

CDIO opgave forår 2014 - Del 3

Alternativt projekt

Dataopsamling fra tests af piezoelektriske krystaller.

Morten Hesselbjerg, s017704

Christian Budtz, s134000

Rikke Christina Hansen, s120359

Rúni Egholm Vørmadal, s134004

Martin Nielsen, s123064

Eirik Oterholm Nielsen, s123006

[illegible]

[Indledning](#)

[Status](#)

[Beskrivelse af programflow](#)

[Kravspecifikation](#)

[Use Cases](#)

[Udfør måling af data på batch](#)

[Hent data](#)

[Design](#)

[Implementering](#)

[C# - CDIO Demo.exe](#)

[Java](#)

[Data Transfer Objects](#)

[MainController](#)

[LoginController](#)

[BatchMeasureController](#)

[NewBatchController](#)

[DasyFileReader](#)

[CConnector](#)

[DataBaseController](#)

[SQLConnector](#)

[LoginGui](#)

[BatchMeasureGui](#)

[NewBatchGui](#)

[Begrænsninger i nuværende implementering](#)

[Test](#)

[Udviklingsplan](#)

[Sidste iteration](#)

[Bilag](#)

[Protocol for java to C# socket communication.](#)

[Screenshots](#)

Indledning

Jævnfør aftalen har vi arbejdet på vores alternative projekt og forsøgt at implementere features, der modsvarer de stillede opgaver i vægtprojektet. Særligt har vi forsøgt at arbejde på vores TCP/IP datakommunikation mellem en C# komponent og vores hoved-Java program og forsøgt at sikre robust kommunikation og mulighed for at genoptage arbejdet efter uventet afslutning af dataopsamling. Vi anvender i vores løsning en SQL-server og gemmer kontinuert data i databasen, således at data ikke går tabt undervejs. Lukker man programmet, eller går det ned, kan man loade sin måleserie og fortsætte fra sidste succesfulde måling.

Status

Vi har nu udviklet det grundlæggende programflow i projektet. Data kan opsamles fra to forskellige kilder, hhv. fra et USB-device tilsluttet computeren og fra DasyLab filer genereret af kundens egen software. USB-modulet kan opsamle analoge data fra en tilsluttet mikrometerskrue. Selve tilslutningsprogrammet til USB device er udviklet i C#, da der kun findes drivere til dette. Når C# programmet kører, opstarter man et Java-program og kommunikation med C# programmet foregår så over en TCP/IP forbindelse via vores egen protokol.

Al data gemmes for hver måling i DTU's MySQL database og det er muligt at genoptage en måleserie (batch), der er blevet afbrudt før tid.

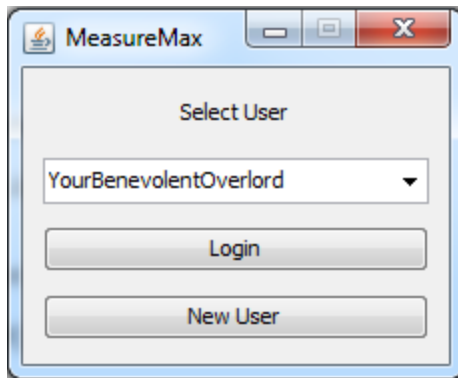
Beskrivelse af programflow

For at få et indtryk af hvad vores projekt aktuelt kan, er det nødvendigt at starte vores C# server, der er konstrueret til at opsamle data fra det tilkoblede USB-device. Denne startes fra filen CDIO_Demo.exe, placeret i projektmappen C#Code\CDIO_Demo uden mccdaq v2.zip.

C# -serveren opretter en TCP/IP serversocket og afventer en forbindelse. Bruger-interaktion og al anden logik ligger i vores java-program. Den vedlagte udgave af C# programmet er en testudgave, der kan returnere tilfældige værdier, når man ikke har tilsluttet det relevante måleapparat. Vores produktionsudgave af C# serveren, nødvendiggør installationen af et 256 MB stort driverprogram, så til test og opgavebedømmelse har vi udviklet vores letvægtsudgave.

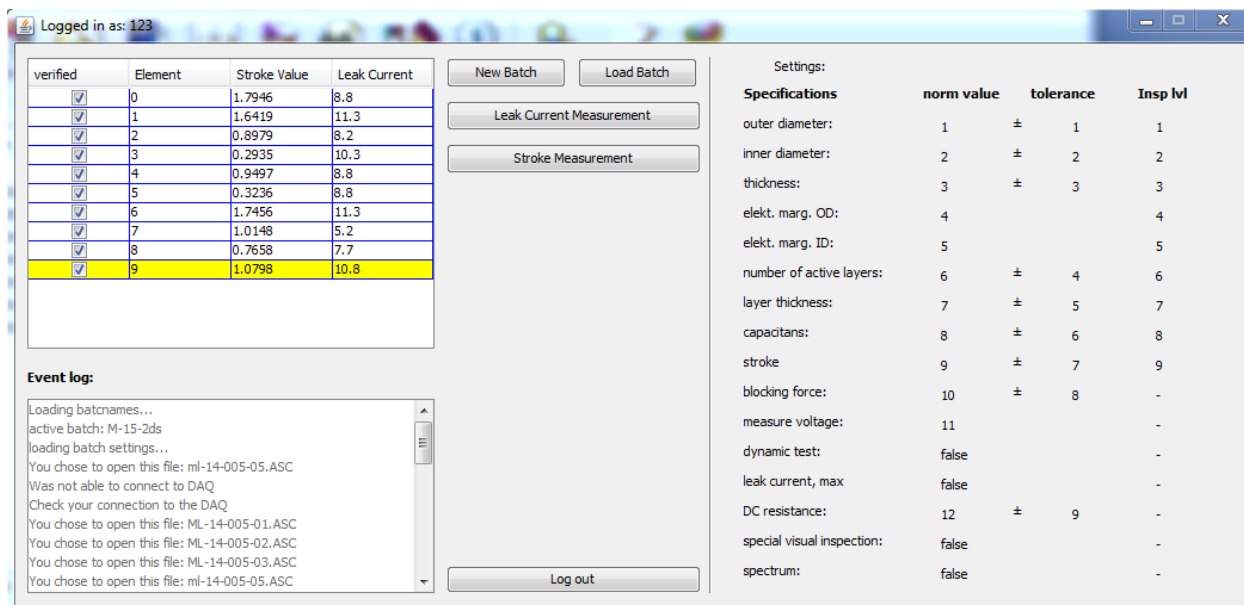
Vores javaprogram startes fra default package fra MeasureMain klassen.

Kører man programmet som det er nu, gives man først muligheden for at logge ind med sit brugernavn, eksisterer brugeren ikke kan man oprette en ny bruger. Brugere er gemt i MySQL-databasen. Vi har endnu ikke gjort det muligt at slette en bruger.



Figur 1: Loginscreen Når man logger ind kan man enten vælge en eksisterende bruger, eller skrive et navn i tekstfeltet og oprette en ny bruger. Brugerlisten indhentes fra SQL-databasen.

Herefter kan man oprette eller loadet et batch fra databasen og lave målinger tilhørende det batch. Hvert batch er associeret med en måleprofil, bestående af et antal settings der ligeledes gemmes i databasen. Målinger kan ikke udføres uden at være knyttet til et batch og en tilhørende måleprofil, men det er ikke nødvendigt at indtaste noget i profilen - man kan altså starte målinger uden at have indtastet settings og disse gemmes blot blanke.



