previous paragraph result in the possibility to predict upcoming states for certain measurement patterns, and would that be usable together with PHDs sent home with patients for longer-term monitoring?

8. Nomenclature

Android Debug Bridge (ADB) –A command line tool for communication with Android devices and Android emulator instances.

Agent – An agent in the ISO/IEEE 11073 protocol family is a device that measures medical data in some way. The agent is responsible for setting up connections to a manager when it has new medical data recorded, which it not yet has shared.

Android Native Development Kit (NDK) – A software suite for compiling C/C++ code into native machine code for the different platforms on which Android runs.

Android Software Development Kit (SDK) – A set of software for developing Android software, including e.g. a handset emulator, libraries, a debugger and sample code.

Collector unit – A unit that can collect data from a concentrator and transfer it to a hospital information system.

Concentrator – See gateway unit.

Constrained Application Protocol (CoAP) – An application layer protocol which is easily translatable to HTTP, but has much smaller overhead and is used over UDP instead of TCP, thus making it lightweight enough for IoT devices (things).

Gateway unit – A unit that acts as a gateway for (multiple) generic personal health devices, enabling them to communicate with hospital information systems.

Integrated Development Environment (IDE) – A computer program or program suite for making programming easier, it usually contains at least a text editor a compiler and a debugger. Well-known examples include Microsoft Visual Studio, Eclipse and NetBeans.

Manager – A manager in the ISO/IEEE 11073 protocol family is a device capable of receiving data from agents.

Model View Controller (MVC) – A software design pattern where the code is divided in different classes depending on whether the code has mostly to do with the Model (data model, more or less pure information), the View (the interface that the user sees) or the Controller (code that manipulates, or controls, the model and to some extent the flow of the program, and hence the view).

Myco – An Android smart phone developed by Ascom, designed to be used by nurses at a hospital.

Near Field Communication (NFC) – Wireless transfer protocols for short range (typically up to 10 centimetres) radio data communication. Has support for so called tags, which are passive data stores, powered by the electromagnetic field that is the RF signals from the reader device.

Object-Relational Mapping (ORM) – A technique in object-oriented programming for conversion between incompatible type systems, can for instance be used to make a database seem like a regular collection type.

Personal Health Device (PHD) – A small medical or fitness sensor device, designed to be worn. It is often a consumer product, but could also be intended for medical use.

Protected Health Information (PHI) – Health information of any kind bundled with an identifier for either the person regarded by the information or any of that person's relatives, according to the HIPAA patient data handling law.

Quick Response (QR) code – A two-dimensional bar code, can be used to contain e.g. hyperlinks, contact information or just plain text.

Server – A unit that receives collected data from personal health devices, it could for example be a hospital information system or a patient records system.

Uniform Resource Locator (URL) – What usually (informally) is referred to as "web address", it generally consists of a scheme followed by "://" followed by a host followed by a path, e.g. http://chalmers.se/.

Unite – A messaging system protocol developed by Ascom, features for instance message priority.

9. References

9.1 Reports and articles

Castellani, A. P., Bui, N., Casari, P., Rossi, M., Shelby, Z., & Zorzi, M. (2010). Architecture and Protocols for the Internet of Things: A Case Study. *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on* (pp. 678-683). Mannheim: IEEE.

Day, B. (2011, January 12). *Standards for Medical Device Interoperability and Integration.* Retrieved January 13, 2016, from Patient Safety & Quality Healthcare: http://psqh.com/standards-for-medical-device-interoperability-and-integration

de Gouveia, F. (2013). *Transmission and presentation of medical sensor-data.* Aveiro: Universidade de Aveiro.

Department of Health. (2013). *Reference costs 2012-13.* London: Department of Health.

Haselsteiner, E., & Breitfuß, K. (2006). Security in Near Field Communication (NFC). *Workshop on RFID Security 2006.* Graz: IAIK, TU Graz.

Martins, A. F., Santos, D. F., Perkuisch, A., & Almeida, H. O. (2014). IEEE 11073 and Connected Health: Preparing Personal Health Devices for the Internet. *Consumer Electronics (ICCE), 2014 IEEE International Conference on*, (pp. 274-275). Las Vegas.

Park, C., Lim, J.-H., Jung, H.-Y., & Park, S. (2010). ISO/IEEE 11073 PHD Standardization of Weighting Scale Using Nintendo's Wii Balance Board™ for Healthcare Services. *Consumer Electronics (ICCE), 2010 Digest of Technical Papers International Conference on*, (pp. 195-196). Las Vegas.

Roman, D. H., Conlee, K. D., Abbott, I., Jones, R. P., Noble, A., Rich, N., . . . Costa, D. (2015, June 29). The Digital Revolution comes to US Healthcare. *Equity Research, Internet of Things, Vol. 5*, pp. 1-54.

