UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Lauris Kruusamäe

Optimizing Contextual Sensor Data Collection for Mobile Users

Bachelor Thesis (6 EAP)

Supervisor: Satish Narayana Srirama, PhD
Co-supervisor: Huber Flores, MSc

Author:..... "....." Month 2013

Supervisor:..... "....." Month 2013

Professor:..... "....." Month 2013

TARTU, 2013

Abstract

Nowadays, mobile applications are becoming more context aware due to technological achievements which enable the applications to anticipate users' intentions. This is achieved through using the device's own micromechanical artifacts that can be used to perceive the environment. However, this is constrained to the hardware limitations of devices.

A proposed solution for this has been made in the thesis "Context Sensor Data on Demand for Mobile Users Supported by XMPP" by Kaarel Hanson. The solution is to use XMPP for transporting sensor data from Arduino microcontroller to the cloud. Arduino provides low-cost hardware, while the cloud offers the reliable and high- availability means for storing and processing sensor data.

This solution shows that running on a 9V battery the microcontroller lasts for 101 minutes when using an Ethernet module for communications, and 161,5 minutes with a WiFi module. These results are not good enough for remote data collection with limited access to the microcontroller.

This thesis proposes an optimisation for the system so that instead of reading and sending sensor data every 10 seconds, the cloud server would notify the controller when to start sending data and when to stop. This means implementing an algorithm for detecting similar sensor data readings and notifying the microcontroller of needed operations. With similar readings, the microcontroller could be put to an idle state for limiting power consumption, which would prolong battery life.

The aim is to optimise the sensor reading process enough to prolong the Arduino microcontroller's battery life on a 9V battery.

Contents

List of Figures **v**

1 Introduction **1**

1.1 Introduction 1

1.1.1 Motivation 1

1.1.2 Contributions 1

1.1.3 Outline 2

2 State of the Art **3**

2.1 Arduino 3

2.1.1 Arduino Mega ADK 4

2.1.2 Wireless SD Shield 4

2.1.3 RN-XV WiFly Module 5

2.1.4 TinkerKit 5

2.2 Fuzzy Logic 6

2.2.1 Fuzzy Set 6

2.2.2 Fuzzy Control Systems 6

3 Problem Statement **7**

3.1 Current Solution 7

3.1.1 Arduino 7

3.1.2 XMPP Communication 8

3.1.3 Data Collection Server 9

3.2 Problems 10

3.2.1 Hardware 10

3.2.2 Software 11

3.2.3 Power Consumption 11

4 Problem Solution **12**

4.1 Power Consumption 12

4.1.1 Sleep 13

4.1.1.1 Watchdog Timer 13

4.1.1.2 Arduino Mega ADK 13

4.1.1.3 WiFly module 14

4.1.2 Communication 14

4.2 Server-side Client 15

4.2.1 Simple Linear Regression 17

4.2.2 Fuzzy Logic Engine 18

4.3 Results 22

5 Conclusions **23**

5.1 Conclusions 23

5.2 Summary of Contributions 23

6 Related Work **24**

7 Future Research Directions **25**

8 Sisukokkuvõte **26**

Bibliography **27**

List of Figures

3.1 XMPP Session Lifecycle 9
3.2 Server-side Data Model 10

4.1 Power Consumption at 30 Second Interval 16
4.2 Updated Server-side Data Model 17
4.3 Regression Confidence Fuzzy Sets 19
4.4 Sensor Value Fuzzy Sets 20
4.5 Idle Time Fuzzy Sets 20

1

Introduction

1.1 Introduction

Briefly summarize the question (you will be stating the question in detail later), and perhaps give an overview of your main results. (it is not just a description of the contents of each section)

1.1.1 Motivation

The developed prototype (6) works well when the general idea and goal is considered. However, the lack of flexibility of data transmission intervals and high power needs constrain the implementation usages. In order to take full advantage of the prototype, power usage should be reduced and more flexibility added to the communication between the server-side client and sensor modules.

1.1.2 Contributions

A more flexible data collection solution is developed based on HTTP. The changes made enable more energy-efficient data collection by predicting sensor data when possible. A fuzzy control system was developed to decide if sensor data is predictable and for what time period. Moreover, the prototype's Arduino-based sensor module has been improved to support sleep mode while no sensor data is sent to the server-side client.

Power consumption before and after the changes was tested to indicate changes' effect on battery life. 1. mention peaktech power supply here?

1.1.3 Outline

Chapter 2: introduces the Arduino framework and modules used in the prototype. Finally, a description of fuzzy control systems and simple linear regression is given.

Chapter 3: describes the existing prototype and gives an overview of its implementation details. Later, an overview of identified areas of improvement is given.

Chapter 4: explains the improvements made to the prototype. Firstly, describes changes made to the Arduino sensor module. Secondly, changes in the communication between the sensor module and the server-side client are discussed. Lastly, an overview of the server-side fuzzy control system and data prediction techniques is given. The chapter is ended by a power consumption comparison between the initial and improved prototype.

2. add more chapters here when they're ready

2

State of the Art

The state of the art used in the thesis highlighted the advances in the cloud computing domain and the mobile domain...

Description of Jabber and XMPP protocol and its usage.

