GeoLog

Data Logger for Geological Research

Páll Helgason, pallsh12@ru.is

Sindri Ólafsson, sindrio12@ru.is

Sveinn Elmar Magnússon, sveinnm12@ru.is

Þór Tómasarson, thortom12@ru.is

T-411-MECH Mechatronics 1

Instructor: Dr. Joseph Timothy Foley

November 10, 2014

SVN Revision 48

https://projects.cs.ru.is/projects/t-411-mech-2014/repository/changes/group/DataCollector/FinalReport/GeoLog.tex

Contents

Introduction

This open source project is a development of wireless data collecting equipment called the GeoLog. The project was cooperated with the geologist Dr. Andrew D. Wickert [1], seen in figure 5, who needed an affordable and light-weight field instrumentation for his research. Dr. Wickert had developed an Arduino-based[2] low-power field environmental data logger as show in figure 1. He needed a communication module to fit his data logger and sought assistance from Reykjavík University with the development.
The goal of the project is to design Arduino based communication module which sends data through GSM network to HTTP server. Later on the project might be integrated with Dr. Wickert low power data logger. With that accomplished the project adds an affordable light-weight equipment that seems to be unavailable on the commercial market. The project's solution is open source see `https://github.com/GeoLogTeam` and is therefore modifiable for anyone who chooses to use the GeoLog.
This report is written in a technical manner and is intended for those with understanding of the matter.

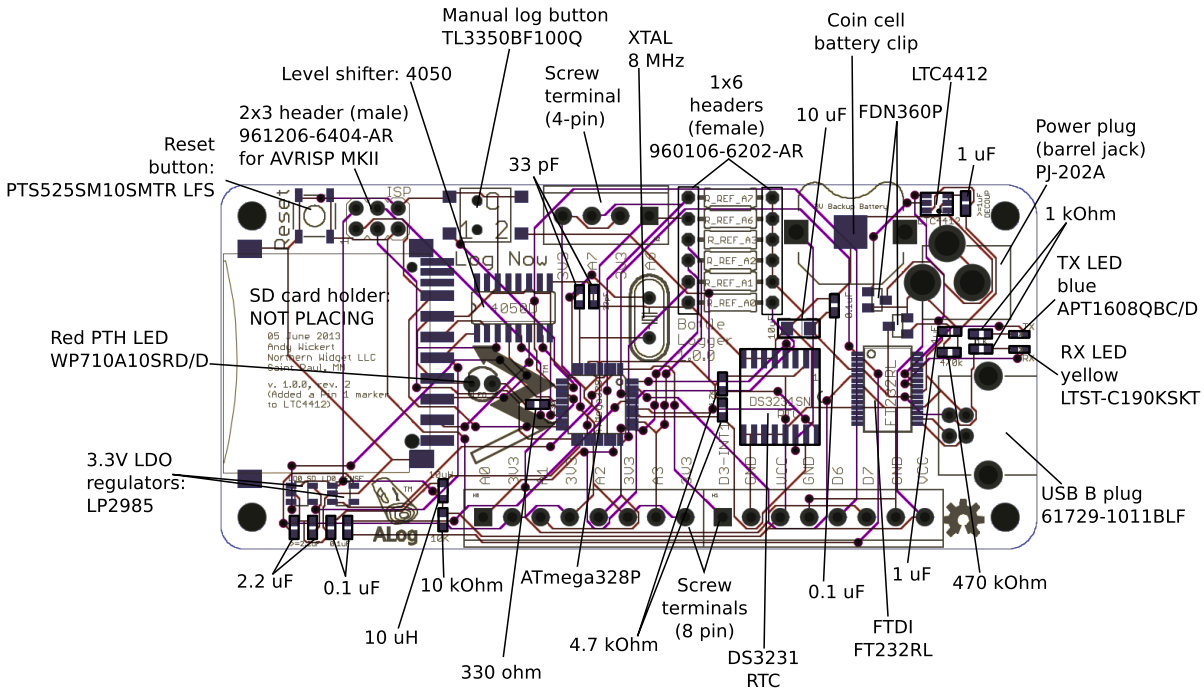

Figure 1: Arduino-based low-power field environmental data logging platform, circuit board layouts[3]

1 Background

Data sampling is a well know method to help scientists understand nature's behavior. Back in the days, scientists sampled data in real time in the field by writing measurements down to log book. With time automatic measurement equipment was developed which made it possible for scientists to leave the field and come back later to collect the data in order to process. Nowadays, with wireless communication technology the automatic measurement equipments allows scientist to access data in real time without going to the field.

Similar equipment that fulfills the requirements of being light-weight and affordable with long-range transmit capabilities is hard to find. The available equipment which is capable of long range communications through GSM network can be seen in figure 2a and 2b costs around $3,000. More affordable equipment as shown in figure 2c and 2d costs around $500 has the disadvantage of limited wireless range only up to 300 meters.

(a) $3500 [4] (b) $2800 [5] (c) $658 [6] (d) $595 [7]

Figure 2: Various price of portable weather stations

The goal of this project is to make an affordable, long-range communication data logger which results in more dense coverage in data logging and to assist the scientific community. By offering an low cost solution that is modifiable for different needs.

2 Requirements

Project requirement is listed in table 1 along with design parameters. It is essential to make the product as modular as possible with well planned software architecture. There fore FR8 is then most challenging.
Additionally three wireless sensors modules are to be added to expand the coverage of the mother hub. The sensors should use Pololu Wixel to communicate to the mother hub that samples data from them and sends through GSM network to a HTTP server.

Table 1: FR-DP

Functional requirements (FR):	**Design parameter (DP):**
1) Collect measurable data	Get temperature with TMP36[8] sensor
2) Store data local	Write temperature values to Arduino MEGA EEPROM
3) Keeps data for amount of time	Arduino MEGA [2] has 4kB of EEPROM
4) Runs on own power source	Have battery pack and minimize power consumption
5) IP67 proof	Fit all hardware into IP67 fiber box
6) Sent data wireless through GSM network	Send Json formated data trough GSM module
7) Store data on server	Set up HTTP API server
8) Modular Design	Well planed software design architecture

3 Design

The concept for the GeoLog originally came from Dr. Andrew D. Wickert, who came to Reykjavík University's Mechatronics class for help developing his design of a low cost and low power datalogger[3]. Dr. Wickert had two problems needed to be solved. Firstly to minimize the power consumption of the device he had already developed and secondly to have means of getting the data back without having to travel through difficult terrain. Decision was made to help Dr. Wickert improve the latter with emphasis on making the system as modular as possible see figure 3.
This design assumes there is GSM/GPRS coverage in the area the system is deployed. It is also assumed that HTTP server has been set up for receiving data from GeoLog. The reason GSM was selected as a communication method was because there is very good GSM coverage in Iceland, see figure 4 The GeoLog datalogger was designed in nine phases. Phases seven to eight are iterated until the product is ready for deployment.

- Phase 1: Initial brainstorming and high level design phase.
- Phase 2: Hardware selection phase, where hardware is selected and ordered.
- Phase 3: Software design phase, where the classes and interfaces is designed.
- Phase 4: Hardware hacking phase, trial and error in software writing for the Wixels[10] and the GSM/GPRS module[11].
- Phase 5: Building phase, where hardware is assembled.