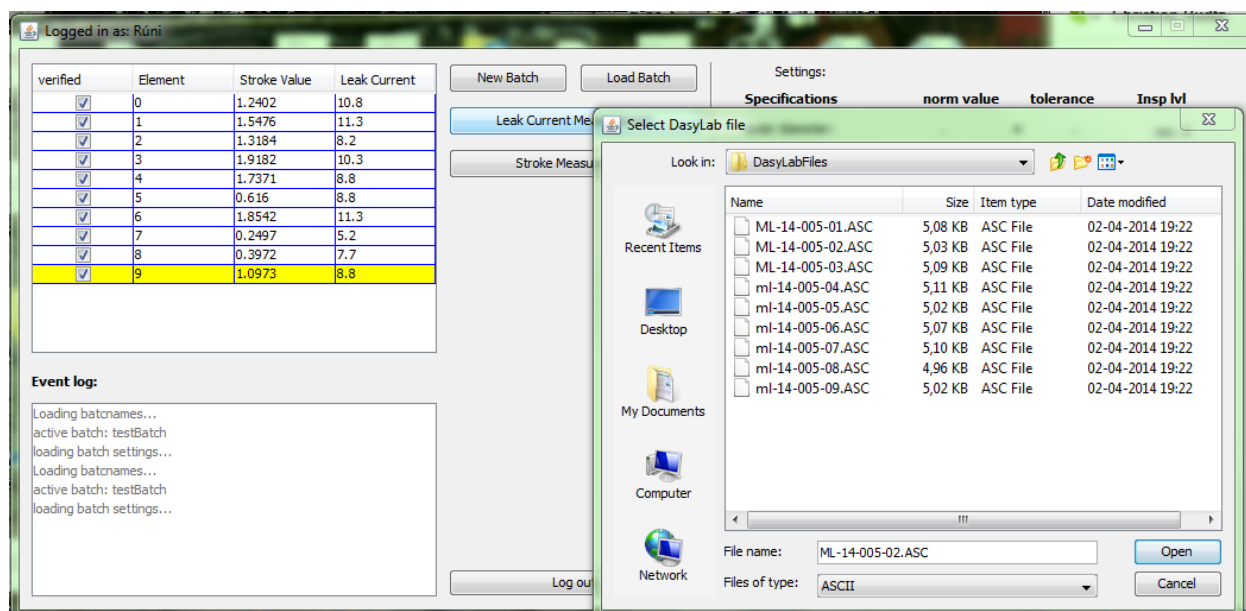
Figur 2: Main Menu. Når man er logget ind kommer man til Main Menuen. Man kan vælge imellem at loadet en eksisterende batch eller at oprette en ny batch. Hvis man vil lave en måling skal man vælge en ASC-fil, som så optræder som et element på listen.

Specifications	norm value	tolerance	inspection level
outer diameter:	<input type="text"/> ±	<input type="text"/>	min. 5
inner diameter:	<input type="text"/> ±	<input type="text"/>	min. 5
thickness:	<input type="text"/> ±	<input type="text"/>	min. 5
elekt. marg. OD:	<input type="text"/>		<input type="text"/>
elekt. marg. ID:	<input type="text"/>		<input type="text"/>
number of active layers:	<input type="text"/> ±	<input type="text"/>	<input type="text"/>
layer thickness:	<input type="text"/> ±	<input type="text"/>	<input type="text"/>
capacitans:	<input type="text"/> ±	<input type="text"/>	min. 10
stroke:	<input type="text"/> ±	<input type="text"/>	AQL 0.65
blocking force:	<input type="text"/> ±	<input type="text"/>	<input type="text"/>
measure voltage:	<input type="text"/>		<input type="text"/>
dynamic test:	<input type="checkbox"/>		<input type="text"/>
leak current, max:	<input type="checkbox"/>		min. 20
DC resistance:	<input type="text"/> ±	<input type="text"/>	<input type="text"/>
special visual inspection:	<input type="checkbox"/>		AQL 0.65
spectrum:	<input type="checkbox"/>		<input type="text"/>

Figur 3: New Batch Menu. Her indtaster man specifikationer for et batch. Man kan også loade gamle batch specifikationer, hvis de er blevet gemt først.

Når batch og måleprofil er fremsøgt, kan målingerne starte. Er der allerede foretaget målinger på den givne batch (eks. ved tidligere login), loades disse og man kan fortsætte målearbejdet. Målingerne bliver så snart de er taget, gemt i databasen.

Måling af lækstrøm (Leak current) foregår ved indlæsning af en fil, som man vælger i en dialogboks (Figur 4). Vi har gjort, at det er kun muligt at hente ascii/asc filer, da det er dette format kunden bruger, og så har vi for brugervenlighed gemt stien til sidste mappe man loadede en fil fra, så man ikke skal browse så langt næste gang. Dette gemmes i en property fil. Når programmet afprøves kan test ASC filer findes i mappen DasyLabFiles i projektmappen. Når filen er valgt, parses denne og data fra filen omsættes til en værdi, der straks gemmes i databasen og vises i målevinduet i øvre ve. hjørne.



Figur 4. Indhentning af data fra DasyLab filer til Leak Current Measurement. Data parses fra filer genereret af DasyLabProgramet.

Stroke målinger foretages direkte ved tryk på Stroke Measurement knappen. Målingen gemmes med det samme i databasen og vises i målevinduet.

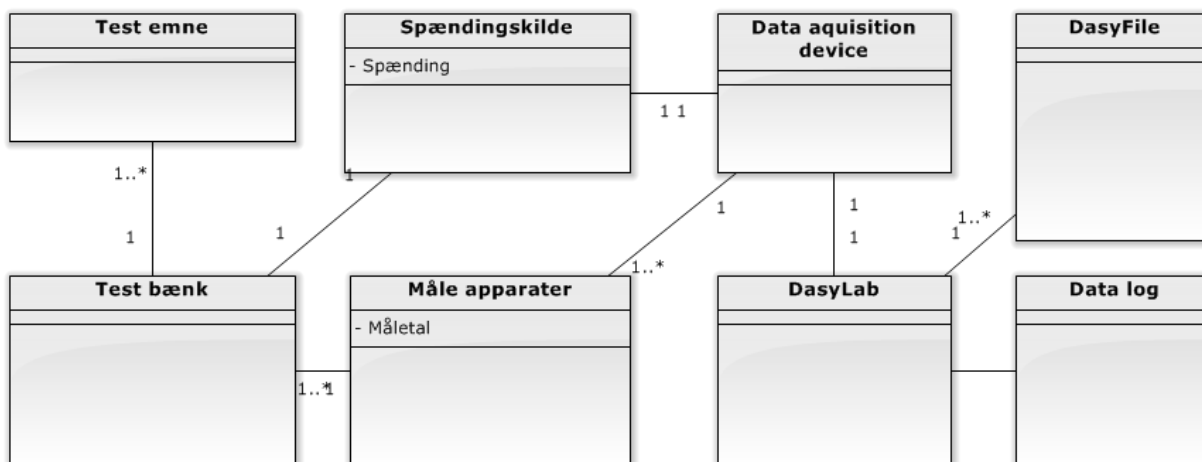
Det er muligt at navngive en måleprofil i databasen og senere hente den frem igen - således at hvis det er indstillinger man bruger ofte, så slipper man for at taste dem alle ind hver gang de skal bruges.

Vores hovedvindue indeholder en log (se evt. Figur2, kan ses nede i venstre hjørne), der udskriver hvad der aktuelt foregår, mht. kommunikation med C# del og indlæsning af DasyLabFiler. Når kommunikationen foregår, udskrives dette til brugeren, og opstår der fejl, udskrives dette ligeledes her. Er C# serveren eks. ikke opstartet eller fejler forbindelsen, gives en besked her - ligesom andre fejl i dataopsamlingen meldes her - således at brugeren kan korrigere problemer. Aktuelt er brugerens valgmuligheder begrænsede - i de næste iterationer planlægger vi at udvide brugergrænsefladen til at understøtte en mere fleksibel arbejdsgang. Som nævnt har vi - for at vi/underviser/hjælpelærer skal kunne teste og demonstrere at forbindelsen virker - i projektet gjort sådan at C# komponenten sender tilfældige værdier mellem 0,5 og 2, da vi ikke har USB device'net koblet til.

Kravspecifikation

Vi har ingen væsentlige ændringer i kravene til vores projekt. Vores måleproces består fortsat af to separate målinger - hhv. en *stroke* måling og en *leak current* måling. Stroke målingen forudsætter at vi opsamler data over USB porten via et data aquisition device. Leak current

målingen foretages i kundens software og genererer en .asc-fil, som vi så indlæser i vores program og trækker relevante data ud af.



Domæne model. Ingen væsentlige ændringer.

