Software Project Plan

Pond Environmental Measurement
SSNS - Smart Sensor Network Systems
Frankfurt University of Applied Sciences

Alexander K.,
Sabrina B.,
Rozana A.,
Alexander V.D.

July 5, 2013

Contents

1 Introduction

The Idea of this Project is to build a System, based on a Wireless Smart Sensor Network, to measure a ponds water Temperature and the weather conditions that influence the ponds temperature. The Data from the measurements shall be stored in some thing like a database and be visualizable on a PC. The goal is the visualization of a reliable 24/7 data aquisition.

For the breeding and keeping of fish in a pond it is often relevant to know the ponds temperature and the range in which the temperature stays over a longer period of time. We also expect some interesting results from a system like that.

2 Project Specification

2.1 Project Description

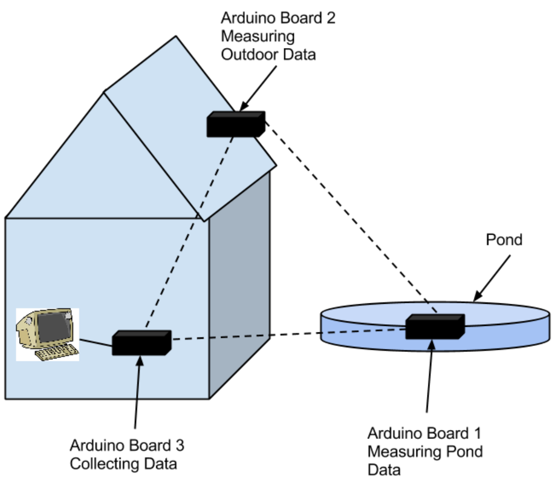
The System consists of the following Components:

- one underwater Temperature measuring Sensor Node
- one over water weather measuring Node (Temperature, Light)
- one indoor Node collecting and storing the measured Data, connected to ethernet
- A PC running a visualizing Application

The sensors measure the weather conditions and pond temperature. The values get forwarded to an indoor Data Collecting Node with ethernet access. There the Data is going to be logged. Using a Computer a history of the measurements (seen as tables and graphs) can be viewed via a special application.

The Three Nodes will communicate via the ZigBee Protocol, using XBee Modules.

Figure 1: Pond Envrionmental Measurement



The outdoor sensor nodes are going to operate independant of a power source meaning that these Nodes will be equipped with Solar panels and batteries. They should be able to run 24 hours every day just by the power provided by the solar panel and the batteries (that get charged by the solar panel).

In this project we want to concentrate on the Software part of the System. To build the Outdoor Nodes in a Weather Proof way and designing them independant of a Power Supply by using Solar is a complex task. Finishing the Hardware is an optional aspect of our project. We will design the necessary circuits for attaching the sensors to the arduino Nodes and use the Nodes to test our Software on. The task of building weatherproof, solar powered Nodes based on the sensor circuit design is optional or can be done in a future project.

2.2 Process Model

As Process Model we chose the Rational Unified Process (RUP). RUP has the advantage of being iterative and incremental while also being relatively lightweighted. All Project activities can be used any time when needed, this gives us flexibility.

The Rational Unified Process is an Interactive software development process framework. It was created by Rational Software Corporation, a division of IBM since 2003. It enhances team productivity and creates and maintains models. It is also a guide to effectively use the Unified Modeling Language. Its goal is to deliver a high quality product that the customer actually wants.

The RUP model is mainly used in Telecommunications, Transportation, aerospace, defense, manufacturing ,financial services and in system integrators.

Well-defined and well-documented software development processes are key to the success of software projects. CMM(Capability Maturity Model) by the software engineering Institute (SEI) has become a beacon. Theoretical know-how fails to materialize in practice. Sometimes there is no process know-how at all. Resulting in chaos, failure and Loss. In this cases RUP can help. It is a mature, rigorous and flexible software engineering process.

We choose RUP for our project because of its luxary features, usage of Best Practices and also for its development cycle process. The mensioned terms are explaing as follows:

Features of RUP :

- Iterative Development
- Requirements Management
- Visual Modeling of Systems
- Quality Management
- Change Control Management

Compare to the traditional waterfall process, the iterative process has the following advantages:

- Risks mitigated ealier
- Change is more manageable
- Higher level of reuse

- The project team can learn along the way
- Better overall quality

The 6 Best Practices are as follows :

1. Develop software iteratively
2. Manage requirements
3. Use Component-based architechtures
4. Visually model software
5. verify software quality
6. Control changes to software

In every Iteration, all of these tasks can be performed with a certain weight. In the first iterations a lot of weight will be on the Requirements and less on the Deployment. In later iterations Deployment is going to gain weight while the Requirements are almost done. So it is possible to react to change, and to correct errors, with the beginnig of every iteration and it will be accomplished at the end of the iteration. It is also intended that the software gains functionality with each iteration (incremental development).

All iterations of a project are grouped into 4 phases.