2.1 Arduino

Arduino (1) is an open-source electronics prototyping platform based on a simple microcontroller board and a development environment for writing software for the board. It is intended for anyone interested in creating interactive solutions. Arduino can take inputs from a variety of sensors and control various actuators and lights. The microcontroller is programmed using the Arduino programming language (based on Wiring) and the Arduino development environment (based on Processing). Arduino IDE enables to choose between different board models, microcontroller programmers and communication ports.

3. add an image of the IDE here... Programs (called sketches) are written and uploaded to the board using the Arduino IDE. Each sketch must have two functions- *setup*() and *loop*(). *setup*() is the first function called after Arduino is started or rebooted. It is called once and afterwards the function *loop*() is called consecutively until the board is stopped, restarted or crashes. When a crash occurs, the program is restarted, which means calling *setup*() again.

Since programs are written in C/C++, there are a lot of libraries available for use.

2.1.1 Arduino Mega ADK

Arduino Mega ADK (2) is one of the most capable boards available. The Arduino ADK is a microcontroller board based on the ATmega2560. Similar to the Mega 2560 and Uno, it features an ATmega8U2 programmed as a USB-to-serial converter. It has a USB host interface to connect with Android based phones, 54 digital input/output pins, 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, USB B, micro B connections and a 2.1mm center-positive power jack. The ADK is designed to be compatible with most shields designed for the Uno, Diecimila or Duemilanove

The ADK has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

The Arduino ADK can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter or battery. The board can operate on an external supply of 5.5 to 16 volts. The recommended range is 7 to 12 volts.

4. format the text a bit, remove describe what is EEPROM, add an image

2.1.2 Wireless SD Shield

The Wireless SD shield (3) allows an Arduino board to communicate using a wireless module. The module can communicate up to 100 feet indoors or up to 300 feet outdoors.

The shield has an on-board switch which allows to select between USB and Micro modes. In USB mode, the shield bypasses Arduino board's microcontroller and communicates directly to the USB-to-serial converter. In Micro mode, data sent from the microcontroller will be transmitted to the computer via USB as well as being sent wirelessly by the wireless module. The microcontroller will not be programmable via USB in Micro mode.

2.1.3 RN-XV WiFly Module

The RN-XV module (4) by Roving Networks is a certified Wi-Fi solution especially designed for customer who want to migrate their existing 802.15.4 architecture to a standard TCP/IP based platform without having to redesign their existing hardware. In other words, if your project is set up for XBee and you want to move it to a standard WiFi network, you can drop this in the same socket without any other new hardware.

The RN-XV module is based upon Roving Networks' robust RN-171 Wi-Fi module and incorporates 802.11 b/g radio, 32 bit processor, TCP/IP stack, real-time clock, crypto accelerator, power management unit and analog sensor interface.The module is pre-loaded with Roving firmware to simplify integration and minimize development time of your application. In the simplest configuration, the hardware only requires four connections (PWR, TX, RX and GND) to create a wireless data connection.

5. Add an image of the wireless shield and rn-xv wifi module

2.1.4 TinkerKit

TinkerKit (5) is a tool used to build interactive products using Arduino boards. It consists of modules (sensors, actuators) and a sensor shield. The tool greatly simplifies product assembly, because instead of building circuits out of low level components, all the modules can be attached to the TinkerKit sensor shield with a snapping cable.

Here is the mega sensor shield used in this project... **6. Add a figure of the mega sensor shield**

The modules are divided into sensors and actuators.

7. Thermistor Module, Light Dependent Resistor Module,Hall Sensor Module

2.2 Fuzzy Logic

2.2.1 Fuzzy Set

2.2.2 Fuzzy Control Systems

Description of fuzzy logic and its uses.

8. Add simple linear regression here, too?

3

Problem Statement

In this chapter, the prototype developed in Context Sensor Data on Demand for Mobile Users Supported by XMPP (6) is described. The prototype was successful, but had some areas of improvement. Secondly, an overview of these improvements is given. **9. Add a small introduction of who wrote the previous thesis etc**

3.1 Current Solution

The current solution (6) has three main components:

1. Arduino sensor module
2. OpenFire XMPP server
3. Data collection server in the cloud (referred to as the server from here on)

There were two separate configurations described in Context Sensor Data on Demand for Mobile Users Supported by XMPP - one using Wi-Fi and the other Ethernet for network communication. Only Wi-Fi configuration is considered in this thesis due to the fact that the availability of an Ethernet connection (a cable) usually means that there is a power outlet nearby.

3.1.1 Arduino

The first component is the Arduino sensor module. The hardware configuration is based on the Arduino Mega ADK board. Wireless Shield with RN-XV WiFly

module is mounted on top of the board. Wireless Shield and Arduino board communicate over UART (hardware serial). TinkerKit Mega Sensor Shield is mounted on top of the Wireless Shield with 7 modules attached to it: 4 LED indicator lights, Hall, thermistor and LDR sensors. The external power source used is a 9V battery.

On the software side, the implementation mainly relies on a XMPP library and WiFly module library called WiFlyHQ. The WiFlyHQ library is responsible for creating a TCP connection and communication over the network, the XMPP library handles all the XMPP implementation details.

