Luleå University of Technology

Third year project

Sensor data aggregation through CoAP

***A*uthors:**
Sophia Bergendahl
sopber-8@student.ltu.se

Edvin Bruun
edvbru-9@student.ltu.se

William Gustafsson
wilgus-9@student.ltu.se

Christoffer Holmstedt
cihhol-7@student.ltu.se

Marcus Rådman
marrdm-9@student.ltu.se

Kristoffer Svensson
kirsev-9@student.ltu.se

Ludwig Thurfjell
ludthu-7@student.ltu.se

***S*upervisors:**
Ulf Bodin
ulf.bodin@ltu.se

Jens Eliasson
jens.eliasson@ltu.se

Rumen Kyusakov
rumen.kyusakov@ltu.se

March 4, 2012

Project Description

Background

Luleå University of Technology conducts research on lowpower wireless microprocessors called "Mulle". These microprocessors can be used for various things depending on which type of sensors you connect to it, everything from measuring temperature or vibrations in a car to analyzing the quality of the road that you drive on.

Every year northern parts of Sweden are used for testing cars during winter conditions. To test a car you first decide what you want to test, then you test with local sensors logging within the car. When enough data is collected you return back home. At the testing facility the data is now available for analysis. Depending on the results from the previous runs you might want to test some parts in more detail so you re-configure all sensors and go out for another test run.

This process is time consuming when you need to return to testing facility to be able to analyze and re-configure all sensors. In todays society most computers are connected to internet and/or other private networks, most of these computers have the ability to be remotely configured and maintained. The goal with this project is to be able to analyze data from sensors in realtime and re-configure them on the fly while testing is in progress.

Project Targets

1. Be able to send live sensor data from multiple "Mulles" to an online logging server/service.

2. Be able to read sensor data on the web with both a PC (web browser) and through an Android mobile device.

3. Be able to re-configure the sensors through a web interface and through an Android mobile device.

Technical limitations

The technical limitations for this project was mainly to restrict how flexible the finished solution was. The Mulles can be used together with any kind of sensors. In this project the main goal was to get it operational in a car and measure temperature and some other variables.

Execution of the project

Scrum and how it has been used

It was decided back in november that the entire project would be divded into three sprints. The exact dates were to be decided in the beginning of each sprint. In cooperation with the client the scope of the project and the scope of the first sprint was decided upon in november. During the first projectmeeting the first sprint goal was divided into eight sprint stories. It soon became clear that those eight stories were way to big, at the end of the sprint none of the stories had been finished.

Lesson learnt, the second sprint was divided into smaller stories which gave immidiate result when the first 69 sprint story points finished during the second sprint.

To decide upon size for each sprint story, for the second and third sprint, "planning poker" [1, p. 42] was used. For every sprint story each project member wrote down an estimate on the scope for each story. With planning poker it became clear that each project member had a different vision for each story. A short discussion after each estimate made it more clear on how big the scope was, an agreement was usually made within a few minutes.

One project, three sprint goals

As mentioned earlier the project was divided into three sprints. This meant that three different sprint goals had to be divided into smaller sprint stories which in turn had to be assigned to a project member. Due to all project members being new to most of the tasks at hand the first team division was made with focus on components [1, p. 106]. The goal with this was that each smaller team within the project could sit together and dive deep into their specific part such as the Mulle, the server parts or the android code. Later on a split into cross-component teams [1, p. 107] was aimed for but due to some persistent bottlenecks in some components this was never done.

For all sprints the sprint planning meeting were used to categorize each sprint story into the different components (Mulle, Server, Android). It was then up to each component based team to split their stories between themselves. This ended being a very flexible solution, in some cases, to flexible when a team didn't use the Scrumboard online at Scrumdo.com the team could wander off from the sprint story they were supposed to work on. For the last sprint everyone got at least one sprint story assigned to themselves directly during the sprint planning meeting. This was made to put more focus on using the Scrumboard at Scrumdo.com.

A move to cross-component teams was never made instead an attempt to increase speed for the Mulle and the Android component was made by moving one team member from the server team to the Mulle team and another to the Android team. The server component at this time was way ahead of the other components. Another week later it became clear that the additional team member for Mulle team was not needed, cause the problem was still a bottleneck that only one or two team members could work on at a single point in time, a move back to the server team was made.