1. Inception
2. Elaboration phase
3. Construction phase
4. Transition phase

- Each phase is concluded with a well-defined milestone
- Each phase has a specific purpose

Inception phase:

- A vision document
- An intial use-case model
- An Initial project glossary
- An initial business case
- A project plan, showing phases and iterations .
- A business model

Project milestone : The Lifecycle Objectives Milestone

Construction Phase:

The construction Phase is concerned with moving from the executable architechtures created in the Elaboration phase to an operational system

- the focus here is to develop the application to the point where it is ready for deployment
- Focus is on implementation of the design

Project Milestone : Initial operational Capability

Elaboration Phase:

- A use-case model (at least 80% complete) supplementary requirement (non functional requirement)
- A software architechtural description
- An Executable architechtural prototype
- A revised risk list and a revised business case
- A development plan for the overall project
- An updated development case specifying the process to be used
- A preliminary user manual (optional)
- Domain model

Project milestone : the Lifecycle architechture Milestone

Transition Phase:

- "Beta testing" to validate the new system against user expectations
- Parallel operation with a legacy system that it is replacing
- Conversion of operational database
- Training of users and maintainers
- roll-out the product to the marketing , distribution , and sales team

Project milestone : The product Release Milestone

Pros of RUP:

- Regular feedback from and stakeholders
- efficient use of resources
- we can deliver exactly what the customer wants
- Issues are discovered early in your project
- Supports iterative development
- Improved risk management

Cons Of RUP:

- The process may be too complex to implement .
- Development can get out of control
- It is a heavyweight process
- You need an expert to fully adopt this process

2.3 Requirements

2.3.1 General Requirements

There are Three (Software) components to be developed:

- Software for the two measuring Nodes (only different in respect to the sensor configuration)
- Software for the collecting Node, with connection to Ethernet
- PC Application for Data visualization

Ofcourse there are also Hardware components that have to be developed:

- Weatherproof, Solar powered water temperature measuring Node
- Weatherproof, Solar powered weather measuring Node
- Data colloecting Node

When discussing the Requirements of the system it is necessary to distingush the components and to distingush the requirements for the Software from the requirements to the hardware.

2.3.2 Overall System Requirements

- reliable 24/7 data Aquisition
- Data being storaged on the collecting Node (Some sort of Database)
- Graphical visualization on the PC, reading Data from the collecting Node over Ethernet

2.3.3 Data Measuring Nodes Requirements

- Taking measures as the mean of multiple measurements over a short time period
- Measures get taken after a demand by the collecting Node
- Measures get stored localy until the collector acknowledges the receipt of the measure
- Based on Arduino UNO and Wireless Protoshield with an XBee Module

Optional:

- Weather proof housing
- independant from Power Supply by using Batteries and Solar

2.3.4 Collecting Node Requirements

- Coordinator Role in the ZigBee Network
- knows Date and Time, set via NTP
- Demands Measures from the measuring Nodes every half an hour
- Measuring Nodes have 30 sec. time to answer before their measures get stored as unknown.
- stores measures on SD-Card as CSV-File

- Sends CSV-File over Ethernet on demand
- Acts as TCP Server (On static IP Adress and fixed Port)
- Based on a Arduino Ethernet with Wireless Protoshield and XBee Module

2.3.5 PC Application Requirements

- Developed in Java, GUI using Swing
- Connects to the Collecting Node via IP/Ethernet
- Connects as TCP-Client to collecting Node on its static IP-Adress and fixed Port
- Data Visualization as Tables and Graphs
- Selectable Timeframe for shown Data
- Selectable Data to be shown in the same Graph (Pond Temperature, Air Temperature, Light Level)
- Data Export as CSV File (Coma-Seperated-Values)

3 System Architecture

3.1 Architecture

The Hardware Architecture describes the circuits for using the Sensors and the configuration of the three Network Nodes.

3.1.1 The Sensors

We are using two kinds of Sensors. The kind of Temperature Sensor and one kind of LDR Sensor.

- Temperature Sensor TMP36
- LDR GL5528

We chose these Sensors because of their easy sourcing and their wide usage. They are also relativel easy to use.

The Temperature Sensor is a solid State component which doesnt need calibration. It Outputs a voltage depending on the temperature. The relation between the temperature and the voltage is linear, so its very easy to calculate the temperature based on analog readings of the voltage.

Some relevant facts taken from the Datasheet:

- specified operating Temperature Range: -40°C to 125°C
- scalefactor of 10mV/°C
- offset of 0.5V
- Accuracy of ±1°C at 25°and ±2% in the range of -40°C to 125°C

This gives us the Formular: Temp [°C] = $\frac{Voltage[mv]-500}{10}$

The LDR (Light Dependant Resistor) is for Measuring the light Level. LDR's are usually not precise enough to really measure the light level, but are rather used to distinguish darknes from lightness. So we wont get very precise readings with it, but iwe will try to get the best out of it.

Some relevant specifications:

- not precise enough to measure the light level in Lux
- only measure darkness from light
- Reliable performance
- linear relation

To get the light value in Lux you have to execute following:

- Get voltage of Resistor
- Get Resistor Value with formula: $\frac{5.0-LightVoltage}{LightVoltage} * 1000$

- Get Luxy with formula:

$$10 * {\frac{14000}{LightResistor}}^{\frac{1}{0.7}}$$

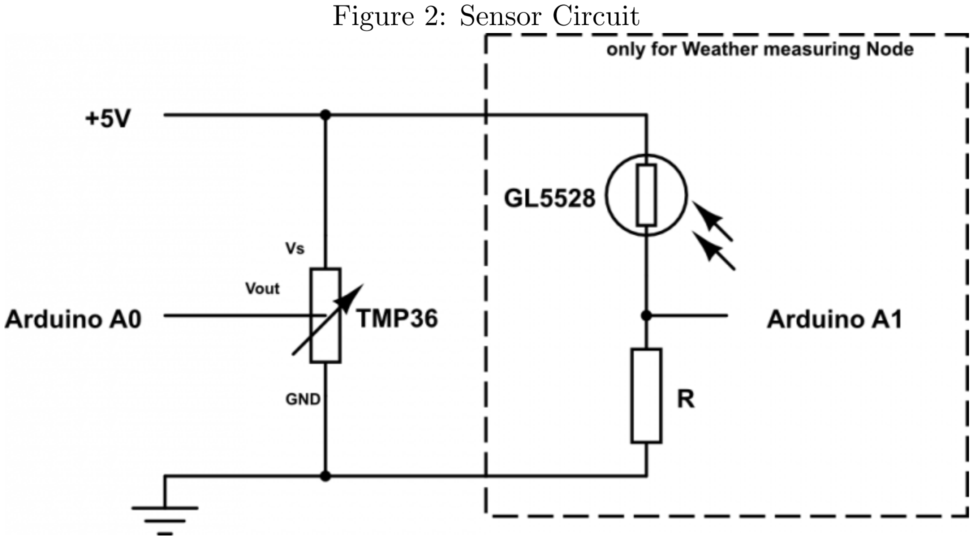
If you calculate the Error of Light and Temperature you will see, that Temperature has a quantisation error of $\pm 1°C$ and the light has a very big quantisation error. Ths let us see, that you cannot measure really exact.