The sketch itself is fairly straightforward - in *setup*() a wireless network is joined, a TCP connection established and lastly a XMPP session is initialized. In *loop*() all connections are checked and reestablished if needed. Then the last transmit time variable is compared to the current time and if the report step amount (currently 10 seconds) has passed, data is collected from the sensors, formatted to appropriate JSON format string and sent to the server-side client.

```
{"SensorData": {"location": 1, "data": [{"type": 1, value: 5.00},
{"type": 2, "value": 500.00}, {"type": 3, "value": 312.00}]}
```

10. check if it really is like this

3.1.2 XMPP Communication

Both the data collection server and Arduino module are XMPP clients. The XMPP OpenFire server runs in the cloud and provides XMPP communications to both clients. Clients connect to the same chat where the server listens for messages from the sensor module. When a message is received by the server, sensor data is parsed from it and saved in a database.

XMPP session lifecycle can be seen in Figure 3.1. With the current implementation, steps 1 - 5 are done once when establishing the connection or when the connection drops. Data transmission step is done every 10 seconds and the last 2 steps are done when the connection is closed.

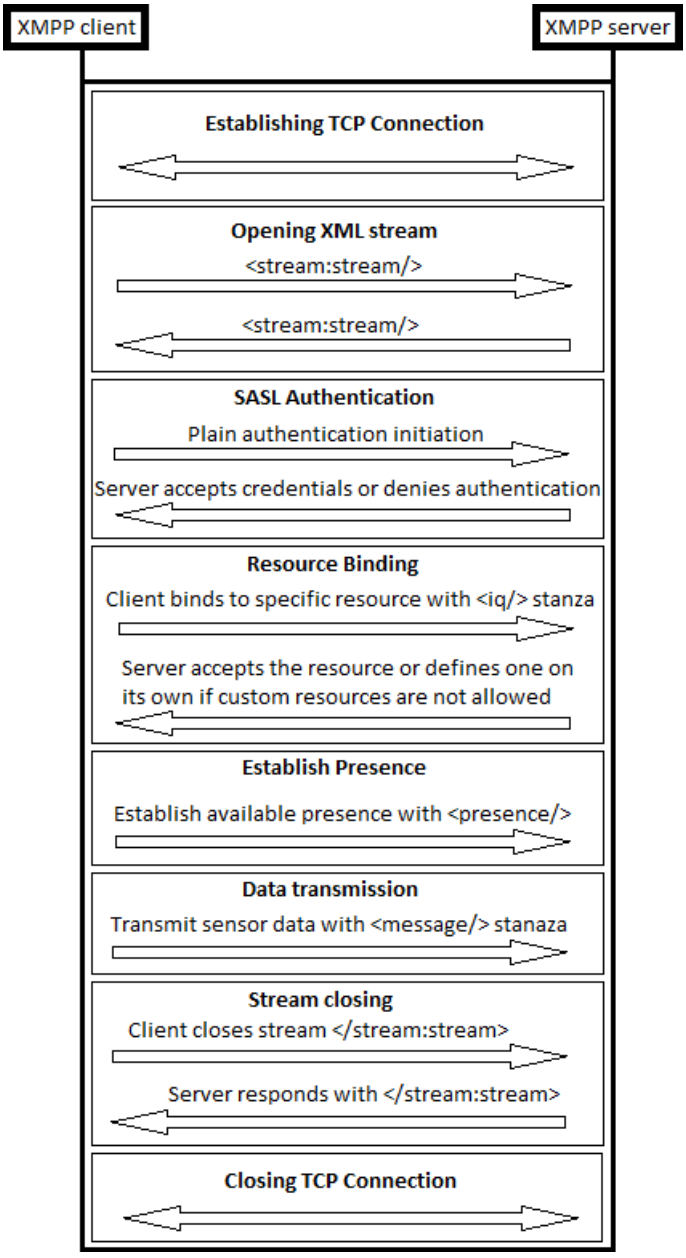

Figure 3.1: XMPP Session Lifecycle

3.1.3 Data Collection Server

The data collection server is responsible for gathering sensor data and saving it in a database. Its implementation is written in Java and uses Smack XMPP

API to communicate over XMPP. Data is stored in a H2 database, because of its simplicity and suitability for prototyping.

The database has three main tables - sensors, data and locations. Locations table has different sensor module locations, sensors table has different types of sensors and data has data gathered from different locations and sensors. The data model can be seen in Figure 3.2.

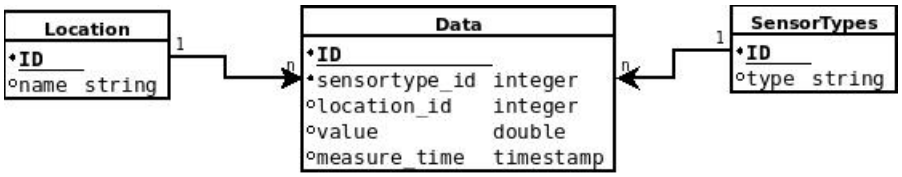

Figure 3.2: Server-side Data Model

3.2 Problems

Previously mentioned implementation has a few problems which will be discussed in this section. The two main points are power consumption and data collection flexibility.