Seo, D. S., Kim, S. S., Lee, Y. H., & Kim, J. M. (2014). Implementation of Personal Health Device Communication Protocol Applying ISO/IEEE 11073-20601. *International Journal of Distributed Sensor Networks, 2014*(1), 176-179.

Weber, R. H., & Weber, R. (2010). *Internet of Things - Legal Perspectives.* Berlin: Springer-Verlag GmbH.

9.2 Standard specifications

Bluetooth SIG. (2007, July 26). *Bluetooth Specification Version 2.1 [vol 2] + EDR - Radio Specification.* Retrieved February 4, 2016, from Bluetooth Technology Website: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=241363&_ga=1.121150856.1431713180.1441200829

Bluetooth SIG. (2007, July 26). *Bluetooth Specification Version 2.1 + EDR [vol 1] - Architecture.* Retrieved February 4, 2016, from Bluetooth Technology Website: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=241363

Bluetooth SIG. (2007, July 26). *Bluetooth Specification Version 2.1 + EDR [vol 2] - Baseband Specification.* Retrieved February 4, 2016, from Bluetooth Technology Website: https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=241363

Bluetooth SIG. (2012, July 24). *Bluetooth Specification - Serial Port Profile.* Retrieved from https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=260866&vId=290097&_ga=1.149442998.1431713180.1441200829

Bluetooth SIG. (2016). *Health Device Profile.* Retrieved January 14, 2016, from Bluetooth Technology Website: https://www.bluetooth.com/specifications/assigned-numbers/health-device-profile

CEN. (2016). *CEN - Advanced search - Publications and Work in Progress.* Retrieved January 14, 2016, from CEN - European Committee for Standardization: http://standards.cen.eu/dyn/www/f?p=204:105:0:::::

Denso Wave Inc. (2015). *Information capacity and versions of the QR Code | QRcode.com | DENSO WAVE*. Retrieved December 21, 2015, from QRcode.com | DENSO WAVE: http://www.qrcode.com/en/about/version.html

The Institute of Electrical and Electronics Engineers, Inc. (2009). *IEEE Std 11073-10404™-2008, Health informatics—Personal health device communication—Part 10404: Device specialization—Pulse oximeter.* New York: The Institute of Electrical and Electronics Engineers, Inc.

The Institute of Electrical and Electronics Engineers, Inc. (2014). *IEEE Std 11073-20601™-2014, Health informatics—Personal health device communication, Part 20601: Application profile—Optimized Exchange Protocol.* New York: The Institute of Electrical and Electronics Engineers, Inc.

9.3 Books

Cohn, M. (2004). *User Stories Applied.* Boston: Pearson Education, Inc.

Leffingwell, D., & Widrig, D. (2000). *Managing Software Requirements.* Upper Saddle River: Addison Wesley Longman, Inc.

9.4 Interviews

Bentzer, J. (2016, February 3). Follow-up meeting. (A. Svanström, Interviewer)

Snyder, M. (2015, September 23). E-mail correspondence about WristOx development kit. (A. Svanström, Interviewer)

Withings, W. (2015, September 22-29). Conversation about Withings Bluetooth-enabled devices. (A. Svanström, Interviewer)

9.5 Other sources

Android Open Source project. (2016). *Android NDK.* Retrieved January 15, 2016, from Android Developers: http://developer.android.com/tools/sdk/ndk/index.html

Android Open Source project. (2016). *Android Studio Overview.* Retrieved January 15, 2016, from Android Developers: http://developer.android.com/tools/studio/index.html

Android Open Source project. (2016). *Connecting to the Network.* Retrieved January 20, 2016, from Android Developers: http://developer.android.com/training/basics/network-ops/connecting.html

Ascom. (2015). *Master Thesis Proposal - IoT Communication Protocols in Healthcare.* Retrieved March 19, 2015, from Ascom Sweden: https://delta.hr-manager.net/ApplicationInit.aspx?cid=1011&departmentId=8589&ProjectId=66922&MediaId=5

Bluetooth SIG. (2011, March 6). *Bluetooth Basics.* Retrieved February 4, 2016, from Bluetooth Technology Website: https://web.archive.org/web/20110306131542/http://www.bluetooth.com/Pages/Basics.aspx

Bluetooth SIG. (2016). *Bluetooth.* Retrieved February 3, 2016, from Bluetooth Technology Website: https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth

Bluetooth SIG. (2016). *Bluetooth Technology Basics.* Retrieved February 4, 2016, from Bluetooth Technology Website: https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics

