Renewal of data collection system at the Department of Environmental Science

and renewal of relevant connected systems

Morten Toudahl

Summary

This document describes how we worked as consultants for the Department of Environmental Science at Aarhus University and how we created prototypes as proof of concept for how they could improve upon their existing air measurements data gathering system.

A conflict

You may notice that this report is similar to an other report that has been turned in. The reason for this is that two days before the deadline, the two members of this group had a serious conflict. Because of this it was no longer possible to work together nor take a joint exam.

The majority of the report has been rewritten by Morten Toudahl. I have however edited the remaining sections that was written by Lárus Þór Jóhannsson, some more than others.

So please, take extra care when reading both mine and Lárus' report, as they are similar, but by no means identical.

Distribution permissions

Table of contents

Introduction

This report is written as a graduation dissertation at Zealands Institute of Business & Technology by the following students:

Morten Toudahl (b. 1985), local student from Denmark. He has previously completed a Webintegrator + 3D education. He started on the Computer Science AP degree in September 2014, and is currently working as a student developer for the Department of Environmental Science at Aarhus University, improving their website for the presentation of the data gathered by their measuring stations.

Lárus Þór Jóhannsson (b. 1987), international student from Iceland. Finished Icelandic upper-secondary education in 2008 and completed a film production diploma 2010 from Prague Film School. Attended Zibat AP Computer Science program since 2014. Currently working as a contracted student developer for the danish engineering company FLSmidth A/S.

The dissertation supervisor from Zibat is Anders Kristian Børjesson.

The client is the Danish Centre for Environment and Energy, a department in the Science & Technology branch of Aarhus University.

Morten had already established good relations with the client during his time there as an intern, and he realized the potential for improvements on the system that ensures the collection of environmental data.

Our assigned manager from Aarhus University is Keld Mortensen, the IT manager for the department, who is responsible for maintaining the current system.

Project establishment

Morten had initially made an arrangement with the dept. of Environmental Science to look at, and if possible, improve their data collection system. Later on, Lárus joined Morten. They have both worked together before, and compliment each other well in a project. Nevertheless, this new project meant that it would now be beneficial to agree on some working terms.

Team contract

Due to Lárus' job, the team is unable to meet up and work every day. So we agreed to the following conditions.

- Until week 49, the team will meet up twice a week to work together and be able to discuss various topics face to face.
    - The days may change to suit each participant's schedule, but will be Monday and Tuesday by default.
- When the team agree that it is no longer necessary to meet twice a week, they will start meeting face to face only once a week.
    - Default will be Tuesday, but this can be changed to fit each participant's schedule.
- Each person will also work alone, when not meeting up with the other person.
    - This will be at least one day a week, but preferably two.
    - Which day(s) is to be determined individually.
- Additional days can be agreed upon verbally or written should the need arise. I.e. in case of crunch time.
- Communication
    - The team will use appropriate means of communication depending on the urgency and type of information that needs to be passed along. E.g. email for documents, phone call/text to inform of tardiness.
    - If a member is late or cannot come for whatever reason, he must notify the other person.
    - The team will utilize a kanban board for project management, daily reports and keeping track of our progress. Specifically Trello[1]
- Version control
    - Version control must be used. The implementation of version control will be git. We will use the host bitbucket.org
        - It is forbidden to check in code that does not compile or otherwise does not work.

[1] Kanban board - https://trello.com/b/NnBT42jG/dissertation

System Introduction

On the day Lárus arrived at the department of Environmental Science we were given a tour of their campus and general tour of the Risø research area, the facility where the department is located.

This included a thorough presentation of the stations, and the architecture of the system. Both the overall architecture, but also at the application level of the data collection.

It also included a closer look at the test station located at Risø, and a good look at all of the hardware located at these stations, plus a look at the applications running on the instruments and station computers.

This allowed us to quickly come up with some ideas on what to improve, and gave us a good understanding of how their system operates.

Client

Our client is the Department of Environmental Science at Aarhus University. The university is considered among the top 100 universities in the world[2]. It is involved in education, research and has contracts with the danish government in multiple fields of study.

Under the university's administration we have the rector, the prorector, the university director and the deans of each field of study (e.g. Arts, Health, Science and technology). Inside of these fields there are multiple departments. Most of these departments provide one or more of the following: academic education, research and/or monitoring for the government i.e. for data regarding public health and environmental safety.

A good example of one of these monitoring agencies is the Department of Environmental Science / Institut for miljøvidenskab. This agency was previously called Denmark's Environmental Research (DMU - Danmarks miljøundersøgelse). It was contracted by the government to provide it with atmospheric data, that adhere to standards set by the European Union for measurements of air quality to be used in larger atmospheric models to monitor and predict global atmospheric conditions. This contract remained with the agency when it merged with Aarhus University which gives this department the atypical standing of being a monitoring agency that provides data for research activities, but has little to no involvement in actual education, aside from providing thesis students with assistance.

Since the 70's the department has gathered data regarding various compounds in the air, along with basic measurements of temperature, wind speed and direction. The measurements are carried out at 15 stations across the country, portable campaigns, as well as a single station in Greenland.

[2] Times Higher Education World University Rankings - https://www.timeshighereducation.com/world-university-rankings/2017/world-ranking#!/page/0/length/25/country/2256/sort_by/rank_label/sort_order/asc/cols/rank_only

Problem definition

Talking with members of the department we have discovered that most parts of the system have been developed as the need for them arose. The core architecture was developed back in the 70's. Some renewal of the system has been implemented, but most of it works like it has since the beginning with ad hoc changes. E.g. changing from communicating over the phone lines to using a mobile modem.

The result of this is a slow and unnecessarily complicated system, as a direct result of the patchwork of upgrades and modifications applied over the years, leading to our problem definition:

How can the system be modified to decrease complexity and increase effectiveness?

To answer this, we plan to look at the following:

- How do the instruments integrate with the current data gathering system, and how can this be improved?
- How does the current system collect the data from the instruments, and how can this be improved?
- How is the data stored in the current system, and how can this be improved?
- What issues does the current system have?
- What issues could the new system have?

We also wish to expand our own skills by answering the following questions:

- Where can I improve as IT system developers?
- What can I do better when designing a system for a client?
- How do we best ensure a high quality communication with the client?

We will do this by analyzing information provided by the department, and building prototypes and proof of concept programs that we will deploy on new hardware and test with instruments provided by the department.

Methodologies

The team established certain methodologies to assist us in managing workload, follow a value driven development mind-set, and ensuring the quality of our work. We would be drawing upon what we had learned at Zibat about software development methodologies, researching a viable research methodology and taking a critical eye at every one of them.

The software methodologies will be the guiding aspects of our project work and the research methodology will come into play when we are gathering data and gathering our conclusions from that data for this dissertation.

Research

The department already has all the knowledge we need. For this reason, I took an approach inspired by eXtreme Programming's principles, KISS and YAGNI.

I requested copies of the documentation for the instruments, and other relevant documents or scripts. These documents was then examined on a need to know basis, using something akin to the hermeneutic circle.

That is. When we needed to do work with an instrument, I would read what I needed to to complete my task. If I then found out we needed more information, I would then continue to that, until such at time I felt informed enough to finish my task.

Development

There must be a clear understanding that we are not shipping a deliverable product, but incrementally creating prototypes. These prototypes will perform their core function, and they will interact together. However, they will not be complete, and thus not ready for production. Should the department chose to implement the system we suggest, they must expect to put in more work to polish the pieces and add all of the needed functionality.

Below I will sum up the methodologies that we considered for our project.

eXtreme Programming (XP)[3]

XP is a very flexible and agile methodology. It is easily adaptable, and feels natural for both of us to work with. For this reason, it was the first methodology we thought of. It consists of the following practices:

- **Whole team**
  - This practice dictates that everyone should work closely together, to ensure that the software written fulfills the needs of the client.
- **Pair programming**
  - This practice helps with collective ownership of the code, while also improving the quality of the code written.
- **User Stories**
  - User stories document the requirements and the needs of the client, while also serving as a checkbox to verify that a feature has been completed.

[3] Agile Principles, Patterns, and Practices in C#, chapter 2
ISBN: 978-0131857254

- **Short cycles (iteration plan, release plan)**
    - Deliverable software every two weeks.
- **Acceptance tests**
    - Used to verify the completion of User Stories.
- **Test-driven Development (TDD)**
    - Write a test, before you write a piece of code.
- **Collective ownership**
    - Anyone can work on any module to improve it.
- **Continuous integration**
    - All code written is properly integrated into the existing code. Any finished code is pushed to a testing or production server as soon as it has been accepted.
- **Sustainable pace**
    - *"Software development is not a sprint; it is a marathon"*[4]. XP teams are not allowed to work overtime, as this is detrimental to the quality of the software.
- **Open workspace**
    - Everyone works together in the same rooms, communicating about progress and issues concerning the development process.
- **Planning game**
    - The planning game helps with assigning points to a user story, which in turn will help with the release plan.
- **Simple design**
    - "Do the simplest thing that could work". As in, do not implement a database before it is absolutely required.
    - "You aren't going to need it". Do not try to predict what you will need. Wait until you need it.
- **Refactoring**
    - Any time you work on code, you must improve it by refactoring. This is not done at the end of the project.
- **Metaphor**
    - The practice of using metaphors to describe and understand the project as a whole. This can also help with choosing meaningful names for methods and classes, etc.

[4] Agile Principles, Patterns, and Practices in C#, p 18
ISBN: 978-0131857254

Unified Process[5]

Unified process was the first software development methodology we learned at our school. It is a very documentation heavy, with a lot of artifacts and disciplines. However, it is stated that any of the artifacts only to be created if they provide value. It is a risk driven process using iterative incremental development. This means that the Use case with the highest risk associated with it will be focused on first. While reserving the right to go back to previous parts of the system to elaborate or change, based on feedback or new knowledge.

It describes how to gather the requirements of the system, how to analyze the requirements and use that analysis to design the code. It also has instructions for how to plan and manage releases.

There are four phases in Unified Process

- **Inception**
  - Smallest phase, used to explore viability of the product. Establish project scope, identify risk and come up with candidate architectures.
- **Elaboration**
  - Complete the use cases with the highest risk, while identifying the majority of the system requirements and compiling them into Use Cases.
- **Construction**
  - This is the largest phase in the project. The remaining part of the system is build.
- **Transition**
  - The system is deployed, feedback is gathered and users are trained. Corrections and changes are also implemented when needed.

[5] Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)
ISBN: 978-0131489066

Scrum[6]

Scrum is not specifically a software development tool, but rather a tool for planning your work, and measuring the progress of your work. It is however, often used by software development teams to structure their work, which naturally made us take it into consideration.

In scrum, you will find the following practices and concepts.

- **Product owner**
  - Has knowledge about the domain that is being worked on, and manages contact with stakeholders.
- **Sprint planning meeting**
  - Meeting where the entire team agrees on what needs to be done in the coming sprint.
- **Product backlog**
  - A list of work that needs to be done. Managed by the Product Owner.
- **Sprint backlog**
  - List of items that need to be completed in the current sprint.
- **Scrum master**
  - Ensures that everyone follows the scrum principles, facilitates interactions with people outside the team and removes obstacles that the team runs into.
- **Daily scrum meeting**
  - Short meeting where everyone informs the others of the progress they have made, and any potential issues they are having.
- **Sprint review**
  - The result of the sprint are shown to the Product Owner, and the stakeholders for approval.
- **Sprint retrospective**
  - The team talks about any potential issues, and how to improve their work.
- **Burn down/up chart**
  - Can be used to track overall project progress.

[6] Scrum guide - https://www.scrumalliance.org/why-scrum/scrum-guide

Kanban[7]

Kanban is a planning framework much like scrum, with a few differences. It focuses on a continuous workflow without iterations or sprints. It utilizes continuous delivery by default and embraces change at any point.

To achieve this, practitioners limit themselves to a set amount of work-in-progress items in each category of the kanban board. E.g Two items in the "code review" category, 4 items the "in progress" category.

- **Planning flexibility**
  - No fixed length iterations. Anyone,at any time, can pick an item off the top of the backlog.
- **Shortened cycle times**
  - By having overlapping skills, cycle times decrease.
- **Fewer bottlenecks**
  - Set a limit on how many items can be in each state. E.g only two items in code review.
- **Visual metrics**
  - Visualize the workflow using charts and diagrams, to better identify bottlenecks.
- **Continuous delivery**
  - With only a few work-in-progress items, the speed of continuously delivery increases.

Our way

Having considered the above software development methodologies, we decided we would pick the practices that best fit our work mentality and the project itself.

We decided against using any of the process and activities from Unified Process. Having roughly one year's worth of school work experience with unified process, we knew that it was too documentation oriented for our need. We had no use for the details of Use Cases which is the central point of the Unified Process. Nor did we need the artifacts that is the main product, next after the software solution it self.

We ended up using a mix of Scrum, Kanban and eXtreme Programming.

From Scrum, we used the concept of a Product Owner, but modified to better fit our needs. Each section of the system have their own experts. Therefore we had multiple "Product Owners" depending on what part of the system we needed information and feedback about. Which is not exactly the same tasks a scrum Product Owner manages. The rest of the normal Product Owner tasks, was taken care of by us.

For managing our workflow we utilized kanban. But again, not in its full form. We did not feel the need to visualize our workflow, since we are only two people. The availability and use of the digital kanban board "Trello" was enough, because of our two man team, and the ease at which we could communicate.

Following the continuous delivery practice of kanban was not relevant to us, since we were only creating prototypes. As these prototypes are not intended to be delivered for the client to implement, it did not make sense to follow this practice.

[7] The kanban methodology - https://www.atlassian.com/agile/kanban

We took the most inspiration from eXtreme Programming. The following practices were used:

- Whole team
- Pair programming
- Collective ownership
- Continuous integration
- Sustainable pace
- Open workspace
- Simple design
- Refactoring

With only two people sitting in the same room most of the time, and our clients sitting one floor below us, it was easy for us to follow the practices, *"whole team", "pair programming", "open workspace" and "collective ownership".*

Following the days we had to work individually, we would go over the work each of us had done to get up to speed.