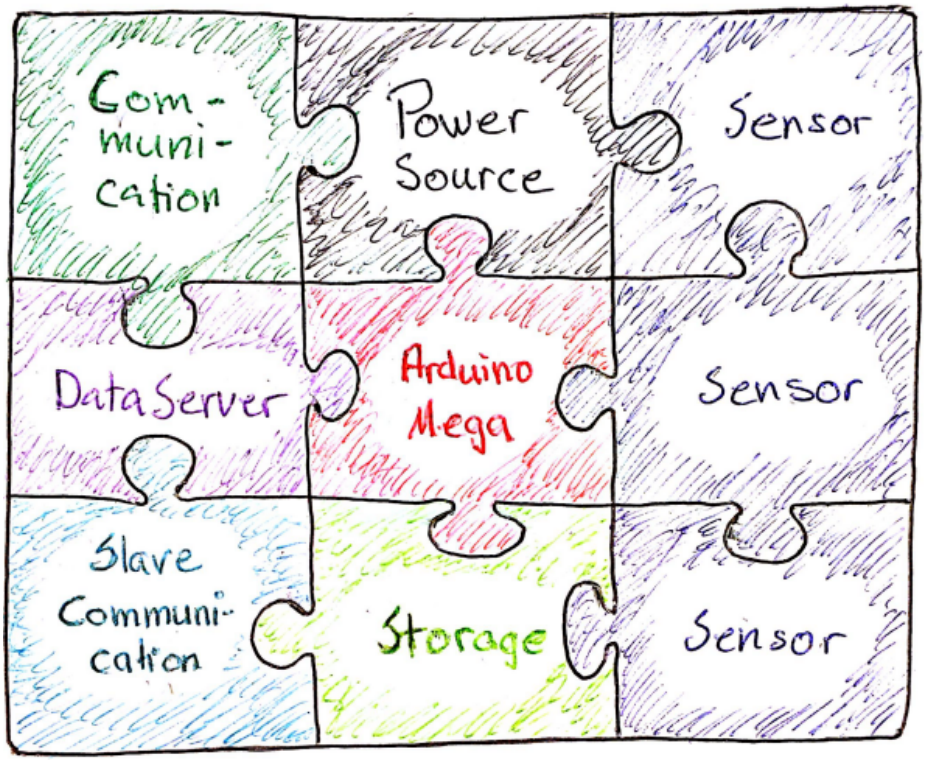

Figure 3: Modular design explained

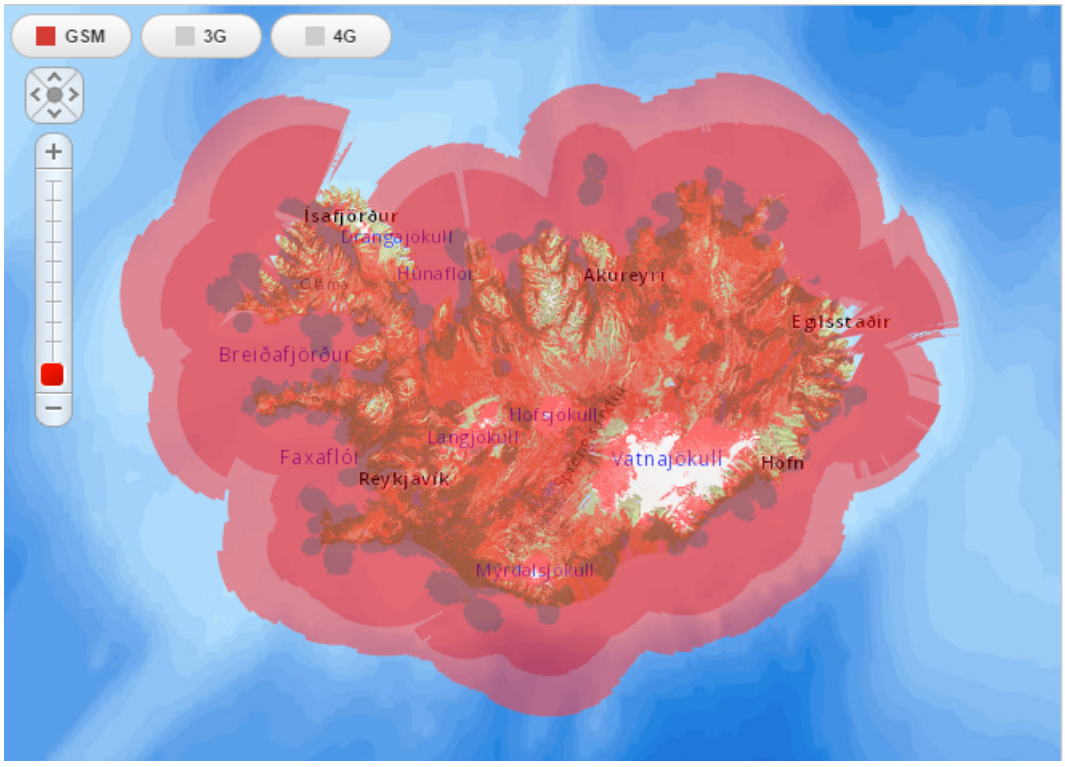

Figure 4: GSM coverage in Iceland[9]

Figure 5: Dr. Wickert[1]

- Phase 6: Software integration phase, all the software integrated to one.
- Phase 7: Field testing phase, testing of the system in real life environment.
- Phase 8: Fixing phase, fixing bugs found in the field testing phase.
- Phase 9: Deployment phase, system is functional and can be deployed.

3.1 Hardware

The hardware is divided into sensor modules and the mother hub. The sensor modules designed for this system were three wireless temperature sensors using Wixel[10] for means of communications. The mother hub consists of a Arduino Mega[2] that communicates with the sensor modules with a Wixel, collects the data and sends it to a HTTP server via the GSM/GPRS module[11]. See overview of the system in figure 8. All modules are encased in a IP67[12] plastic enclosure for protection against difficult weather conditions. Figure 6 shows hardware setup and figure 7 shows the assembled product.

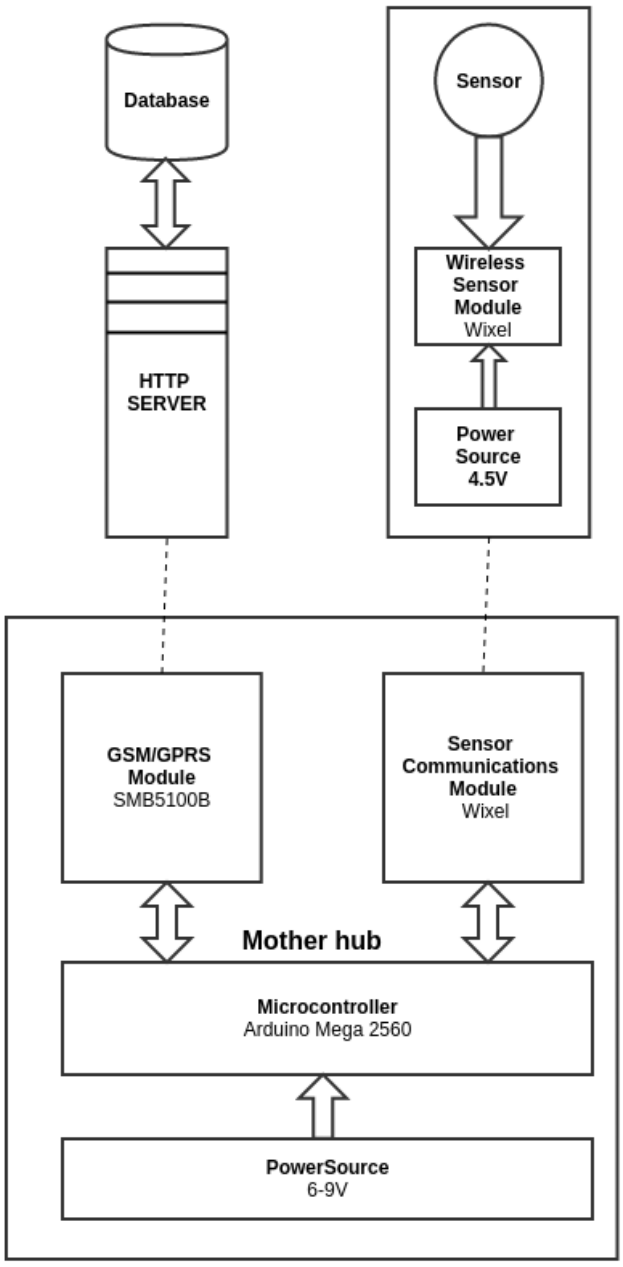

Figure 6: Hardware Module Design Diagram

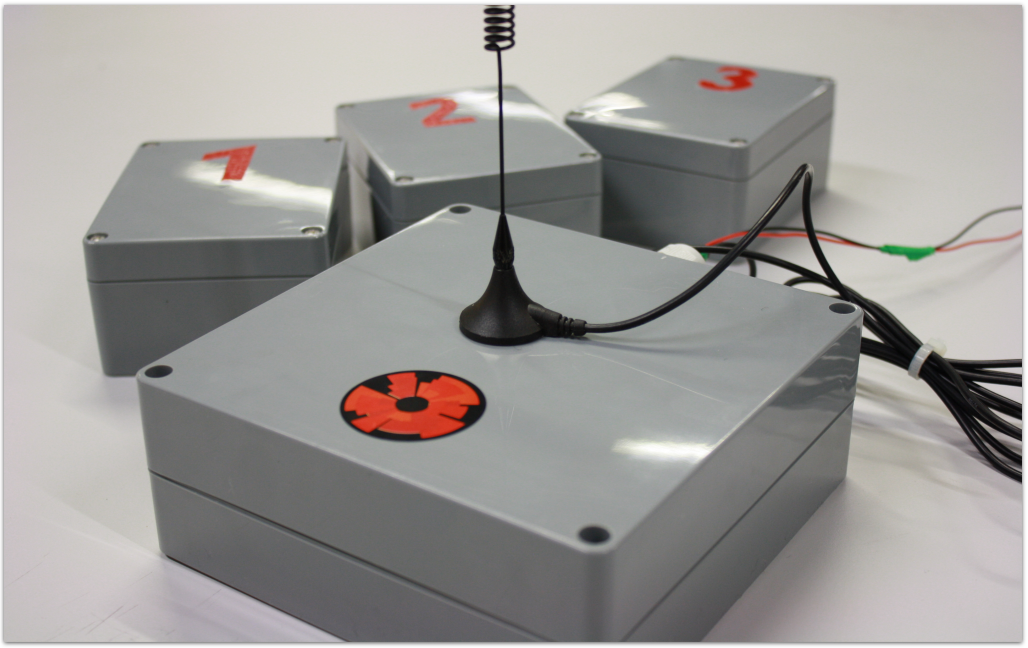

Figure 7: Assembled product

Figure 8: Demonstration drawing of wireless data collecting.