Individual time monitoring and speed

Sophia Bergendahl

Edvin Bruun

Throughout the project I, as well as my fellow group members, have been gradually learning to work with the Scrum project-model. This work-model includes a tactic for distributing work called "stories". These stories are given an time estimate and a actual time when they are done, and following this I will explain three stories that I've encountered in this project. For convenience I've chosen one story from each iteration to roughly show how the work-model was more and more used. As a quick reminder, my stories have revolved around getting the Mulle-communication to work.

To start things off, I chose the first story that I had, which was titled "CoAP communication over bluetooth from Mulle to server". This story was estimated to take the entire first sprint which was roughly eight weeks long. In this story there was a pretty hefty start-up time included, as most of us were clueless as to how much time we would spend on learning the new technologies in each of our assignment-fields(Mulle, Server, Android). When the eight weeks were up, the progress on the story was horrible to say the least. During our first sprint we struggled with issues that were very much out of the scope of what we should have been doing. The reason for this was mainly because we needed to get these things working before we could start progressing on the actual story. However when the time was up we agreed that we'd have to take a different route to achieve the communication so the actual time for this story was pretty accurate.

Moving on to the next story, this story took place in the second iteration and revolved around sending UDP packets from the Mulle to the server. The point of this story was to check if we could achieve the simplest of communication(with our new approach) and then build from there. This story was

scheduled to run over a week which also was our estimate. The actual time for this story was however a few days over the estimate. The reason for this was that at this point we were working on multiple stories as well as some issues with our testing methods. Looking back on the things that caused the delay I don't see how we could have avoided them.

The third story to be explained in this documentation is a story that was very small and which took place in the third iteration. The task in the story was to get the Mulle to run a certain function every time it received a UDP package. This was estimated at roughly ten hours, were as the actual time it took was closer to five. The reason for the shorter time than anticipated was that I had gained knowledge how to do this indirectly through another story which I'd worked on earlier.

William Gustafsson

Christoffer Holmstedt

What I've learnt during this project is that there is no way to estimate a reasonable time without knowing something about the topic of concern beforehand. I've had several stories assigned to me in the topic of linux installation and configuration e.g. installing a webserver and a mysql database. Both the webserver and database I've installed several times before and knew exactly which steps I was supposed to do to get it up and running as soon as possible. Of course this made it really easy to estimate an expected time for these stories, or at least most of them. One story was about installing Ubuntu. Even though I have installed Ubuntu several times before I didn't anticipate that an entire installation could be so slow, I had simply missed to take into account that we were running our server on a very old machine. The time estimate for this story was of by 100%.

The other stories that didn't go so well was mainly about python programming and working with the coapy server implementation. It took generally more time than I expected to get started. In the future I will increase the expected time before I get going e.g. the time to read about a new software I will work with and the time to try out the current functionality. Though I will keep my estimate for the actual coding parts.

In the future when it's up to me to do another estimate concerning a topic I've never worked with I hope I have a few colleagues with knowledge in that area to help me out with my first estimate. As long as I guess an estimate instead of chosing "I don't know" I will improve and eventually I will now my "speed".

Marcus Rådman

Kristoffer Svensson

Assignments

1. "Restructure the loading of new services for the python server"

   I had some previous experience with python on top of already been dealing with the actual server implemention we decided to use prior to dealing with this issue so I was pretty certain I would be able to complete it in about 3 hours. Without any major issues I managed to complete it in roughly 3 hours and 30 minutes. The question at that point was rather if the solution was adequate, which after some consultation it was deemed to be.

2. "Figure out how service discovery works in CoAP"

   The estimated time for this was set between 8 and 10 hours. Reason being that it was new ground for me but still didn't feel like it was that big of an issue. It turned out to be a relatively small fix after 6 hours of work. The issue of whether the solution was good enough or not for the end-purposes was brought up again but it was decided that it was good enough for the time being. With that said, I doubt that the estimated 8-10 hours would have been sufficient if a end-purpose qualifying solution was to be made.

3. "Implement EXIficient"

   Originally the time for this assignment was set to 20 hours, when the assignment was still "Implement EXI parser". In that situation we assumed that there was more work to be done making the parser from the ground up.

   When EXIficient was discovered we found out that it had a ready-made demo that almost suited our needs. It turned out that the actual time needed to get something that would do the job for us would be around 10 hours modifying that demo, which is exactly what ended up happening.

4. "Individual documention"

   I assumed about 2-4 hours for the actual writing with an added hour or two for the research needed to know what was to be included in the report. In the end, this all came down to about 4-5 hours which is well within the expected range.

General thoughts concerning time estimation I feel as if my personal time estimations, contrary to what I expected, have been quite accurate. I reckon the reason to this being that I had the fortune of having some kind of knowledge concerning the assignments prior to doing them. Without any idea on the scope or size of the assignments I think that the estimations wouldn't have been this accurate.