"Continuous integration" came naturally as we required compilable code before check in to our version control system.

"Sustainable pace" was very important to us. We cannot make high quality work if we burn ourselves out.

"Simple design" and "refactoring": Both of us put personal pride in making quality code. To do so, we believe that these two principles are an important tool in achieving that.

As you can see from the list, some of the principles are missing from the above list.

- User Stories
- Short cycles (iteration plan, release plan)
- Acceptance tests
- Test-driven Development (TDD)
- Planning game

We have never used *"Test-driven development"*, which we found to be a big problem because it would slow our development speed down significantly, if we had to learn how to do this while developing this project. Morten had also read a research paper[8] regarding *"Test-driven development"*, which concluded:

> "TDD does not affect testing effort, software external quality, and developers' productivity."[9]

[8] An External Replication on the Effects of Test-driven Development Using a Multi-site Blind Analysis Approach

[9] Ibid, taken from the conclusion

Which we interpreted as; unless we purposely skip testing, it would not matter if we implemented *“Test-driven development” or not.*

Using *“Short cycles”* contradicts the purpose of us adopting kanban. Therefore it was not used.

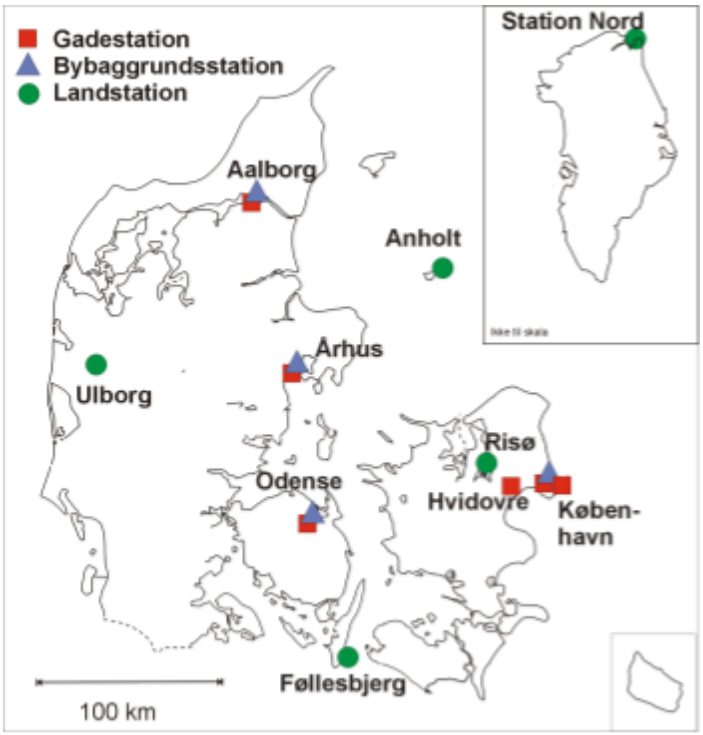
Since we did not use *“User stories”*, there was no basis for *“Acceptance Tests”* nor the *“Planning Game”.* From the beginning we analyzed the system, to map out the architecture, which we again broke down into components, and talked with the users to find out what kind of requirements they have.

After having done so, we removed the unnecessary parts, and added tasks to our kanban board to complete in order to implement the new architecture, we came up with.

Having regular conversations with the technicians and other users of the system, we modified and updated the requirements for each part in a continuous flow, following the principles of kanban.

Note: For lack of a better word, the phrase “user story”, “story” or a variation thereof will be used to refer to the tasks we created for the rest of the report.

Existing system



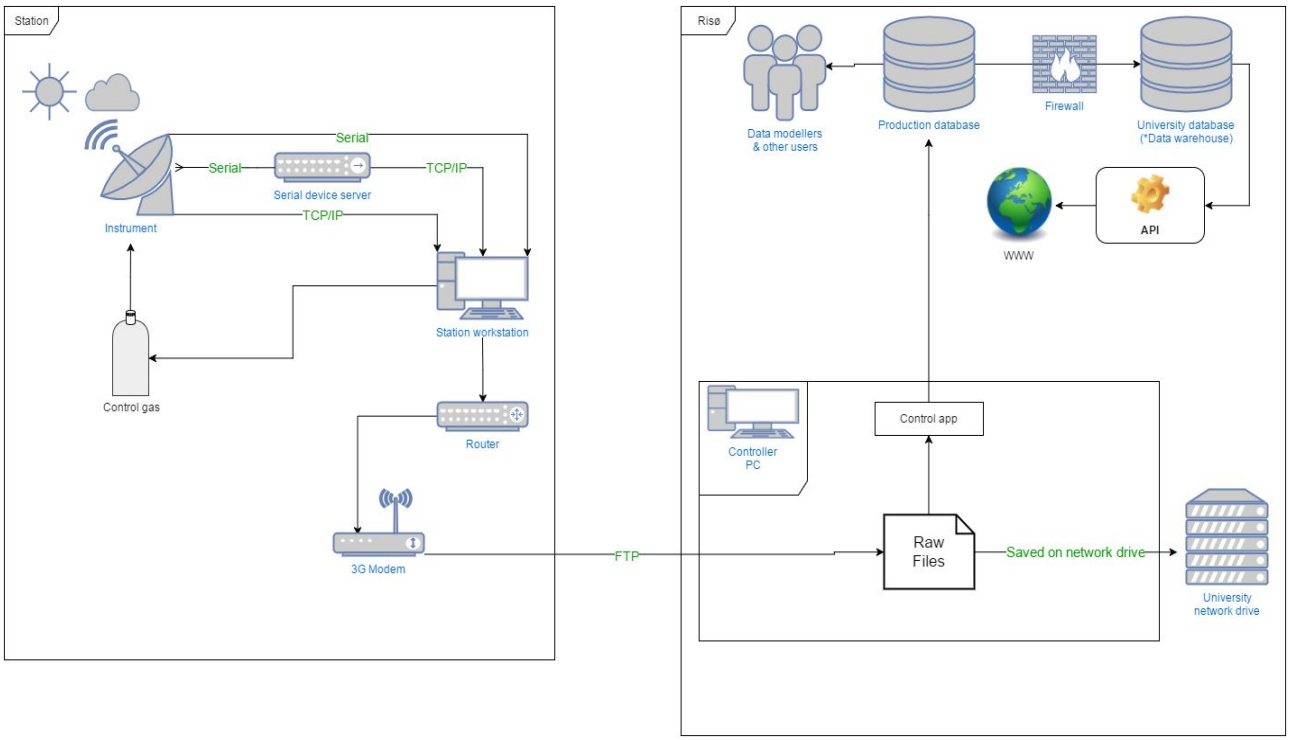
Instrument stations in Denmark (2016)

At Lárus' welcome meeting with Keld Mortensen, Keld made a presentation of the existing system. In the following I will try to sum up the information provided by him.

Currently there are 15 stations. These stations contain at least 3 instruments, however most contain more than that. The instruments are then connected to a local desktop computer, either directly through a serial cable, through router + serial device server, or router + ethernet cable. Bottles of gas are also connected to the computer, and hooked up to the instruments with tubes. These are used to calculate the inaccuracies that each instrument gets over time. This is called zero/span measurements.

The computer then collects the data from the instruments. Once an hour a desktop computer, located at the department itself, will then synchronously open up ftp connections to each station computer to collect the data.

At the local desktop computer, the data is saved into raw data files, and saved to disk. The raw data files will then be read, and the data in them will be added into the database. After which a store procedure will run, based on a predefined schedule, and make the necessary correction based upon the zero/span measurements, and add the corrected values to a table called Data_Warehouse. The procedure will also look at older data to see if any manual corrections has been made to the data.



Overview of the full existing system

Internet connectivity

Aside from the station in Greenland, all the stations are connected to the internet using a Dovado PRO AC [10]router, and an USB mobile network modem.

The connection is set up to use a mobile access point name (APN). The use of an APN means that the mobile modem is connecting directly to a server in Aarhus University's intranet.[11] This means that there are no immediate concerns about WAN access to the routers, as the AU firewall does not allow that.

Although the feature list of the router boasts of a SPI firewall[12], the menu does not offer any way to configure this, nor does the manual mention any firewall, except to inform that it will be off if the router is in Bridge Mode, and circumvented by UPnP.

The wireless feature of the router is not used by the instruments or the station desktop, so it has been disabled to prevent wireless attacks.

[10] http://www.dovado.com/images/PDF/DOVADO_PRO_AC_DATASHEET.pdf
[11] Difference between APN and VPN - http://smallbusiness.chron.com/difference-between-apn-vs-vpn-38815.html
[12] What is an SPI firewall - https://www.techwalla.com/articles/what-is-an-spi-firewall

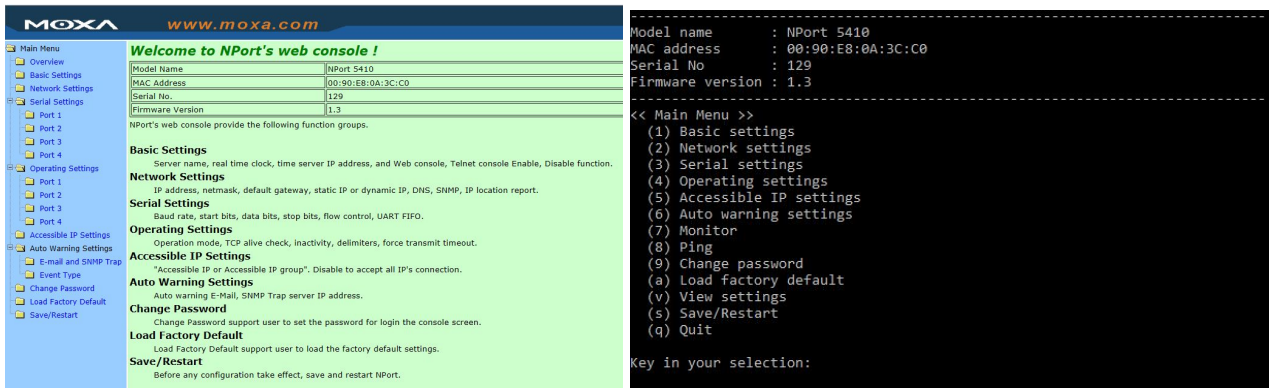
Serial (RS-232) and Serial device server

"In computing, a serial port is a serial communication interface through which information transfers in or out one bit at a time (in contrast to a parallel port). Throughout most of the history of personal computers, data was transferred through serial port"
- Wikipedia: https://en.wikipedia.org/wiki/Serial_port

Attached to the router you will find a Moxa NPort 5410. This device is known as a serial device server (Moxa).

A moxa is capable of interfacing with the older instruments that does not have a RJ45 connection - that is the normal well known network cable. It provides two interfaces, web and telnet based.

However, if the station desktop computer itself has a serial port, the instruments are attached directly to the computer.



Web and Telnet interface of the moxa

Measuring instruments

At each station we have measuring instruments that are by various manufacturers, have different measurement processes and measure different things. ie NO_2, NO_X, wind speed

Instruments include but are not limited to:

- LVS - Low volume sampler

  Measures the weight of particles of different sizes in the air.

- SM200

  Is being outphased, very old instrument that uses radioactive materials emitting beta radiation.

- Tapered Element Oscillating Microbalance
- Temperature

  Every station has a thermometer inside the station close to other instruments. In each station there is an air conditioning unit that regulates the temperature. The thermometer is used as a warning in case of malfunction of the air conditioning unit. Because the instruments will overheat, and break down.

- Meteorology instruments

  At each station they have a few meteorology instruments that measure wind speed, wind direction and humidity.

- Teledyne API T100U, T200U, T300U

  These are the newest instruments that measure gas concentrations of SO_2 (sulfur dioxide), (NOx/NO (nitric oxide and nitrogen dioxide), O_3 (Ozone), and CO (carbon monoxide). They are replacing older models from the same manufacturer. These instruments are the most common on the stations. They have a lot of new features, like accepting commands through TCP/IP connection and caching of data.

Look in the appendix for a full list of instruments and their capabilities.

Data gathering

The local station computer is regularly communicating with instruments located at the station, to gather the various data. Once an hour, a computer located at the campus will synchronously open ftp connections to all of the station computers to gather the raw data files, so it can start processing them.

Database

The departmentis using a Microsoft SQL Server to host their database. The server itself is running on a desktop computer located in the office of Keld and Rune - the IT employee and database administrator of the department. Aarhus University has provided a database in their datacenter, which they maintain and backup, and offer a high Service Level Agreement. However Keld and Rune have had concerns regarding access rights to the server, and have thus decided to only use it as a backup of their locally run database.

The computer at the department of Environmental Science will run through all the files that it collects from the stations, interpret the content and add it to the appropriate tables in the database. After the data has been added to the database, a schedule will start the database procedures that make corrections to the measurements, based upon the result of the span and zero measurements on the instruments.

When the procedures are done correcting, a schedule will run a script to synchronize some of the data in the database with the database hosted and maintained by Aarhus University's IT department. This is done to provide up-to-date data for the API and the website, that the department are providing as part of their duties to inform the public.

Once every 24 hours the entire database are synchronized with the one hosted at Aarhus University IT, which in turn makes regular backups of the data.