3.1.2 Measuring Nodes

The Measuring Nodes consist of Arduino Uno's (or similar boards like Leonardo) which the Software is running on ant the Sensors are connected to. The Arduino Board is expanded using a Wireless Proto Shield with an XBee Module to provide ZigBee Network conectivity.

The Sensors are being used according to the following Circuit Diagram. The **TMP36** is directly connected to the Arduino's 5V Power source and GND, the output voltage of the **TMP36** is read on the A0 Analog Input of the Arduino.

Figure 2: Sensor Circuit



On the weather measuring Node the circuit is enlarged by the **GL5528 LDR**. The LDR is used in a voltage divider circuit. The Voltage of 5 Volt is being divided by the propordion of the resistances of the LDR and the fixed Resistor. This "proportional divided" voltage is supplied to the A1 Analog Input of the Arduino.

3.1.3 Collecting Node

The Collecting Node consists of an Arduino Ethernet Board (A simple Arduino with Ethernet Shield would also be possible). The Arduino Ethernet board is also extended with a Wireless Proto Shield and XBee Module to provide ZigBee connectivity. The Arduino Ethernet features a built in micro SD Card slot.

3.2 Software Architecture

3.2.1 Messaging

ZigBee Network Setup

The ZigBee Network consists of three Nodes

- Pond measuring Node
- Weather measuring Node
- Collecting Node

The Collecting Node will act as the ZigBee Coordinator, so its XBee Module will have to be loaded with the coordinator Firmware. the two measuring Nodes will be either End-Nodes (which means low power consumption), or routers (which gives them the ability to extend the Networks Radius by routing). Since both measuring Nodes only communicate with the collecting Node (very simple star-network-configuration) no additional routing Nodes are necessary. Also making the measuring Nodes act as routers will increase the power consumption of the Nodes.

So as long as both measuring Nodes are in reach of the collecting Node, it is best to stick to the end-device firmware for the measuring Nodes XBee modules, otherwise the routing firmware might enhance the networks reach.

In our Project we are going to use the ZigBee Network standard. This means that we won't concern ourself with the lower layers of the protocol stack. Because we only work on the upper Layers (Application Layer), the choice of firmware for the measuring Nodes is not relevant for our Software. ZigBee will take care that our packages get to their destination, either directly or via hopping the other measuring Node.

In order to work with the XBee Modules we will use the xbee-arduino Library:
http://code.google.com/p/xbee-arduino/

The XBee Modules are connected to the Arduino over a Serial Connection. The protocol to control the XBee Modules over the Serial connection is pretty easy, however the Library makes it even easier.

Every XBee Module comes with a 64 Bit long Serial number that is also used as its own ZigBee Network Adress. The coordinator must know the adresses of the two measuring Nodes. It is possible for the Collector to discover the other Nodes using broadcasts, however we are going to hardcode the Meaurement Nodes adresses into the collectors Sourcecode. The Measuring Nodes are going to adress the collector either by taking its adress from the collectors packets, or by using the reserved Adress 0x0000000000000000 (coordinators 64 Bit adress).

There is also a 16 Bit PAN (Personal Area Network) Adress that has to be set up on all Nodes. The exact adress is irellevant (as long as its 16 Bit long and the same on every Node). The Adress we are using will be dec 56154 = 0xDB5A. By the way, this adress was chosen by multiplying 42 by 1337.

ZigBee Network Messages

Between the Measuring Nodes and the Collector there are two Messages that have to be defined. The first there is the Measurements Request that the Collector sends to the Measuring Nodes. The second Message is the answer from the Measuring Nodes to the collector.

Packages in the ZigBee Network transfer a payload of a certain number of bytes. In the Arduino language the payload would be an array of the type uint8_ t.

The following Messages with their specified structure are going to be transfered:

Measurement Request

Byte(s)	content	meaning
1	'R'=0x52	identifier for Measurement Request

1 Byte consisting of the uppercase letter 'R', indicating that this is a measurment request from the collector.

Weather Nodes Measurement Response

Byte(s)	content	meaning
1	'W'=0x57	identifier for Measurement response
2-5	float	float for temperature measurement in Celsius
6-9	float	float for light intensity in Lux

The first Byte indicates that this message is a measurement response from the Weather Measuring Node. The next 4 Bytes contain a float value of the measured temperature. The last 4 Bytes are are a float value for the measures light intensity.

Pond Nodes Measurement Response

Byte(s)	content	meaning
1	"P'= 0x50	identifier for Pond measuremnt response
2-5	float	float for temperature measurement in Celsius

The first Byte indicates that this is a measurement response from the Pond Measuring Node. The next four bytes contain a float value for the measured temperature in Celsius.

Communication between the Collector and the Application

The Collector is hosting a TCP Server (static ip Adress and portnumber, hardcoded into sourcecode). The Application is going to connect to the collector as a client. After the TCP 3-Way handshake id over the collector is immediately going to transfer its CSV-File over the TCP conection to the Application. After the complete File is transfered the connection is being closed.