3.1.1 Sensor module

The sensor module is based on Wixel[10] which reads sensor value from it's analog inputs when the mother hub calls for reading. To save energy the sensor module does not send any data until it's id is called by the mother hub. The sensor module then gathers several readings from the sensor over a short period of time and sends back the average reading of the data collected. After sending the the data it falls back to suspend mode until the mother hub calls again for data. Electrical schematic diagram of the sensor module can be seen in figure 15.

3.1.2 Mother hub

The mother hub is based on Arduino Mega 2560[2] which can be switched out for Dr. Wickert's Bottle Logger[3], a Wixel[10] used for communicating with the wireless sensor modules and a SM5100B GSM/GPRS module[11]. The mother hub gathers data from the Wireless sensor modules via the onboard Wixel by calling them by id and wait till corresponding sensor module answers with newly gathered sensor data. The data gathered is then saved to its onboard EEPROM [13]. At a preset time the data is sent to a HTTP server via GPRS [14] where it can be processed by the researcher. Electrical schematic diagram of the mother hub can be seen in figure 16.

3.2 Software

When designing the software, modularity was the main interest and is better described as a framework ready for different implementation. The design aims for making it as versatile as possible and ready for implementation of new ways of communications, sensors and storage. This is done by building the software up of three interfaces the user needs to implement if for example another means of communications is needed. The system comes with two ways of communications to the outside world, a SMS gateway and a GPRS HTTP gateway. It also comes with one way of storing data, to the EEPROM and one way of communicating to wireless sensors. Figure 9 shows the layered design of the software. The GeoLog project is open source and anyone can contribute to the project at `https://github.com/GeoLogTeam/geolog`.
To take a closer look in to the software and how the interfaces work see the class diagram shown in figure 17. Next sections will describe more thoroughly how each module works. All code is written in C++ except the HTTP server which is written in Python/Flask-Restful [15].

3.2.1 Sensor Service interface

The sensor service is an interface the developer needs to implement for every new kind of sensor module. The sensor service interface has three virtual functions that the developer needs to implement by inheriting from the SensorService class.The functions that needs implementation are the following:

- **int getSensorData(int sensorId)**
  - Takes in the id of the sensor and returns the raw value.
- **int sendDataToSensor(String data)**
  - Takes in data to send to sensor and returns 0 for OK and -1 for error. This could be used to remote calibrate sensor.

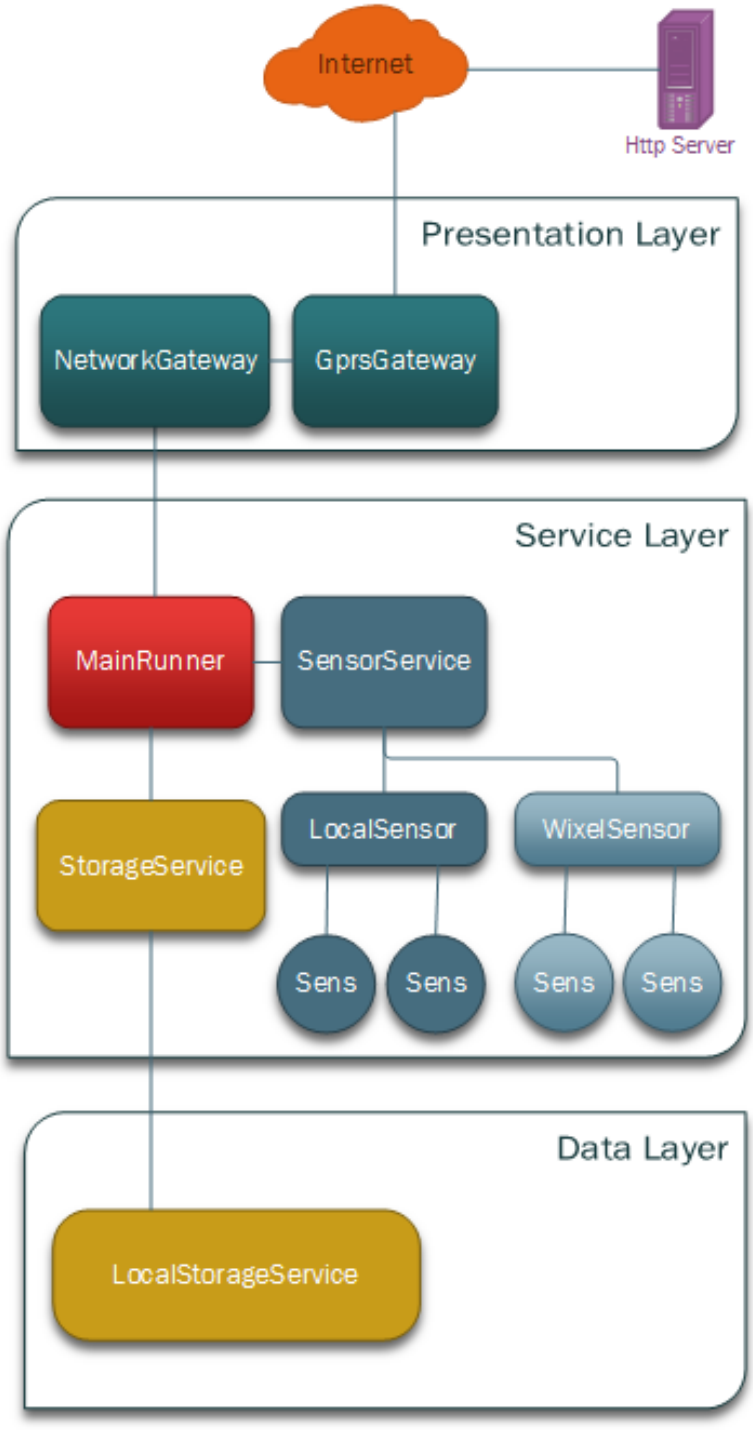

Figure 9: Layered Design Diagram of the system

- **int setDateTime(long DateTime)**
  - Takes in UNIX date/time for setting the time on the sensor if needed.

The sensor service now implemented is the WixelSensorService. This service communicates with the Wixel Sensor Modules, converts the raw data to a value of interest like temperature and finally parses the converted data to Json[16] strings. Sample of Json parsed data can be seen in figure 10. Pseudocode of the getSensorData() function can be seen in listing 1. Calibration of the sensor data was done with equation 1.

$$\frac{data \cdot 4.0V}{1023bit} \cdot 100\frac{bit \cdot^{\circ} \mathrm{C}}{V} - 50^{\circ}\mathrm{C} \tag{1}$$


```
{
    "date": "51032",
    "sensorId": "1",
    "temp": "23"
}
```


Figure 10: Json[16] string received from the WixelSensorService

Listing 1: Pseudocode of WixelSensorService getSensorData() function

```
Function(getSensorData(sensorId)):
    define wixelSerialPort as desired serial port
    define TimeOut as desired timeout in milliseconds
    define startTime as current time in milliseconds
    define data as 0;
    while data equals 0 or ((current time - startTime) > TimeOut:
        write sensorId to wixelSerialPort
        wait for 100 milliseconds
        if wixelSerialPort read buffer is not empty:
            data = data read from wixelSerialPort
            flush wixelSerialPort
        end if
    end while
    return data converted to celsius
end Function(getSensorData)
```

3.2.2 Storage Service interface

The storage service is an interface the developer needs to implement if a new way of storing data is needed. The storage service interface has four virtual functions the developer

needs to implement by inheriting from the StorageService class. The functions that needs implementation are the following:

- **int save(String data)**
  - Takes in data in String format and saves it to storage medium. Returns number of bytes saved, returns -1 if not enough space.
- **String load()**
  - Loads all data in memory and returns it in string format.
- **int erase()**
  - Erases all data from storage medium. Returns number of bytes free after erase.
- **bool isFreeSpcace(int dataSize)**
  - Takes in size of data to be saved in bytes. Returns true if enough space is available, false if not.