System analysis

We approached the analysis by identifying natures/causes of complexity and then extracting the resulting issues. Subsequently we looked at the system's strong points. In the analysis, we will use the following legend:

- ⁝ - Nature / Cause of complexity
- ➢ - Resulting issue
- ✔ - Existing system's strong point

- ⁝ Out of 15 stations there are 11 combinations of the required measurements that require different instruments.
  - ➢ The installation instructions for new stations are provided in a centralized word document that is 36 pages long, a lot of it is not relevant to all stations. This is very difficult to keep up to date.
  - ➢ The installation instructions include a lot of software and hardware installations that are described step by step in danish by a department employee. This kind of personal narrative by an employee can result in misunderstandings and/or errors in the steps. As opposed to a more technical standardized document.
  - ➢ Stations are distributed over the country so to maintain them multiple technicians are needed. Their salary and transportation cost is needed but is a big expense factor.
  - ➢ Repair and replacement requires a person from the department to drive to the station.
- ⁝ The instruments vary greatly in output, interface and work processes.
  - ➢ The instrument measure very different inputs (e.g. gas concentration, particles, meteorological data (wind speed, wind direction), temperature...). That includes a lot of different algorithms for each data set that are hard to maintain. The system executes these algorithms at the controller server who already has a lot of responsibilities.
  - ➢ The gas concentration measuring devices has 2 phases:
    - Measurements phase: Does measurements over the course of a minute and averages the readings to represent a period of 5 minutes.
    - Calibration phase. Depending on the gas being measured the sources for these readings can either come from an internal tube storing the air or an external tube controlled through the instrument or the station computer:
      - Zero (nul): takes in air with zero concentration of the measured gas. The offset is then accounted for when storing the normal measurements.
      - Span: takes in air with a very dense concentration of the measured gas. The offset is then accounted for when storing the normal measurements.
- ⁝ The danish government regularly updates requirements of how the measurements are gathered and how they are delivered. In recent years the European Union also has legislation in place to make sure that measurement procedures from each member state are able to meet a certain

level of quality, and conform to one another, since they are then used for global environmental research.

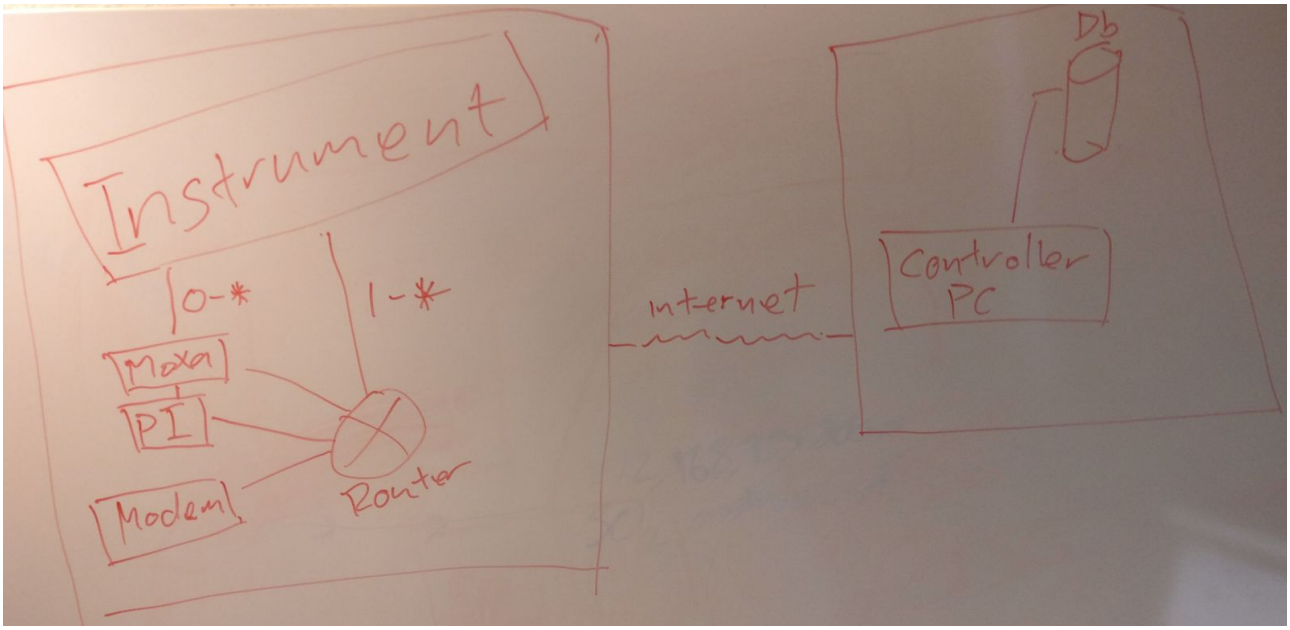
- ➢ Each revision of the contract between department of Environmental Science and the danish government have introduced changes. These can affect how our system should run. The existing system has difficulty in implementing new changes issued by EU about changing the zero/span phase duration due to them being specified in a very old script file using a deprecated language with discontinued support. One of the scripts can be seen in the appendix.

- ❗ The production database is old and has been extended upon multiple times, and with the introduction of a centralized university database the roles of these two databases and their interactions have not been clearly defined.
  - ➢ It takes up to 1 hour and 35 minutes to get the data to the production database and then up to 20 extra minutes to move selected data to the Aarhus University IT database.
  - ➢ The relationship between the production database and the Aarhus University IT database was instigated by the need to retain liberal user access. This liberal access introduces some security issues.
  - ✔ The retention of the production database on a server directly controlled by the department means lesser restrictions in place regarding access and management.
  - ➢ 4 aspects for db issues
    - The schema has been extended upon for many years without refactoring.
    - It doesn't follow best-practices. A lot of normalization needed.
    - Database best-practices have changed since the database was created. Specifically, the normal forms was defined by Edgar Codd in 1971[13]
    - A lot of redundant/legacy data that can be done away with.
- ❗ Municipalities have opposed moving or setting up new stations in city centres due to aesthetic considerations and the size of them. The department has started work on limiting the size of new stations.
  - ➢ It results in a lot of downtime of stations working with the municipalities to find a suitable place for a station.
  - ➢ Dealing with the bureaucracy involved in getting the stations where they are needed, takes up time that could be better spent elsewhere.
- ✔ There have not been any system-wide breakdowns.
- ✔ Data modellers are pleased with the access to the data they need.
- ✔ The system has documentation for a person with rudimentary computer knowledge to understand how to set up all components in the stations, maintain them and how they work.
- ✔ Has not needed a large team to implement and maintain.

[13] Reference to book/publication - https://en.wikipedia.org/wiki/First_normal_form#cite_note-2

Updated system

After having worked on the system for several weeks, we believe that we have created a good suggestion for how to build an alternative data gathering system. The department are moving away from having multiple brands of instruments and slowly switching over to using teledyne instruments. This means that our suggested solution will become more attractive.



Overview of proposed architecture

Value proposition

After we analyzed the existing system we wrote the value proposition. Here we discuss what we could improve upon with the updated system and use that as a reference to realize what we can contribute and how to prioritize the work.

One thing that we found would contribute to all 4 points is to put high emphasis on code quality. The IT people working at the department of Environmental Science have done a great job in building the existing system and maintaining it, and we can hopefully make it easier for them doing both with the existing system, by writing code that is easy for them to scale, performs well, is easily reusable and maintainable.

1. Scalability
    a. **Fewer software components, more standardized station layout, fewer hardware components and fewer drivers:** Setting up new stations or changing the setup of existing stations is easier. Fewer man hours that go into that and the risk of errors during setup are lowered.
    b. **Utilizing well established and current technologies:** Support in the form of documentation and updates can assist us if any problems arise.
2. Performance
    a. **Create a new persistency system for instruments:** Some instruments does not have the ability to persist the data in their buffer. This means that in case of equipment or power failure, the data would be lost. With the new system, it would pose a problem not implementing persistence in a new way. Because of the removed station computer.
    b. **Time to database:** Decrease the time it takes to get measurements from the instruments to the production database, by writing directly into the database before creating raw data files.
    c. **Make the controller program asynchronous:** Now the controller is looping through each station and requesting data. By making it asynchronous we can make the data acquisition faster.
    d. **Increased accuracy:** By redefining the format and aggregation of the data between the instrument and database we can increase resolution of the measurements. This can be used to more accurately remove single erroneous measurements and provides the correction algorithm with more data to base its corrections on.
3. Usability
    a. **Utilizing web applications:** Easy to use graphical interface where we can easily create a front-end application both for looking at the status and directly control of one or more stations. Visual graphs in web applications are very powerful and customizable. This allows users to access currently local applications to be accessed remotely.

4. Maintainability
    a. **Utilizing a single board computer:** Hardware that is cheaper and easier for quick replacement in case of hardware failure.

System description

We have created an architecture that we believe is flexible, and easily expandable. The system has fewer points of maintenance than the original system, because it is comprised of fewer parts. At the same time, the important parts of the system is run locally on campus, while still providing access to them while on the road, e.g in case they need to be accessed while at a station. In case of breakdowns on the stations themselves, it will be quick to get back up and running with preconfigured routers and preconfigured images.

In the new system, you will find:

- Station
    - Router using an USB modem.
    - Instruments.
    - Moxa, in case of instruments that does not have ethernet capabilities.
    - Raspberry PI[14], in case of instruments without the ability to store their buffer.
- Main server
    - Control program.
    - Logging/control website.

Station

We have seen no reason to change the way that the stations are connected to Aarhus University's intranet. So the network connectivity should still be delivered through a router using a usb modem. There are however some changes necessary to the set up, which will be mentioned in one of the following sections.

The instruments themselves also remain unchanged, seeing as they are not in our scope of modifications. However, the way they are set up at the station has changed.

Before they were all connected to a local desktop computer which would handle the gathering of data from the instruments. Either through serial, moxa or ethernet cable. The data is placed in files, which are collected once an hour through an FTP connection, initiated by a server at the department.

In the new system, we suggest removing the desktop computer and put the responsibility elsewhere, which I will elaborate on shortly.

Keld initially asked us to focus on the teledyne instrument, which is where we put our main effort with prototyping and proof of concept. However, there are other instruments in the system, so we still made a plan on how to handle those.

In case of a station that houses instruments, that does not have an ethernet cable, the station will naturally maintain their moxa. The moxa it self can handle 4 instruments through serial and also hosts a web server to configure said instruments.

Some of the instruments lack the ability to persist their measurements in case of power outage or instrument failure.

[14] What is a Raspberry PI - https://pimylifeup.com/what-is-raspberry-pi/

> *"In the virtual network computing (VNC) system, server machines supply not only applications and data but also an entire desktop environment that can be accessed from any Internet-connected machine using a simple software NC. Whenever and wherever a VNC desktop is accessed, its state and configuration (right down to the position of the cursor) are exactly the same as when it was last accessed."*[15]

On those stations there should be a Raspberry PI running raspbian. This PI, would be running a program to constantly gather data from the instruments that cannot persist data, which would then be stored locally in case of failure. The data could then be retrieved through FTP, and added to the main database at a later point. Running linux on the PI would also enable the department to use VNC, and access the PI using remote desktop in case the FTP program failed.
This could obviously be achieved through the terminal too, however we feel it would be more user friendly, and provide a lower learning curve to just offer remote desktop.
Keeping in mind that the department has moved to the Microsoft platform, we would also suggest using the new .NET core framework in the PI. This would make it possible to write C# code to execute on the PI.

Main server

On the main server, there will be a control program running in the form of a console application. This program will be doing multiple things at the same time. The most important of these tasks is talking with all the instruments.

It will, with regular intervals, spin up multiple threads and establish a TCP/IP connection with all of the instruments, collect the data they have gathered since last connection and insert that data into the database.

At the same time, it will also host a TCP/IP server, to which it is possible to send commands defined in a protocol of our own making. This allows for control of the console application, and basic create, read, update and delete operations of stations and instruments.

[15] Virtual Network Computing - https://www.cl.cam.ac.uk/research/dtg/attarchive/pub/docs/att/tr.98.1.pdf

The protocol is defined as the following:

Custom protocol
Format:

command parameters\r\n
payload\r\n

List of commands:
add <station|instrument> <payload>
delete <station|instrument> <payload>
update <station|instrument> <payload>
start
stop

Payload:
The payload is a json serialized object, without line breaks.
The payload is required for add and update.

Example:
update instrument 1234\r\n
{"Id":1234,"InstrumentType":0}\r\n

Evidently, the protocol is simple, and easy to work with. In case someone would wish to build their own program to manage the control program, it should be a small task to complete.

The main server will also be hosting a website using IIS - Internet Information Services - this website will centralize the logging into one database. The logging are required to keep track of events that occurred at a station. The website provides an easier overview of all the stations, and the events that has occurred there. This task is currently done on each station using the desktop computer on location with the current system.

At the same time, the website also serves as the frontend for the control program. Using an implementation of the proprietary communication protocol, it will communicate events to the control program that is relevant to it. E.g. creating a new station, or modifying an existing one. This is done after having applied the change to the database.

Although not implemented in the prototype we have made, the website will also be able to detect if the control program is running or not. And in case that it is not running, it will be able to instantiate a Process, and start the control program. This should mean that it will not be necessary to remote into, or otherwise actively use the computer hosting the website/control program at any point.

Requirements

The requirements are quite low for the new system to run. One computer running on the local network to host the website, and the control program. Aside from that a properly configured router and possibly one raspberry pi on some of the stations.

How to set it up

At the station, one would need to install instruments and the supporting equipment as usual. The instruments will then need to be connected to the router. In case of instruments that do not support ethernet connectivity, a moxa would need to be installed. In case of instruments that are unable to persist the data in their buffers, a raspberry pi would also need to be installed.

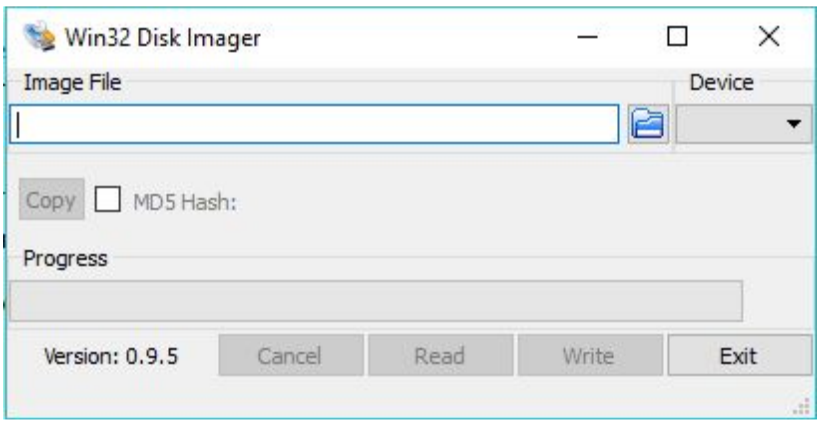
After that is in place, the control program and website needs to be placed on a server that is only reachable when connected to Aarhus University's intranet. Either from AU's premises, or through VPN, if located elsewhere.