3.2.2 Collecting Node

CSV File Format:

The Data Collecting Node will store a CSV-File (Coma Seperated Values) on a SD Card. Every half an hour the collecting Node will require measurements from the measuring Nodes. The Measuring Nodes then adds a Line to the CSV-File. Also the PC Application will require the collecting Node to send the CSV-File over Ethernet. Each Line represents a complete measurement of the whole System.

The Lines will have the following form:

hh,mm,ss,dd,mm,yyyy,ppppp,aaaa,llll

- **hh, mm, ss, dd, mm, yyyy** represent the point in time when the measurements were taken.
- **ppppp** is the temperature of the pond in °Celsius in the form of 12.5 meaning +12.5°C
- **aaaaa** is the air temperature in °Celsius and is coded in the same way as the Pond's temperature
- **llll** is the light level in Lux

In principle all values can be signed floating point numbers of arbitrary length, however only certain subsets of the possible values make sense. hh will be integer and of the interval [0 - 23]. mm and ss will be integer and of the interval [0-59]. For the temperatures float values from -30 to +45°C are realistic. Negative values get marked by a -. Unknown values might be left out, but the necessary comas will stay.

Example Line:

00,30,15,13,05,2013,11.5,9.7,

At 0:30h and 15 seconds on the 13th of May of 2013, the pond temperature was 11.5°C, the air temperature was 9.7°C, and the lightlevel is unknown. The Lines will be seperated by a Newline '\n'.

In this Format, for every complete measurement, a line size of approximately 34 Bytes will be written to the File. Based on this Line Size the following estimations on Filesize can be made:

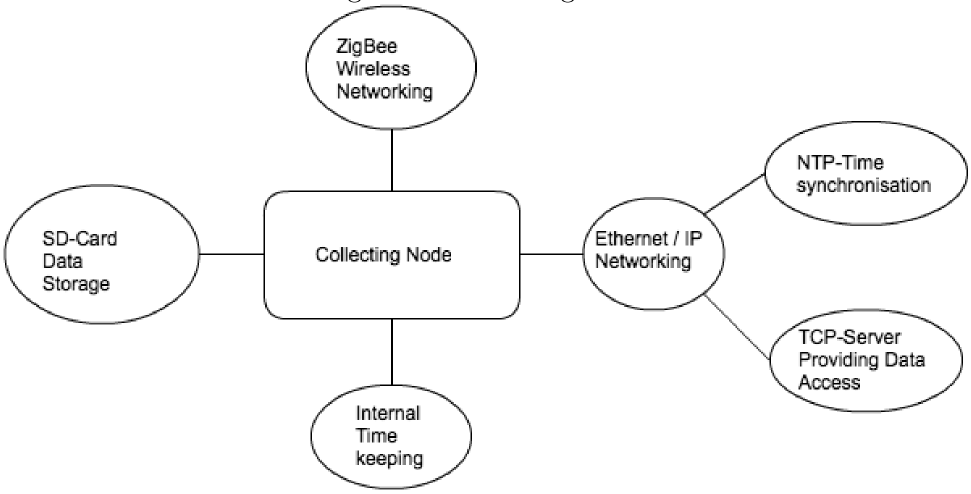
- 1 Measurement = 34 Bytes
- 1 Day of measurements = 48 * 34 Bytes = 1632 Bytes
- 1 Month of measurements = 31 * 1632 Bytes = 50.592 Bytes
- 1 Year of measurements = 12 * 50.592 Bytes = 607.104 Bytes ≈ 593 kByte

With just a 32MB sized SD Card this System could collect Data for about 55 years. Since we will hardly be able to find an SD-Card of a size lower than 1GB it is obvious that Filesize is not going to be a problem.

Software Components:

Since the Collecting Node has several responsibilities in respect to functionality, it's Software consists of several Components: Those components get realised in the Software by writing so called

Figure 3: Collecting Node



Arduino Libraries. The Arduino Plattform provides several Libraries for different Services, but it also allows users to write their own Libraries. User defined Libraries get implemented by defining C++ classes that get instanciated in the Arduino sketch. This means that Arduino Libraries are just a way of using object oriented software development to modularize Arduino Sketches.

The ZigBee Wireless Networking component takes care of everything the Collector has to do on the ZigBee Network, meaning to Send Measurement Requests and to receive the Measurement Responses from the Measuring Nodes. The Ethernet Networking component provides everything that has to be done using the Ethernet/IP Network connection. There are two functionalities provided by this component. First is the possibility to get the current Daytime and Date from a NTP Server. The second functionality is the hosting of a TCP Server, waiting for incomming connections. As a connection gets established the component sends the Data, which is stored using the SD-Data storage component, to the connected TCP-client (Our WSN-Accessing Application).

The Component for intrnaly keeping the current Time and Date is not coded by us. It is the openly available Time Library by Michael Margolis. It uses the Arduino's internal function millis(), so no additional Hardware is needed to keep the time. The Time can be set by retrieving the UNIX standard Time (seconds since the 1st of January, 1970 00:00:00) and providing it to the Time library. It is then possible to retrieve the current Time in Hours, minutes and Seconds, as well as the calender date.

The last component is the SD-Data storage component. It provides access to the SD-Card that can be inserted into a socket on the Arduino Ethernet board. On the SD-Card, there is one File created called "data.csv". The SD-component adds CSV-Lines to the File, and can retrieve the File byte-wise.

Application Flow

Arduino Sketches are always based on two core functions, setup() and loop(). Setup() contains all code that should run when the Arduino Board powers on, and loop() contains all the code that will run repeatedly until the Board gets powered off.

