

CDIO opgave forår 2014

*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

CDIO_51_alternativ						
Time-regnskab	Ver. 2008-09-03					Bemærk
Dato	Deltager	Design	Impl.	Test	Dok.	Andet
19/2	Christian				3	3 Kundemøde
	Morten				3	3 -
	Rikke				3	3 -
	Martin				3	3 -
	Rúni				3	3 -
	Eirik				3	3 -
24/2	Christian				2	2 Projektplanlægning
	Morten				2	2
	Rikke				2	2
	Martin				2	2
	Rúni				2	2
	Eirik				2	2
25/2	Rúni				0	
	Christian				2	2 RoadMap
26/2	Christian				2	2 RoadMap
1/3/2014	Christian	2				2
2/3/2014	Christian	1		2		3
						0
1/3/2014	Morten		2			2 C# kode
2/3/2014	Morten		2	2		4 C# kode
2/3/2014	Rikke			2		2
						0
	Sum	3	4	2	2	34 47

Indholdsfortegnelse

[Indholdsfortegnelse](#)

[Indledning](#)

[Kravspecifikation](#)

[Domænemodel](#)

[Main use cases:](#)

[Opsaml data](#)

[Hent data](#)

[BCE-diagrammer](#)

[Udviklingsplan](#)

[Kritisk use case:](#)

[Tidsplan](#)

[Projektstatus](#)

[Plan](#)

Indledning

Vi har som projekt valgt en alternativ opgave. Opgaven løses for firmaet Noliac, der fremstiller piezokeramiske elementer til en lang række industrier. Noliac udfører kvalitetskontrol på en delmængde af de producerede elementer, for at sikre at de lever op til specifikationerne. Disse tests/målinger inkluderer bl.a.:

- Måling af udbøjning - dvs. den spænding der genereres, når elementet udsættes for en deformation
- Måling af geometri
- Måling af lækstrøm

Målingerne udføres en ad gangen og de opsamlede data indtastes manuelt i et Excel-ark, for derefter at blive brugt til en batch-rapport og statistik.

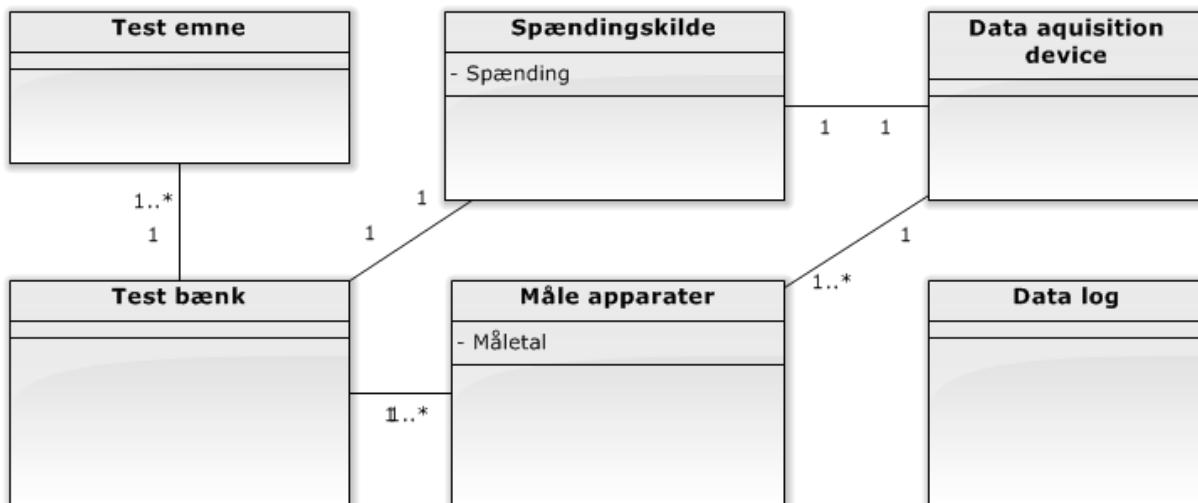
Noliac ønsker at få måleprocessen automatiseret i videst mulige omfang, således at det dels kan sikres at kvalitetsprocessen foregår hurtigere, men også så den udføres ensartet.

Kravspecifikation

Forsøgsopstillingen består af en testopstilling, hvor emnet der skal testes fastgøres i en testbænk. Elementet tilsluttes efter tur tre måleapparater, der er tilsluttet en computer via et USB-Data Aquisition Device (USB-DAQ). Elementet påvirkes med en spænding fra en spændingskilde.