This is because the control program is hosting a TCP/IP server, which does not implement authentication or authorization of any kind. The server itself should implement a firewall, that only allows TCP packages on ports 80, 443, and both TCP and UDP on port 3389. Since the server would be physically close to the people responsible for it, there should be no need to allow wake-on-lan. However, if that is a feature that is desired, one would also need to open port 9 for the UDP protocol.

This set up will ensure that it is not possible to communicate with the control program, except using the website. The website will also enforce the use of https, while enjoying the protection of Aarhus University's corporate grade firewall.

The department asked us to focus mainly on the instruments of the teledyne brand. They have a lot of functionality that some of the other instruments do not have. However, if our new system were to support all of the instruments, it would also be necessary to set up a moxa and a Raspberry PI. I will only be going into the detail of our proposed Raspberry PI set up, since configuring the moxa is not going to be different than what the department usually does.

To install the operating system on the Raspberry PI, one would need to use a program that can flash images to SD cards. The one that we used is called Win32DiskImager and can be downloaded from sourceforge[16].



[16] Win32DiskImager - https://sourceforge.net/projects/win32diskimager/

This program is also able to create an image from an SD card, which is a helpful feature if you want to be able to deploy fast.

Raspberry PI installation instructions:

1) Go to: https://www.raspberrypi.org/downloads/raspbian/
    a) Download the latest image, which is not the LITE version.
2) Extract the image from the zip file.
3) Use Win32DiskImager to flash the image to an SD card.
4) Insert SD card in a Raspberry PI and boot it.
5) In the terminal, run the following command:
    sudo apt-get update && sudo apt-get upgrade

After having installed, updated and upgraded the operating system the next step would be to put the collection program on the PI, and set it up to run automatically[17] on boot and enable VNC[18].

Since we were told to focus on the teledyne instruments, we have not prototyped this program. However, the code would be near identical to what we have already written for the InstrumentCommunicator, except it would be implemented in .NET core, which would allow it to run on linux.

The data itself should be placed on a USB thumb drive. This allows for larger amounts of data, and it also means that in case of redeployment of a Raspberry PI, the technician does not have to worry about extracting data before swapping out the SD card with another one. If the data was located on the SD card, it might be accidentally deleted.

Once these steps are complete, an image of the operating system should be created in its current form using Win32DiskImager. This would enable rapid deployment in the case of hardware failure, and would only necessitate a few station specific configurations, before deployment.

Router configuration

To be able to talk to the instruments some port forwarding is required. However, it seems that the dovado routers that the department are currently using with that feature, contains a bug with its port forwarding. It is only able to do a one-to-one forward of the port, as in forwarding the external port 3000 only to the internal port 3000. However, this setup requires that ports are forwarded differently. I.e. port 7000 to port 3000.

Extensive testing was done with the routers provided by the department , however it was just not possible to forward like that, even after flashing the newest firmware provided by dovado. Morten then brought his spare router from home, to show to Keld that it is possible to do so normally.

The router D-Link DIR-855 had no problems forwarding the ports like we desired, despite using a different name for it. D-link has provided an emulator for the DIR-855[19], where you can play around with, and examine the menus and capabilities of the router.

[17] Raspbian: Run a Program at Startup - http://www.mikeslab.net/?p=176
[18] Raspberry PI VNC documentation - https://www.raspberrypi.org/documentation/remote-access/vnc/
[19] Router emulator - http://www.support.dlink.com/emulators/dir855/Virtual_Server.html

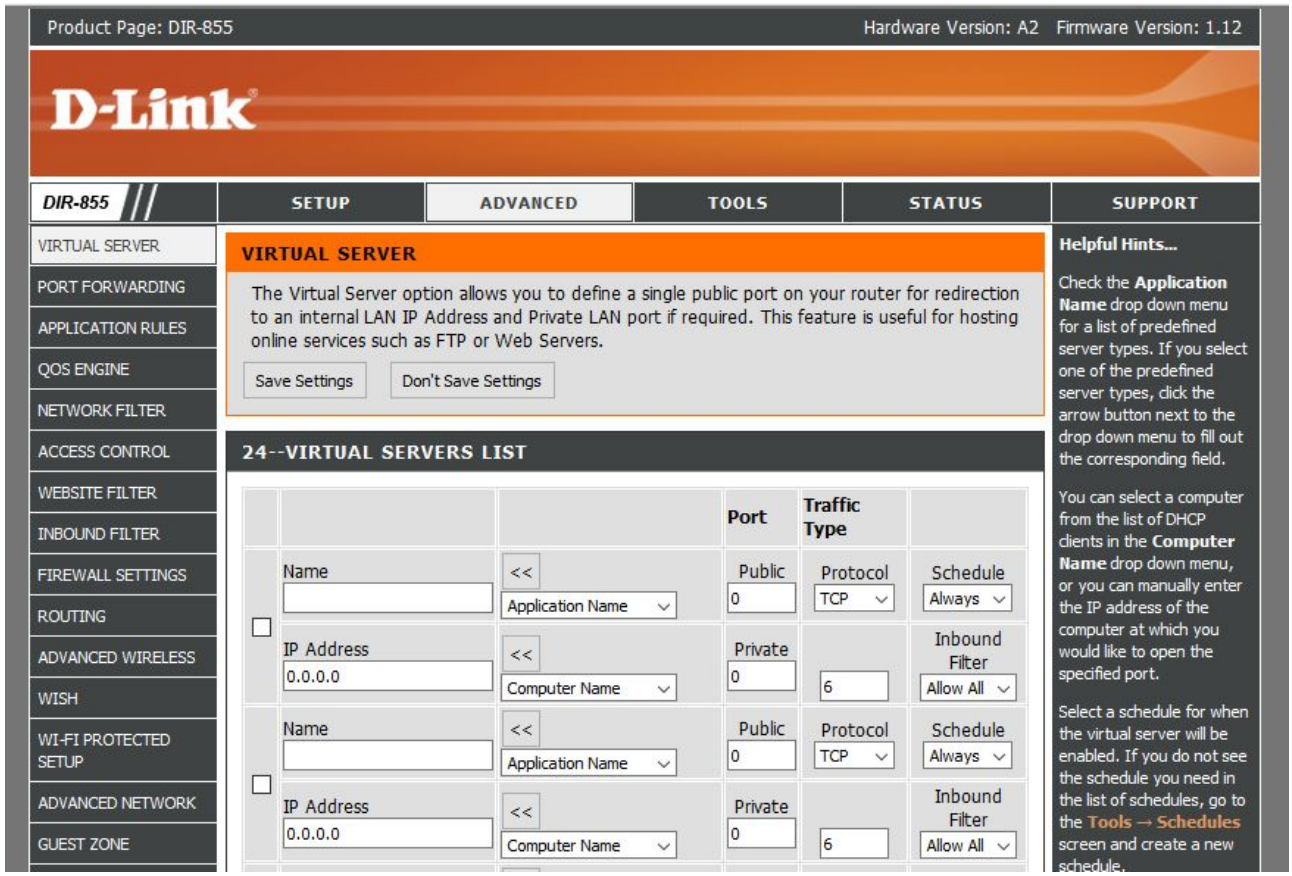


Image of the menu used to configure port forwarding.

In essence, this means that the department will have to replace the routers, or inform dovado of the bug and wait for them to fix it in their firmware, which might take a while.

The alternative is not much different. It would require the department to buy new routers like in the replacement scenario. But instead of replacing the dovado routers, the new routers would be put in a DMZ behind the dovado router, which would change the Dovado's role to that of a modem.

The argument for this set-up could be that the new routers do not posses the ability to use USB modems and APN's like the dovado do. This ability is an essential requirement of the router.

Our recommendation is to inform dovado of the bug that we have found. While waiting for them to fix the issue, the set up of the new system should continue. When the set-up reaches the point where it is just the router part missing, they should be replaced, in case Dovado have not fixed the bug yet.

Even if dovado fixes the bug within a reasonable amount of time, we would still recommend that the routers are exchanged for a different and well known brand. They only have a few[20] routers under their belt, and the features they claim in the documentation do not seem to exist in the menus. Combined with the inability to forward ports like we want - suffice it to say, they have not inspired trust in their product. According to conversations with some of the employees, the smaller router which they used at first, have also proved to be unreliable in maintaining internet connectivity.

[20] Dovado products - http://www.dovado.com/en/products

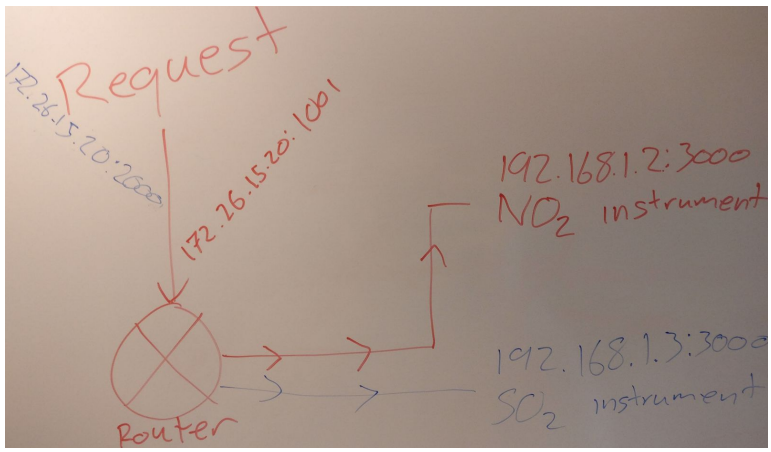
There is also the issue of the dovado router only having 4 ports, which limit the stations to 4 instruments or 4 moxa's. Or a combination thereof.

It would be a good idea to future proof the station by buying a router with several network ports. E.g Asus RT-AC88U, which has 8 network ports and supports USB modems.

Port Ranges

To keep track of what is what, we also propose the following.

Each type of instrument should belong to a specific range. E.g the external port for all of the NO_2 instruments should be in the 1000-1999 range, SO_2 should be in 2000-2999. This way it would be very easy to differentiate what kind of measurement is being gathered. The reason for this is that, according to Keld, there is no way to see what type of measurement you are actually getting, when requesting data from the teledyne instruments. And the added benefit of this would also be that the configuration of the instruments and routers can be generic.



The same goes for the moxas web servers. They should be in a specific external port range like the instruments. E.g. 8000-8999

These port ranges are not implemented in any of our prototypes. However, in the event that this new system was to be implemented, we would highly recommend doing this.

Issues

The proposed system does of course have its weak points. The most important one is that it has a single point of failure. The main server.

In case of the server crashing, or otherwise becoming unavailable, no data will be gathered from the stations. However, despite this we still believe that it is an improvement over the current system, with a computer placed at each station.

In the current system, if a computer goes offline for some reason - someone will have to travel to the station and attempt to fix the problem. Which can happen for all 15 stations.
But with the new system, the technicians will just have to go in the next room to fix whatever problem occurred. The data gathered meanwhile will be stored in the instruments and the Raspberry PI for collection as soon as the collection system comes back online.

Aside from this issue, the other known issues, such as lack of persistence in instruments, heat issues, connectivity issues to the stations, are also present in the current system, and will therefore not be mentioned here in relation to issues with the new system.

Prototypes for the new system

The product represents the whole redesigned data gathering system that is to potentially replace the existing data gathering system. The system includes a lot of sub-systems which we will refer to as sub-products. The scope of this project is not to replace the whole system, but to provide the client with as much information as possible, to make the replacement of the existing system as easy for them as possible.

At the start of each week we will be looking at the current sub-products we have created, and evaluate which sub-product we should create or update to a higher version.

Our approach to the project is to create incremental prototypes of sub-products. They act as a proof-of-concept. We try to make the best core architectural design we can, so we can present the client with a good base to build upon, should they chose to implement our suggested system.

The prototype method we will be using is called Throwaway Prototyping[21]. It is a rapid form of iterated prototyping where we re-define requirements of the product at the end of each week.

[21] Throw away prototyping - https://en.wikipedia.org/wiki/Software_prototyping#Throwaway_prototyping

Steps of Throwaway prototyping:

1. Write preliminary requirements.
2. Design the prototype.
3. User experiences/uses the prototype, specifies new requirements.
4. Repeat steps 1-3 if necessary.
5. Write the final requirements.

The process of writing the preliminary requirements consists of going back to our value proposition and the discussions we have had with the users. Then we write up all the requirements we wish to achieve, but which might not necessarily end up in that version of the prototype. We will be focusing on the functional requirements due to their relatability to the existing system. We might draw upon non-functional requirements, but only for prototypes we consider of high complexity.

During the design we will draw upon the software methodologies listed in our Methodologies chapter, adopting Extreme Programming especially, as it helps us keep the design of the prototype minimal and simple, so we don't waste any time creating something that holds no value.

At the end of the week we might schedule an informal meeting with one or more users, where we ask for their input. We have a casual conversation about the product and ask them their general thoughts on whether the requirements have been met. The success criteria will be highly influenced by the capabilities of the existing system. The users are so few that we do not find the time spent on formulating a questionnaire to be worth it. The users are also primarily used to a more informal mode of communication within the department, so we expect we can engage them better in casual conversation, thus getting more input during a discussion, than through a written questionnaire.

When writing the final requirements the sub-product might not have all of the requirements declared in preliminary requirements. We will list all of the requirements we have finalized, and all of the ones that have not been met. Those requirements are intended to be implemented by the client after the end of the dissertation, in case they wish to implement the system. Any requirement that we do not finalize are considered to be of low complexity and should not create any problems for the client in terms of implementation.