The storage service now implemented is the LocalStorageService. This service stores data to the Arduino's EEPROM. The Arduino Mega 2560 has 4096 bytes of EEPROM [2]. The LocalStorageService stores the data as series of Json strings. A typical use of the LocalStorageService is first to use the isFreeSpace() function to check if enough space is available for the data to be stored. If space is enough the save() function is used to save the data to the EEPROM. If space is not available the developer has to take some action like loading all existing data using the load() function, send the data to the HTTP server before using the erase() function to erase the EEPROM for new data. Pseudocode of the save(), load() and erase() functions can be seen in listings 2

Listing 2: Pseudocode of LocalStorageService save(), load() and erase() functions

```
Function(save(data)):
    define iterator as 0
    define i = 0
    for i < EEPROM size:
        read byte i from EEPROM
        if byte i equals 0:
            break
        end if
        iterator = iterator + 1
        i = i + 1
    end for
    if not enough space for data:
        return -1
    else:
```

```
        define dataSize as 0
        define j as 0
        for j < data size in bytes:
            if iterator >= (EEPROM size -1):
                break
            end if
            write byte at index j in data to EEPROM address that equals iterator
            iterator = iterator + 1
            dataSize = dataSize +1
        end for
        write ',' to EEPROM address corresponding to iterator
        dataSize = dataSize + 1
        return dataSize
    end if
end Function(save(data))

Function(load):
    define data as string
    define i as 0
    define letter
    for i < EEPROM size:
        letter = byte at EEPROM address that equals i
        if letter equals 0:
            break
        end if
        append letter to data
    end for
    return data
end Function(load)

Function(erase):
    define counter as 0
    define i as 0
    for < EEPROM size:
        counter = counter + 1
        write 0 to EEPROM address that equals i
    end for
    if counter not equals EEPROM size:
        return -1
    else:
        return counter
    end if
Function(erase)
```

3.2.3 Network Gateway interface

The network gateway is an interface the developer needs to implement for each new way of communications with the outside world. The network gateway interface has five virtual functions the developer needs to implement by inheriting from the NetworkGateway class. The functions needing implementation are the following:

- **bool connectToNetwork()**
  - Connects to network. Returns true if connected, false if not connected.
- **bool connected()**
  - Checks if connected to network. Returns true if connected, false if not connected.
- **int sendData(String address, String data)**
  - Sends given data to given address. Returns number of bytes sent, -1 if error sending data.
- **String reciveData(String address)**
  - Gets data from given address. Returns the data.
- **long getTime()**
  - Gets the current time/date from the network. Returns the time in UNIX time/date as long.

The Network Gateways now implemented are the SmsGateway and the GprsGateway. The GprsGatway is based on code from Toby Fox[17] and the SmsGateway utilizes the SerialGSM library from Meir Michanie[18]. Since SmsGateway is not yet fully developed the GprsGateway will be described in detail here.
The GprsGateway sends gathered sensor data as Json objects via HTTP to a HTTP API mentioned in chapter 3.2.5. The GprsGateway uses AT commands[19] to control the GPRS module. A typical use of the GprsGateway is first to check if the system is online by calling the connected() function. If the system is offline the connectToNetwork() function is called to connect to the GPRS network. When connected there are three functions to choose from; sendData(), receiveData and getTime. The sendData() function takes in the data as Json string and posts it to a given HTTP address as REST[20] message. The receiveData() function receives data from a given HTTP address. This can be used to update or change settings of the GeoLog. The getTime()function is used to get the current time from the network to sync the onboard clock. Pseudocode for connectToNetwork() and sendData() functions can be seen in listings 3

Listing 3: Pseudocode of GprsGateway connectToNetwork() and sendData() functions

```
Function(connectToNetwork):
    boot up GPRS module
    send "AT+CGATT=1" to the GPRS module
    wait for OK from the GPRS module
    if not received OK from GPRS module:
        return false
    end if
    send "AT+CGDCONT=1, IP, YOUR_APN" to the GPRS module
    wait for OK from the GPRS module
    if not received OK from GPRS module:
        return false
    end if
    send "AT+CGACT=1,1" to the GPRS module
    wait for OK from the GPRS module
    if not received OK from GPRS module:
        return false
    end if
    send "AT+SDATACONF=1, TCP, YOUR_SERVER_IP_ADDRESS, YOUR_SERVER_PORT_NUMBER"
    wait for OK from the GPRS module
    if not received OK from GPRS module:
        return false
    end if
    send "AT+SDATASTART=1,1" to the GPRS module
    wait for OK from the GPRS module
    if not received OK from GPRS module:
        return false
    end if
    return true
end Function(connectToNetwork)

Function(sendData(address, data)):
    define timeOut as desired timeout in milliseconds
    define data as "POST /geolog HTTP/1.1
                    Host: YOUR_HTTP_SERVER_ADDRESS
                    Content-Type: application/json
                    Content-Length: SIZE_OF_DATA_IN_BYTES

                    contents of data"
    if not connected to network:
        connectToNetwork()
    end if
    define inTime as time now in milliseconds
    while (time now in milliseconds - inTime) < timeOut:
        send "AT+SDATASTATUS=1" to the GPRS module
        get socketstatus from GPRS module
```

```
    if not received OK from GPRS module:
        return -1
    end if
    if socketstatus = "+SOCKSTATUS:  1,0,0104,0,0,0":
        wait for 1000 milliseconds
    else if socketstatus = "+SOCKSTATUS:  1,1,0102,0,0,0":
        break
    else:
        return -1
    end if
end while
define packetlength as size of data in bytes
send "AT+SDATATSEND=1, packetLength\r" to the GPRS module
if not received ">" from the GPRS module:
    return -1
end if
send data to the GPRS module
send ctrl-z to the GPRS module
if not received OK from GPRS module:
    return -1
end if
define inTime as time now in milliseconds
while (time now in milliseconds - inTime) < timeOut:
    send "AT+SDATASTATUS=1" to the GPRS module
    get socketstatus from GPRS module
    if socketstatus equals "+STCPD:1":
         get socketstatus from GPRS module
    end if
    if socketstatus equals "+STCPC:1":
        get socketstatus from GPRS module
    end if
    if not received OK from GPRS module:
        return -1
    end if
    if socketstatus not starts with "+SOCKSTATUS":
        return -1
    end if
    if socketstatus packet length equals packetlength:
        break
    else:
        wait for 1000 milliseconds
    end if
end while
send "AT+SDATASTATUS=0" to the GPRS module
send "AT+SDATASTART=1,0" to the GPRS module
if not received OK from GPRS module:
    return -1
```

```
    end if
    send "AT+CGATT=0" to the GPRS module
    if not received OK from GPRS module:
        return -1
    end if
    send "AAT+CGATT?" to the GPRS module
    set connected as false
    return 1
end Function(sendData)
```

3.2.4 Main Runner

The main runner is the .ino file, e.g. the Arduino sketch. It is up to the researcher how it is set up although default setup is provided under the filename MainRunner.ino. Users are encouraged to modify the MainRunner to their needs. The MainRunner comes with number of helper functions and are as following:

- **void collectData(int sensorId)**
  - Used to collect data from sensor with given id. This function should be modified if using other kind of sensor than provided.
- **String parseDataToJson(int data, int sensorId)**
  - Used to parse values from a given sensor and its id to Json object.
- **String concatToJson(String data)**
  - Makes a Json array of given data. Takes in data as stringified comma separated Json objects.
- **int sendToServer(String data)**
  - Used to send given data to server. This might have to be modified if using another kind of communications than provided.
- **bool eraseRequest()**
  - Used to erase the memory manually by pushing the erase button.
- **bool manualSendRequest()**
  - Used to send data to the server manually by pushing the manual send button

A typical application of the MainRunner could be as shown in the pseudocode shown in listings 4

Listing 4: Pseudocode of typical application of the MainRunner

```
define GSM_RESET_PIN as 22
define SAMPLE_DATA_FREQ as 300000 ms
define SEND_DATA_FREQ as 900000 ms
define TIME_OUT as 45000 ms
define MAX_TRIALS as 5
define ERASE_PIN as 50
define MANUAL_SEND_PIN as 51
define LED_PIN as 13
define sendRequest
define lastDataSample
define lastDataTransmit
define faildTransmits

Function(setup):
Set GSM_RESET pin as OUTPUT
Set ERASE pin as INPUT_PULLUP
Set MANUAL_SEND_PIN pin as INPUT_PULLUP
currentTime = current time
lastDataSample = current time
lastDataTransmit = current time
failedTransmits = 0
sendRequest = false
end Function(setup)

Function(loop):
if (current time - lastDataSample) > SAMPLE_DATA_FREQ or sendRequest equals true
    ↪ :
    collect data from sensor with id 1
    collect data from sensor with id 2
    collect data from sensor with id 3
    lastDataSample = current time
end if
if (current time - lastDataTransmit) > SAMPLE_DATA_FREQ or sendRequest equals
    ↪ true:
    load data from memory
    parse data to Json array
    send data to server
    if data was succesfully sent:
        erase memory
        lastDataTransmit = current time
        failedTransmits = 0
        sendRequest = false
    else if faildTransmits >= MAX_TRIALS:
        failedTransmits = 0
        lastDataTransmit = current time
```

```
[
    {
        "date": "51032",
        "received": "05. November 2014 15:24",
        "sensorId": "1",
        "temp": "23"
    },
    {
        "date": "51295",
        "received": "05. November 2014 15:24",
        "sensorId": "2",
        "temp": "24"
    },
    {
        "date": "51558",
        "received": "05. November 2014 15:24",
        "sensorId": "3",
        "temp": "21"
    }
]
```


Figure 11: Json[16] array received by the HTTP Server

```
        sendRequest = false
    else
        faildTransmits = faildTransmits + 1
    end if
end if
if manual erase button is pushed:
    erase memory
end if
if manual send data button is pushed:
    sendRequest = true
end if
end Function(loop)
```

3.2.5 HTTP Server

The HTTP server designed for GeoLog is a Python Flask-Restful HTTP API [15]. The design was minimal and should only be able to receive data from the GeoLog, store it and present it as a REST API and as more human readable. The data arrives as a Json array as seen in figure 11, the server parses the data and publishes it if the data is not corrupt. The human readable version is called GeoBlog. The code for the server is quite short and can be seen in the appendix under listings 5.