According to the tests run in the previous thesis, the battery is able to power the module for 161,5 minutes using Wi-Fi. (6, p. 50-51) This, however, is not sufficient to enable actual data collection from a remote location. The problem can be addressed by using a larger battery, but the actual power consumption should be optimized, too. Additionally, the 10 second data transmission interval is hard coded into the Arduino sensor module, which does not provide enough flexibility. The following chapter described changes to the hardware, software and communication methods which improve on these error points.

3.2.1 Hardware

The hardware configuration has two main problems. Firstly, communication over the wireless module can be unstable at times, because parts of the messages might be missing or scrambled when read from the UART (6, p. 47). Since XMPP

session initialization is quite verbose, the possibility of receiving scrambled or incomplete messages effects the process.

Secondly, all parts of the hardware configuration are run in full power mode. As there is a 10 second gap between data transmission and sensor reads, there is a time period when most components are idling. This, in turn, means that some parts of hardware could use less power during these intervals and therefore reduce overall power consumption.

3.2.2 Software

With the existing implementation *loop*() is called consecutively and time differences are checked to determine when 10 seconds has passed using the internal *millis*() function, which gives the current Unix time. Since we have a 10 second interval, we do not need to waste cycles on time difference checks we now will fail for the next 10 seconds. A way to delay program execution could be used to stop the execution until we know the defined amount of time has passed.

In addition, as messages from the Wireless Shield might not be complete and XMPP session initialization is a verbose process, the current XMPP implementation hangs when scrambled messages are received during session start up. The library checks from complete XML tags, but when the attributes are incomplete, connection will not be successful and the session initialization code hangs.

3.2.3 Power Consumption

Power consumption is the main problem this thesis focuses on. As a 9V battery has an average capacity of 400 mAh at 100 mA current, the resource is quite limited. With the existing configuration, the average current drawn is 108 mA. Battery lifetime tests in the previous thesis showed that the module runs for 161,5 minutes on a 9V battery (6, p. 50). However, 161,5 minutes is not enough to gather contextual data for the proposed data collection system. The batteries need to be changed too often for it to be a viable solution. 11. how should i reference all the data collected?

4

Problem Solution

In this chapter improvements to the existing implementation are discussed. First, changes in the Arduino sensor module are described. Secondly, an overview of changes in the communication between the module and data collection server (the server from here on). Lastly, an overview of the fuzzy control and data prediction systems that were implemented to reduce communication overhead and power consumption.

The main goal is to reduce the average power consumption measured by the PeakTech power supply.

12. reference it somehow

13. introduction

4.1 Power Consumption

The first problem addressed is power consumption. There are two main ideas behind reducing it - put the sensor module to sleep mode when not transmitting data and reduce the need for data transmission. For this, the server-side client was improved to predict sensor data when possible and notify the sensor module of the next data transmission time. This enables the module to enter sleep mode for the given time period and thus reduce power consumption.

4.1.1 Sleep

Since the module consists of 3 components - Arduino Mega ADK, Wireless SD Shield with WiFly module and TinkerKit Sensor Shield. Both the Mega ADK and WiFly module support sleep modes.

4.1.1.1 Watchdog Timer

A watchdog timer (7) is an electronic timer used to recover from computer malfunctions. They are found in automated systems where human interference is not possible and therefore the system must be able to recover from malfunctions on its own. A watchdog timer essentially performs a timing function producing a delayed response to an input trigger. The most common implementation has a digital counter that counts from a specified value down to a terminal value. Usually the initial value is programmable. When the counter reaches the terminal value, the timer timeouts and triggers a timeout signal. Usually this means restarting the program from the start. A program can restart the watchdog timer at any time. The act of restarting is usually referred to as "kicking the dog". In this way, a program can be written witch never lets the counter reach the terminal value.

4.1.1.2 Arduino Mega ADK

In case of an Arduino board, when the watchdog timer timeouts, the sketch is restarted (new call to *setup*()). Furthermore, a watchdog timeout signal is sent and this signal can be captured by the sketch. In the Arduino sketch, this is implemented by the JeeLib library (8). JeeLib is a library written for experimenting with JeeLabs products, however, some parts of the library are written for Arduino boards and can be used with them. Specifically, the Ports class (9) is the one used in this implementation to put the Arduino Mega ADK into sleep mode. The sketch execution stops for the specified sleep time and afterwards execution continues from the call to sleep function.

14. Add a sample overview of the sketch here maybe?

4.1.1.3 WiFly module

The RN-XV WiFly module can be put to sleep in two ways - sleep timer or sleep command. With the sleep timer, the shield will enter sleep mode after a specified time period has passed since all TCP active connections have closed. With the sleep command the module will enter sleep mode immediately, unless an active TCP connection exists. (10).

This means that in order to put the WiFly module to sleep, all active connections must be closed. For the XMPP session, this means closing the active stream and the underlying TCP connection. Once this has been done, the module can successfully enter sleep mode.

The module can be waken up by either sending characters of the UART or by using the wake timer. In our implementation the activity on the UART wakes the module up when the sketch execution continues after the Mega ADK wakes up from sleep mode and establishes a new TCP connection in the start of *loop*() method call. This effectively means going through the first 5 stages of XMPP session lifecycle on every wake up. Since XMPP session initialization is quite verbose and the communication over Wi-Fi is unstable, the possibility of receiving scrambled or incomplete messages creates a problem.