Bluetooth SIG. (2016). *Low Energy.* Retrieved February 4, 2016, from Bluetooth Technology Website: https://www.bluetooth.com/what-is-bluetooth-technology/bluetooth-technology-basics/low-energy

Cisco. (2013, July 29). *Connections Counter: The Internet of Everything in Motion.* Retrieved February 3, 2016, from the network - Cisco's Technology News Site: http://newsroom.cisco.com/feature-content?type=webcontent&articleId=1208342

Ericsson (Director). (2015). *Ericsson launch: Accelerating IoT* [Motion Picture]. Retrieved February 3, 2016, from https://www.youtube.com/watch?v=FyfKxzLbmHc

Free Software Foundation. (1999, February). *GNU Lesser General Public License v2.1 - GNU Project - Free Software Foundation.* Retrieved December 18, 2015, from GNU Project - Free Software Foundation: http://www.gnu.org/licenses/lgpl-2.1.html

Höller, J., & Arkko, J. (2012, June 14). *Internet of Things Propels the Networked Society.* Retrieved February 3, 2016, from Ericsson Research Blog: http://www.ericsson.com/research-blog/internet-of-things/internet-things-propels-networked-society/

Larmo, A. (2015, July 14). *Wi-Fi for the Internet of Things.* Retrieved February 3, 2016, from Ericsson Research Blog: http://www.ericsson.com/research-blog/internet-of-things/wi-fi-for-the-internet-of-things/

Livio, A., Martins, A., Bezerra, D., Pfützenreuter, E., Silva, F., Martins, J., . . . Almeida, H. (2012, March 21). *Antidote: Program Guide.* Retrieved December 18, 2015, from Signove OSS wiki: http://oss.signove.com/images/c/c7/AntidoteProgramGuide.pdf

Mathias, C. (2015, January 2). *Wi-Fi® and the Internet of Things: (Much) more than you think.* Retrieved January 25, 2016, from Wi-Fi Alliance: http://www.wi-fi.org/beacon/craig-mathias/wi-fi-and-the-internet-of-things-much-more-than-you-think

NFC Forum. (2013, December 17). *What are the data transmission rates? - NFC Forum*. Retrieved December 21, 2015, from NFC Forum: http://nfc-forum.org/resources/what-are-the-data-transmission-rates/

OData. (2015). *Libraries.* Retrieved January 14, 2016, from OData - the Best Way to REST: http://www.odata.org/libraries/

OData. (2015). *OData - the Best Way to REST*. Retrieved January 14, 2016, from OData - the Best Way to REST: http://www.odata.org/

Oracle. (2015). *Primitive Data Types.* Retrieved January 21, 2016, from The Java™ Tutorials: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html
Personal Connected Health Alliance. (2015). *About Continua*. Retrieved January 13, 2016, from Continua: http://www.continuaalliance.org/about-continua
Personal Connected Health Alliance. (2015). *Certified Product Showcase.* Retrieved January 22, 2016, from Continua: http://www.continuaalliance.org/products/product-showcase
StatCounter GlobalStats. (2015, December). *Top 7 Desktop OSs on Dec 2015 | StatCounter GlobalStats*. Retrieved December 18, 2015, from StatCounter GlobalStats: http://gs.statcounter.com/#desktop-os-ww-monthly-201512-201512-bar
U.S. Department of Health & Human Services. (2013, January). *Summary of the HIPAA Privacy Rule | HHS.gov*. Retrieved December 21, 2015, from HHS.gov: http://www.hhs.gov/hipaa/for-professionals/privacy/laws-regulations/index.html
Withings. (2016). *Withings Wireless Blood pressure Monitor.* Retrieved January 21, 2016, from Withings: https://www.withings.com/uk/en/products/blood-pressure-monitor?
Volvo Car Group. (2015, March 2). *Volvo Cars' connected car program delivers pioneering vision of safety and convenience.* Retrieved January 13, 2016, from Volvo Cars: https://www.media.volvocars.com/global/en-gb/media/pressreleases/159478/volvo-cars-connected-car-program-delivers-pioneering-vision-of-safety-and-convenience
Xamarin Inc. (2016). *Simple and transparent pricing.* Retrieved January 21, 2016, from Store - Xamarin: https://store.xamarin.com/

10. Appendixes

10.1 GitHub repository

The GitHub repository containing the full source code for this project can be found at https://github.com/daemondeas/phdIntegration

10.2 Gateway/Concentrator unit reasoning and definition document

The gateway/concentrator unit

A very vital part of the system upon which the thesis "IoT Protocols in Healthcare" is based, is the gateway unit or concentrator unit. It has these two names because depending on the scenario in which it is used, it acts more like one than the other, but it should still be possible to use the same device for both functionalities.