3.3 Safety

The GeoLog is designed to withstand difficult weather conditions although care must be taken when setting it up. Before taking the GeoLog out to the elements make sure every screw and fitting is tightened.
Warning: Water or dirt might slip into the enclosure and damage the equipment if the sealing is not tight.
Warning: Do not use acid batteries cause they may leak and cause damage to your equipment.
The software has been tested to meet the requirements stated in chapter 2. All modifications and changes are possible but can cause fail in the equipment. The GeoLog team is deprived from all liability for any kind of misuse of the equipment. The GeoLog is open source and the GeoLog Team welcomes contributors at `https://github.com/GeoLogTeam/geolog`.

4 Testing

During the development of the GeoLogger multiple test where performed. The most important tests will be listed in the following sections along with the data obtained from the tests.

4.1 Wixel range test

The test was performed in the Reykjavík University electronics lab. The room is 18 meters across and marked lines with 2.5 meters between them were laid down. A Wixel connected to a serial monitor was placed at one end of the room and another Wixel was moved between the lines regularly sending data strings. The table below shows if the data was received on the serial monitor. Each test was performed two times.

Table 2: Wixel range test

Number	Distance	Received	Number	Distance	Received
1	2.5m	yes	8	2.5m	yes
2	5m	yes	9	5m	yes
3	7.5m	yes	10	7.5m	yes
4	10m	yes	11	10m	yes
5	12.5m	yes	12	12.5m	yes
6	15m	yes	13	15m	yes
7	18m	no	14	18m	no

4.2 Lab test one

Lab test one was performed in the Reykjavík University electronics lab. The mother hub was placed at the centre of the room and the wireless sensor modules were placed in three corners of the room. The mother hub was programmed to poll the wireless sensor modules for data every five minutes and then send the gather data to the HTTP server every 15 minutes. The aim of the test was to see if the system was ready for field testing and to detect errors. No errors were detected during the test and the GeoLog was made ready for field testing. Table 3 shows the results of lab test one.

Table 3: Lab test 1

Transmit time	**Timestamp [sec]**	**Sensor ID**	**Temperature °C**
05. November 2014 15:24	51	1	23
05. November 2014 15:24	51	2	24
05. November 2014 15:24	51	3	21
05. November 2014 15:24	56	1	23
05. November 2014 15:24	57	2	24
05. November 2014 15:24	57	3	21
05. November 2014 15:24	58	1	23
05. November 2014 15:24	58	2	24
05. November 2014 15:24	59	3	21
05. November 2014 15:58	137	1	23
05. November 2014 15:58	137	2	24
05. November 2014 15:58	137	3	20
05. November 2014 16:15	438	1	23
05. November 2014 16:15	438	2	24
05. November 2014 16:15	438	3	20
05. November 2014 16:15	740	1	23
05. November 2014 16:15	740	2	24
05. November 2014 16:15	741	3	20
05. November 2014 16:15	1041	1	23
05. November 2014 16:15	1041	2	25
05. November 2014 16:15	1041	3	21

4.3 Field test one

Field test one was performed in Öskuhlið next to Reykjavík University. The mother hub was attached to a tree along with a battery pack containing five parallel connected 9V battery's.

The wireless sensor modules were also attached to trees with 15 meters radius around the mother hub. The mother hub was programmed to gather data from the wireless sensor modules with five minutes intervals and transmit gathered data every 15 minutes. The idea was to let the GeoLog run while transmitting data in order to monitor problems and to measure power consumption. As can be seen on the time stamp in table 4 the system was running when it was placed in the field, this was because the system was started up in the lab to confirm that it was going to send data before it was placed in the field.
Table 4 shows that the mother station was able to send two whole data packs before stopping. When no data was received from the GeoLog for two hours a decision was made to restart the system and manually send data. As can be seen on the time stamp in table 4 included in the last data pack that was sent are measurements that where taken before the system stooped. This shows that the function that stores data on the EEPROM is doing what it is designed to do, that is to make sure that even though the system stops and is not able to transmit data no information is lost.
The likely reason why the system stopped sending data is the mother hub did not receive data from one or more of the wireless sensor modules. More research has to be done to confirm this.

Table 4: Field test 1

Transmit time	**Timestamp [sec]**	**Tensor ID**	**Temperature °C**
05. November 2014 17:13	2295	1	0
05. November 2014 17:13	2298	2	1
05. November 2014 17:13	2299	3	3
05. November 2014 17:13	2619	1	-5
05. November 2014 17:13	2619	2	-3
05. November 2014 17:13	2620	3	-1
05. November 2014 17:13	2924	1	-7
05. November 2014 17:13	2925	2	-6
05. November 2014 17:13	2925	3	-5
05. November 2014 17:31	3234	1	-8
05. November 2014 17:31	3235	2	-8
05. November 2014 17:31	3235	3	-6
05. November 2014 17:31	3558	1	-10
05. November 2014 17:31	3559	2	-8
05. November 2014 17:31	3560	3	-8
05. November 2014 17:31	3892	1	-10
05. November 2014 17:31	3893	2	-9
05. November 2014 17:31	3893	3	-9
05. November 2014 19:35	4198	1	-10
05. November 2014 19:35	4212	2	-9
05. November 2014 19:35	4212	3	-9
05. November 2014 19:35	4521	1	-11
05. November 2014 19:35	4526	2	-10
05. November 2014 19:35	4527	3	-10
05. November 2014 19:35	12	1	-14
05. November 2014 19:35	15	2	-12
05. November 2014 19:35	15	3	-12

4.4 Routine user tests

Tests needed to be perform regularly are: The user needs to calibrate any sensors he wants to implement with the system, he needs to do ranges tests for different wireless sensor modules and for different conditions the plans on using them in. The user also needs to perform tests on the GSM connection on the field where the mother hub is placed to ensure

data delivery.

5 Usage

The GeoLog was designed for the field scientist or anyone how is interested in monitoring the enviroment. In the development of the GeoLog it was taken into account the system would be able to perform in rough environment and could be left outside in rugged terrain. The knowledge need for setting up and maintain his own GeoLog is a background in software C, C++ and knowledge of electric circuits.

5.1 Installation

When preparing the system for first use setup of necessary software is needed to be able to configure the sampling rate of the desired data. The user needs to be able to modify the Arduino sketch but should not be required to modify the Wixel software. Thus set up of the Arduino development enviroment is needed but for the Wixel it is optional.
All the required code for this system can be found under `https://github.com/sveinnel/geolog`. The link `https://github.com/sveinnel/geolog` will be referred as geolog-folder from here on.

1. Install the software
   (a) For the Arduino:
      i. Start a web browser
      ii. Go to `http://arduino.cc/en/main/software`
      iii. Install the appropriate version of Arduino IDE for your platform. Information on how to set up the arduino can be seen here `http://arduino.cc/en/Guide/HomePage`
   (b) For the Wixel:
      i. Start a web browser
      ii. Go to `http://www.pololu.com/docs/0J46`
      iii. Install the Wixel IDE, see part 3 in Pololu user's guide
      iv. Follow the instructions listed in part 10.a, 10.b and 10.c, for setting up the development enviroment
2. Set up the hardware
   (a) Connect the components for the mother hub:

i. Follow the schematic in figure 16

ii. Position the mother hub in its box as in figure 12

(b) Connect the components for the wireless sensor module/modules:

i. Follow the schematic in figure 15

ii. Position the components in its box as in figure 13

3. Upload the code

(a) For the mother hub:

i. Upload the application wireless_serial.wxl on the Wixel. The application can be found under geolog-folder/Wixel/Receiver

ii. Upload the script MainRunner.ino on the Arduino. The script can be found under geolog-folder/Software

(b) For the wireless sensor module/modules:

i. Upload the application wireless_serial.wxl on the Wixel. The application can be found under geolog-folder/Wixel/Transmitter

4. Place the batteries in each module:

(a) The mother hub needs a power source of 6-9V

(b) The wireless sensor modules need a power source of 2.7-6.5V

Set the sampling frequency and the data transmission time for the GeoLog to the desired values. To set the values the MainRunner.ino sketch needs to be modified. The sketch can be found under geolog-folder/ArduinoSoftware. Change the defined constants SAMPLE_DATA_FREQ and SEND_DATA_FREQ in line 13 and 14, the values for the sampling time should be given in milliseconds. Then compile the modified MainRunner.ino sketch on the Arduino then the GeoLog should be ready for collecting data.