4.1.2 Communication

Because all TCP connections and therefore the XMPP session have to be closed after every data transmit, XMPP session lifecycle steps 1 - 5 shown in Figure 3.1 are executed multiple times. When testing the XMPP implementation with Arduino Mega ADK and WiFly module sleep modes enabled, a troubling fact was discovered - the XMPP session negotiation fails at least once for every 30 minute test. The reason for these connection failures are scrambled authentication or stream opening stanzas.

From the tests, it could be seen that the average XMPP session start up time was 15 seconds. 15. show a graph here i guess? or whatever Data needs to be transmitted every 10 seconds, which means that the module can never be put to sleep as it will not be able to go through the sleep and wake up cycle during the available time period. Moreover, data transmission took on average 1 second,

meaning that 15 seconds spent on wake up would result in a second of actual work, which is not efficient.

During testing the sleep and wake up cycles with XMPP, it was found that XMPP session negotiation will hang approximately once in 15 minutes. As the maximum sleep time in our implementation is 65 seconds, this means that there are at least 13 separate session negotiations. Of course, this is the problem of the XMPP library in use and its lack of error handling. Problems with the library could be addressed with a better implementation, however, there was another factor discovered during the tests.

As a result, XMPP as a means of communication was not viable when trying to minimize power consumption. As an alternative, 16. look how to reference these web sockets, raw sockets and HTTP was considered. Web sockets were left out because opening a web socket connection is opened with a HTTP request, making using that one request to actually send the data more efficient. Therefore, HTTP was preferred to web sockets. Raw socket implementation in Arduino would add needless complexity to the sketch and was therefore not implemented.

The main advantages of HTTP are connection initialization speed and simplicity. The average time to wake up from sleep mode, send an HTTP request and receive response from the server, was measured to be around 5 seconds. In the previous scenario of 10 second transmission interval, it would mean 5 seconds could be spent in sleep mode, 5 seconds to wake up and transmit the data. This implementation would be more energy efficient, which can be seen in Figure 4.1. The data was gathered while running the prototype for 30 minutes with a data sending interval of 30 seconds. The sleep times were adjusted for both configurations based on their wake up times. As seen in the figure, the HTTP configuration consumes approximately 20% less power than the one using XMPP.

4.2 Server-side Client

With the move from XMPP to HTTP, the server-side client's implementation changed. Instead of using XMPP Smack library, a web server was needed. Jetty

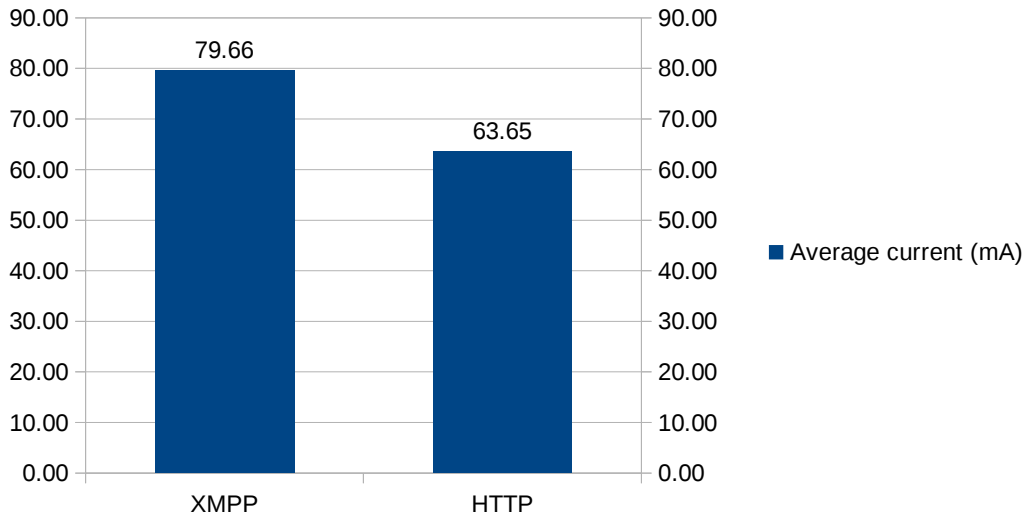

Figure 4.1: Power Consumption at 30 Second Interval

was selected because of its simplicity and possibility to embed it into the application. A web server embedded in an application is useful when prototyping, because it saves time on configuration and deployment.

Furthermore, to take full advantage of the newly developed sleep mode functionality, the client was further developed to predict sensor values for some time. Two modules were added the server for this - simple linear regression and fuzzy control engine. In addition, the data model was modified to suit the new modules. The new data model can be seen in Figure 4.2. *Data* table now has *measured* field, which indicates if the value has been measured or predicted. *SensorTypes* table has two new fields - *regression_error* and *measure_error*. *regression_error* is the acceptable regression model error for future predictions. *measure_error* indicates the acceptable variance for measured values.