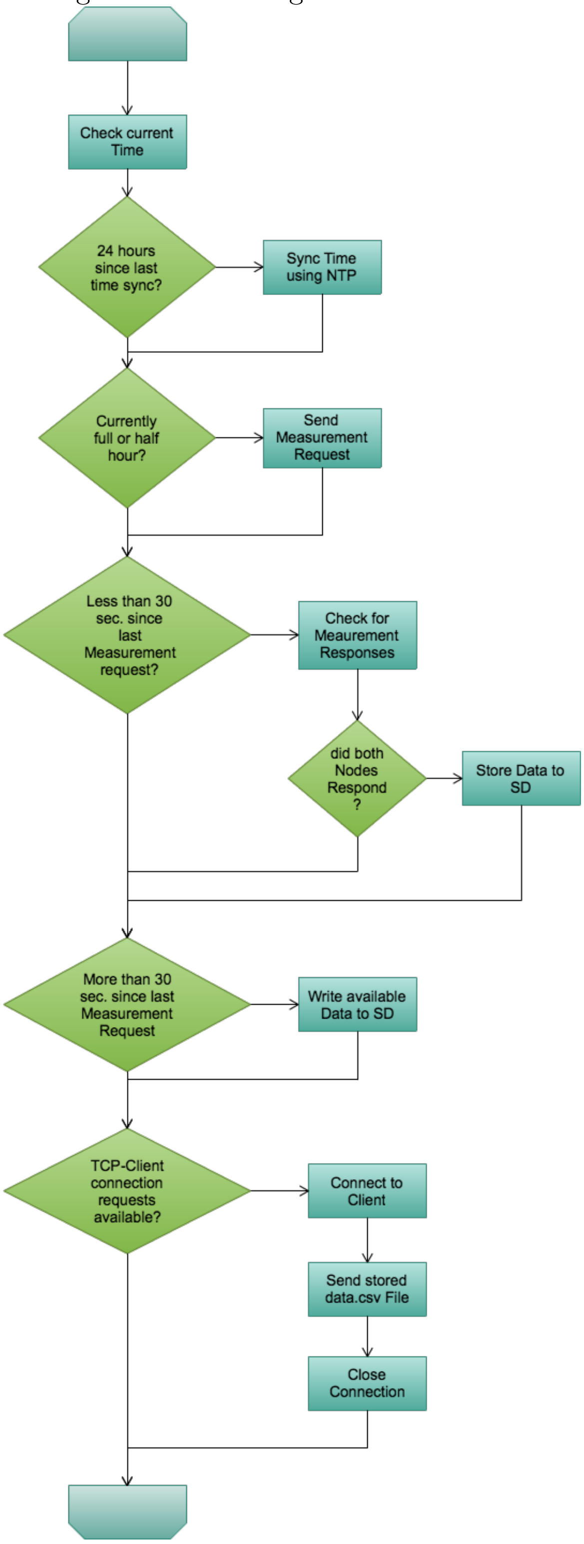
Besides a lot of general initilization, inside setup() the collector does two things:

- Synchronize the Time using NTP
- Create data.csv on the SD-Card if not present

For the operation of the Collector it is absolutely necessary to know the current Time and Date, thats why the collector first synchronizes using NTP. Then the data storage File gets created. If the file is already present, it will just be opened. The File does never get deleted by the collector!

In the loop() function the collector works according to the following Algorithm:

Figure 4: Collecting Node Flow Chart

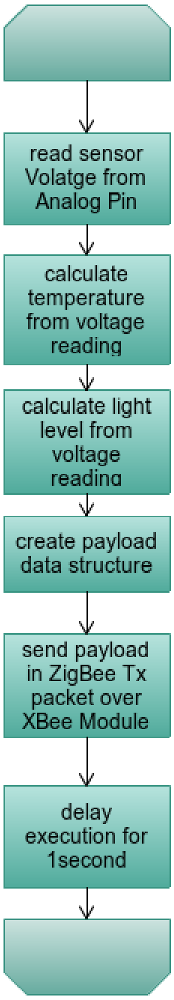


3.2.3 Measuring Nodes

The Measuring Nodes are seperated into two Parts. One Measuring Node measures Light and Temperature as an Weather Station outside. The other Measuring Node collect the temperature of a pond. The measured data is send via a tx package in the ZigBee Network to the Collector Node. Inside the Measuring Nodes there are executing following Functions:

- read Voltage from Analog Pin
- Calculate Temperature
- Calculate Light
- Send Temperature and Light Values via payload in a TX packet to Collector

Figure 5: Measuring Nodes: Flow Chart

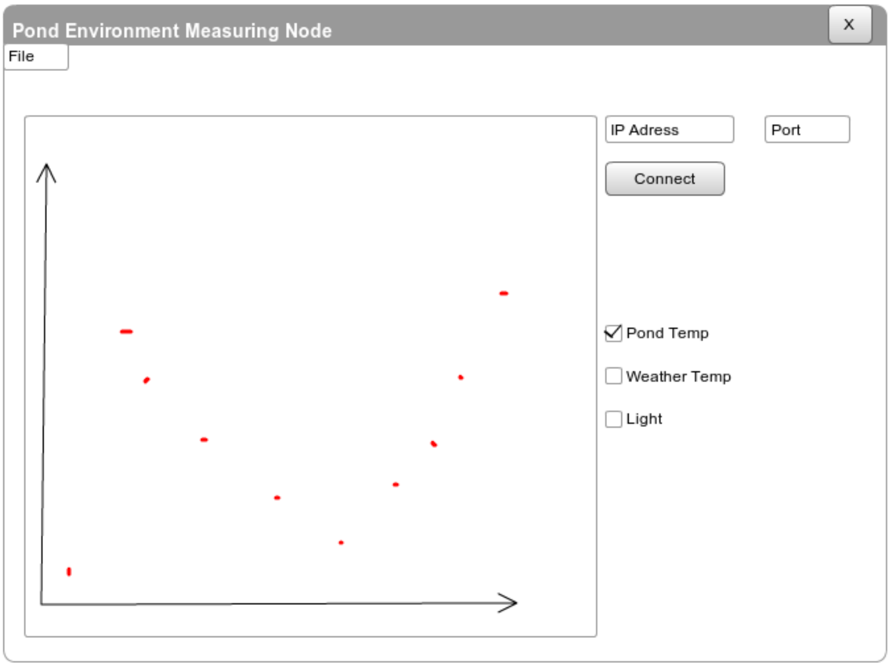


The Flow Chart shows, that the Measuring Nodes has a linear sequence of functions. These functions were repeated again and send temperature data and light data via a tx packet to the Collector Node.