The estimations themselves have been useful tools for knowing when to ask for help or guidance. When you're approaching a time limit and you know that you're kind of stuck that's a really good indicator that help is needed. I think it's been quite a useful tool, at least for me personally.

Ludwig Thurfjell

Reflection about Scrum usage during this project

Alot has been learnt during this project and to keep it short the following lessons learnt and improvements that can be made are a chosen few.

At the end of all projects when the deadline is closing in the pace of the development often increases. A downfall of this is that when the speed increases the quality of the code often decreases. Scrum is a solution to this problem with the main goal of keeping a steady development pace and always keeping the quality of the code as good as possible above some minimum criteria. With to long sprints, projects will still end up with a big deadline, this is what happened in this project. In total the project lasted about 17 weeks including the winter holidays. Instead of three sprints where each lasted about five weeks a project split into smaller sprint would have been better. If time travel was possible this project would have had two smaller sprints in the beginning each would have lasted for two weeks. The first sprint with goal of configuring all required software for everyone such as setting up git, the different IDEs, gcc and other required software/tools. The second sprint with the goal of understanding what was available at the time and get all basic functionality working. If the project had found any big bottlenecks at the end of the second sprint that would be the point in time to halt the project and really rethink what to focus on. The remaining weeks should have been divided into three evenly sized sprints.

Another big improvement could have been made in the relation between Scrum team and product owner. From the beginning it was unclear who was the product owner out of three supervisors/clients, this should have been defined to one single person as early as possible before continuing with any other work. The absence of the product owner before and during our sprint planning meetings resulted in alot of confusion when it was up to the team to

decide an estimate for each sprint story. The lesson learnt from this is that without a product owner the scrum team will fumble in blindness forever or as Kniberg writes [1, p. 25] *"...each story contains three variables that are highly dependent on each other"*. There is no way the team can choose a good estimate when there is no product owner that has already chosen an importance and scope for each sprint story to start with and is available to change that during the course of a sprint planning meeting. This also lead to the project growing in scope without any clear bounds.

The last and perhaps the most important improvement that will be mentioned is the importance of having a team member taking the role as Scrum master. Without the Scrum master during the first sprint everyone was on their own. For the second and third sprint the Scrum master role was appointed to one team member. This improved the communication within the Scrum team alot and also made it possible for individual team members to have someone to go to in case of general questions and/or other problems.

Results

Deliverables

Mainly due to being unable to setup communication from the Mulle to the server, project target number one listed in the beginning of this document is not met. What is delivered from the project concerning this is available in Appendix A where a guide on how to get started where this project ends is available.

Project target number two is not met. What is delivered from the project is a basic webapplication with a simple database structure as the back-end where future real sensor data can be stored.

Project target number three is not met. Basic functionality on how to use EXI as encoding for sensor data is available for the Mulle and the Android application but hasn't been tested. No work on EXI has been made for the server though there exist a simple webpage to turn a value on or off.

Testing

During this project testing has been made by each project member. The scope of the testing for each story has been up to each project member to decide upon. Right before each sprint demo a project meeting was scheduled were each one showed what was completed and what was not. Depending on what was ready at that point in time the entire Scrum team decided what was going to be presented at the demo and a test was made to confirm that it was possible to show the parts that were decided upon. No further testing was made during this project.

Lessons learnt

Communication is always troublesome, it's usually easy to get a system to transmit data and another to do the same. The hard part is when you want different system to actually communicate with each other. This boiles down to the lesson learnt that the project should have focused more on specific components instead of spreading out on all three components (Mulle, Android and the server) from the beginning. The goal should have been to get the Mulle communicate with either the server or the android device to start with. When that was operational new systems could have been added to the mix.

Testing is always hard to do, it might be easy to test specific test cases but is very hard to test all possible usages. To make sure that your codebase doesn't become useless in the long run you need somekind of automated

testing to make sure that all previous bugs found is tested automatically with all new improvements and add-ons. It might be time consuming in the beginning of a project but it's priceless in the end. An important part of testing is to test both individual components and interaction between different components. If only testing is made to the interaction between different components a finished component cannot be tested until the other one is finished aswell, this creates bottlenecks for the entire project.

When is something done? Do not let this question be up to the individual programmer, it puts the programmer in a difficult situation. Either the programmer wants to create the perfect spot and keeps going forever or the programmer will take shortcuts that will comeback and haunt you later on. It must be up to the team to decide when something is done or not. This will make it alot easier in the day to day work for everyone, if anybody is uncertain if it's completed or not, just look into what was decided earlier.