17. Do i really need this here? The final web server consists of 4 modules:

1. Request handler
2. Linear regression model
3. Fuzzy control engine

4. Data storage

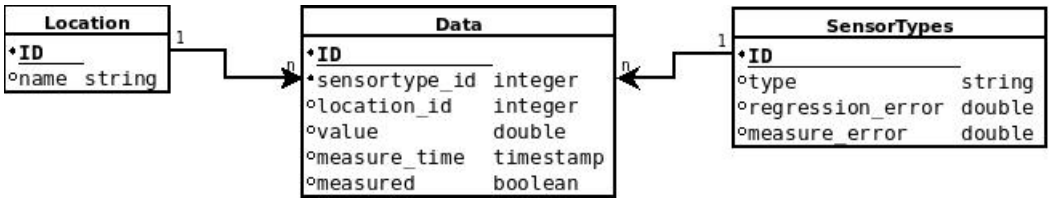

Figure 4.2: Updated Server-side Data Model

4.2.1 Simple Linear Regression

A simple linear regression model was selected to predict future data, which uses the least squares method to calculate the future values. 18. add a mention of the paper the idea was taken from? The model has a single explanatory variable - Unix time. This variable is used to predict the future values of sensor readings based on previous measurements.

The model sample is taken from previous measurements during the last 19. actual time 5 minutes. The 5 minute interval is selected to balance out errors caused by extreme values. This is necessary because a single value with big enough deviation can cause the regression model to become inaccurate.

To measure the accuracy of the model, an confidence level of 90% was introduced. If the sample for last 5 minutes provides an accurate enough regression model, then the predictions can be used. Otherwise, fresh data should be queried and added to the model until the error threshold is satisfied. The equation to calculate regression model's confidence level α:

$$\alpha = 2\Phi\left(\frac{\epsilon}{RMSE}\right) - 1.$$

Here Φ is the CDF (Cumulative Distribution Function) of residuals, ϵ is the allowed error (in the database model named as $regression\_error$ and $RMSE$ is the root mean square error the regression model. 20. maybe describe what rmse is?

Regression models are created separately for each sensor. Each model has a sample of previous measurements and the allowed error ϵ defined in the database. The confidence level α is calculated for each model and has to be over 90% which is the selected confidence threshold. If all regression models satisfy the confidence level, then the fuzzy control system is initialized with data from previous measurements and calculated confidence levels.

4.2.2 Fuzzy Logic Engine

The next step in predicting future sensor values is to calculate the time interval for the next measurement request. To calculate the time, a fuzzy control system was introduced to provide flexible decisions based on multiple input variables. The input variables are:

1. *temp*
2. *temp predictability*
3. *light*
4. *light predictability*
5. *hall*
6. *hall predictability*

Here each sensor has a pair of input variables - measured sensor value and sensor value predictability (regression model confidence level α). All these variables are mapped into fuzzy sets, a procedure call fuzzification. The predictability variables are mapped into sets which all have the same membership functions shown in figure Figure 4.3.

The fuzzy control system is initialized with a previous measurement for each sensor. This effectively sets the bounds of each sensor's fuzzy sets as seen in Figure 4.4. The *PREDICTABLE* fuzzy set's DOM (degree of membership) reaches maximum at a confidence level of 0.95 or 95% because for our prototype anything above that level is highly predictable.

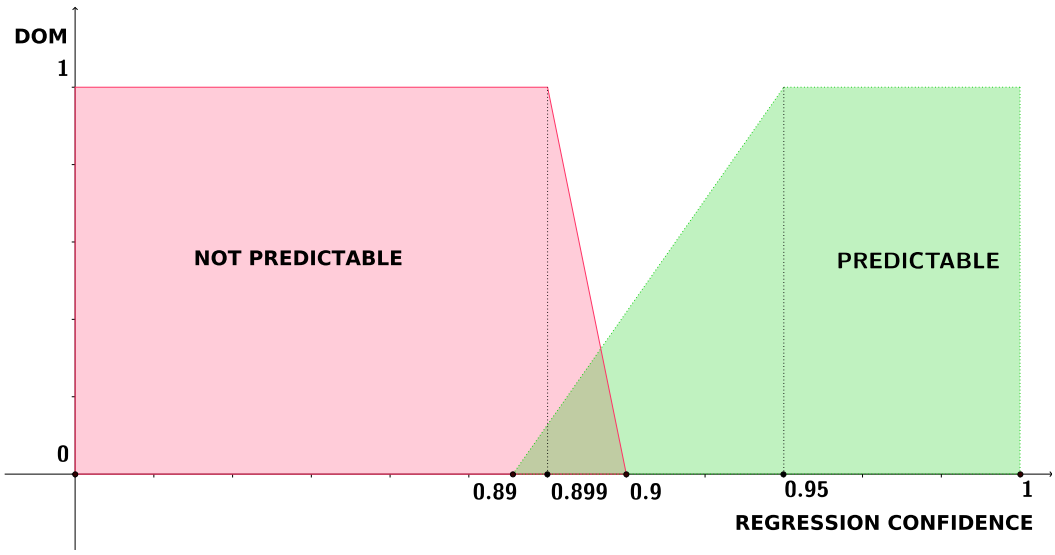

Figure 4.3: Regression Confidence Fuzzy Sets

The measured sensor values are fuzzified into sets, which each have different membership functions. The membership functions are calculated on the last measurement received as seen in Figure 4.4. Here X in set labels notes the sensor type currently used as the sets are the same relative to each sensor's previous measurement. The center point O is the previously measured value. $||X_2X_3||$ is the predefined sensor measurement error ($sensor\_error$ field in the data model). This means that if the new measurement is between the points X_2 and X_3, there is no change in the measurement for the fuzzy system. The lengths $||X_0X_2||$, $||X_1O||$, $||OX_4||$ and $||X_3X_5||$ are defined by the same variable $slope\_width$.