3.2.4 Application

The features of the Application are described in the list of requirements. To imagine how the application behaves and how to modularize it we begin by creating a mock-up of the GUI: This is the

Figure 6: Mock-Up: Application



graphical user Interface. On the left side is a **plotting area** where the history of measured Data is shown as a plot. On the right side are some necessary controlls.

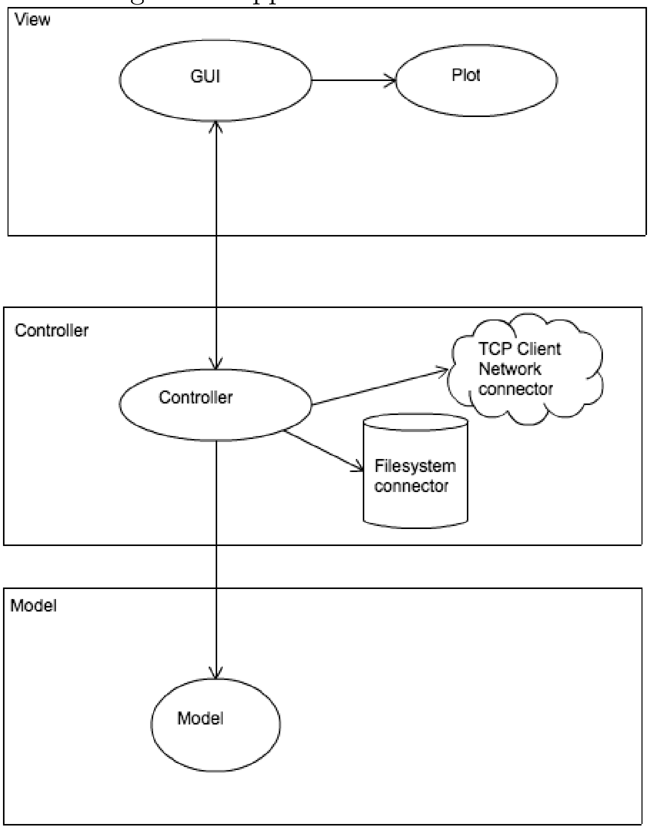
First there are two **input boxes** where the IP Adress and Port number, to reach the Collecting Node's TCP Server, are specified. After this input is made the connection can be established by clicking the **connect button**. If the connection fails an error dialog box will pop up, if it works the application will receive the CSV File's contents as byte stream (maybe show a progress bar dialog) and store it in local memory. After that the Data is being processed and plottet in the **plotting area**.

On the right side are also three **check boxes** to select the values to be plottet. So by selecting the checks, curves are being drawn or removed from the **plotting area**. The curves are drawn in different colors so that they are distinguishable.

On top of the GUI is a **menue bar** with a menue **"file"**. This menue only contains the entry **"Export as CSV"**. On selection of this entry a "save file" Dialog will pop up which allows to save the CSV files contents in a CSV-File on the PC's hard drive (like a copy of the CSV File on the Collecting Node).

The Software Architecture follows the MVC Pattern and has the components shown in the following Graphic:

Figure 7: Application-Architecture



4 Project Estimates

4.1 Estimation technique Function Point

Function Point Estimation was developed by A.J. Albrecht of t he IBM Corporation in the early 1980s. The idea behind Function Points is to identify and quantify functionality required for a project. This procedure is decomposed into three parts:

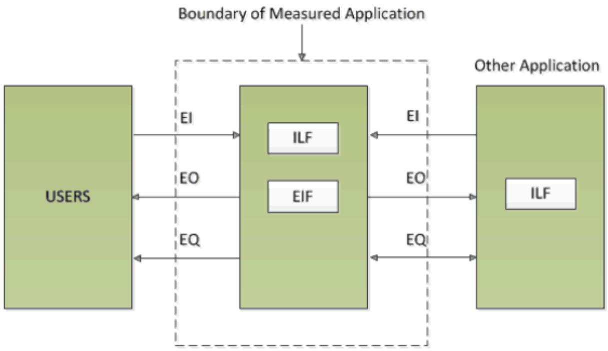
1. Defining the Unadjusted Function Point Count
2. Determining the Value Adjustment Factor
3. Determining Function Points

Defining the Unadjusted Function Point Count

The application boundary defines what is external to the application. The number of Unadjusted Function Points is derived from the number and complexity of transactions functionalities like:

- **External Inputs (EI)** which are elementary processes (an elementary process is the smallest unit of activity that is meaningful to the user) in which data crosses the boundary from outside of the application. They come from input screens or other applications.
- **External Outputs (EO)** which are elementary processes in which derived data passes across from inside to outside. These screens, reports, graphs, messages are created from one or more logical files and external interface files.

Figure 8: Function Point Model



- **External Queries (EQ)** which are processes with both input and output components that result in data retrieval from one or more internal logical files and external files.

and data functionalities like:

- **Internal Logical Files (ILF)** which are user groups of logically related data that reside within the application boundary. They are maintained through external inputs.
- **External Interface Files (EIF)** which are user groups of logically related data that are use for reference only.

Using above data we can calculate the Unadjusted Function Points. After having all basic data and transactional functionalities of the system we can use the following table below to calculate.