Gateway functionality

When this device works as a gateway, it will relay data from Personal Health Devices (PHDs) to a Unite messaging system over the Internet, thus working as an ISO/IEEE 11073/Bluetooth -> Unite/IP gateway. This could be done in various ways.

One way to do this is to have the gateway unit function only as a gateway, i.e. to just relay the ISO/IEEE 11073 encoded data to the Unite Connect server, and let the Connect server translate it into Unite messages.

Another way is to let the gateway unit be a bit "smarter" and let it handle the translation and relay the data via for example the Unite REST API.

Concentrator functionality

When functioning as a concentrator the device will concentrate data from multiple PHDs into medical data for one patient. For a future implementation it might be interesting to use a single device to concentrate data for multiple patients, but as far as this proof-of-concept goes, it is deemed out of scope.

System design

As stated above, there are different possibilities on how to design the gateway/concentrator unit, however, that doesn't only hold for the software but also for the hardware.

Hardware design

As the unit in question is intended to be portable it cannot be too large in terms of volume and mass, perhaps with the exception of a possible future concentrator for multiple patients (such a device would most likely be stationary in anyway). However there are still multiple available options when considering a small, powerful enough device to handle the above described gateway and concentrator tasks. To summarise, we know that at least the following requirements must hold for the hardware in question.

- Low mass (max 200g should be a good aim)
- Small volume (either possible to put in a pocket or to wear on the body)
- Bluetooth
- Wi-Fi or mobile data connection
- Possible to run both on battery and external power

As of today there are quite many available products on the market that conform to those requirements, e.g. smartphones, single-board computers with a battery pack and standalone smartwatches.

Using a single-board computer, like for example the raspberry pi, gives huge flexibility when it comes to the choice of platform. Considering the raspberry pi, there are lots of Linux and BSD distributions as well as Windows 10 available as operating systems to use for it, making it possible to choose virtually any framework or programming language for implementing the gateway/concentrator software. Another possibility when using a single-board computer would of course be to write a small operating system designated for just the tasks in question. On the other hand, the single-board computers are often just a single board and the development process would also have to include casing and integration with some kind of battery pack. Although this product packaging would be outside the scope of this thesis project, it's still worth taking into consideration in case parts of the project will actually be used for a future product.

In the case of smartphones, there is an advantage in that the market is full of devices pre-packaged with all necessary hardware and a high-capacity battery. Another advantage is that the person using the gateway/concentrator might already have a smartphone that could run the software, and would thus not need any extra device for obtaining this functionality. A downside is lack of flexibility; today there are two major platforms, Android which uses Java (and C/C++) and iOS which uses Objective C or Swift. There is also the Windows phone platform, with a couple of percent market share, using the .NET framework, and a few others all below one percent market share. There is actually also a framework called Xamarin, with which it is possible to create applications for Android, iOS and Windows phone with mostly the same C# code, however when reading discussions about it on different developer forums, it doesn't seem to be production ready yet.

Lastly when looking at standalone smart watches, there are a few different with very varying operating systems, but for example those which runs full Android could use the exact same application as that for an Android smartphone based system. The watches have a clear advantage in terms of being mobile and easy to wear, but due to their small size they also have quite small batteries in comparison to their power consumption.

Software design

When it comes to the software design, one question is whether the ISO/IEEE 11073 decoding should be performed at the gateway unit or at the Connect unit, but whichever decision is made here, a decoding module will be needed. To make the software as portable and reusable as possible it is good to use a modular design, and therefore the different parts of the software system will be described per module.

Bluetooth module

This software module will handle the Bluetooth part of the communication, it should at least have these functions/methods (or equivalent depending on framework/programming language):

- void write(byte[] message) – transmits the data of message to the connected Bluetooth device
- void read() – receives messages from the connected Bluetooth device and sends them via message passing to the control module
- bool connect(BluetoothDevice device) – tries to connect to device and returns success status

- void disconnect() – disconnects from connected device

Control module

Contains the main logic of the Gateway, i.e. what to do with received data, controlling of connected devices etc. It should contain at least the following functions/methods:

- void handleReceivedMessage(Message msg) – decides what to do with msg, depending on its type and contents
- void initiate() – initiates connections to all configured PHDs and to any configured Connect server
- void addDevice(BluetoothDevice device) – adds device to the list of PHDs to connect to at initiation
- void removeDevice(BluetoothDevice device) – removes device from the list of PHDs to connect to
- bool configureConnect(IPNumber nr) – configures nr to be the IP address of the Unite Connect server to connect to, returns whether connection was possible
- void unConfigureConnect() – clears Connect server address
- PatientData add(PatientData pd1, PatientData pd2) – returns the combination of all medical data in pd1 and pd2, when a value is to be found in both of them (e.g. when combining data from a blood pressure monitor and a pulse oximeter, there could be duplicate pulse values) the one from pd2 is used