21. maybe calculate this automatically somehow, atm predefined variable used

The output variable is defuzzified from the idle time sets which have membership functions described in Figure 4.5. The sets *request* and *predict* are symmetrical to enable easy defuzzification. The maximum idle time is 65 seconds, which is the maximum length of one sleep cycle for the sensor module.

When all input variables have been fuzzified and degrees of membership calculated, then the next step is to fire all predefined rules. For each sensor, there is a collection of 3 rules, making a total of 9 rules (here $x \in \{hall, temp, light\}$):

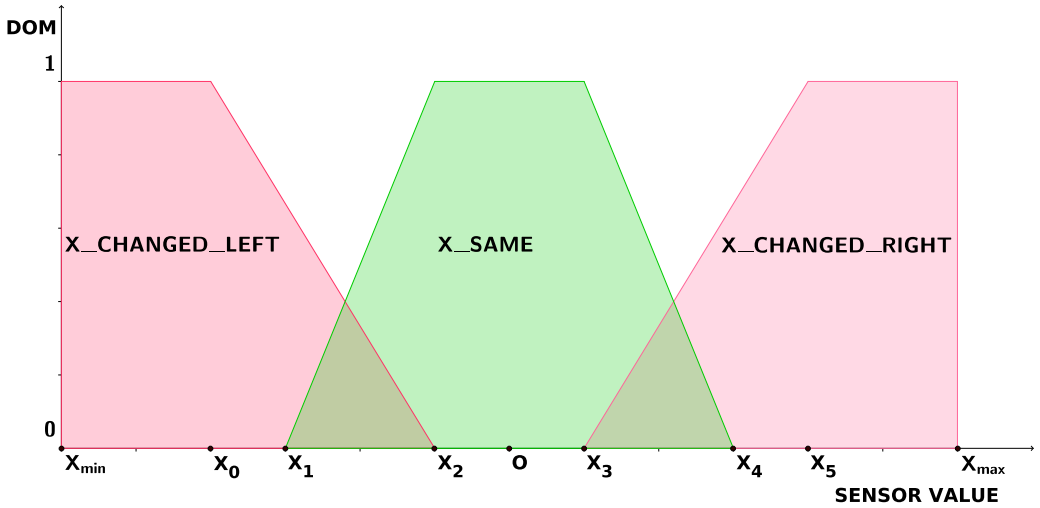

Figure 4.4: Sensor Value Fuzzy Sets

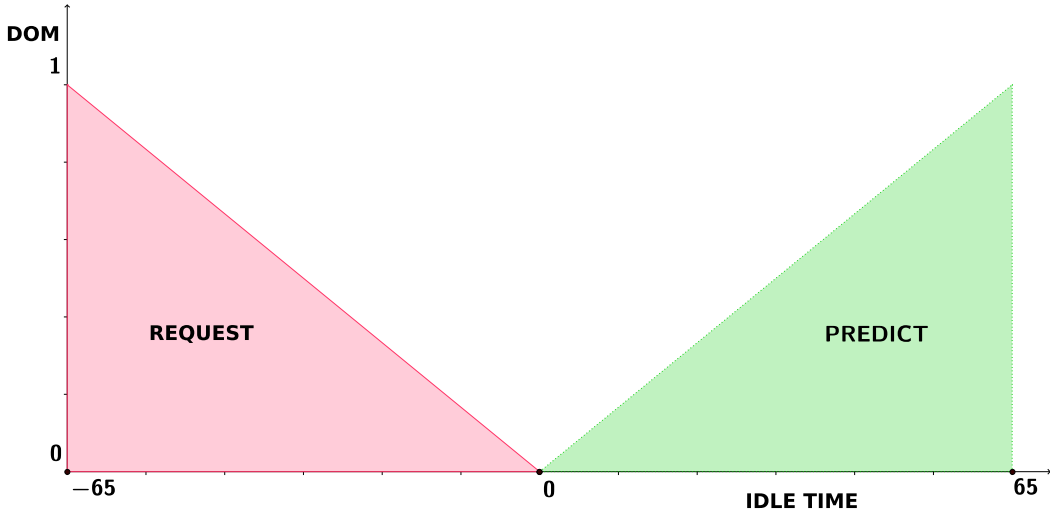

Figure 4.5: Idle Time Fuzzy Sets

1. IF x IS $x\_changed\_left$ OR $x\_changed\_right$ AND $x\_predictability$ IS $not\_predictable$ OR $predictable$ THEN $request$

2. IF x IS x_same AND $x\_predictability$ IS $not\_predictable$ THEN $request$

3. IF x IS x_same AND $x\_predictability$ IS *predictable* THEN *predict*

The previously calculated degrees of membership are fed into these rules. For AND relationship, the minimum value is selected, for OR relationship, the maximum value is selected. For each rule, the output variable received a DOM equal to the DOM of the premise. When all rules have been evaluated, the results are defuzzified into a single crisp value, which is the idle time decision. This process is called defuzzification.