For the git, it was decided we would be using a component focused structure sub-divided by features. With the master branch acting as a release branch.

1st week (28th nov - 2nd dec)

The technicians and other technical personnel visiting the stations are using a program that is running locally on each station computer, with no centralized storage of data. This program is used for logging the actions taken at the stations, ie. swapping a bottle of gas with a fresh one.

Seeing as our intention is to remove the station computer, it would be necessary to find an alternate solution. Creating a website to take care of the responsibilities of the aforementioned program seemed like a good idea (see Updated System -> Value Proposition 3.a), as it would also centralize the data generated from the technician's visits.

Based on comments and conversations with various technicians, both passing in the hallways of the department, and at meetings, we decided that this would provide a high amount of business value, especially considering the complexity of the user story.

For that reason we decided to tackle the station logging web site first.

Using the default ASP.NET template with user accounts enabled, it was very quick and easy to implement the CRUD operations and user rights' management on both stations and logs using Entity Framework[22] and ASP.NET Identity[23].

Controller web site v1

Requirements/Features

- ✓ Create log for stations, containing text, initials and date/time.
- ✓ Create stations, containing name and referencing logs.

New requirements/features:

- Manage span gas.
  - Which type of gas.
  - Bottle number.
  - Measured result.
  - Log event.
- Add monitor.
  - Which type of monitor.
  - Monitor number.
- Task list.
  - Create a list of tasks that need to be completed on the monthly "station trip".

[22] What is entity framework - https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx#What%20is%20Entity%20Framework
[23] Introduction to ASP.NET Identity - https://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity

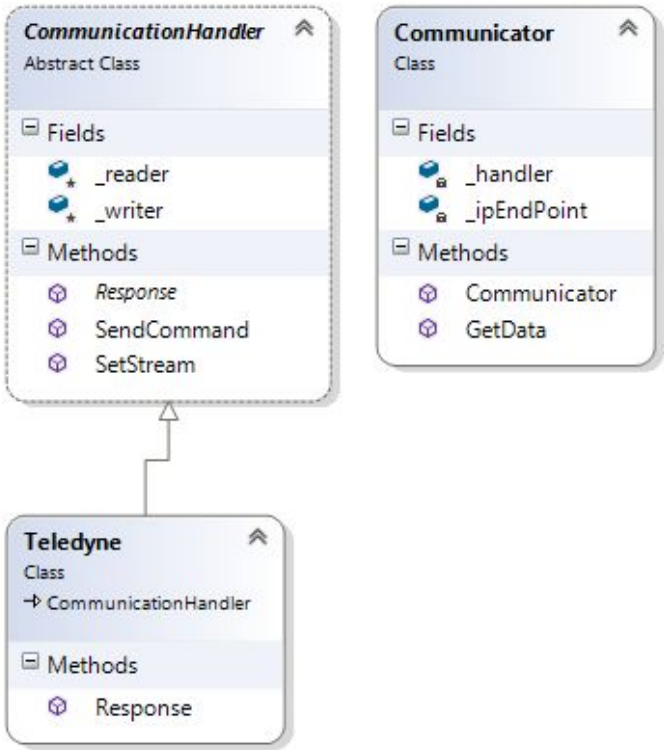
2nd week (5th dec - 9th dec)

After having created the logging website, we decided that it would be best to start with the instruments next, because of the data flow starting there. Relying on the dependency inversion principle[24] and the strategy pattern[25], this would allow us to either consume mock data or data from an actual instrument. This makes integration testing very quick, while also making it easier to write unit tests for the program.

We agreed to put the responsibility of interfacing with the instruments into a class library, which would decouple the components from each other.

Lárus had worked with .NET Core web applications before, and advocated creating a .NET Core Class Library that targets only the .NET standard library, which means it would be usable on linux operating systems, like the Raspberry PI. There was a concern about the nature of the architecture[26] and whether it was truly cross-platform, but upon closer inspection of the .NET core system architecture we were assured that that was the case, and we could see examples of this in action.

Instrument interface class library v1



[24] Agile Principles, Patterns, and Practices in C#, chapter 1
ISBN: 978-0131857254
[25] Strategy pattern - http://www.gofpatterns.com/behavioral-design-patterns/behavioral-patterns/strategy-pattern.php
[26] .NET Architectural Components - https://docs.microsoft.com/en-us/dotnet/articles/standard/components

Requirements/Features

- ✓ Validate the format of the address:
  - ✓ Using the Microsoft defined class IpEndPoint, testing this is redundant, as it is handled by the class.
- ✓ Connect to instruments using TCP/IP:
  - ✓ Establishes a TCP/IP connection with a TCP/IP server.
- ✓ Connect to Moxa adapter using TCP/IP:
  - ✓ Establishes a TCP/IP connection with a TCP/IP server.
- ✓ Send commands:
  - ✓ Sends data to the instrument through a TCP/IP network stream.
- ✓ Return the results:
  - ✓ Retrieves data from the instrument through a TCP/IP network stream.

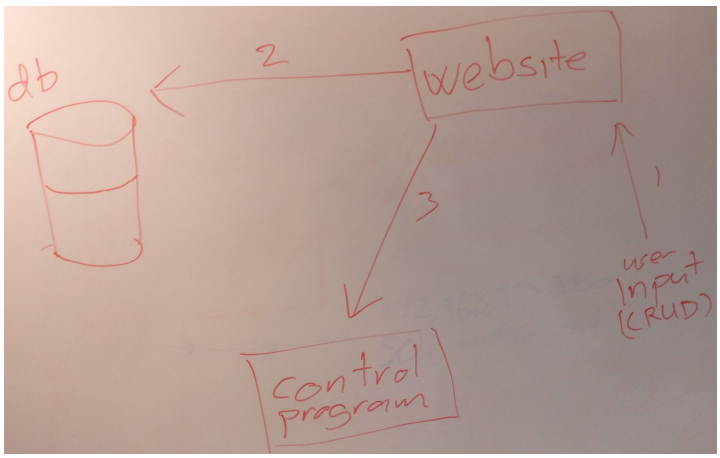
Error handling

- ✓ Prevent user input that is invalid, both in initializers and in any object that it is passed to.
- ✓ Handle not getting any response:
  - ✓ Handles any exceptions resulting from unavailable instruments.
- ✓ Logging errors
  - ✓ Used the Microsoft defined class Trace to handle logging.

3rd week (12th dec - 16th dec)

Having created the means to communicate with the instruments, the next natural step was to handle the communication. We had a very short discussion about what type of application it would be and decided on doing a WPF (Windows Presentation Foundation[27]), even though it was the first prototype. The decision was made because we envisioned the client running it on their servers and needing to manipulate the list of stations and instruments from within it.

We quickly realized that a WPF application was not necessary. The job could easily be handled by having a console application that would communicate with the database to retrieve the current stations and instruments. The program could then be controlled from the logging website[28], which would decrease the amount of programs/interfaces that the users would need to know about.



When discussing how this shift in responsibilities would affect the controller program we started drawing out on paper a system diagram of how we envisioned the controller program; its major components, and how those components communicated with different sub-products. In accordance to our expectations, working on the controller program early on, made it easier for us to understand its interactions with other sub-products, and helped shape our design decisions.

We created a class library that would hold entity models (representing the actual business data) and hold DTOs (Data Transfer Object[29]), which would hold data when passing it along different components. The models were moved out of the controller site, to allow other projects in our solution to consume them.

Instrument Interface Class Library v2

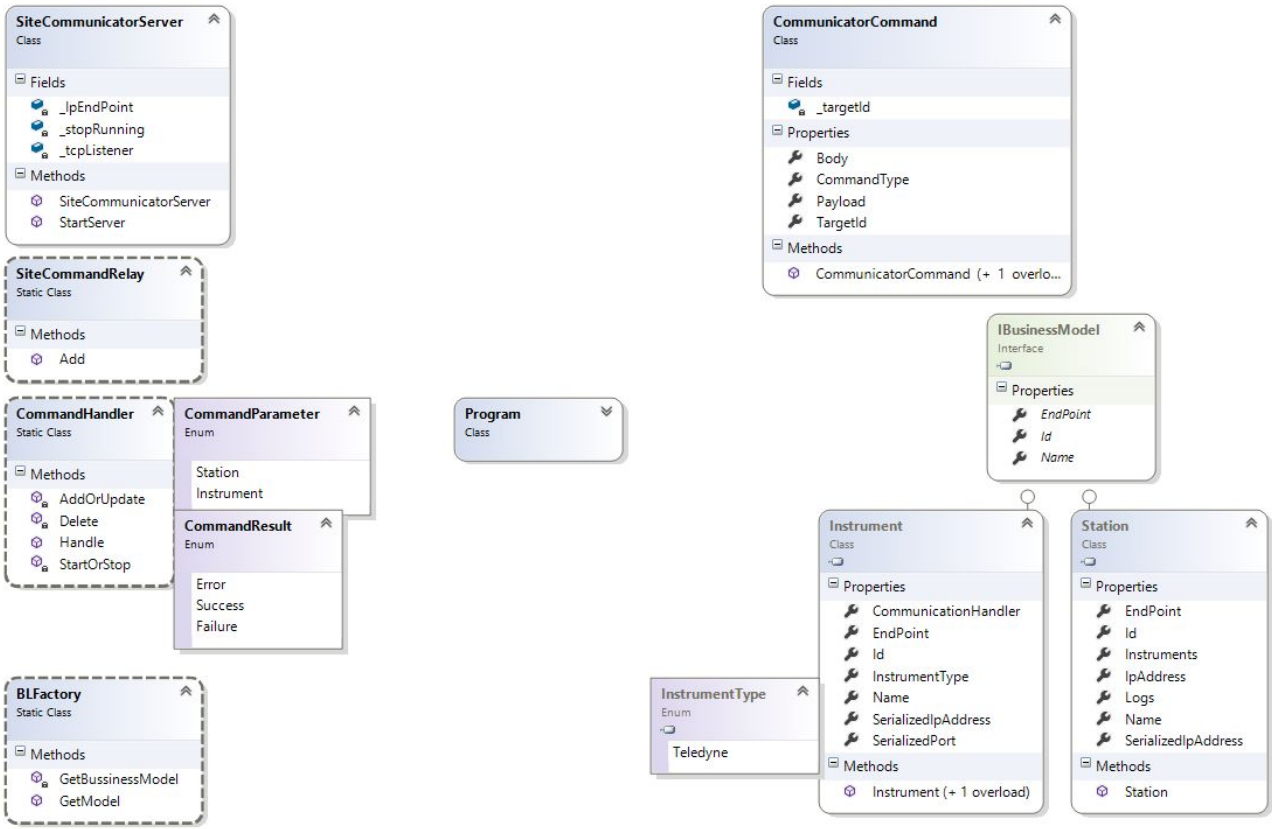
This version simply entailed re-creating the Instrument Interface Class Library without targeting .NET Core, because referencing a .NET Core class library within .NET Frameworks is problematic as the solution and project formats are incompatible between .NET Frameworks and .NET Core. There are ways around this, but they did not seem worth going into at this point, since the client had not asked for cross-platform functionality .The framework for working with these 2 project formats will in all likelihood see updates. In the meantime this will hopefully make the process a lot easier.

[27] Introduction to WPF - https://msdn.microsoft.com/en-us/library/mt149842(v=vs.110).aspx
[28] From now on, referred to as controller site
[29] Data Transfer Object - https://en.wikipedia.org/wiki/Data_transfer_object

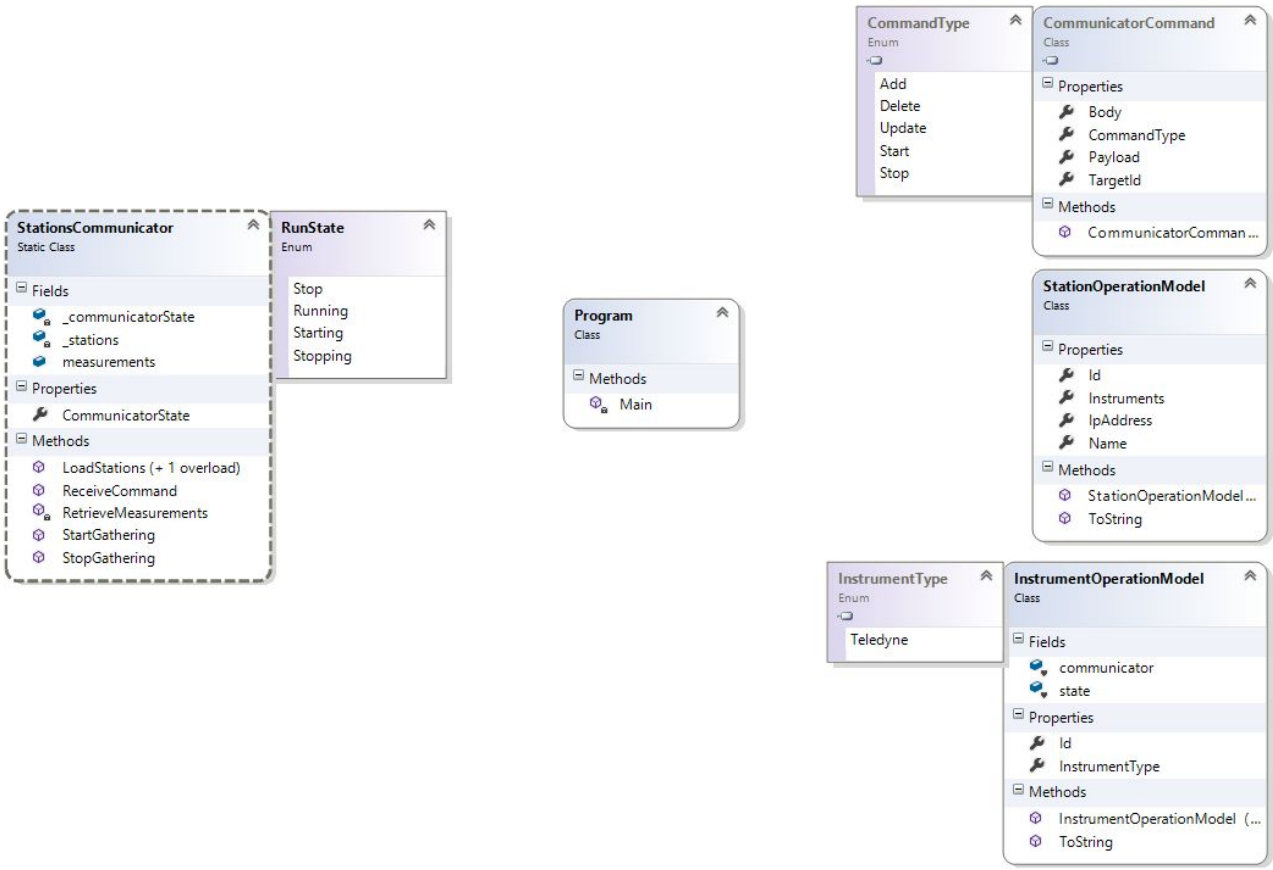
Controller program v1

Components

- Runner: Handles the running of the program. Initialization, termination and execution order.
- Database handler: Handles communication with the database using Entity/Stored procedures to and from the control program.
- Controller site handler: Handles communication to and from the controller web site using TCP/IP connection with our own defined protocol.