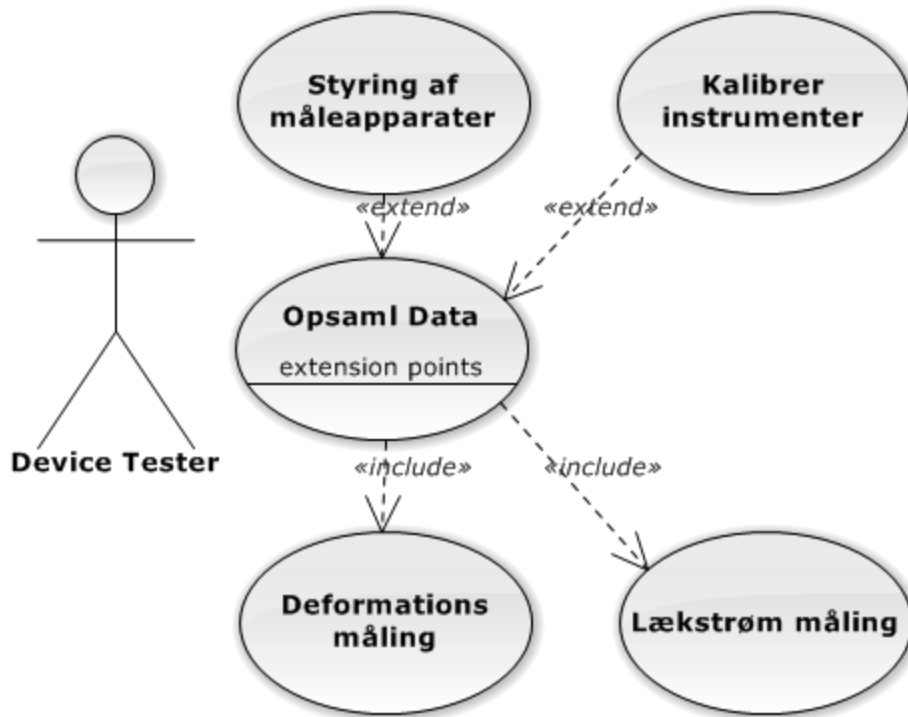
Use Cases

Main use cases er i væsentlige træk uændret, men er opdateret til at reflektere den løbende udvikling i implementeringen.

Udfør måling af data på batch

Preconditions:

Der er fremfundet et batch af piezoelektriske elementer der skal testes. Testemnerne er monteret i testbænken. Måleinstrumenter (mikrometerskrue) tilsluttet korrekte porte på USB-Data Aquisition Device'et. Måleinstrumenter er tændt. Alle nødvendige apparatindstillinger er rigtige (senere evt. implementering af styring fra PC).



Aktør

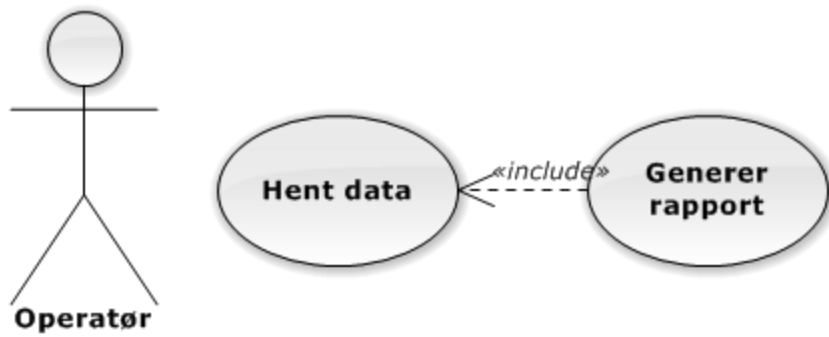
1. Opstarter program (incl. C# Server)
2. Logger ind.
3. Opretter Batch måling
4. Opretter en profil med måleindstillinger.
5. Starter måling af Leak Current
6. Godkender data eller gentager måling
7. Starter måling af Stroke.
8. Godkender data eller gentager måling
9. Gentager 4-7 indtil alle emner er målt
10. Afslutter Batchmåling og logger ud

System

1. Modtager nødvendige brugerindstillinger. C# server opretter kontakt til USB -DAQ.
2. Modtager Brugerdata
3. Opretter Batch i database
4. Gemmer ny måleprofil.
5. Opsamler og lagrer data i DasyFile. DasyFile Indlæses og data for lækstrøm præsenteres for bruger. Gemmer data i database.
6. Sletter evt. tidligere måling.
7. Sender besked om måleserie til C# server, der igen henter måledata fra mikrometerskrue. Gemmer data i database
8. Sletter evt. tidligere måling.
9. ds.
10. Logger bruger af

Extensions:

- 1a: Apparater kalibreres.
- 4a Alle måleapparater tilsluttes parallelt og softwaren kontrollerer spændingskilde og måleapparatur automatisk, så alle data logges i en arbejdsgang.
- 5/7b: Data lagres i en enten lokal eller cloud baseret SQL database til senere udtræk.



Hent data

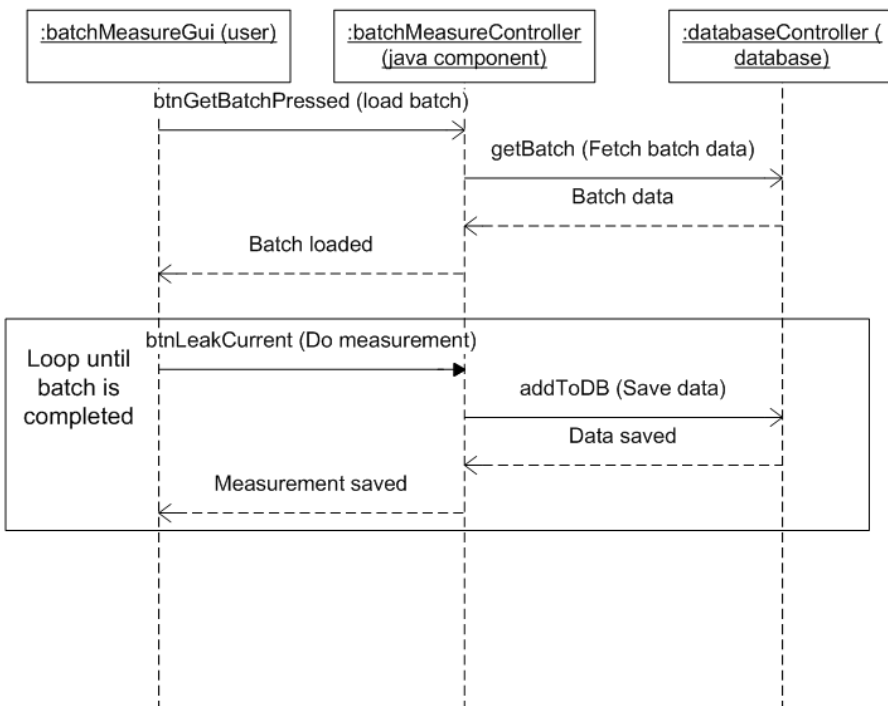
Preconditions: Data er opsamlet og lagret.

Aktør 1. Starter browser og tilgår web-side 2. Vælger udtræk af data	System 1. Modtager login 2. Præsenterer data.
---	--

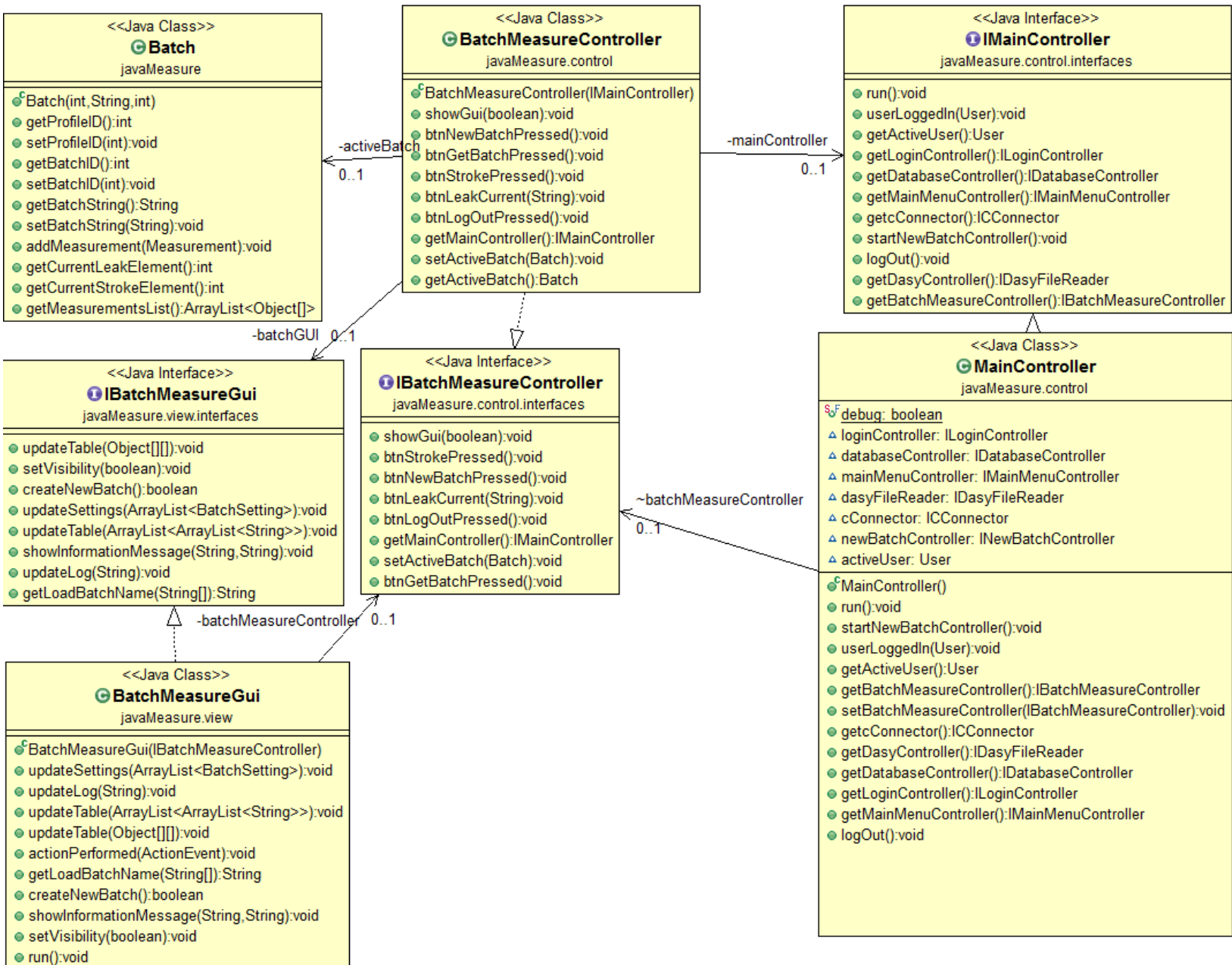
Extensions:

2a) Forskellige brugerdefinerede rapporter/udtræk

Design

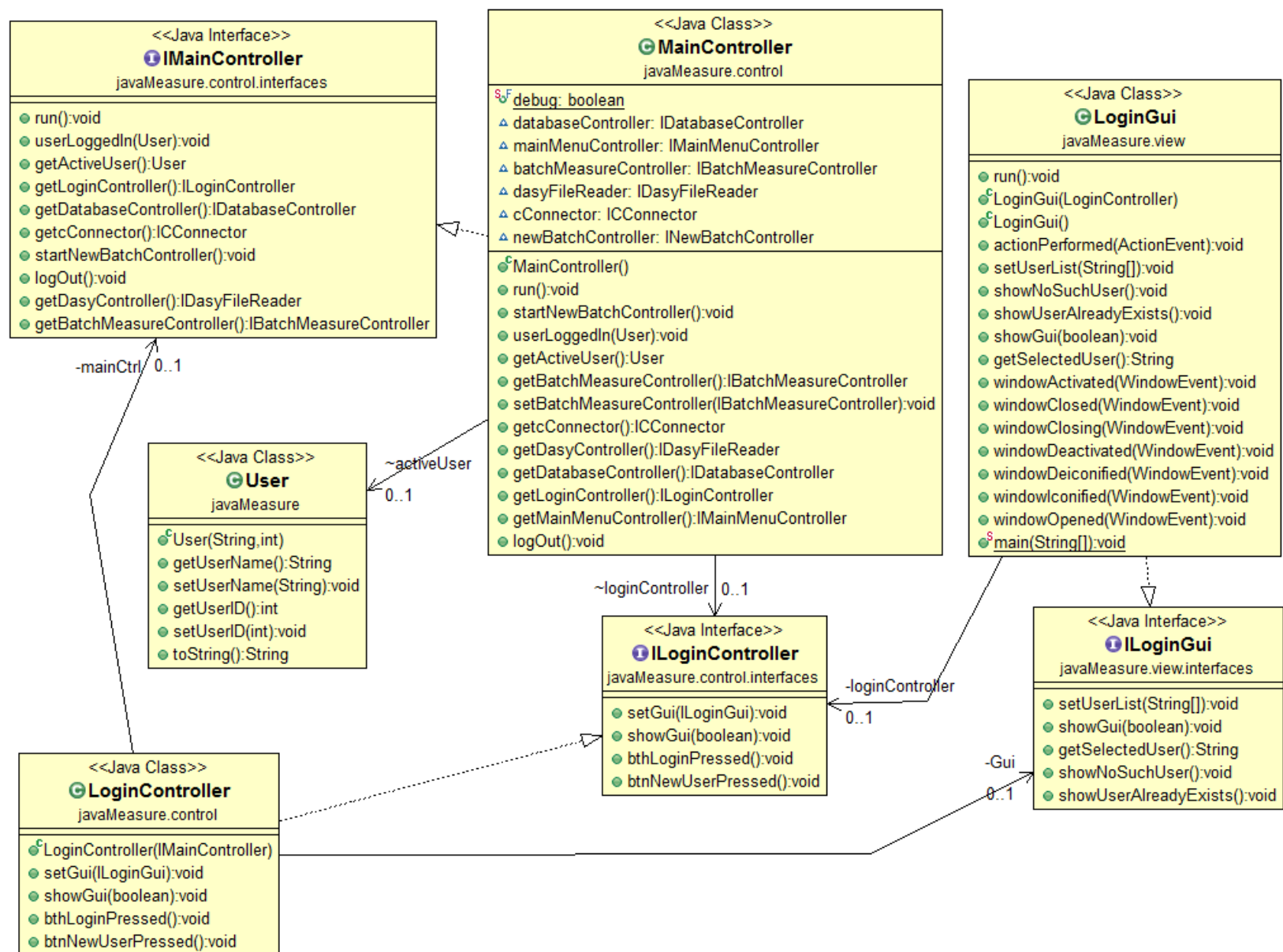


System sekvens diagram, der viser forløbet i systemet, når brugeren udfører en måling.