IP module

This part of the system handles communication over IP networks, in other words it is the part that connects to Connect. It should feature at least the following functions/methods:

- void setAddress(IPNumber nr) – sets nr to be the IP address of the server
- IPNumber getAddress() – returns the current server address
- bool send(PatientData pd) – sends pd to the configured server and returns whether transmission was successful
- void read() – receives messages from the server and sends them via message passing to the control module

Decoding module

This module will handle decoding of ISO/IEEE 11073 messages into something useful. It is proposed to use the Antidote library as this module, as it is already used in several Continua certified manager software and is also available under the GNU lesser general public license, meaning it is allowed to be used as is in commercial products. Whatever choice is made upon which module to use for decoding, it must at least contain the following function/method:

- PatientData decode(ISO/IEEE11073Data md) – returns the decoded values of md

10.3 Test plan

Test plan

The purpose of this document is to identify a way to check that the system developed in the context of the thesis "IoT Communication Protocols in Healthcare" is really a proof-of-concept for a solution of the problem described in the thesis proposal and in the system description.

As the system is described by four scenarios with associated user stories and acceptance criteria, this plan will be based upon one of those scenarios, namely the "permission from hospital" scenario. It is essentially, in terms of system setup, the same scenario as the "monitoring in a ward" scenario, with the sole difference that the Internet is used in place of a local network. The only system part not included in this scenario is the collector unit, which is not really needed to prove the concept of integrating small health devices into Unite, but more of a special case with an extra unit along the line of transmission.

Test setup

The setup for these tests will be equivalent to the system setup shown for the "permission from hospital" scenario in the system description document.

The tester will wear the PHD instead of a patient, and the tester will also initiate measurement.

The Gateway unit shall be paired with the PHD and configured to communicate with the Connect server.

The Connect server shall be configured to receive data from the Gateway unit.

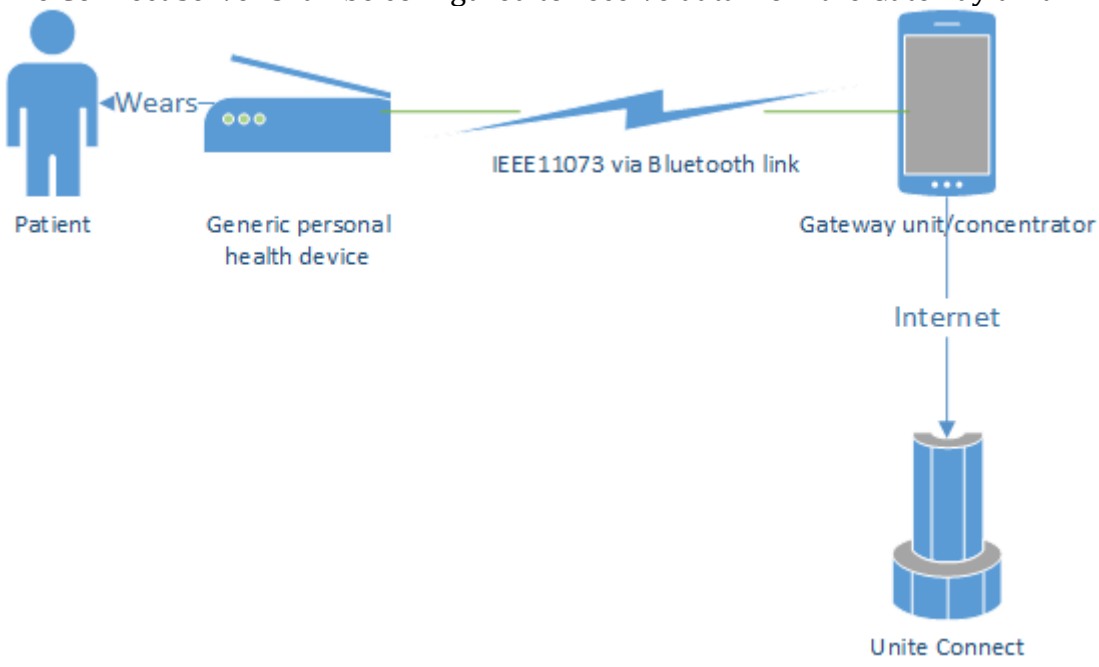


Figure 27 Picture of test system setup

Decisions about what to test

This plan goes through all acceptance criteria in the scenario "permission from hospital" and decides if the criterion is needed for the proof-of-concept or not, and if it is, it describes a way to test if the criterion holds.