- Station communicator: Handles communication to and from the stations, and their active instruments, using our own Instrument Interface Class Library.



- Input manager: Handles keyboard input by the user.

Requirements/Features

- ✓ Create a protocol that we can use between cross-component communication.
- Get station and instrument data from MS SQL database to Station communicator:
  - Create a connection to the database.
  - CRUD on the business data stored in the database (to be split later).
  - Retrieve station and instrument data.
  - Send measurement data.
- Pass commands from the controller site to Station communicator.
- Communicate with the instrument interface library:
  - ✓ Have the communicator receive commands from other components (database, site).
  - ✓ Have the communicator get measurements from station library.
  - Have the communicator handle these communications with stations asynchronously.

- Logging events:
    - ✓ Log status events.
    - ✓ Log errors:
        - ✓ Log to console.

Error handling

- ✓ Log errors to console.
- ✓ Log errors to file.
- Handle miscommuncation with Database.
- ✓ Handle controller site ⇔ controller program unintended commands.
- Handle asynchroneous conflicts (thread safety).
- Handle exceptions thrown from the instrument interface library:
    - Connection errors.

Retrospective

We had unexpected issues with our version control which resulted in more time being spent on dealing with merge conflicts. To simplify the process we branched out from master into component branches. Any time there was an update to a component we would go to that branch, implement the update, run the unit tests, merge into master, resolve merge conflicts that might arise and run the unit tests again. Only re-basing and merging to and from master branch was permitted unless there was an explicit need to do otherwise. This helped us keep track of when updates had been made, and ensured that the latest code could always be located in the same place.

There were a lot of heated arguments regarding design decisions. We discussed them, with both parties arguing their case, but in the end the person responsible for the code had the final say.

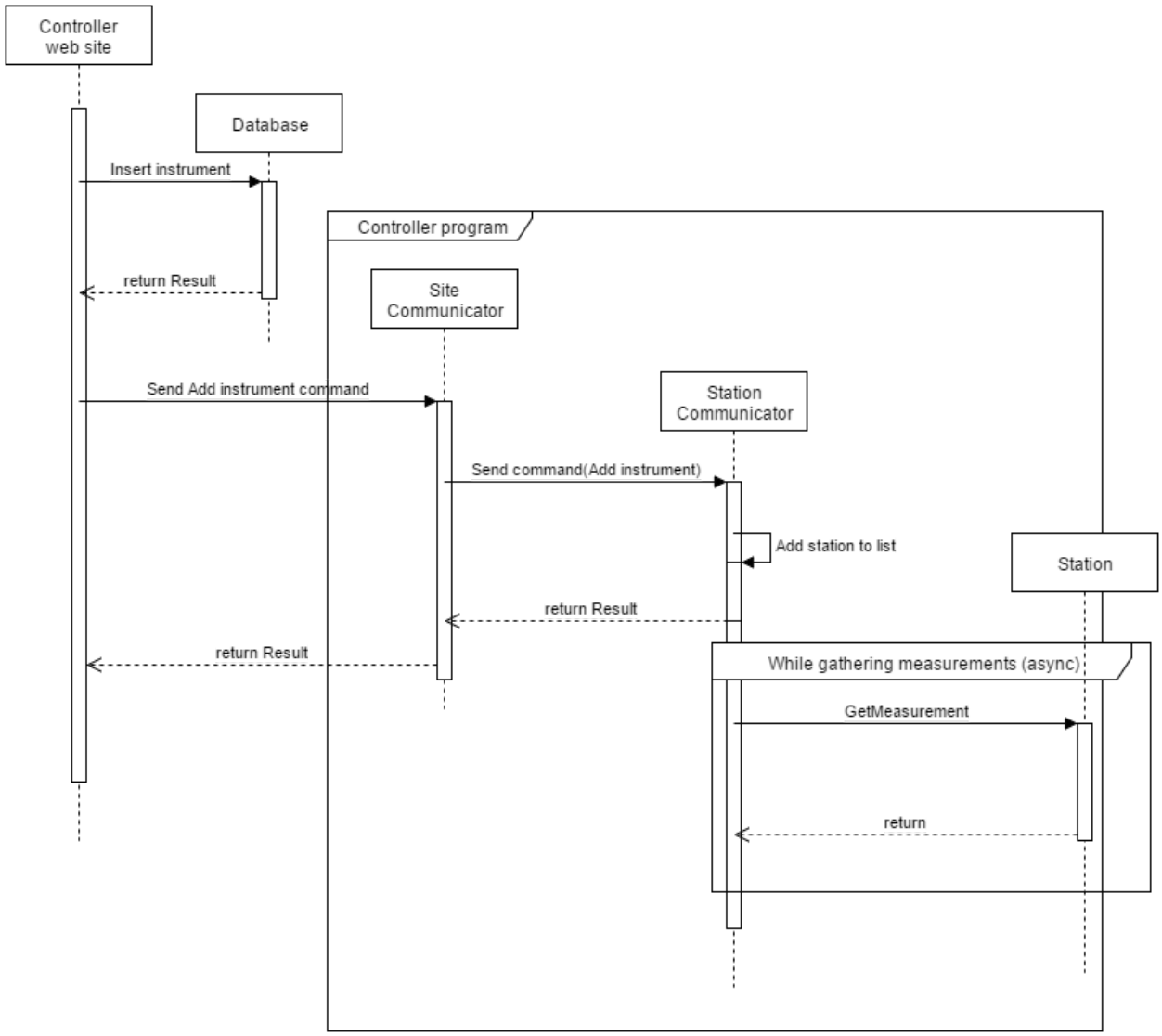
4th week (19th dec - 23rd dec)

This week coincided with the last week before Christmas. We had discussed this the week before, and were aware that our personal and social obligations would impact our cycle time. As such, we we focused on report writing instead of the products.

5th week (26th dec - 30th dec)

Controller program v2

After having defined the core architecture and the cross-component and cross-product interactions in version 1, we spent this week on implementing these interactions and creating the Database Handler and the Controller Site Handler. Alongside this version we also worked on version 2 of the Controller Site. We were able to test integration, ensuring the functionality and stability of the whole pipeline from Controller Site to Station Communicator.



System Sequence Diagram for adding instruments through the Controller site.
This is the same procedure for adding stations

The Database Handler uses Entity Framework object relational modelling with 'Code First models' that are then translated to a local database for testing, while we are working in development. It returns a list of stations from the database.

Requirements/Features

- ✓ Create a protocol that we can use between cross-component communication.
- Get station and instrument data from MS SQL database to Station communicator:
  - ✓ Create a connection to the database.
  - ✓ CRUD on the business data stored in the database (to be split later).
  - ✓ Retrieve station and instrument data.
  - Send measurement data.
- Pass commands from the controller site to Station communicator.
- Communicate with the instrument interface library:
  - ✓ Have the communicator receive commands from other components (database, site).
  - ✓ Have the communicator get measurements from station library.
  - Have the communicator handle these communications with stations asynchronously.
- ✓ Logging events
  - ✓ Status
  - ✓ Errors
  - ✓ Information

Error handling

- ✓ Log errors to console.
- Log errors to file.
- ✓ Handle miscommunication with Database (using Entity means we gain security in error handling).
- ✓ Handle controller site ⇔ controller program unintended commands.
- ✘ Handle incorrect keyboard input (keyboard input requirement put on indefinite hold).
- Handle asynchronous conflicts (thread safety).
- Handle exceptions thrown from the Instrument Interface Library:
  - Connection errors (we pass along a false boolean value, but we need to deal with the possibility of extended period of connection problems).

Retrospective

The team was well refreshed after Christmas and ready to tackle more programming, so we saw a better output of code both in terms of functionality and quality.
In this version we were happy with how well our Controller site and Controller site handler communicated with the Controller Program so the Input manager has been put on an indefinite hold.
In the end we have one unit test failing due to the Unit Testing Project not having access to Entity Framework provider. We modified the unit test to bypass the need for accessing the database, and then re-instated it as was. The permanent solution is to base our Entity testing environment on a pre-existing testing project that Morten has implemented.

New requirements

- Process raw measurement responses.
- Aggregate measurements.

6th week (2nd jan - 6th jan)

Controller Program v3

This was the last week and we had to put a lot of resources into writing the report. We did however decide we would create one more version of the Controller Program as a way to be able to break out of monotonous report writing. We had looked at running the gathering of measurements asynchronously, a complex task, which, if successful, would be of high value to the client.
When the StationCommunicator gets raw measurement data from a station's instrument it is added to a ConcurrentQueue collection. ConcurrentQueue is a Queue class part of .NET library that is thread-safe. This Queue is contained in a handler that starts a thread that starts taking from the queue, until the queue is empty and is restarted when the queue has content.

Requirements/Features

- ✓ Create a protocol that we can use between cross-component communication.
- Get station and instrument data from MS SQL database to Station communicator:
  - ✓ Create a connection to the database.
  - ✓ CRUD on the business data stored in the database (to be split later).
  - ✓ Retrieve station and instrument data.
  - Format and aggregate raw measurement responses.
  - Send measurement data.
- ✓ Communicate with the instrument interface library:
  - ✓ Have the communicator receive commands from other components (database, site).
  - ✓ Have the communicator get measurements from station library.
  - ✓ Have the communicator handle these communications with stations asynchronously.
- ✓ Logging events:
  - ✓ Log status events.
  - ✓ Log errors:
    - ✓ Log to console.
    - Log to file.

Error handling

- ✓ Log to console.
- ✓ Handle miscommunication with Database.
- ✓ Handle controller site ⇔ controller program unintended commands.
- ✓ Handle asynchronous conflicts (thread safety).
- ● Handle exceptions thrown from the instrument interface library.

Retrospective

Improvements were done on the testability. Using dependency injection[30] we provided the classes with concrete implementation, that created mock data or gave mock responses.
There is no method in the DatabaseHandler to insert measurements in the database. That would be a very high value requirement, however the way client wants to insert into their database is a lot more complex than simply working with a development database, which suits our needs for prototyping.

[30] What is deendency Injection - https://stackoverflow.com/questions/130794/what-is-dependency-injection#140655

Code Examples

Here i will show some of the code pieces that i have created for the prototypes

Instrument communicator

As you can see here, the communicator is quite simple. It implements dependency injection through the strategy pattern, while also enabling the dependency inversion principle. Higher level components will only need to rely on the abstract class CommunicationHandler, and not the individual implementations.

```
namespace InstrumentCommunicator
{
    12 references | Lalli-Oni, 2 days ago | 2 authors, 4 changes
    public class Communicator
    {
        private readonly IPEndPoint _ipEndPoint;
        private readonly CommunicationHandler _handler;

        8 references | 0/5 passing | Morten Toudahl, 21 days ago | 1 author, 1 change
        public Communicator(IPEndPoint ipEndPoint, CommunicationHandler handler)
        {
            if (ipEndPoint == null) throw new ArgumentNullException(nameof(ipEndPoint));
            if (handler == null) throw new ArgumentNullException(nameof(handler));
            _ipEndPoint = ipEndPoint;
            _handler = handler;
        }

        /// <summary>
        /// Send command, and get result
        /// </summary>
        /// <exception cref="SocketException">An error occured when accessing the socket</exception>
        /// <exception cref="ObjectDisposedException">TcpClient is closed</exception>
        /// <exception cref="InvalidOperationException">TcpClient is not connected</exception>
        /// <param name="command"></param>
        /// <param name="timeOutInMiliseconds">default 2000</param>
        4 references | 0/2 passing | Lalli-Oni, 3 days ago | 2 authors, 3 changes
        public List<string> GetData(string command, int timeOutInMilliseconds = 2000)
        {
            using (var client = new TcpClient(_ipEndPoint.AddressFamily))
            {
                var clientTask = client.ConnectAsync(_ipEndPoint.Address, _ipEndPoint.Port);

                Task.WaitAll(clientTask);

                _handler.SetStream(client.GetStream(), timeOutInMilliseconds);
                _handler.SendCommand(command);
                return _handler.Response();
            }
        }
    }
}
```