Suggested improvements

Without meeting any of our project targets it's hard to suggest improvements except the obvious one to complete what has been started. Instead of going straight at it and try to finish it all as soon as possible some thought process need to be put into the question, what really needs to be finished?

From this projects point of view the first thing to prioritze is to get some communication with the Mulle. The Mulle in its current state has alot of bugs which makes it hard to debug, there seems to be alot of "random" bugs occuring when you lest expect them. The code that is finished "should work" but hasn't.

The second priority should be to rethink the servers purpose. At the current state the python implementation is very simple and if the only thing you want to do is add new CoAP services which just return some simple results, then keep going with the current server implementation. A new service is very easy to add. What has been realized at the end of this project is that in the long run it might be better to create a new C server implementation instead. The main reason for this is that when you want to implement custom communication protocols such as EXI alot of code is available in C and with a server implementation in C it will be relatively easy to share code between the server and Mulle system.

As the third and last priority, work should be put into the Android system. With the Mulle up and running for testing purposes it should be easy to get going with the android application but without the Mulle the android application isn't worth anything.

Conclusions

This project started out to be of reasonable size for our project group but ended up way to big mainly due to some persistent bottlenecks in some components. This lead to no component (Mulle, CoAP server and Android application) being fully functional in the end. This is exactly what Scrum tries to prevent, as discussed earlier in this report a better solution to improve quality would have been to do a few more, but smaller sprints.

Concerning usability and future improvements there is alot more work that needs to be done. The current state of the the software is that all components are lacking small parts in different areas, this makes it unusable in it's current state. Future improvements should start with taking a step back and really rethink what the goal is and start working with one component at a time. Highest priority should be to get the Mulle operational and able to communicate with it's surroundings. Without sensors there is no data to analyze or to reconfigure "on the fly".

References

[1] Henrik Kniberg, *Scrum and XP from the Trenches*. C4Media Inc, Publisher of InfoQ.com, 978-1-4303-2264-1, http://infoq.com/minibooks/scrum-xp-from-the-trenches, 2007.

Appendix A - How to build upon our codebase

This appendix include information on how to build upon our codebase for the Mulle (C), server code (Python, PHP/HTML5 and C) and Android Mobile phone (Java).

Mulle

The Mulle communication via COAP is far from finished and therefore here is a guide how to get started followed by what should be further built. The instruction on how to get started are done in Ubuntu, the versions that were tested are 10.04 and 11.11. Windows or any other OS will not be covered in this guide.

The Mulle software required is all in the Software-PAN-NAP folder which contain necessary libraries and applications. It is assumed that you have acquired this because it's essential for any development.

Getting started

The first step you will take is to download the code from a repository on Github.com, if you don't know how to do this there are several guides how to do that on their site. Since there are several ways to do this here are the links to the repository:

- Download zip: `https://github.com/christofferholmstedt/vehicletesting/zipball/master`
- Git: `git://github.com/christofferholmstedt/vehicletesting.git`
- SSH: `git@github.com:christofferholmstedt/vehicletesting.git`

When you've done this note that you'll only need files from the ../vehicletesting/code/Mulle folder. In this folder there are two subfolders named Coap and PAN-Router_demo. These two folders are going to seperate locations in your Mulle software folder, since the Coap folder contains the code for the protocol and PAN-Router_demo code for the application.

The folder Coap should be copied into ../<Mulle Software folder>/Libary/misc/apps/ and the PAN-Router_demo folder goes to ../<Mulle Software folder>/Applications/. The PAN-Router_demo folder is optional to use, if you want to use you're own Mulle program you need to include coap.h and run coap_init() somewhere in your applications c-file. Since the coap-protocol isn't official you also need to put a "#define LWIP_COAP" and set it to 1 in proj_arch.h(as well as turn off any other protocol that are used for communcation) and

finally you need to add coap.c in the SOURCES.mk file. This is done by adding "$(LIBDIR)/misc/apps/coap/coap.c" to the LWIP_Apps field.

Now when you have the code set up you must install the gcc compiler, namely m32c-elf-gcc, for this type of hardware. Do the following steps in your terminal with sudo, alternatively make a script. Be prepared to redo this if you're doing it in a script. It's preferable to do this step by step because it has a higher rate of succeeding.