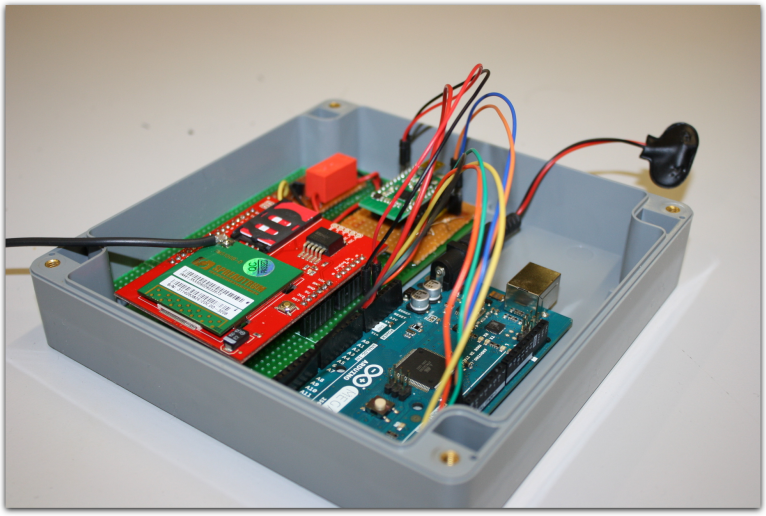

Figure 12: The mother hub displayed in its box

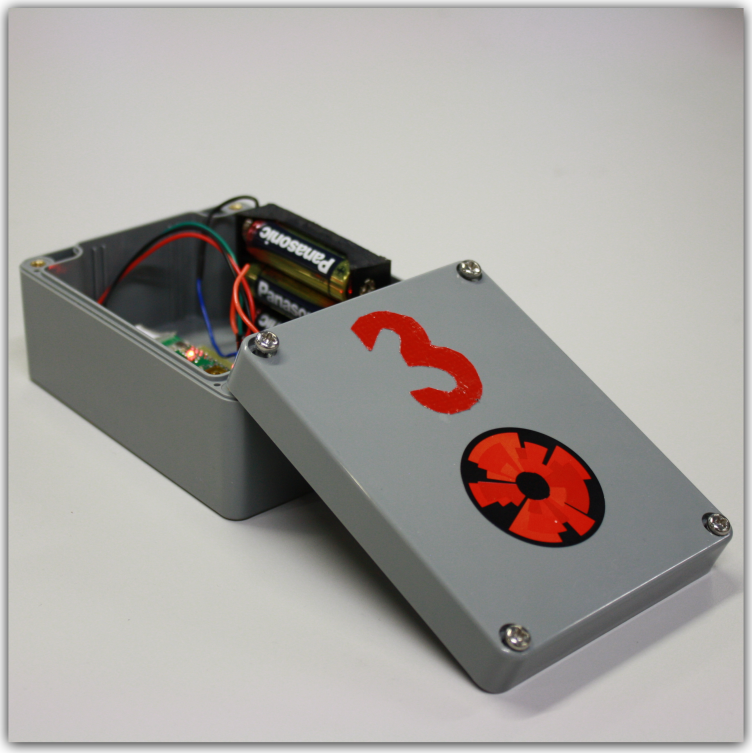

Figure 13: The wireless sensor module in its box

To prepare the GeoLog for transport the batteries need to be removed out of the box for each module. The boxes should be securely closed with the four screws in each corner. Then the GeoLog is ready for travel.
To unpack the GeoLog open the boxes, insert the batteries and securely close the boxes again.
Warning: If the boxes are not closed securely, they might start leaking. That could damage the electronics.

5.2 Instructions

Before the system is set up the area chosen needs to be checked for GSM signal.
Warning: There has to be GPRS coverages for the GeoLog to send data to the HTTP server.
Once the system is setup the user should be able to position the GeoLog in the site where data will be collected.
Spread out the wireless sensor modules around the mother hub, such they can sample the desired data.
Warning: the Wixel sensor module can not be placed more than 15 meters away from the mother hub.
The Wixels will not be able to transmit the data over a distance of more than 15 meters in clear line of sight.

If everything has been setup correctly the raw data will be transmitted over to the HTTP server.
Extra preparation or calibration should not be needed. The only routine maintenance needed is replacement of battery packs when the ones installed run out.

If the GeoLog is not transmitting the data over to the server the user can manually request a transmission to be sent. In order to request a transmission the user needs to connect a wire from pin 51 on the Arduino to the ground see figure 14. The red light on the GSM module should turn on for a short while (10 to 30 seconds) and turn off again. If the data does not arrive on the HTTP server the user might be forced to clear the EEPROM on the Arduino and try again. To clear the EEPROM the user needs to connect a wire from pin 50 on the Arduino to the ground, see figure 14. The led at pin 13 on the Arduino board will be turned on until the EEPROM has been fully cleared. Once the EEPROM has been cleared the user should attempt once again to manually request a transmission of data to the HTTP server. If the data does still not arrive on the HTTP server the user is advised to go over the installation process once more to ensure everything is correctly connected and is operating the correct software on each module.

If any further errors or malfunctions occur please contact our development team through our github account `https://github.com/GeoLogTeam/geolog`.

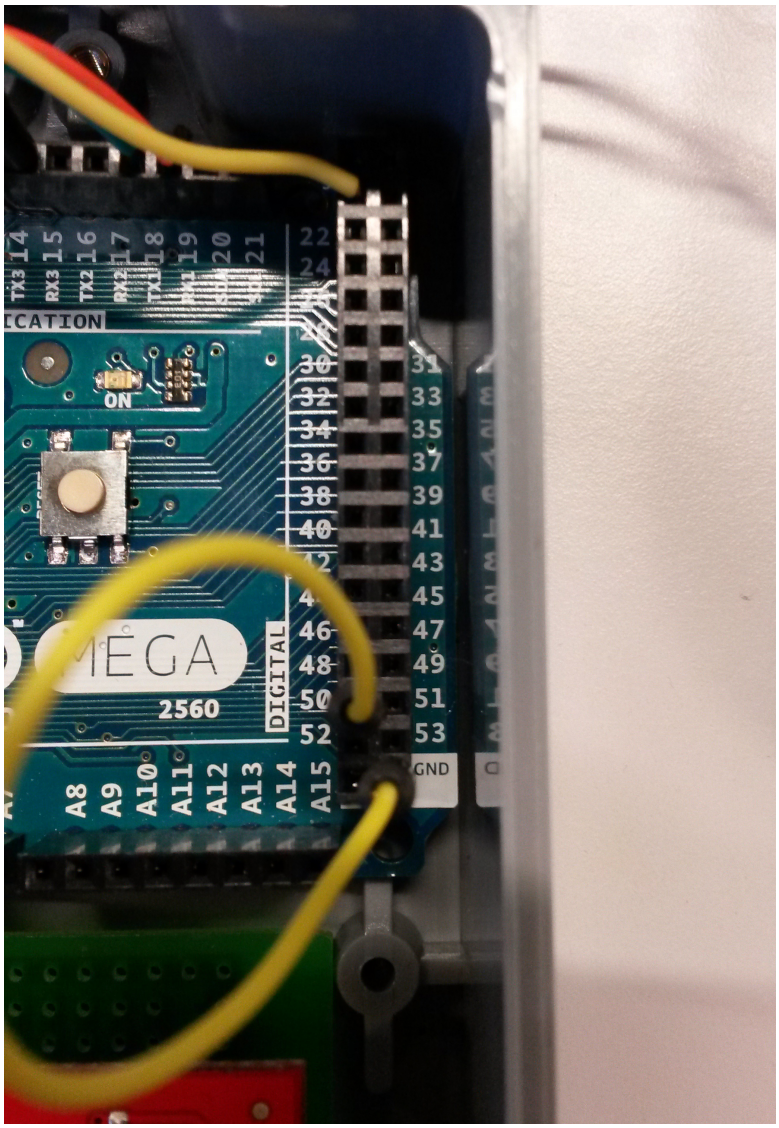

Figure 14: The request transmission pin 51 and clear EEPROM pin 50 shown

6 Results and Discussion

There are multiple obstacles a designer stumbles upon when designing a mechatronic product. One of the biggest obstacles that came up while designing this product was insufficient documentation for the GSM module [11]. A GitHub library [18] was found that helped with the development of the HTTP protocols for the GSM module. With the library at hand the obstacle was overcome but with small steps at a time. The design is not perfect some of the issues are still present and need to fixed if the design is to be fully functional. For example the Wixel in the wireless sensor module has only range of 15 meters [10]. The range of the Wixel limits the capabilities of the GeoLog device to spread out an array of sensors for observing large area. The power consumption of the device is quite a lot see table 5, but there were some actions taken to lower the power consumption of the device. The device was modified to tun on the GSM module for a short time, just enough time to send the data

and then turn off the GSM module again. The device might lower its power consumption if it were using the low power Arduino based board from Dr. Wickert.