Criteria

A1.1 Gateway unit shall be able to communicate with sensor units – *needed*

A1.2/A2.2 Gateway unit shall be able to send monitoring (sensor) data over the Internet to the hospital in real-time – *needed*

A1.3 Gateway unit shall notify the patient if a sensor unit falls off or stops transferring data – *not needed*

A1.4 Server shall be able to receive all data sent by the gateway unit – *needed*

A1.5 Server shall have the data presentable to nurses in real-time – *not needed*

A1.6 Server shall generate a notification in case of sensor data out of accepted ranges or not received sensor data at defined data reception times – *not needed*

A1.7 Gateway unit shall be easy to configure – *not needed*

A2.1 Gateway unit shall collect all sensor data sent by the sensor units – *needed*

A2.3 Gateway unit shall notify the patient and the hospital if it is unable to collect or transmit data – *not needed*

A2.4 Gateway unit shall be easy to use for the patient – *not needed*

A2.5 Sensor units and gateway unit shall not make the patient too immobile – *not needed*

Tests

T1. Sensor unit communication

Tests criteria:

- A1.1 Gateway unit shall be able to communicate with sensor units

Course of action:

- The tester starts the gateway unit
- The tester starts the PHD
- The tester initiates measurement

Success criteria:

- Gateway unit receives a correct measurement from the PHD

T2. Relaying data over the Internet

Tests criteria:

- A1.1 Gateway unit shall be able to communicate with sensor units
- A1.2 Gateway unit shall be able to send monitoring (sensor) data over the Internet to the hospital in real-time

Course of action:

- The tester starts the Unite server
- The tester starts the gateway unit
- The tester starts the PHD
- The tester initiates the Unite connection
- The tester initiates measurement

Success criteria:

- Gateway unit receives a correct measurement from the PHD
- Unite server receives a correct measurement from the gateway unit
- It is possible to find the measurement in the Unite system at most three seconds after it has been received by the gateway unit

T3. Continuous measurement

Tests criteria:

- A1.1 Gateway unit shall be able to communicate with sensor units
- A1.2 Gateway unit shall be able to send monitoring (sensor) data over the Internet to the hospital in real-time
- A1.4 Server shall be able to receive all data sent by the gateway unit
- A2.1 Gateway unit shall collect all sensor data sent by the sensor units

Course of action:

- The tester starts the Unite server
- The tester starts the gateway unit
- The tester starts the PHD
- The tester initiates the Unite connection
- The tester initiates measurement
- The tester lets measurement run for a minute
    - If it is a type of test where point measurements are performed (e.g. blood pressure), the tester instead performs three measurements
- The tester ends the measurement session

Success criteria:

- Gateway unit receives all measurements transmitted by the PHD
- Unite server receives all measurements that the gateway receives
- It is possible to find all measurement data in the Unite system at most three seconds after the last measurement was received by the gateway unit

10.4 Risk analysis

Risk and consequence analysis

This document is intended to identify larger risks associated with the project, analyse their possible consequences and give an idea on how to tackle them, should they occur.

Project risks

A time plan unit is not finished on time

Consequences:

- Deliverables needed for the next unit in the time plan might not be finished, thereby delaying also that unit
- The extra time needed to finish the unit will almost inevitably take working time from units later in the plan
- In the end the project might not be finished on time

Measures:

- Keep track of the time plan while working
- Should a unit seem to take longer time than planned:
    - Work extra hours
    - Prioritise what's left and do what's necessary first
    - Prioritise and remove what's not so important from this or coming units
- Let the time plan be a "living document", i.e. modify it if new revelations make it seem that some unit will take longer or shorter time

Technical risks

Communication with PHD(s) is impossible to implement

Consequences:

- At least one of the deliverables about PHD communication will not be delivered
- The name of the project will be misleading as no integration will be done

Measures:

- Generate data to be able to continue with the rest of the project

Communication with Unite is not possible to make secure

Consequences:

- Implementation cannot be used in a real product

Measures:

- Don't send sensitive data over open networks
- Continue with the rest of the project

Communication with Unite is not possible to implement

Consequences:

- At least one of the deliverables about Unite communication will not be delivered
- The name of the project will be misleading as no integration will be done

Measures:

- Write a detailed report about why it doesn't work

Android will have stability issues (e.g. killing the service(s) that handle(s) communication)

Consequences:

- Implementation cannot be used in a real product

Measures:

- Try getting it as stable as possible
- State that Android is not a suitable platform for integration of PHDs
- Think of what could be a better platform for PHD integration

Data will often be corrupted

Consequences:

- Data will be unusable

Measures:

- Also send checksums and allow retransmission requests

Communication links will be unstable (i.e. connection is dropped often)

Consequences:

- Data might be missed
- System might "go down"

Measures:

- Re-connect automatically
- Check if it's possible to make connections more stable
- Save data until transmission is confirmed

Fall-back points

The fall-back points are indicators for measuring how big percentage of a project part is finished, should the part not be fully completed.