Alternatively you can follow this guide(however, it's strongly advised not to): `http://www.eistec.se/docs/wiki/index.php?title=Mulle\_software\_with\_GCC`

Setup Development Host:

- apt-get install build-essential
- apt-get install m4 autoconf libtool gawk bzip2 bison flex gettext texinfo zlib1g-dev
- apt-get install libmpc-dev libmpfr-dev

When this is done you need to download binutils and install it:

- wget http://ftp.gnu.org/gnu/binutils/binutils-2.19.1.tar.bz2
- tar xjf binutils-2.19.1.tar.bz2
- mkdir binutils-obj
- cd binutils-obj
- ../binutils-2.19.1/configure –target=m32c-elf
- make
- sudo make install
- cd ..

In the next step you'll download gcc-4.3.3 and newlib 1.17 and install them:

- wget `ftp://ftp.nluug.nl/mirror/languages/gcc/releases/gcc-4.3.3/gcc-4.3.3.tar.bz2`
- wget `ftp://ftp.nluug.nl/mirror/languages/gcc/releases/gcc-4.3.3/gcc-core-4.3.3.tar.bz2`
- wget ftp://sources.redhat.com/pub/newlib/newlib-1.17.0.tar.gz

- tar xjf gcc-4.3.3.tar.bz2
- tar xjf gcc-core-4.3.3.tar.bz2
- tar xzf newlib-1.17.0.tar.gz
- cd gcc-4.3.3
- ln -s ../newlib-1.17.0/newlib
- ln -s ../newlib-1.17.0/libgloss
- cd ..
- mkdir gcc-obj
- cd gcc-obj
- ../gcc-4.3.3/configure –target=m32c-elf –with-newlib –enable-languages="c"
- make
- sudo make install
- cd ..

When you've done all these steps you should try typing this into your terminal: "m32c-elf-gcc –version" and check that it says "m32c-elf-gcc (GCC) 4.3.3" just to be sure.

Now you're good to go, a final note on how get get set up is how to push your code onto the Mulle. Pushing your programs is done in a few steps, first you need to orient your way to the folder containing the program you're pushing in the terminal, then do these steps:

- make clean
- make all
- make program
- make debug

The "make clean" is only needed if you changed something in the system files and when running "make program" the Mulle must be set to "Program Mode". If all your modified files are correct your code will be pushed to the Mulle, if you get a "Clock Validation Error" make sure the Mulle is set to "Program Mode" and try again. The "make debug" command starts mulle_term, which is a terminal debugger for the Mulle, so when you put it back on "Run Mode" you'll see what it is doing(in the form of printouts).

Further developement

You will need some sort of device that can share internet through Bluetooth to the Mulle, whether it be a telephone or a personal computer with an USB-dongle. To test COAP protocol, you can download the Copper addon for Firefox. The coap_init() function in coap.c initializes so that when you receive UDP packets on the Mulle the function coap_input() is run.

First the coap_input() function needs to be properly made, since it does nothing useful at this time. What needs to be done is that the function should put all the correct values into a coap struct and based on what values are in each field the function should decide where to send the infomation next for handing.

A suggestion for handling the COAP payload is by working with EXIP. You create and decode exi files directly on the mulle. EXI-files which have the same structure as XML are interesting for this type of projects since it's light and you can easily write scripts with XML structure.

You can find out more about EXIP here: http://exip.sourceforge.net/ . A final suggestion is to finish the debug_coap_print() function since it will be very helpful for debugging.

Server

Coapy server

Existing implementation The server is based on CoAPy, which is a python implementation of the CoAP protocol. The actual server implementation is a modification of the example server provided along with CoAPy. Some minor changes were made to it allowing it to accept all forms of CoAP messages, provided they are taken care of properly. We also added a way for new services to be easily added and used.

The procedure when adding new services is as follows:

1. Create a new .py file with the name corresponding to the actual service name (e.g. "TestService.py") in the "services" directory

2. The file should contain only the necessary imports (including CoAPy parts) along with a class name the same thing as the file ("TestService", in this case)

3. The class is to define a single function "process" in which the actual actions of the service are to be made

Using an existing service, like "CounterService.py", is highly recommended for understanding how a service should be structured and laid out.

Further development The server itself should be set to add new services to for operation. What needs more work is the incorporation of a few things:

1. An actual XML scheme for configuration of mulle nodes. This point is not particular to the server, however.
2. Implement ways for the server to make use of the EXIP application to translate the XML that's to be sent into EXI via the command line.
3. Make services corresponding to the functionality you would like to have for configuration and communication with mulles and android devices, respectively.

Webpages and database

Android Mobile Phone application