CommunicationHandler

The abstract class implements general functionality, while both allowing for overwriting of two method, and requiring individual implementation of the Response method. This is because how the instruments reply is different. This approach means that the InstrumentCommunicator should be able to communicate with any instrument that can communicate with TCP/IP

```
namespace InstrumentCommunicator
{
    11 references | Morten Toudahl, 22 days ago | 1 author, 1 change
    public abstract class CommunicationHandler
    {
        protected StreamWriter _writer;
        protected StreamReader _reader;
        /// Sets the stream to work with.
        /// <param name="stream"></param>
        /// <param name="timeOutInMiliseconds"></param>
        2 references | Morten Toudahl, 22 days ago | 1 author, 1 change
        public virtual void SetStream(Stream stream, int timeOutInMiliseconds)
        {
            if (stream == null) throw new ArgumentNullException(nameof(stream));
            stream.ReadTimeout = timeOutInMiliseconds;
            _writer = new StreamWriter(stream)
            {
                AutoFlush = true,
                NewLine = "\r\n"
            };
            _reader = new StreamReader(stream);

        }

        4 references | Morten Toudahl, 22 days ago | 1 author, 1 change
        public virtual void SendCommand(string command)
        {
            _writer.WriteLine(command);
        }

        6 references | Morten Toudahl, 22 days ago | 1 author, 1 change
        public abstract List<string> Response();
    }
}
```

Unit tests of instrument handler

Here you can see some of the unit tests that i created to verify my code. As you can see i am utilizing the strategy pattern for my testing.

```
namespace UnitTesting.InstrumentInterfaceCommunicator
{
    [TestClass]
    0 references | Morten Toudahl, 21 days ago | 1 author, 2 changes
    public class CommunicatorTests
    {
        private readonly IPAddress IP = IPAddress.Loopback;
        private volatile int PORT = 3000;


        [TestMethod]
        0 references | Morten Toudahl, 21 days ago | 1 author, 2 changes
        public void PassIfConnectionOccured()
        {
            var endpoint = new IPEndPoint(IP, PORT++);
            var listener = new TcpListener(endpoint);
            listener.Start();
            var com = new Communicator(endpoint, new SetStreamReturnStaticResponse());
            TcpClient client = null;
            Task.Factory.StartNew(async () => client = await listener.AcceptTcpClientAsync())
            com.GetData("hallo");
            Assert.IsTrue(client?.Connected ?? false);
            listener.Stop();
        }

        [TestMethod]
        0 references | Morten Toudahl, 21 days ago | 1 author, 2 changes
        public void GetDataBack()
        {
            var command = "Hej";
            var endpoint = new IPEndPoint(IP, PORT++);
            var listener = new TcpListener(endpoint);
            listener.Start();
            var com = new Communicator(endpoint, new ReturnSameString());
            TcpClient client = null;
            Task.Factory.StartNew(async () => client = await listener.AcceptTcpClientAsync())
            var result = com.GetData(command).FirstOrDefault();
            listener.Stop();

            Assert.IsNotNull(result);

            Assert.AreEqual(command, result);
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        0 references | Morten Toudahl, 21 days ago | 1 author, 1 change
        public void ExceptionOnNullIpEndPoint()
        {
            var com = new Communicator(null, new ReturnSameString());
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentNullException))]
        0 references | Morten Toudahl, 21 days ago | 1 author, 1 change
        public void ExceptionOnNullHandler()
        {
            var com = new Communicator(new IPEndPoint(IPAddress.Loopback, 3000), null);
        }

    }
}
```

CpClient

The CpClient's job is to send commands to the control program, and receive the reply. In the control site, it is used by the StationController, on any CRUD command. It is quite simple, and as you can see i have omitted the implementation of some of the methods. Their names speak for themselves. It was however the source of a problem i describe in the next section.

```
namespace CPCommunicator
{
    3 references | Morten Toudahl, 8 days ago | 1 author, 2 changes
    public class CpClient
    {
        private readonly IPEndPoint _ipendpoint;

        1 reference | Morten Toudahl, 8 days ago | 1 author, 1 change
        public CpClient(int port = 65535)
        {
            _ipendpoint = new                                    s.Loopback, port);
        }

        3 references | Morten Toudahl, 8 days ago | 1 author, 2 changes
        public CommandResult Command(CommandType command, IBusinessModel payload = null)
        {
            if (IsStartOrStop(command, payload)) return SendRecieve(command.ToString());

            if (payload == null) throw new ArgumentNullException(nameof(payload));

            var parameter = GetParameterType(payload);

            return SendRecieve($"{command} {parameter}{Environment.NewLine}{JsonConvert.SerializeObject(payload)}");

        }
```

(parameter) int port = 65535

CommandResultHandler

When i added the capability to communicate with the command program in the control site, i ran into a problem. Each call to an action on the site, would create a new instance of the controller the action lived it. Therefore, i could not store the response in the ViewBag, or in an instance field. Using a static field was out of the question too, since multiple people might be sending commands to the control program at the same time. Which would result in the displayed response not necessarily being the one meant for you.

I came up with the below solution. Since there are only a few potential users of the control site, it should not be a problem having multiple entries in the dictionary. Keeping in mind that the ApplicationPools[31] of IIS are usually short lived too, i doubt it will ever be a problem since the process will be killed regularly, which means the memory allocation will be freed.

```
namespace StationLogger.Tools
{
    4 references | Morten Toudahl, 8 days ago | 1 author, 2 changes
    static class CommandResultHandler
    {
        private static Dictionary<string, string> _commandResults;

        3 references | Morten Toudahl, 8 days ago | 1 author, 1 change
        public static void SetCommandResult(CommandResult result, IIdentity identity)
        {
            if (_commandResults == null)
            {
                _commandResults = new Dictionary<string, string>();
            }
            _commandResults[identity.GetUserId()] = result.ToString();
        }

        1 reference | Morten Toudahl, 8 days ago | 1 author, 1 change
        public static string GetCommandResult(IIdentity identity)
        {
            if (_commandResults == null) return null;
            if (!_commandResults.ContainsKey(identity.GetUserId())) return null;
            string result = _commandResults[identity.GetUserId()];
            _commandResults[identity.GetUserId()] = null;
            return result;
        }

    }
}
```

[31] IIS Application Pool - https://technet.microsoft.com/en-us/library/cc735247(v=ws.10).aspx

Reflection and conclusion

Overall, this was a very informative and educational experience.

I had the opportunity to look at scientific instruments, and figure out how to interface with them. This gave me a chance to refresh on the socket programming, which I learned during the 3rd semester. It also required that I figured out how to handle unreliable equipment, since the instruments did not always handle network streams correctly. Furthermore I had to utilize my knowledge about routing, and find out how to configure a router from a manufacturer that I did not know about.

I also had the opportunity to look at interprocess communication, and figuring out another way to achieve the same result when it proved not to be usable for our setup. I will elaborate on this in my presentation.

To answer the questions in the problem definition regarding learning outcome:

Personally, I believe that I would benefit from heavier reliance on scrum/kanban boards. Something which Lárus was a great advocate for. During my internship where I worked alone, I had gotten used to managing everything with notes on paper. But when more people are involved, it is something that I need to remember to use.

The communication with the client, was at an acceptable level. We would often seek feedback, and show individual pieces of the system to one or more of the stakeholders. However, there is always room for improvement. I believe that we could have created a product that was better tailored to their expectations with more communication, for instance implementing something similar to sprint reviews. This is not to say that we did not live up to the agreements made. We did create the prototypes and made the recommendation that we promised to do.

Design wise, I believe that I have done a good job. I feel like the lessons in school were both remembered and used while designing the architecture and writing my code. Without further lessons on a higher level or actual work experience, I do not believe that I can improve my coding or software design skills any further.

As for the problems we set out to solve, i believe we have solved all of them.

- We have investigated the integration of the instruments with the current system, and proposed an improvement to this.
- We have investigated how the system collected the data, and proposed a solution to this.
- We have looked into the movements and storage of the data, and proposed an improvement to this.
- We have elaborated on the potential issues of both the current and the updated system.

Note regarding links: *In case i reference a website, i have also included a date. This was the date that i verified that the information i refer to was there. In case the website is no longer there or it has been modified, please use the wayback machine to verify my source: https://archive.org/web/web.php*

Appendix

Instrument overview.

Monitor		**API (GAS)**	**SM200 (10µm, 2.5 µm)**	**Teom (10µm, 2.5µm)**	**SMPS (new)**	**DMPS (old)**
Compound		NOx,NO,O3,CO,SO2	Partikelconcentration	Partikelconcentration	Partikelnumbers and size	Partikelnumbers and size
Built in interface		Seriel or LAN	Serial	Serial	LAN	Serial
Buffer		14 days data	200 days	60 days	No	No
Data		minut and 30 minut average	24 hour	½ hour	2 minutes	2 minutes
Units		ppm or ppb	µg/m3	µg/m3	µg/m3	
Getting data		Send string	Send string	Sends automatically	Compagy software	Compagy software
Calibration		Yes	At Opsis	No	No	No
Compagny		Teledyne	Opsis	RP	TSI	TSI/home made
Note		One monitor for NO,Nox, temp, press	Filter are analyzed in lab			
Filetype		.day	.pm, .p25	.teo, .t25	.tsi	.raw, .dis
Monitor		**Tekran (Hg)**	**Temperature**	**Flow denuder**	**LVS (10µm, 2.5 µm)**	**Meteorologi**
Compound		Mercury (Hg)	oC, humidity	m3 (Volt)	Partikelkoncentration	VS,VD,HUM,RAD
Built in interface		Serial	USB	Analog (Volt) -> USB	Serial	Serial
Buffer		No	No	No	17 days	No
Data		15 minutes average	5 min	5 min	24 hour	Continous

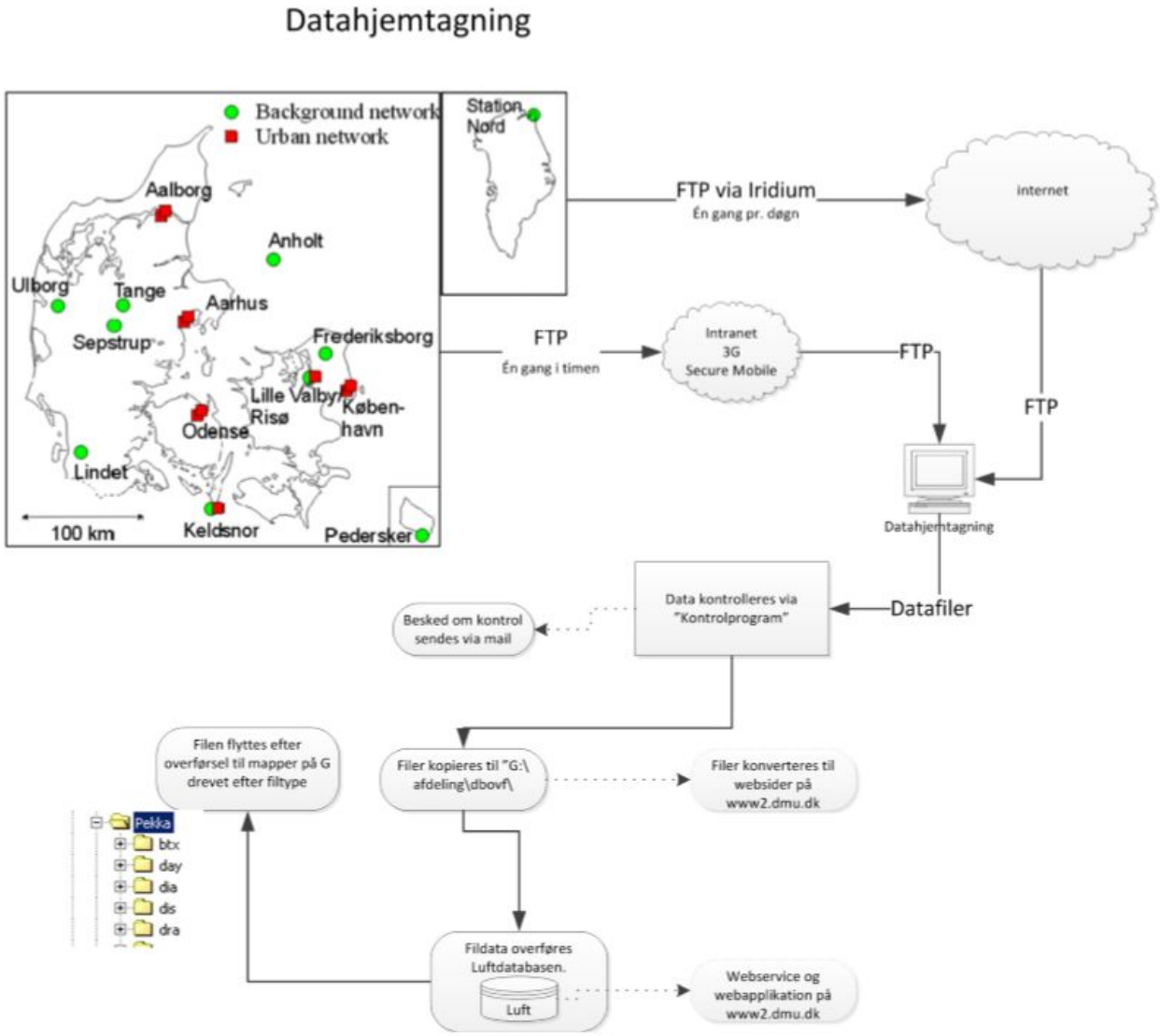
Units		ng/m3	oC, %	Volt	m3	m/s, degrees,%, w/m2
Getting data		Sends automatically	Send String	Send string	Send string	Sends automatically
Calibration		In lab only	No	No	No	No
Compagny				NI	Leckel	RISØ
Note		Only St. Nord	Room temperature	AC/DC converter	Filter must be weighed	Data from several sensors
Filetype		.tek, tk2	.t	.den	.l10, .l25	.met
Monitor		**Sonic (meteorology)**	**Asiaq Meteorologi**	**Flow inlet tubes**	**LVS-FPO**	**LVS-ECOC**
Compound		VD,VS,Temp	VD,VS,Temp,Rad etc	m3 (Volt)	Air volumen	Air volumen
Built in interface		Serial	Serial	Analog (Volt) -> Seriel	Serial	Serial
Buffer		No	month	No	17 days	17 days
Data		continous 10 Hz	10 and 30 min	5 min	24 hour	24 hour
Getting data		Sends automatically	Automatically	Send string	Send string	Send string
Units		m/s, degrees, oC	m/s, degrees,%, w/m2	Volt	m3	m3
Calibration		Metek only	By Asiaq	No	No	No
Compagny		Metek	Cambell	NI	Leckel	Leckel
Note		Data fromseveral sensors	Only St. Nord	AC/DC converter	Filters are analyzed in lab	Filters are analyzed in lab
Filetype		.snc	.met	.fli	.fpo	.fpo
Monitor		**HVS**				