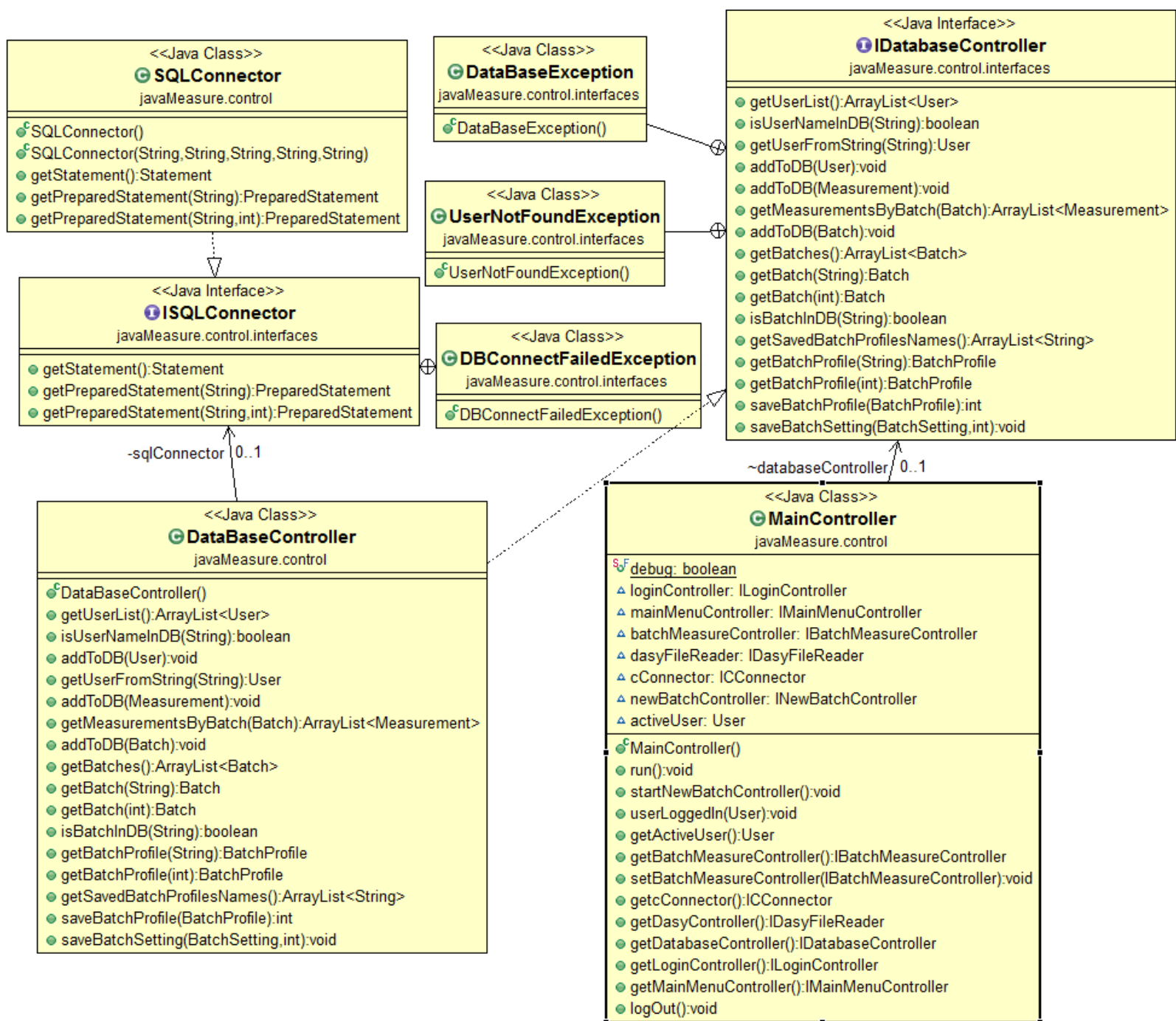
C.D. 1

Her ser vi hvordan hvordan delen af programmet der holder styr på batches er sat op. MainControlleren starter til BatchMeasureController, der starter BatchMeasureGui, og indeholder dens action listener. Batch ligger også i BatchMeasureController. Koden til at hente batches fra databasen ligger i C.D.3, fra DAQ og Daysylab i C.D.4. Koden til at lave nye batches ligger i C.D.5



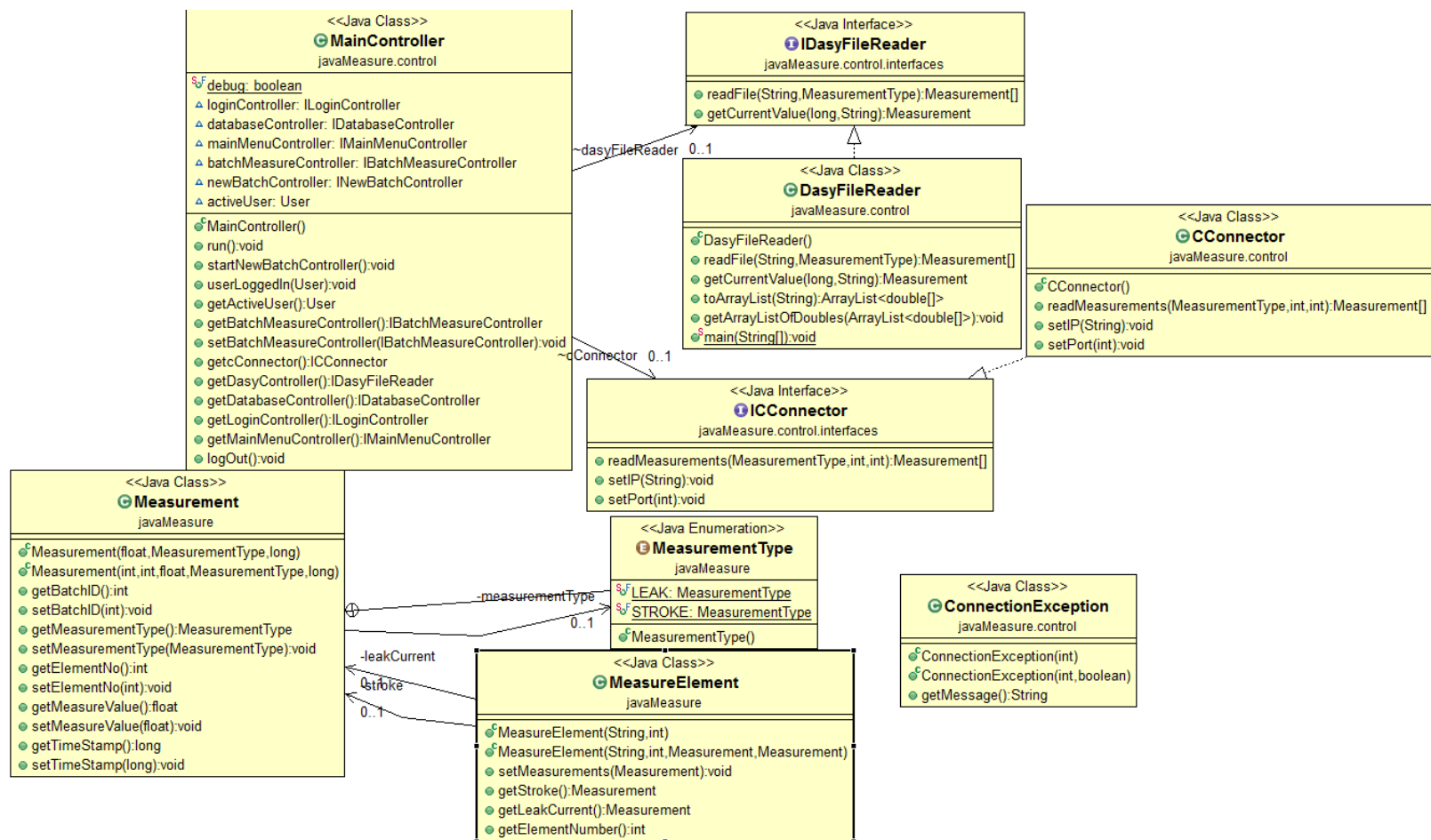
C.D.2

Her ser vi bruger og login delen af programmet. MainController opretter en User, en LoginController. LoginController opretter en ILoginGui.



C.D.3

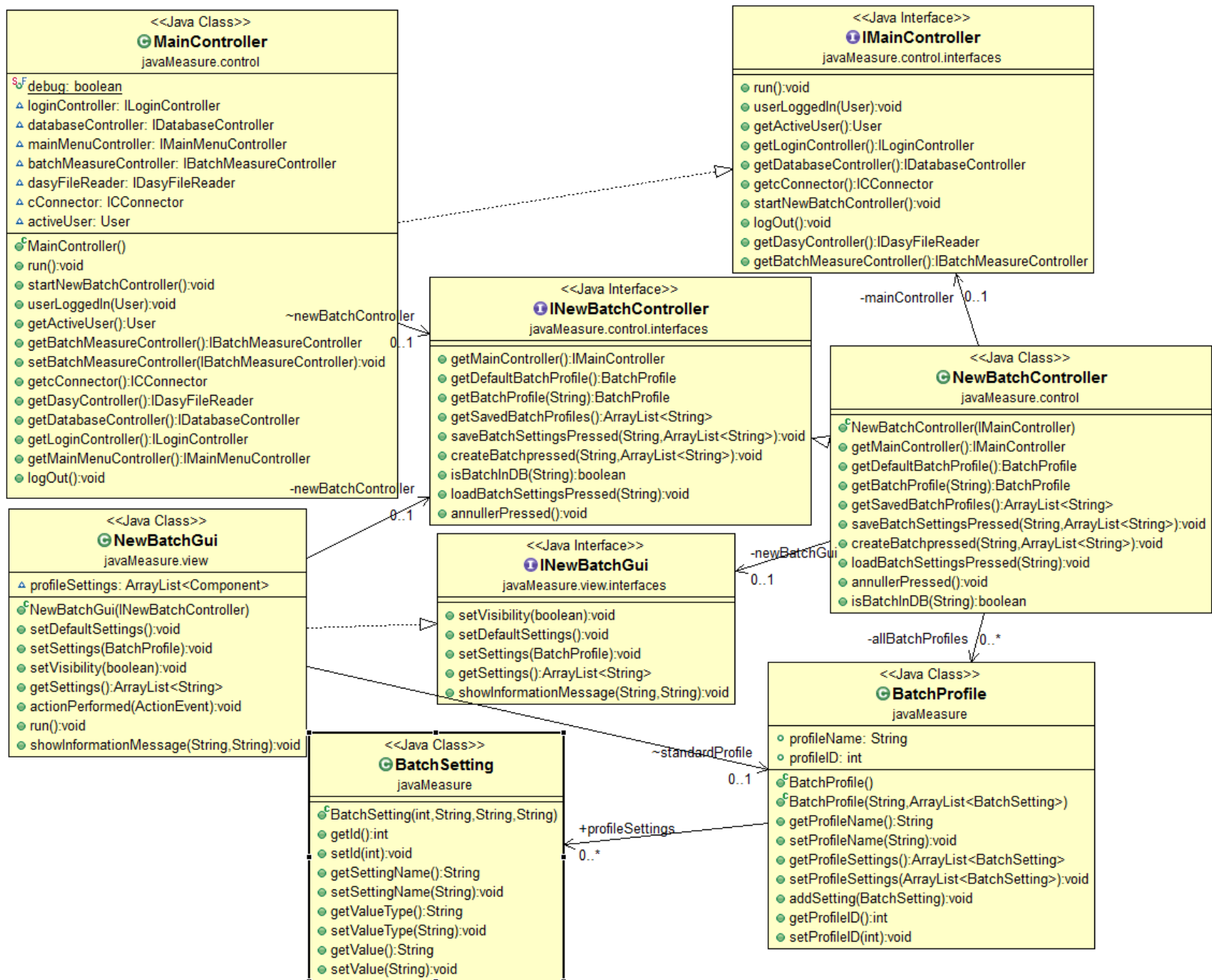
Her ser vi at Main controlleren har styr på DatabaseControlleren, der sender beskæder videre til vores SQL Connector, denne sender information frem og tilbage fra databasen.