Table 5: Current measurements

Mode	**mA**
Arduino + GSM shield idle	230
Arduino + GSM shield off	125
Arduino + GSM + Wixel sending data	340
Arduino standalone	80

The GSM transmission of data is not robust enough as can be seen in table 4. The gap from time 17:31 to 19:35 on November 5. 2014 shows missing transmission. This could be due to a bug in the software or the GSM module its self. This has to be investigated further.

Conclusion

Summary of system capabilities:

- Measure temperature with 3 wireless sensors modules.
- Wireless sensor module sends data through Wixel when mother hub ask for it.
- Mother hub samples wireless data with given sample rate.
- Mother hub stores the data.
- Mother hub sends pre-sampled data through GSM network to HTTP server.
- Mother hub and wireless sensors modules runs on battery power.
- Wireless sensor modules ran through the testing period of 24 hours on 3 AA batteries.
- Mother hub ran on 5x parallel connected 9V batteries during testing period.

Comparison of the system capabilities to the functional requirements (FR) is listed in table 6.

Table 6: Requirements met

	Functional Requirements (FR)	**Evaluation (DP)**
1	Collect measurable data	3x wireless sensor modules measured temperature with TMP36 [8] sensor and sends data to mother hub through wixel.
2	Store data local	The mother hub uses an Arduino MEGA and collects data from the wireless sensor moduls and stores them in EEPROM memory.
3	Keeps data for x time	Mother hubs Arduino MEGA [2] has 4KB of EEPROM and stores data from wireless modules. When the mother hub connects to GSM network and the collected data has been sent the EEPROM is erased. If the mother hub can not send data before the memory gets full, then no data is written to the EEPROM until it has ben restored.
4	Runs on own power source	Wireless sensor modules run on 3x AA batteries. In order to keep power consumptions down, data is only collected and sent to the mother hub when the mother hub requests for temperature. The mother hub runs on 5x parallel 9V batteries. To lower energy consumptions the power to the GSM network is shut down when it's not in use. Temperature sample rate and data send rate have significant impact on power consumption.
5	IP67 proof	Wireless sensor modules consisting of 1x wixel, 1x TMP36 and 3x AA batteries are fitted into 11.5cm x 9cm x 5.5cm IP67 fiber box. The Mother hub that consist of 1x Arduino MEGA, 1x GSM Shield and 1x wixel were fitted into 16cm x 16 cm x 6cm IP67 fiber box. For test phase the 5x parallel 9V batteries were fitted into old 21,5cm x 14cm x 9cm ice-cream box and sealed with duct-tape.
6	Sent data wireless through GSM network	The mother hub was installed with GSM Shield; SM5100B[11] to send data though GPRS network. EEPROM_SIZE format is send to interpret the data.
7	Store data on server	HTTP server is set up to store Json data from mother hub.
8	Modular Design	The software architecture is based on modular design to make it as easy as possible to add or change modules.

All functional requirements in table 6 where fulfilled there for the project is considered a success. There are of course many things that can be improved later and on and will be listed in the chapter future work here below.

Future work

Next steps would be continuing field testing and later on merge this project to Dr. Wickert's Arduino logger.
Other communication modules could be developed, for example UHF, VHF or Iridium.
Optimize power consumption is essential for achieving usable product according to the requirements. Power source is also essential, it has to be functional in glacier environment were temperature can go down to $-15\,^{\circ}$C and withstand high winds and precipitation.
Solar panels and wind generators might be feasible options, depending on the environment.
A user interface should be developed for usability.
To get the system to collect data in remote areas it is estimated to take 4000 man-hours to completed that work.
With this work complete the GeoLog will be affordable and versatile tool for scientists exploring earth's behavior for example to predict natural disasters and overall to make the world a better place.

References

[1] A. D. Wickert, “Earth-surface processes,” October 2014. [Online]. Available: http://instaar.colorado.edu/~wickert/

[2] Arduino, “Arduino Mega,” October 2014. [Online]. Available: http://arduino.cc/en/Main/arduinoBoardMega2560

[3] A. D. Wickert, “Alog-bottlelogger,” September 2013. [Online]. Available: https://github.com/NorthernWidget/ALog-BottleLogger

[4] WeatherShop, “Solar powered cellular weather station,” October 2014. [Online]. Available: http://www.weathershop.com/cellular_weather_station.htm

[5] Texas Weather Instruments, Inc., “Remote weather station solar powered,” October 2014. [Online]. Available: http://txwx.com/product-line/remote-weather-station/

[6] Oregon Scientific, “Oregon Scientific WMR300-7716 Ultra-Precision Professional Weather System with Tripod Mount,” October 2014. [Online]. Available: http://www.oregonscientificstore.com/Oregon-Scientific-WMR300-7716-Ultra-Precision-Professional-Weather-System-with-Tripod-Mou data

[7] Davis, “Wireless Vantage Pro2 with Standard Radiation Shield,” October 2014. [Online]. Available: http://www.davisnet.com/weather/products/weather_product.asp?pnum=06152

[8] A. Devices, “TMP36 Low Voltage Temperature Sensors,” 2010. [Online]. Available: http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/TMP35_36_37.pdf

[9] “Gsm coverage,” Vodafone, September 2014. [Online]. Available: http://www.vodafone.is/simi/staersta/

[10] Pololu, “Wixel programmable usb wireless module,” October 2014. [Online]. Available: http://www.pololu.com/product/1337

[11] Sparkfun, “SM5100B GSM/GPRS module,” November 2014. [Online]. Available: https://www.sparkfun.com/products/retired/9311

[12] DSM&T Co, “IP Rating Chart,” November 2014. [Online]. Available: http://www.dsmt.com/resources/ip-rating-chart

[13] Arduino, “EEPROM Library,” 2014. [Online]. Available: http://arduino.cc/en/Reference/EEPROM

[14] M. Rouse, “GPRS (General Packet Radio Services),” 2014. [Online]. Available: http://searchmobilecomputing.techtarget.com/definition/GPRS

[15] K. Burke, K. Conroy, R. Horn, F. Stratton, and G. Binet, “Flask-RESTful,” November 2014. [Online]. Available: http://flask-restful.readthedocs.org/en/latest/

[16] ECMA-404 The JSON Data Interchange Standard, “JavaScript Object Notation,” Unnknown 2014. [Online]. Available: http://www.json.org/

[17] T. Fox, “SM5100B-GPRS Arduino library,” April 2014. [Online]. Available: https://github.com/tobek/SM5100B-GPRS

[18] M. Michanie, “SerialGSM Arduino library,” December 2012. [Online]. Available: https://github.com/meirm/SerialGSM

[19] Buddy, “SM5100B-D AT Command,” December 2008. [Online]. Available: https://www.sparkfun.com/datasheets/Cellular%20Modules/CEL-09533-AT%20Command_V1%5B1%5D.0.0-1.pdf

[20] R. T. Fielding, “Architectural styles and the design of network-based software architectures. chapter 5 - representational state transfer (rest),” dissertation, UNIVERSITY OF CALIFORNIA, IRVINE, 2000. [Online]. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Appendix