Compound		Air volumen				
Built in interface		Serial				
Buffer		SD card				
Data		24 hour				
Getting data		Send string				
Units		Volume				
Calibration		No				
Compagny		Reimer				
Note		Filters are analyzed in lab				
Filetype		.hvs				

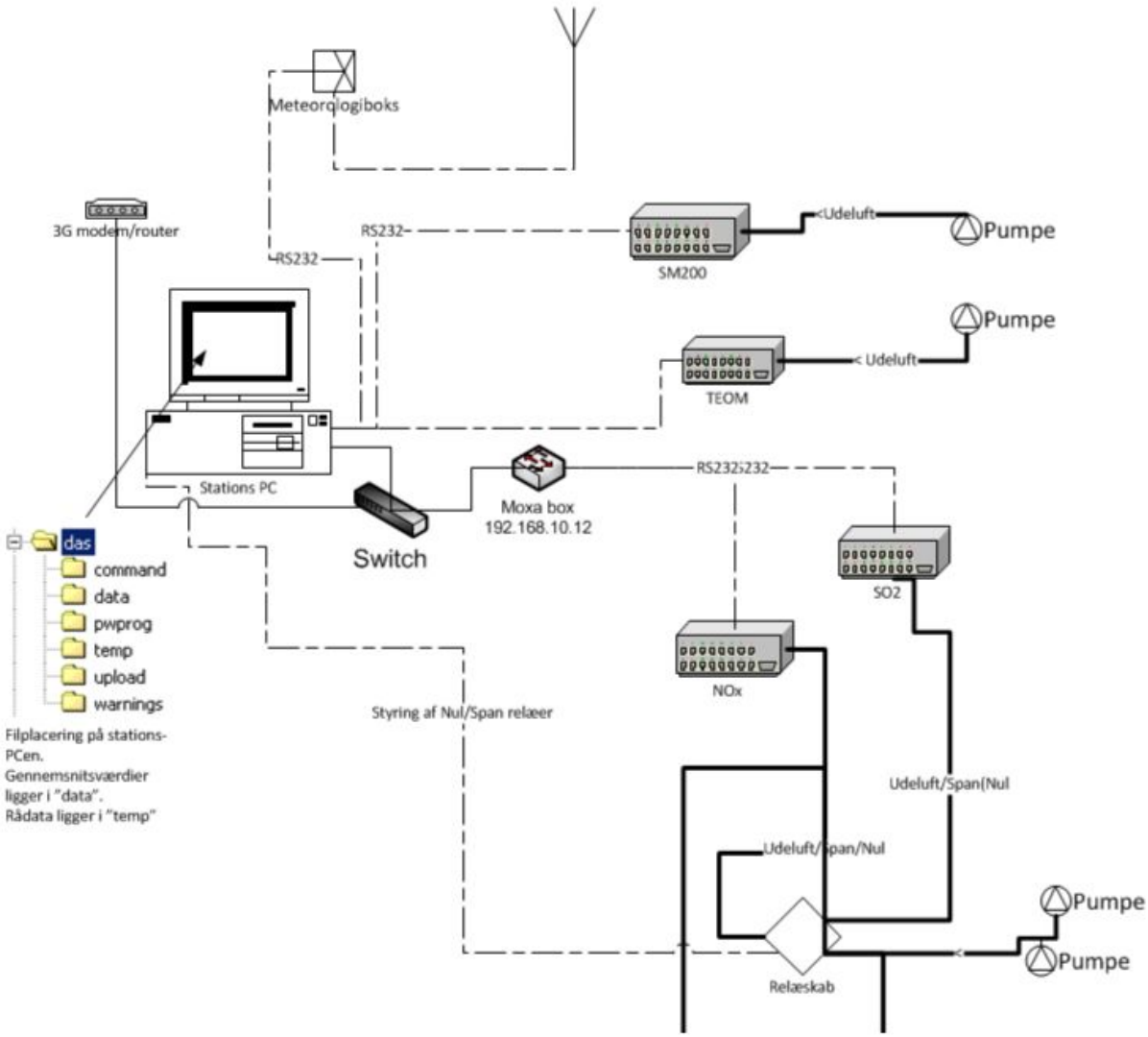
Procomm Plus Script.

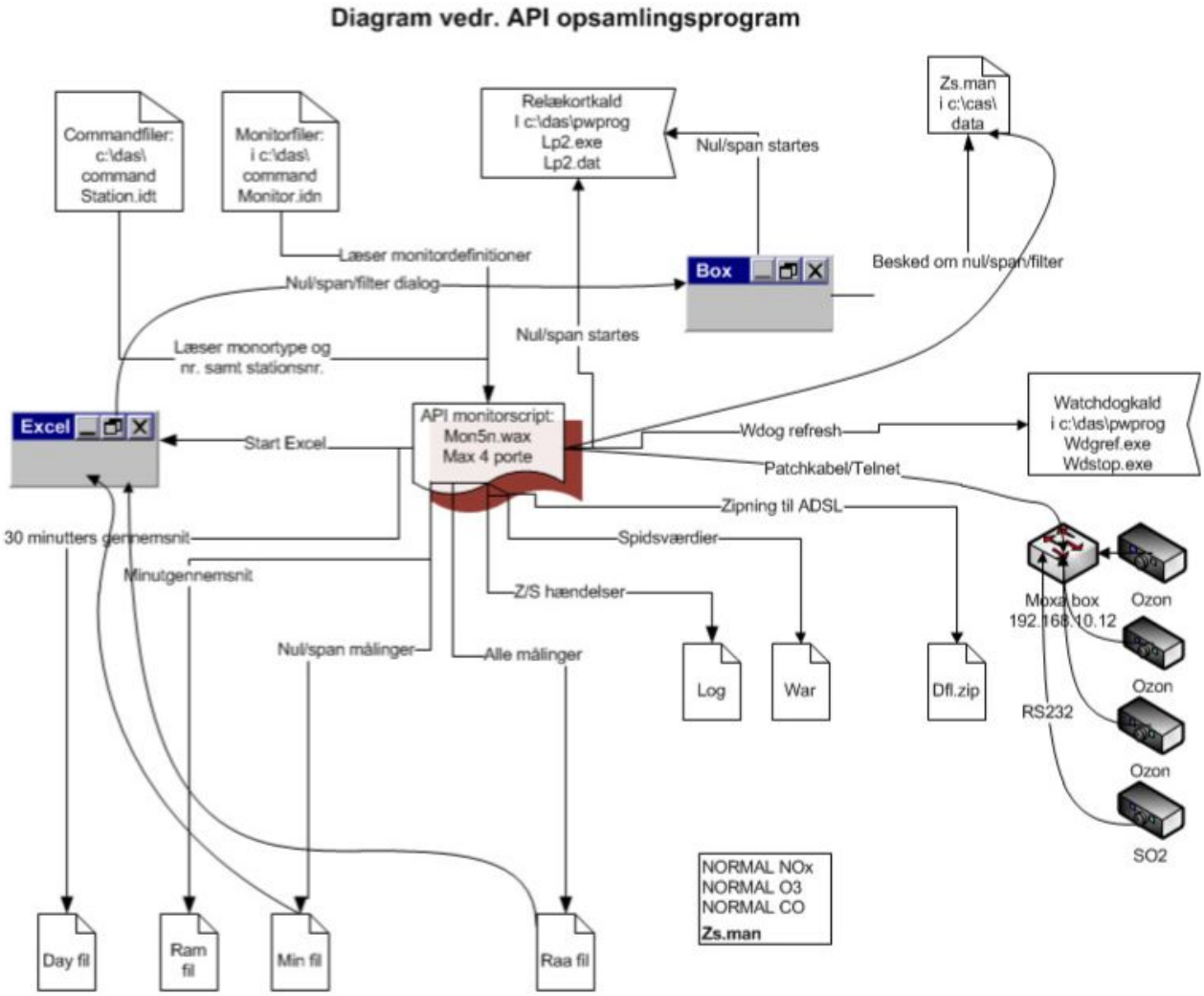
The formatting proved troublesome, so i put the script on pastebin. http://pastebin.com/MPUSktdM
it will expire 06/02-2017

Diagrams


Datahjemtagning
Background network
Urban network
Aalborg
Anholt
Ulborg
Tange
Aarhus
Sepstrup
Frederiksborg
Lille Valby
Risø
Køben-
havn
Odense
Lindet
Keldsnor
Pedersker
100 km
Station Nord
FTP via Iridium
Én gang pr. døgn
internet
FTP
Én gang i timen
Intranet
3G
Secure Mobile
FTP
FTP
Datahjemtagning
Datafiler
Data kontrolleres via "Kontrolprogram"
Besked om kontrol sendes via mail
Filen flyttes efter overførsel til mapper på G drevet efter filtype
Filer kopieres til "G:\afdeling\dbovf\
Filer konverteres til websider på www2.dmu.dk
Pelka
btx
day
dia
ds
dra
Fildata overføres Luftdatabasen.
Luft
Webservice og webapplikation på www2.dmu.dk

Stationsopstilling





Locations

- Trello - https://trello.com/b/NnBT42jG/dissertation
- Win32DiskImager, 05/01-2017 - https://sourceforge.net/projects/win32diskimager/
- Router emulator, 05/01-2017 - http://www.support.dlink.com/emulators/dir855/Virtual_Server.html

Bibliography

In order of use:

- Times Higher Education World University Rankings, 05/01-2017 - https://www.timeshighereducation.com/world-university-rankings/2017/world-ranking#!/page/0/length/100/country/2256/sort_by/rank_label/sort_order/asc/cols/rank_only
- Agile Principles, Patterns, and Practices in C#, by Robert C. Martin and Micah Martin. ISBN: 978-0131857254
- Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition), By Craig Larman. ISBN: 978-0131489066
- Scrum guide, 05/01-2017 - https://www.scrumalliance.org/why-scrum/scrum-guide
- The kanban methodology, 05/01-2017 - https://www.atlassian.com/agile/kanban
- An External Replication on the Effects of Test-driven Development Using a Multi-site Blind Analysis Approach
    - Original link: http://people.brunel.ac.uk/~csstmms/FucciEtAl_ESEM2016.pdf
    - Alternate link: https://drive.google.com/open?id=0BwGU_UCX-H6BSmVHZVlGZnlfZDQ
    - Authors
        - Davide Fucci, M-Group, University of Oulu Oulu, Finland davide.fucci@oulu.fi
        - Giuseppe Scanniello, University of Basilicata, Potenza, Italy, giuseppe.scanniello@unibas.it
        - Simone Romano, University of Basilicata, Potenza, Italy, simone.romano@unibas.it
        - Martin Shepperd, Brunel University, London, United Kingdom, martin.shepperd@brunel.ac.uk
        - Boyce Sigweni, Brunel University, London, United Kingdom, boyce.sigweni@brunel.ac.uk
        - Fernando Uyaguari, Universidad Politécnica de, Madrid, Spain, f.uyaguari@alumnos.upm.com
        - Burak Turhan, M-Group, University of Oulu, Oulu, Finland, burak.turhan@oulu.fi
        - Natalia Juristo, M-Group, University of Oulu, Universidad Politécnica de, Madrid, natalia.juristo@oulu.fi
        - Markku Oivo, M-Group, University of Oulu, Oulu, Finland, markku.oivo@oulu.fi

- Product information regarding the Dovado PRO router, 05/01-2017 - http://www.dovado.com/images/PDF/DOVADO_PRO_AC_DATASHEET.pdf
- Difference between APN and VPN, 05/01-2017 - http://smallbusiness.chron.com/difference-between-apn-vs-vpn-38815.html
- What is an SPI firewall, 05/01-2017 - https://www.techwalla.com/articles/what-is-an-spi-firewall
- Reference to book/publication, 05/01-2017 - https://en.wikipedia.org/wiki/First_normal_form#cite_note-2
- What is a Raspberry PI, 05/01-2017 - https://pimylifeup.com/what-is-raspberry-pi/
- Raspbian: Run a Program at Startup, 05/01-2017 - http://www.mikeslab.net/?p=176
- Raspberry PI VNC documentation, 05/01-2017 - https://www.raspberrypi.org/documentation/remote-access/vnc/
- Dovado products, 05/01-2017 - http://www.dovado.com/en/products
- Throw away prototyping, 05/01-2017 - https://en.wikipedia.org/wiki/Software_prototyping#Throwaway_prototyping

- What is entity framework, 05/01-2017 - https://msdn.microsoft.com/en-us/library/aa937723(v=vs.113).aspx#What%20is%20Entity%20Framework
- Introduction to ASP.NET Identity, 05/01-2017 - https://www.asp.net/identity/overview/getting-started/introduction-to-aspnet-identity
- Strategy pattern, 05/01-2017 - http://www.gofpatterns.com/behavioral-design-patterns/behavioral-patterns/strategy-pattern.php
- .NET Architectural Components, 05/01-2017 - https://docs.microsoft.com/en-us/dotnet/articles/standard/components
- Introduction to WPF, 05/01-2017 - https://msdn.microsoft.com/en-us/library/mt149842(v=vs.110).aspx
- Data Transfer Object, 05/01-2017 - https://en.wikipedia.org/wiki/Data_transfer_object
- What is deendency Injection, 05/01-2017 - https://stackoverflow.com/questions/130794/what-is-dependency-injection#140655
- IIS Application Pool, 06/01-2017 - https://technet.microsoft.com/en-us/library/cc735247(v=ws.10).aspx