C.D.4

Her ser vi at MainControlleren kan sende bede ænten DasyFileReader, eller CConnector om at læse daysy filer, eller af daq'en respectivt. Dette er hvordan vi får nye measurements ind i programmet.

Vi ser også Measurement, MeasurementType og MeasureElement, der ikke er blevet forbundet bedre i programmet i øjeblikket, men vil blive lavet af DasyFileReader og CConnector.



C.D.5

Her ser vi at maincontrolleren har styr på newbatchcontroller, der bliver brugt når vi lægger nye batches i databasen. Den har styr på New pbatchGUI og Batchprofile, der har styr på Batch setting.

Vi har forsøgt at holde vores kodebase opdelt i Data transfer objects (DTO'er), Controllers og GUI kode. Vi har 5 DTO klasser - Batch, BatchProfile, BatchSetting, Measurement og User. Desuden har vi en utility-klasse - PropertyHelper, der anvendes til at hente programindstillinger fra en properties fil.

Det overordnede programflow håndteres af MainController, der fungerer som facadecontroller og delegerer ansvar videre til relevante sub controllers.

Hvert View har en egen controller klasse - BatchMeasureController, LoginController, MainMenuController og NewBatchController.

Indhentning af måledata sker i CConnector og DasyFileReader. Når data skal gemmes/hentes i databasen håndteres det af DataBaseController og SQLConnector.

For at håndtere en robust TCP/IP forbindelse har vi ssepareret forbindelse til C# i en særskilt java klasse - CConnector, der opretter en TCP/IP Socket og skaber forbindelse til vores C# komponent. Aktuelt kan C#-programmet returnere enten data fra 2 analoge porte, eller testdata, da vi ikke har fået fat i den mikrometerskrue som vores kunde bruger. Vi har lavet vores egen protokol til at håndtere fejl i kommunikationen, og har identificeret 5 mulige fejlmeddelelser som kan ses under bilag med titlen "Protocol for java to C# socket communication".

For at sikre robusthed i tilfælde af nedbrud eller præmatur afslutning af måleserie gemmes data efter hver enkelt måling. Vi bruger aktuelt en forbindelse til den mySQL database vi har fået stillet til rådighed i Diskret matematik og databaser. For at få fremdrift i projektet har vi kun én overordnet exception (DatabaseException) som bliver kastet hvis noget går galt - enten med SQL-syntaks eller med forbindelsen til serveren. Dette vil blive subklasset senere for en mere robust fejlhåndtering. En praktisk ekstra feature, kunne være muligheden for at gemme data lokalt, indtil database forbindelsen igen kan oprettes.

Vi har udspaltet håndteringen af SQL-forbindelsen til en separat-klasse, da vedligehold af forbindelse og fejl-håndtering, er så kompliceret at det bedst overskues i sin egen klasse.

Implementering

C# - CDIO_Demo.exe

C# Serveren der afventer forbindelse fra Java programmet- Når connection oprettes og der modtages en besked over TCP/IP enten aflæses målinger fra USB devicet eller returneres test data.

Java

Data Transfer Objects

Som nævnt har vi 5 DTO'er - Batch, BatchProfile, BatchSetting, Measurement og User. Batch har et ID, et navn og en ArrayList af Measurements samt metoder til at tilgå disse. På samme

måde har BatchProfile et ID og en liste af tilhørende BatchSettings. BatchSettings har et ID, navn, valueType og value. MeasureElement repræsenterer det reelle måleelement og har 2 tilhørende Measurement objekter samt et element nummer. Measurement udgør de reelle måledata. de kan enten have den enumererede type LEAK eller STROKE. De har et BatchID og et elementNo, der identificerer hvilket Batch og hvilket element de er udført på og desuden en measureValue, der repræsenterer den reelle måleværdi og et timeStamp, der angiver hvornår målingen er udført.

MainController

Instantierer alle GUIControllerne, CConnector, DasyFileReader og DataBaseController samt holder styr på den aktive bruger i programmet. Delegerer ved programstart ansvaret til LoginController i metoden run(). Modtager fra LoginController den aktive bruger i metoden userLoggedIn(), hvorefter LoginGUI skjules og BatchMeasureController instantieres. Når brugeren vælger at oprette et nyt batch køres startNewBatchController(), NewBatchController vises og BatchController skjules. Håndterer logout i metoden logOut(). Sætter desuden udseendet af GUI'erne til at følge systemstandard med setLookAndFeel().

LoginController

Instantierer en LoginGui, henter en brugerliste fra DataBaseControlleren, hvorefter den sætter GUI'en til at vise den. Har to metoder btnLoginPressed() og btnNewUserPressed() der håndterer tryk på de to knapper på gui'en ved evt. at oprette en ny bruger i databasen og derefter fortælle mainControlleren at en bruger logger ind (mainCtrl.userLoggedIn())

BatchMeasureController

Instantierer en BatchMeasureGui og håndterer bruger inputs. btnNewBatchPressed() beder maincontroller om at instantiere en newBatchController. btnGetBatchPressed() henter en batch fra databasen, således at man kan fortsætte fra en tidligere måleserie. btnStrokePressed() henter en strokemåling fra CConnectoren og sender den til databaseControlleren. Desuden opdateres GUI'en. btnLeakCurrent() beder dasyController om at parse den fil der er valgt aktuelt og udhenter værdien til et særligt tidspunkt, hvorefter den gemmes i databasen. btnLogOutPressed() beder maincontrolleren om at logge brugeren ud.

NewBatchController

Håndterer oprettelsen af nye batches og måleprofiler. saveBatchSettingsPressed() gemmer de værdier, der er indtastet aktuelt i tekstfelterne i NewBatchGui'en. Der oprettes først en batchProfile og derefter tilføjes batchSettings en af gangen, hvorefter batchProfilen gemmes. createBatchPressed gemmer først de batchSettings der er valgt i en unavngiven BatchProfile. databasen returnerer profilens ID og en Batch oprettes med det tilsvarende profil id. Herefter gemmes batchen i databasen. loadBatchSettingsPressed() henter settings fra databasen svarende til det profilnavn der er valgt i gui'en og beder gui'en om at opdatere settings.

Vi har desuden følgende 4 controller klasser, der sørger for data input/output:

DasyFileReader

DasyFileReader læser filer fra DASYLab, gemt i .ASC format. Den parser ASC-filen og udlæser de measurements som er gemt i filen. Herefter kan de relevante measurements udvælges og gemmes i databasen. toArrayList() indlæser en fil fra DasyLabProgrammet. Den søger filen igennem efter målingerne og returnerer dem i en arrayList som par af hhv måleværdi og timestamp. readFile() anvender toArrayList, til at indlæse filen og returnerer et Measurement[] med målingerne - timestampet omregnes til milisekunder.

CConnector

CConnector er den klasse, der taler mellem vores java og C# del af projektet. Vi har lavet en forbindelse, og flyttet data over. I øjeblikket laver vi en request, og får data'en tilbage i en string. Vi håndterer også flere fejl, som fx. der er ingen forbindelse. Den fulde protokol for kommunikation mellem Java og C# delen er vedlagt som bilag.