Figure 9: Unadjusted Function Point Count and Multipliers (Degree of Complexity)

	Measurement Parameter	Count		Weighting Factor Low	Average	High		Total
1.	External Inputs		x	3	4	6	=	
2.	External Outputs		x	4	5	7	=	
3.	External Inquiries		x	3	4	6	=	
4.	Internal Logical Files		x	7	10	15	=	
5.	External Interface Files		x	5	7	10	=	
	Unadjusted Function Point Total							

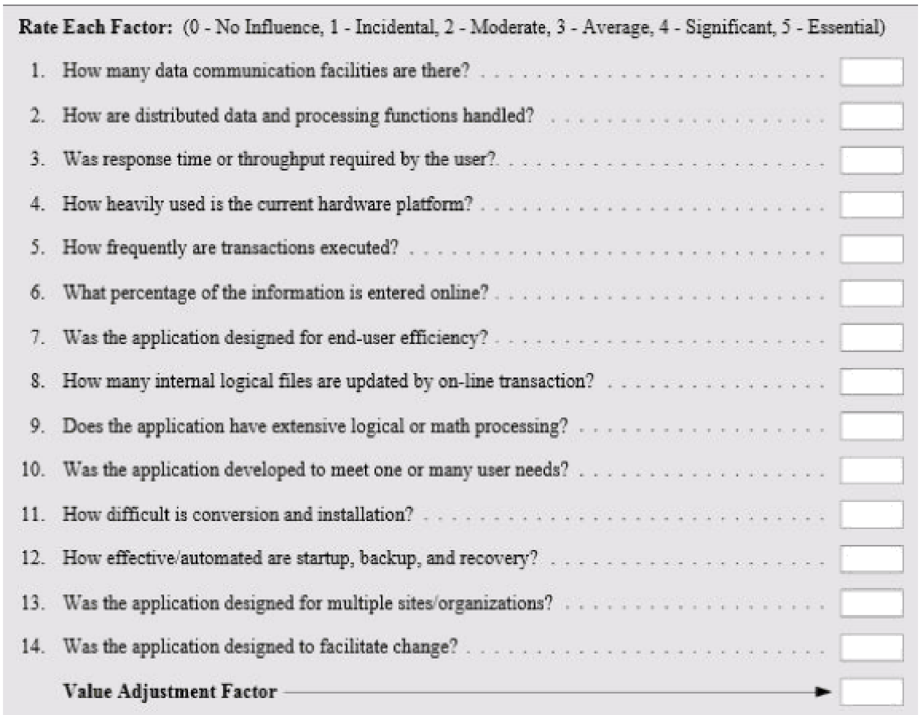
Determining the Value Adjustment Factor

Further we can calculate the Value Adjustment Factor based on 14 general system characteristics:

1. Data Communication Complexity
2. Distributed Data Processing Complexity
3. Performance Complexity
4. Heavily Used Configuration Complexity
5. Transaction Rate Complexity
6. On-line Data Entry Complexity
7. End-User efficiency Complexity

1. On-Line Update Complexity
2. Complex processing Complexity
3. Reusability Complexity
4. Installation Ease Complexity
5. Operational Ease Complexity
6. Multiple Sites Complexity
7. Facilitate Change Complexity

Figure 10: Total Degree of Influence)



Rate Each Factor: (0 - No Influence, 1 - Incidental, 2 - Moderate, 3 - Average, 4 - Significant, 5 - Essential)

1.	How many data communication facilities are there?	
2.	How are distributed data and processing functions handled?	
3.	Was response time or throughput required by the user?	
4.	How heavily used is the current hardware platform?	
5.	How frequently are transactions executed?	
6.	What percentage of the information is entered online?	
7.	Was the application designed for end-user efficiency?	
8.	How many internal logical files are updated by on-line transaction?	
9.	Does the application have extensive logical or math processing?	
10.	Was the application developed to meet one or many user needs?	
11.	How difficult is conversion and installation?	
12.	How effective/automated are startup, backup, and recovery?	
13.	Was the application designed for multiple sites/organizations?	
14.	Was the application designed to facilitate change?	
	Value Adjustment Factor	

The general system characteristics are estimated on a scale of 0 to 5 (Degree of Influence)
Total Degree of Influence = Sum of Degree of Influence

Determining Function Points

Determining Function Points consists of factoring Unadjusted Function Points and Value Adjustment Factor together.
Value Adjustment Factor = 0.65 + (0.01 x Total Degree of Influence)
Function Points = Unadjusted Function Points x Value Adjustment Factor

Example 1: How long will a project take?
Let us assume we have:

- Unadjusted Function Points = 22
- Value Adjustment Factor = 0,84
- Function Points: 22 x 0,84 = 18,48

Here is a table, which shows a general Function Point estimation:

Project	Function Points	Man-Months
ASD	11	1
KWO	24	2
RMD	53	5
WBO	72	6

Having a look at an organization project benchmark we estimate a result of 1.5 Man-Months. At the end of our project we do have in effect an actual amount of 1.2 Man-Months which we then update in our organization estimation chart.

Project	Function Points	Man-Months
ASD	11	1
Arduino	22	1.2
KWO	24	2
RMD	53	5
WBO	72	6

Effort in Person Month = FP divided by number of FP per month (Using an organization or industry benchmark)
The Programmers in a company average 18 FP per month.
197 FP / 18 FP = 11 Man- Months

5 Project Schedule

Look at ../Gantt/ProjectPlan.pdf

6 Staff Organization

6.1 Team structure

The team is organized without any hierarchically behaviors, all team members are equal. The members are:

- Alexander K. - knoetig@stud.fh-frankfurt.de
- Sabrina B. - sabrina.bajorat@gmail.com
- Rozana A. - rozanamail1@yahoo.com
- Alexander V.D. - vallejodirektor@googlemail.com

The Team Members had following Tasks:

- Alexander K. - Documentation, Wiki Administrator, Implementation of Collecting Node and Application (GUI, TestServer)

- Sabrina B. - Documentation, Project Plan, Gantt Chart, Implementation of Measuring Nodes and Application (GUI, TestServer)
- Rozana A. - Domain Expert (Arduino, Xbee)
- Alexander V.D. - Estimation of Project

6.2 Management reporting and communication

There are every week status reports of done and open tasks for the current and next week. The main organization and communctions follows over the Wiki (http://vs1164102.vserver.de/dokuwiki/doku.php?id=ssns:ssns-project).