Functionality goals

- PHD communication 40%
  - Bluetooth connects 10%
  - PHD communicates with Antidote 30%
  - Antidote receives measurements 45%
  - Possibility to configure PHDs 15%
- Data handling 20%
  - Turning measurement data into measurement objects 15%
  - Storing of measurements in a local database 40%
  - Extracting wanted information from database 45%
- Unite communication 40%
  - Connecting to Connect module 20%
  - Converting measurement into Unite message 80%

Robustness goals

- Application doesn't freeze during testing or usage 20%
  - Doesn't freeze at all → 100%
  - Freezes for a short time, but continues to work afterwards → 50%
  - Freezes for a long time, without data loss → 5%
- Application doesn't crash during testing or usage 40%
  - Doesn't crash at all → 100%
  - Doesn't lose any data when crashing → 30%
- Application always receives incoming connections 40%
  - X/100 connections are received X%

Usability goals

- Configuration possible through UI 50%
  - Configuration of PHDs 15%
  - Local configuration (what/when to measure etc.) 45%
  - Configuration of Unite connection 40%
- UI notifies the user of what is happening 40%
  - PHD connection notification 15%
  - New measurement notification 30%
  - Problem notification 10%
    - There is a problem notification 50%
    - Good description of the problem 50%
  - Unite connection notification 15%
  - Receiving messages from Unite (e.g. "registered data for Patient X into medical records system") 30%
- UI keeps an activity log 10%
  - Log contains all notifications 50%
  - Possibility to set filters for what the UI shows from the log file 50%

10.5 Early system sketches

This appendix features two early sketches of how the system could look, they were drawn about 1.5 weeks into the project when the idea of a gateway had just arisen.