Når connectoren instantieres sættes IP'en automatisk til computerens egen IP. Alternativt kan man kalde metoden setIP() for at sætte en alternativ ip. Når readMeasurements() metoden kaldes, sendes en request til C# delen. Herefter ventes på et svar fra C#-serveren, ellers times ud efter et givet interval. Enten returnerer metoden værdierne fra C#, ellers kastes en exception. Forbindelsen termineres med disconnect() metoden.

DataBaseController

DataBaseController (DBC) er mellemlid for databaseoperationer. Vi har så vidt muligt forsøgt at kode controlleren således at den modtager eller leverer et data transfer object (DTO) eller en liste over tilgængelige objekter. Når et DTO overgives til DBC serialiseres det til et eller flere SQL statements der kan eksekveres på en SQL, server. På samme vis kan et objekt deserialiseres ved SQL-statements. Desuden er det muligt at bede om en liste over tilgængeligt objekter af en given slags fra databasen.

Databasen indeholder brugere, måleindstillinger, og målinger. Queries eksekveres via SQLConnector klassen.

SQLConnector

Vedligeholder forbindelse til SQL-serveren. Da vi bruger en remote SQL-server (aktuelt DTU's) og vi kan have flere SQL-queries hurtigt efter hinanden, lukker vi ikke forbindelsen mellem hvert query, da hver forbindelse tager 1/2-1 sekund at etablere. I stedet testes forbindelsen før et query eksekveres og fejler forbindelsen, genoprettes forbindelsen så vidt muligt, ellers kastes en exception.

LoginGui

Implementerer ILoginGui og dermed ActionListener. Har en combobox og 2 knapper. Når der trykkes på login/NewUser aktiveres actionPerformed og derigennem btnLoginPressed() eller btnNewUserPressed() i LoginController. Udstiller metoder til at vise en brugerliste i Comboboxen, til at vise forskellige fejlmeddelelser og til at hente den aktuelle tekst i

comboboxen.

BatchMeasureGui

Er 'hovedviewet' med en JTable til at vise målinger, et tekstfelt til at vise statusbeskeder. 5 knapper til at håndtere oprettelse af batch/load batch og 2 målinger samt logout. Har desuden et dynamisk område, hvor 21 settings tilhørende batchet kan vises.

Udstiller metoden updateSettings() til at opdatere settingslisten i højre side af skærmen. updateLog() bruges til at outputte til logvinduet. updateTable() opdaterer målingerne der vises i JTable'en. getLoadBatchName() viser en JOptionPane med en combobox til at vælge en batch at load. showInformationMessage() viser en popup med en given tekst.

NewBatchGui

Viser 4 knapper til at oprette et batch, load/save af settings og annuller. setSettings() opdaterer de viste settings i gui'en. NewBatchGui har elementer af control, der skal flyttes til controller klassen når tiden tillader. I actionPerformed() håndteres oprettelsen af et nyt batch - createBatch, load af settings fra databasen - loadSettings og save af settings - saveBatchSettings.

Begrænsninger i nuværende implementering

C# programmet skal aktuelt startes manuelt - for brugervenlighedens skyld planlægger vi at det skal startes automatisk. Log-vinduet er ikke up to date, og der kommer intermitterende meddelelser, som ikke stemmer overens med fejltilstanden. Målingerne skal foretages konsekutivt og det er aktuelt ikke muligt at kassere enkeltmålinger. Det er ikke muligt at bruge programmet uden en aktiv SQL-forbindelse. C# serveren skal eksekveres på samme maskine som Java-programmet.

Test

Vi har udført både brugertest af programmet i sin helhed og desuden unittestet de komplicerede datakommunikationsklasser.

Da vi har concurrency på flere niveauer i vores program, har vi også mange muligheder for at få fejl, når forskellige tråde ikke er klar til at håndtere interaktion med andre tråde. Således opstår en null-pointer exception, hvis det lykkes brugeren at trykke på login-knappen, før der er hentet data fra SQL-databasen. Arbejder man på DTU, ses denne fejl ikke, da data hentes for praktiske formål instant, men her er mulighed for at forbedre robustheden. Samme type af problemer kan provokeres andre steder i programmet, men giver ikke anledning til nedbrud - det er muligt at fortsætte med at anvende programmet uden at data korrumpes.

De enkelte metoder til vedligehold og oprettelse af SQL forbindelsen i SQL-connectoren er testet ved test driver metoder i klassen TestMySQLConnection.

Vi har oprettet en JUnit test der tester DatabaseControlleren, ved at tilføje en ny bruger til databasen, søge efter den for at sikre at den er indsat korrekt, og til sidst slette test-brugeren igen.

Udviklingsplan

Vi har fulgt vores plan nogenlunde, og derfor ser vores plan for sidste iteration stort set ud som den gjorde i starten. Vores GUI mangler stadig knapper og labels til de resterende funktioner, der også skal implementeres. Funktionalitet, som vi i dialog med Noliac har tilføjet projektbeskrivelsen, er indsat i planen. Vi har en HTML Mockup, udviklet til at understøtte dialogen med Noliac, men der er ingen reel funktionalitet i denne endnu.

Sidste iteration

Bugfixing, optimering - udvikling af web-brugergrænseflade og evt. udvidelse af implementering

1. Kravspecificering
 - a. Løbende tæt dialog med Noliac, mhp. validering af løsningsforslag og prioritering af udviklingsopgaver.
2. Design
 - a. Nuværende design opdateres efter behov.
3. Implementering
 - a. Optimering af kodebase/GUI
 - i. Test af grænsetilfælde
 - ii. Inputvalidering
 - iii. Bedre exception handling
 - iv. Bedre/mere robust håndtering af SQL-connection
 - b. Manglende funktionalitet i Javaprogram
 - i. Slet bruger(e)
 - ii. Superbruger funktionalitet
 - iii. Måleprofil redigering
 - iv. Slette/redigere gemte profiler
 - v. Verifikation af måledata - mulighed for at kassere fejlagtige måledata
 - vi. Fleksibilitet af måleflow - så man evt. kan bestemme hvilket element, der bliver målt på
 - vii. Program setup - mulighed for at konfigurere eks IP-adresser, måleporte, standard profiler.
 - viii. Gemt data, samles i én property fil (nu gemmes data i flere properties)
 - c. Implementering af SQL/JSP i HTML/CSS grænseflade
 - i. Grundlæggende funktionalitet til at udhente de indsamlede måledata.
4. Test.
 - a. Videre Unit testing af essentielle databehandlingsklasser
 - b. Gerne brugertest med Noliac
5. Evt:
 - a. Implementering af 'custom' rapportgenerering.
 - b. Måske rapportopslag for kundens kunder efter batchnr?
 - c. Statistisk udregning af sample size for at garantere en vis tolerance med en

- given sandsynlighed.
- d. Brugerstyring
- e. Større robusthed af database-interface
 - i. evt. offline mode og synkroniserings funktionalitet (lav prioritet).

Mål: Optimering af løsning, evt. implementering af flere use cases. Kunden skal kunne bruge programmet i drift i en basal udgave.

Slut: 11/5

Bilag

Protocol for java to C# socket communication.

The C# component should be set to listen, and java will connect when the program starts. By default, java will try to connect to localhost port 4567, but this can be overloaded.

When the connection has been established, the following requests and responses are allowed.

Request 1:

Java sends a string to C# in the following format: port+";"+number+";"+period+";<EOF>" where port is the desired port on the DAQ to read from, number is the desired amount of readings, and period is the time between readings
port, number and period are ints.

Response 1:

C# then returns a string in the format: data+";"+timeStamp+";"
this is repeated number times.