7 Project Results

According to our Architecture Designs we started to implement our assigned software products. Since the technology of Arduino and XBee was new to us, implementation followed the principles of trial and error, with a lot of experimental coding. It was difficult for us to estimate the complexity of the task to implement the software according to the planned architecture. At some points we had to do some cutbacks to the requirements to finish the product in time. Unfortunatelly we were unable to get the Wireless Sensor Network to work. The main reason for that is that we were unable to get the collecting Nodes Software to work.
In the following sections we describe what we have archieved in respect to the software components.

7.1 Software for the Collecting Node

The Collecting Node was the most complex Software component, and unfortunatelly it was not finished in time. Alexander Knötig was the developer responsible for that component.

As planned in the Architecture design, the collecting Nodes Software consists of several components implementing subtasks of the collecting Node. Only the components for handling the SD-Card access and the handling of Time and Date have been working as supposed. The components responsible for ZigBee Network connectivity and Ethernet connectivity could not be made working. The problem with implementing those components (and all the other components as well) is related to the Arduino Development Environment.

It is made to be simple and easy to learn for programming beginners. But those simplifications also enforce limitations on the programmer. For example, there is no real control over which object files get linked together in order to resolve dependencies. This is especially problematic when writing libraries (like the components of the collecting Node) that use other libraries. The header files of used Libraries always get included, so the necessary files can get compiled, but when trying to link them together, the necessary object files from the libryries don't get linked. There are workarounds to this problem, but they tend to make the code hard to work with.

Another limitation of the Arduino Development environment is the missing possibility to debug the code. Professional Development Environments for microcontrollers (Like Atmels ATmega) provide possibilities to debug the software while it is running on the microcontroller. This is offcourse more complex that debugging Software for usual PC Operating Systems, but it is a possibility to understand what is going wrong. When using Arduino, it is not possible to see what is running wrong. You can upload your programm to the board and then observe the results, either the board does what it is supposed to be doing, or it doesn't. There are no Exception Messages (like in Java programms), no hex Dumps, and not even a sign if the microcontroller has shut down because of a severe error. If the Arduino Board encounters an error while executing the programm code it just stops the programms execution. There is no possibility to see where it crashed and why.

When trying to implement the Collecting Node we often encountert the sutiation that the software just did not work, but we were unable to find out why. Thats why we were unable to finish it in time.

7.2 Software for the Measurement Nodes

The Softwar for the Measurement Nodes was relatively simple, relative to the collecting nodes Software. But since the collecting Node was not working, it was also not possible to operate the measurement Nodes with it together. The Mesuring Nodes are supposed to be waiting for a Measurement Request Message from the Collecting Node, when receiving this request, the Measuring Nodes send an Response Message.

Because of the missing Collecting Node, we were unable to implement the concept of waiting for an request and answering with an response. Instead, we implemented the measuring Nodes to periodically send a response Message, without listening for Requests.

7.3 WSN Accessing Application

The WSN Accessing Application was supposed to feature a Table and a Graph to visualize the Data that was going to be read in from the Collecting Node. But over the course of the project we changed the applications features a little bit. But the core function of reading the csv file from the Collecting Node and saving it to the computers hard drive is left unchainged. The implemented Application features a textarea displaying the loaded csv file as it is, and also two graphs, one for plotting the curve of the lightlevel and one for plotting two curves for the temperature measurements.

8 Protocol

Week 16

- Team Formation
- Resarch of Hardware (Arduino, XBee)

Week 17

- Research of Hardware
- Collecting Project Ideas
- Experiments with Arduino

Week 18

- Decision on Project Goal
- Project Specification
- Requirement Analysis
- Experiments with Arduino

Week 19

- Requirement Analysis
- Project Plan
- Create Circuit-Design

Week 20

- Project Plan Version 1
- Create Circuits

Week 21

- Create System Architecture: Hardware Architecture
- Continue with Project Plan

Week 22

- Create System Architecture: Software Architecture
- Implementation of Collecting and Measuring Nodes
- Continue with Project Plan

Week 23-25

- Implementation of ZigBee Network
- Implementation of Application Test-Server + Graphical View of measured data
- Continue with Project Plan

Week 26-27

- Test Software
- Finish Project Plan
- Prepare Presentation

References

[1] Arduino Library http://arduino.cc/en/Reference/Libraries

[2] Arduino XBee Library https://code.google.com/p/xbee-arduino/

[3] XBee http://www.digi.com/xbee/

[4] XBee http://xbee.wikispaces.com/

[5] TMP36 http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf

[6] GL5528 https://www.sparkfun.com/datasheets/Sensors/Imaging/SEN-09088-datasheet.pdf

[7] http://www.pjrc.com/teensy/td_libs_Time.html

[8] Building Wireless Sensor Networks: With ZigBee, XBee, Arduino, and Processing", Robert Faludi, O'Reilly Media; Auflage: 1 (31. Januar 2011)

[9] Mc Connell, Rapid Development, Microsoft Press, 1996

[10] C. Jones, Applied Software Measurement, 1991, McGraw Hill

[11] D. Garmus and D. Herron, Measuring the Software Process, Prentice Hall, 1996

[12] Helmut Balzert, Lehrbuch der Software-Technik, Spektrum Akademischer Verlag 2001