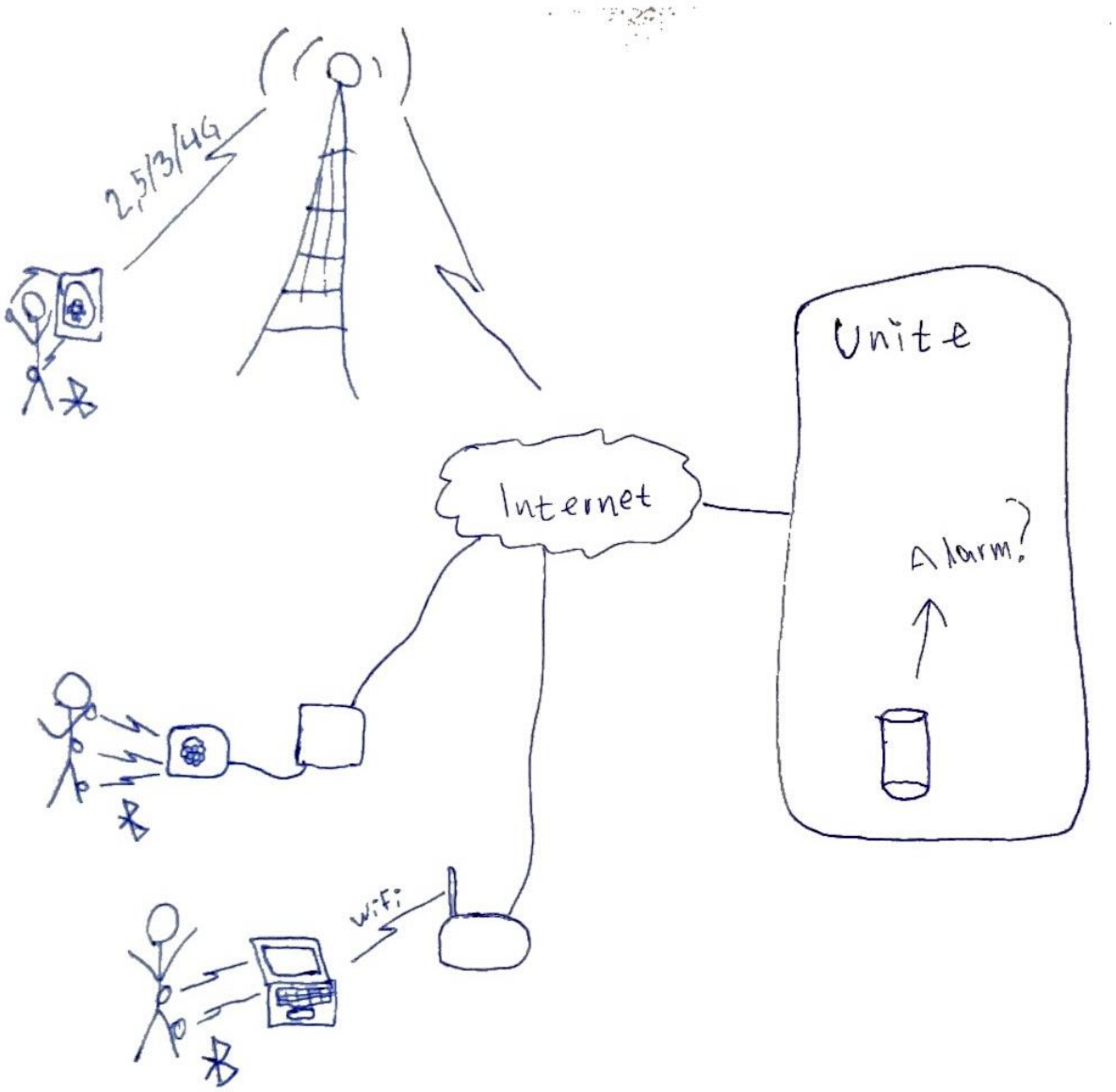


Figure 28 A sketch of an integration over the Internet

This first sketch shows the idea of the same software running on multiple devices, here an Android smartphone a Raspberry Pi single board computer and a pc laptop. The idea was to show a flexible system where IoT medical devices using Bluetooth could connect to a server over the Internet through various types of connection. The server could possibly raise alarms in case of critical measurements,

As the actual gateway application implemented is written in Java, it should be easy to make it run on a single board computer or a pc laptop, as long as they have Bluetooth connectivity. The biggest part would be to adapt the application to the respective operating systems' Bluetooth stacks.

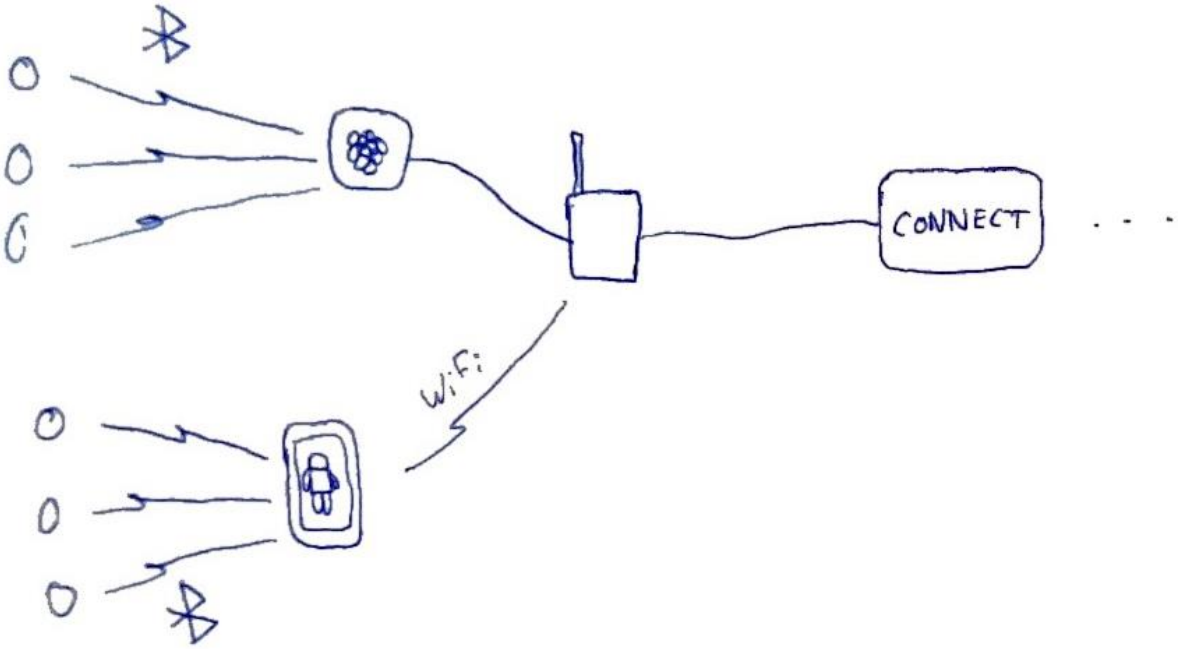


Figure 29 A sketch of an on-site integration over a local network

The second sketch shows two different devices running the same gateway software, connected to an Ascom Unite Connect server on a local network. It is a complement to the first sketch in the way that it shows hospital or care centre usage, rather than remote usage with Internet connections. The essential part is that the software should be written so that it works in both cases.