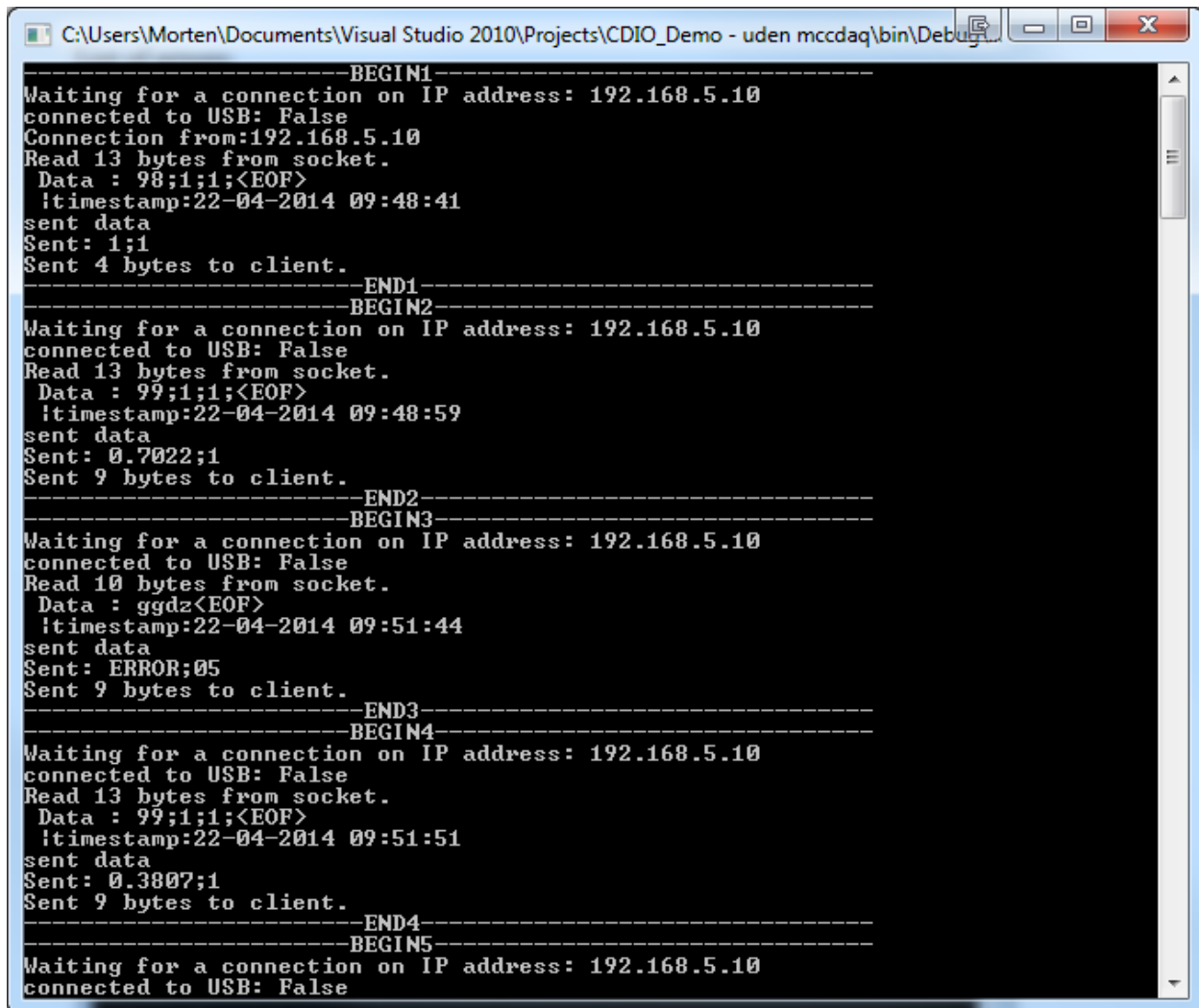
Data is the reading, a float value, and timestamp is the time since the previous reading(long).

If an error has occurred, C# returns a string in the format: ERROR+";"+number+";"
error is a string that reads error, and number is an int, that describes what error has occurred.

List of errors:

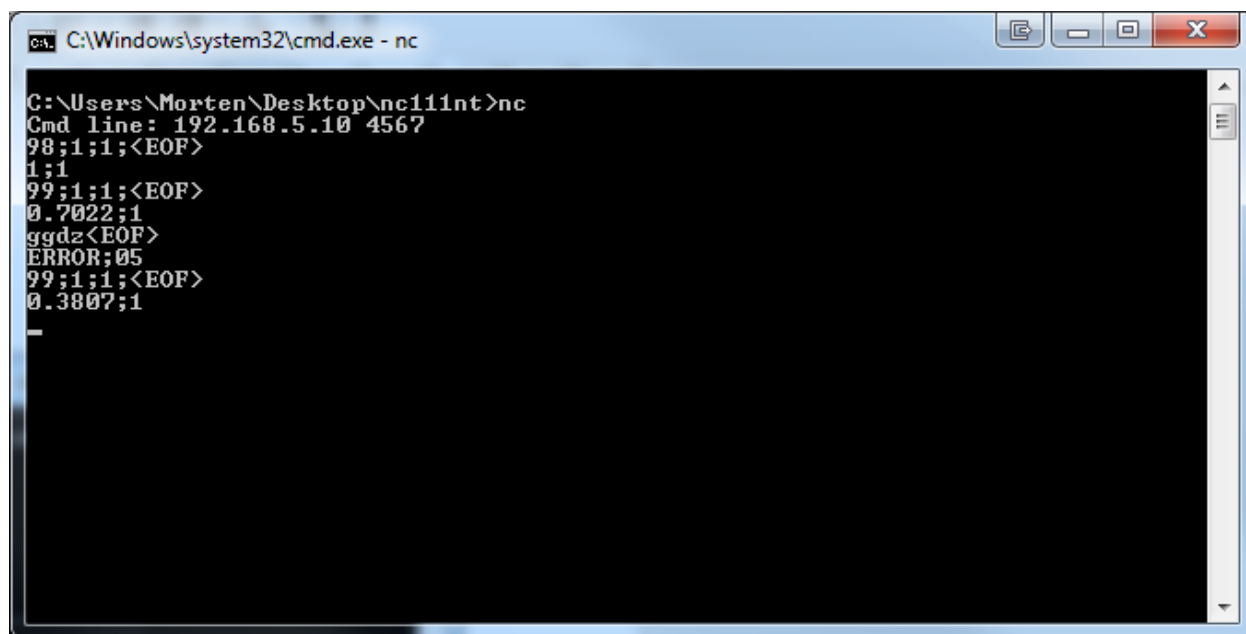
- 01: DAQ not connected or malfunctioning
- 02: Illegal port
- 03: Illegal number of measurements
- 04: Illegal period
- 05: Message does not conform to protocol

Screenshots



```
C:\Users\Morten\Documents\Visual Studio 2010\Projects\CDIO_Demo - uden mccdaq\bin\Debug
-----BEGIN1-----
Waiting for a connection on IP address: 192.168.5.10
connected to USB: False
Connection from:192.168.5.10
Read 13 bytes from socket.
Data : 98;1;1;<EOF>
!timestamp:22-04-2014 09:48:41
sent data
Sent: 1;1
Sent 4 bytes to client.
-----END1-----
-----BEGIN2-----
Waiting for a connection on IP address: 192.168.5.10
connected to USB: False
Read 13 bytes from socket.
Data : 99;1;1;<EOF>
!timestamp:22-04-2014 09:48:59
sent data
Sent: 0.7022;1
Sent 9 bytes to client.
-----END2-----
-----BEGIN3-----
Waiting for a connection on IP address: 192.168.5.10
connected to USB: False
Read 10 bytes from socket.
Data : ggdz<EOF>
!timestamp:22-04-2014 09:51:44
sent data
Sent: ERROR;05
Sent 9 bytes to client.
-----END3-----
-----BEGIN4-----
Waiting for a connection on IP address: 192.168.5.10
connected to USB: False
Read 13 bytes from socket.
Data : 99;1;1;<EOF>
!timestamp:22-04-2014 09:51:51
sent data
Sent: 0.3807;1
Sent 9 bytes to client.
-----END4-----
-----BEGIN5-----
Waiting for a connection on IP address: 192.168.5.10
connected to USB: False
```

Bilag 5: C# komponent. Mellem linjerne Begin3 og End3 vises eksempel på fejlkommunikation



A screenshot of a Windows command prompt window. The title bar reads "C:\Windows\system32\cmd.exe - nc". The window content shows a Netcat listener session. The prompt is "C:\Users\Morten\Desktop\nc111nt>nc". The first line of output is "Cmd line: 192.168.5.10 4567". Subsequent lines show a series of data exchanges: "98;1;1;<EOF>", "1;1", "99;1;1;<EOF>", "0.7022;1", "ggdz<EOF>", "ERROR;05", "99;1;1;<EOF>", and "0.3807;1". A single hyphen "-" is on the line following the last output.

```
C:\Windows\system32\cmd.exe - nc
C:\Users\Morten\Desktop\nc111nt>nc
Cmd line: 192.168.5.10 4567
98;1;1;<EOF>
1;1
99;1;1;<EOF>
0.7022;1
ggdz<EOF>
ERROR;05
99;1;1;<EOF>
0.3807;1
-
```

Bilag 6: Screenshot af kommunikation sendt til C# komponent (via Netcat)