Design documents

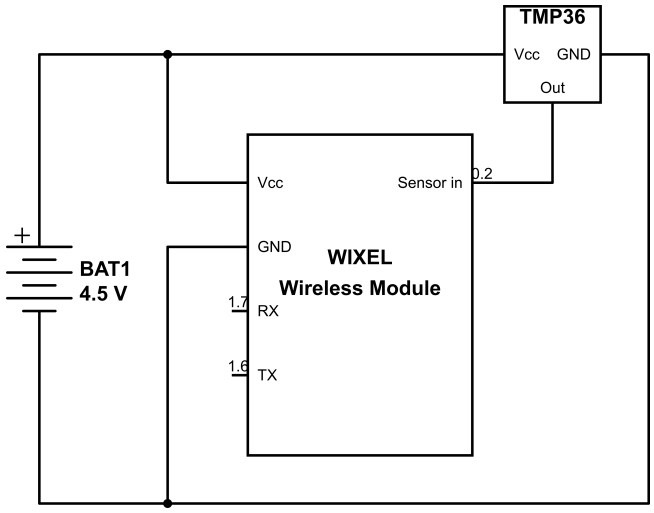

Figure 15: Electrical Schematic Diagram of the Wireless Sensor Module

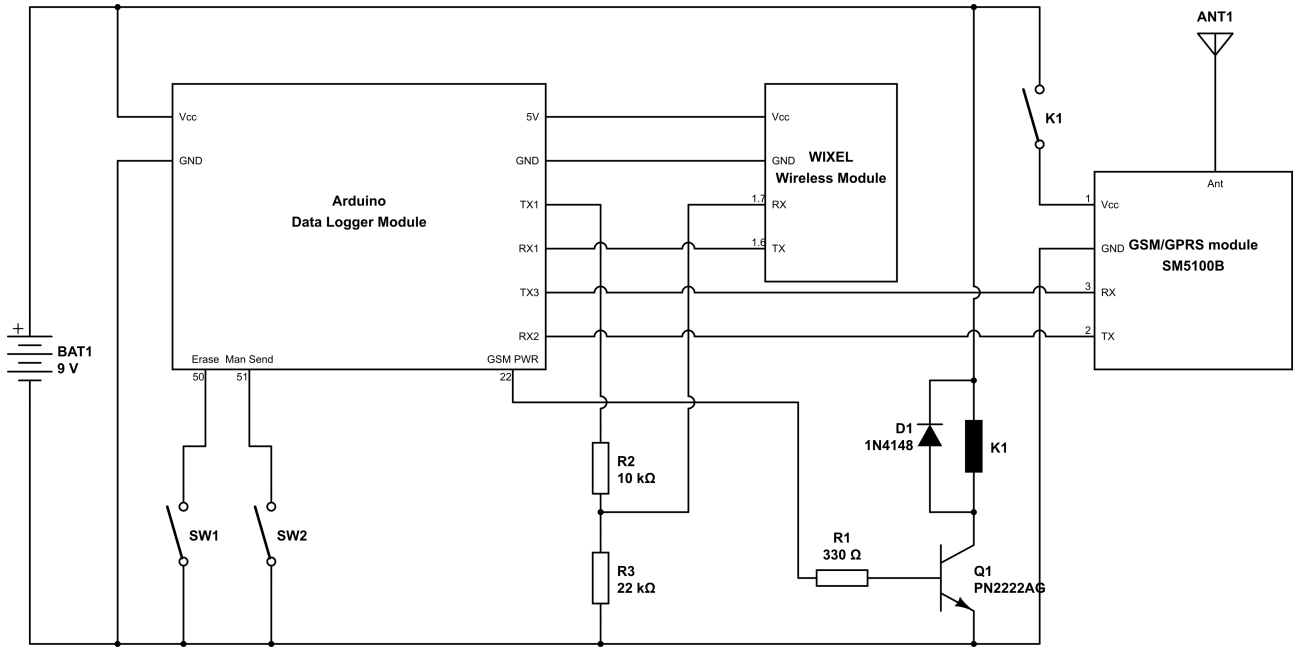

Figure 16: Electrical Schematic Diagram of the Mother Hub

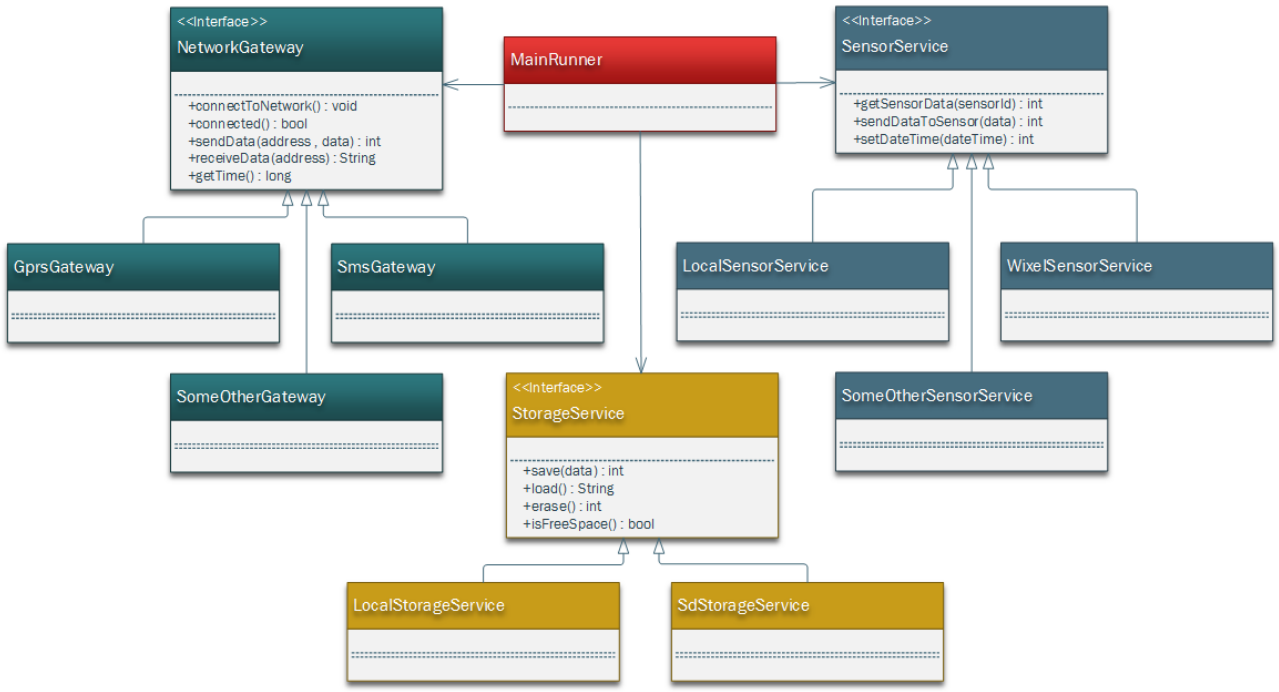

Figure 17: Class Diagram of the system

HTTP Server

Listing 5: Code for the HTTP server language=Python

```
import json, time
from flask import Flask, request
from flask.ext import restful
import datetime
from numpy import mean


class DataStorage(object):

    def __init__(self, file_name):
        self.file = file_name
    def append(self, dict):
        with open(self.file, 'a') as f:
            json.dump(dict, f)

    def get_data(self):
        try:
            with open(self.file) as f:
                json_array = ['{'+x+'}' for x in f.read().strip('{}').split
                    ↪ ('}{')]
                return [json.loads(x) for x in json_array]
        except:
            return []
```

```
app = Flask(__name__)
api = restful.Api(app)
alldata = DataStorage('data.json')

@app.route('/geoblog')
def hello():
    dataGathered =  alldata.get_data()
    lastRecieved = ""
    site = '<!DOCTYPE html>' \
         + '<html class="full" lang="en">'\
         + '<head>'\
         + '<meta charset="utf-8">'\
         + '<title>GeoBlog</title>'\
         + '<link href="http://bootswatch.com/cosmo/bootstrap.min.css" rel="
            ↪ stylesheet">'\
         + '</head>'\
         + '<body>'\
         + '<div class="container"> <div class="jumbotron">'\
         + '<h1>GeoBlog</h1>' \
         + '<h3>Average Temperature: ' + str(mean([int(t['temp']) for t in
            ↪ dataGathered])) + '</h3>' \
         + '</div></div><div class="container">'

    for i in range(0, len(dataGathered)):

        if dataGathered[i]['received'] != lastRecieved:
            site += '<div class="panel panel-default">'\
                 + '<div class="panel-heading"> Received at: ' +dataGathered[i
                    ↪ ]['received'] + '</div>' \
                 + '<div class="panel-body">'\
                 + '<table class="table"><tr><th>Timestamp sek</th><th>Sensor Id
                    ↪ </th><th>Temperature &degC</th></tr>'

        site += '<tr><td>' + str(int(dataGathered[i]['date'])/1000) + '</td>' \
              + '<td>' + dataGathered[i]['sensorId'] + '</td>' \
              + '<td>' + dataGathered[i]['temp'] + '</td></tr>'

        if i < len(dataGathered) - 1:
            if dataGathered[i + 1]['received'] != dataGathered[i]['received']:
                site += '</table></div></div>'

        lastRecieved = dataGathered[i]['received']

    site += '</div>'\
          + '</body>' \
```

```
          + '</html>'
    return site
          #+ '<script src="js/jquery-1.10.2.js"></script>' \
          #+ '<script src="js/bootstrap.js"></script>' \


class ApiGeoLog(restful.Resource):
    def get(self):
       return alldata.get_data()

    def post(self):
        dataRecieved = request.data
        #print str(dataRecieved)
        data = json.loads(dataRecieved)
        for d in data:
            #print str(d)
            if d['date'] != None and d['temp'] != None and d['sensorId'] != None
                ↪ :
                d['received'] = datetime.datetime.now().strftime("%d. %B %Y %H:%
                    ↪ M")
                alldata.append(d)

        return 'ok', 200


api.add_resource(ApiGeoLog, '/geolog')


if __name__ == '__main__':
    #app.debug = True

    app.run(host='0.0.0.0', threaded=True)
```