During defuzzification the DOMs of result sets *request* and *predict* for each rule are combined into a single DOM for the specified set. For this, *request* output has an OR relationship (maximum value) and the *predict* set has an AND relationship (minimum value). These relationships mean that the most inaccurate sensors would be represented in the final result. For example if *hall* has a DOM of 1.0 to *predict*, but *temp* has a DOM of 0.5 to *predict*, then the outcome would be a DOM of 0.5, because we need the output for least accurate sensors. For *request* this logic is reversed as the maximum DOM to *request* represents the output for the least accurate sensor.

The next step is to get the crisp result value. For this process, the idle time values for *predict* and *request* set membership functions at the specified DOM are calculated. x is found the equation $f(x) = DOM$, where x is the idle time and f is the membership function. *request* and *predict* membership functions are symmetric with the axis being in the point $x = 0$ as seen in Figure 4.5. The final defuzzified idle time is equal to $max\{x_{predict} - x_{request}, 10\}$. Meaning that if the DOM to *request* was higher that to predict, then the default idle time of 10 seconds is returned. Otherwise the value at the center point of the two results is returned.

When the crisp result is received from the fuzzy control system and idle time is longer than 10 seconds, future data is predicted for the idle period. This means that using the regression models, future values with 10 second intervals are inserted into the database for each sensor.

Finally, the idle time with a minimum value of 10 seconds is written as a JSON format string to the response body of the HTTP request. A sample response

would look like:

22. add a sample response here

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: 12
Connection: close
Server: Jetty(7.2.0.v20101020)

{"idle": 34}
```

4.3 Results

23. Describe tests carried out

24. Make a test with a 9V battery

25. Graphs with improved power consumption figures on it, description of what changed and how/why

5

Conclusions

5.1 Conclusions

5.2 Summary of Contributions

6

Related Work

Compare your solution with existing projects. How your solution is better than the others?, why to use your solution?, etc.

7

Future Research Directions

Briefly indicate how your current research can be extended, some improvements, etc.

8

Sisukokkuvõte

Eesti abstract...

Bibliography

[1] Arduino - HomePage, http://www.arduino.cc/.
URL http://www.arduino.cc/ 3

[2] Arduino - ArduinoBoardADK, http://arduino.cc/en/Main/ArduinoBoardADK.
URL http://arduino.cc/en/Main/ArduinoBoardADK 4

[3] Arduino - ArduinoWirelessShield, http://arduino.cc/en/Main/ArduinoWirelessShield.
URL http://arduino.cc/en/Main/ArduinoWirelessShield 4

[4] RN-XV WiFly module - wire antenna - SparkFun electronics, https://www.sparkfun.com/products/10822.
URL https://www.sparkfun.com/products/10822 5

[5] 5

[6] Kaarel Hanson, Context sensor data on demand for mobile users supported by xmpp, Tech. rep., University of Tartu, Faculty of Mathematics and Computer Science, Institute of Computer Science, Tartu, Estonia. 1, 7, 10, 11

[7] Jim Lamberson, Single and multistage watchdog timers (2012). 13

[8] JeeLib: introduction, http://jeelabs.net/pub/docs/jeelib/.
URL http://jeelabs.net/pub/docs/jeelib/ 13

[9] JeeLib: port class reference, http://jeelabs.net/pub/docs/jeelib/classPort.html.
URL http://jeelabs.net/pub/docs/jeelib/classPort.html 13

[10] WiFly user guide - WiFly-RN-UM.pdf, http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-UM.pdf.
URL http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Wireless/WiFi/WiFly-RN-UM.pdf 14

ToDo

P.

1. mention peaktech power supply here? . . . 2
2. add more chapters here when they're ready . . . 2
3. add an image of the IDE here... . . . 3
4. format the text a bit, remove describe what is EEPROM, add an image 4
5. Add an image of the wireless shield and rn-xv wifi module . . . 5
6. Add a figure of the mega sensor shield . . . 5
7. Thermistor Module, Light Dependent Resistor Module,Hall Sensor Module 5
8. Add simple linear regression here, too? . . . 6
9. Add a small introduction of who wrote the previous thesis etc . . . 7
10. check if it really is like this . . . 8
11. how should i reference all the data collected? . . . 11
12. reference it somehow . . . 12
13. introduction . . . 12
14. Add a sample overview of the sketch here maybe? . . . 13
15. show a graph here i guess? or whatever . . . 14
16. look how to reference these . . . 15
17. Do i really need this here? . . . 16
18. add a mention of the paper the idea was taken from? . . . 17
19. actual time . . . 17
20. maybe describe what rmse is? . . . 17
21. maybe calculate this automatically somehow, atm predefined variable used . . . 19
22. add a sample response here . . . 22
23. Describe tests carried out . . . 22

24. Make a test with a 9V battery 22
25. Graphs with improved power consumption figures on it, description of what changed and how/why 22