Data opsamles via et program på computeren med det tilsluttede USB-DAQ. Data indføres manuelt på et A4 ark, med en påtrykt protokol. Herefter samles data i et excel-ark og en rapport over emnernes kvalitet og specifikationer generes.

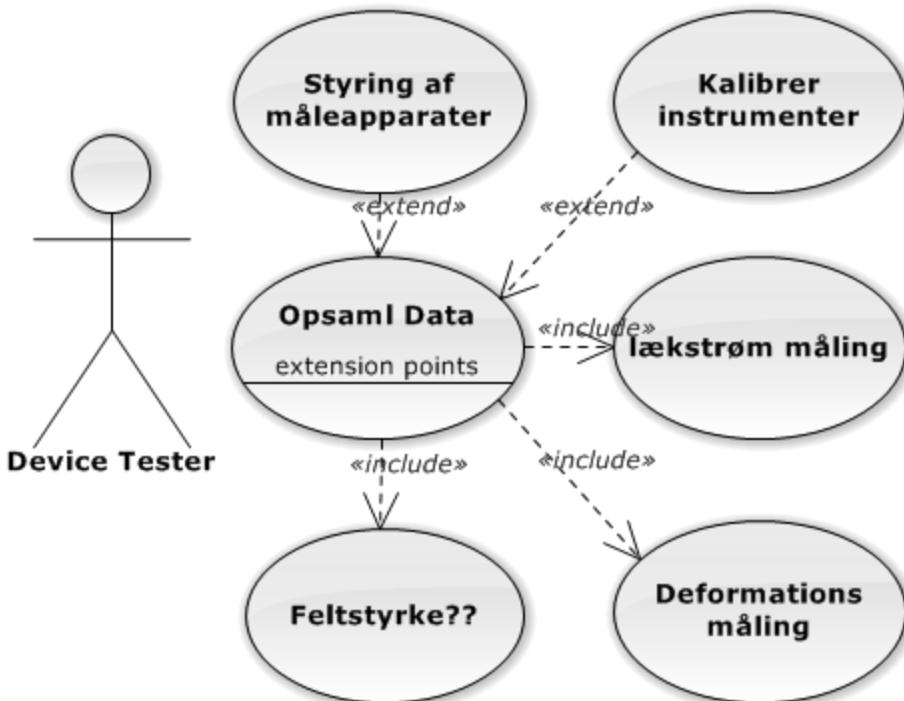
Domænemodel



Main use cases:

Vi har identificeret 2 main use cases som vi har modelleret med mulige extensions:

- Opsaml data
 - Kalibrer instrumenter (extender opsaml data)
- Hent data
 - Generer rapport



OpsamI data

Preconditions:

Testemnet er monteret i testbænken. Måleinstrumenter tilsluttet korrekte porte på USB-Data Aquisition Device'et. Måleinstrumenter er tændt. Alle nødvendige indstillinger er rigtige (senere evt. implementering af styring fra PC).

Aktør	System
1. Opstarter program	1. Modtager nødvendige brugerindstillinger Tilslutter USB -DAQ.
2. Starter måling	2. Opsamler og lagrer data

Extensions:

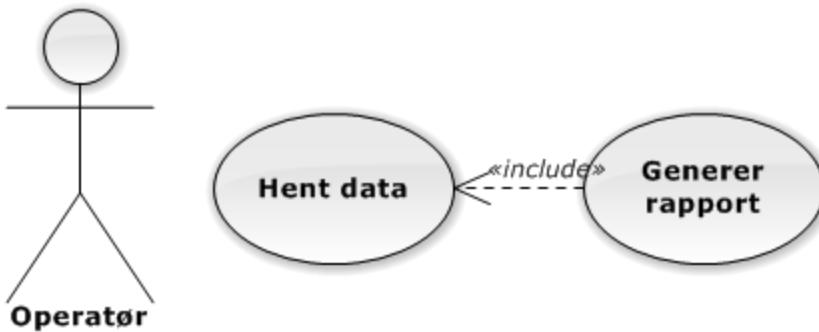
1a: Apparater kalibreres.

2a: Flere forskellige apparater tilsluttes efter tur og data logges separat for hver måling.

i) Alle måleapparater tilsluttes parallelt og softwaren kontrollerer spændingskilde og måleapparatur automatisk, så alle data logges i en arbejdsgang.

2b: Data lagres i en enten lokal eller cloud baseret SQL database til senere udtræk.

2c: Operatøren præsenteres for data og kan godkende/afvise testdata.



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:

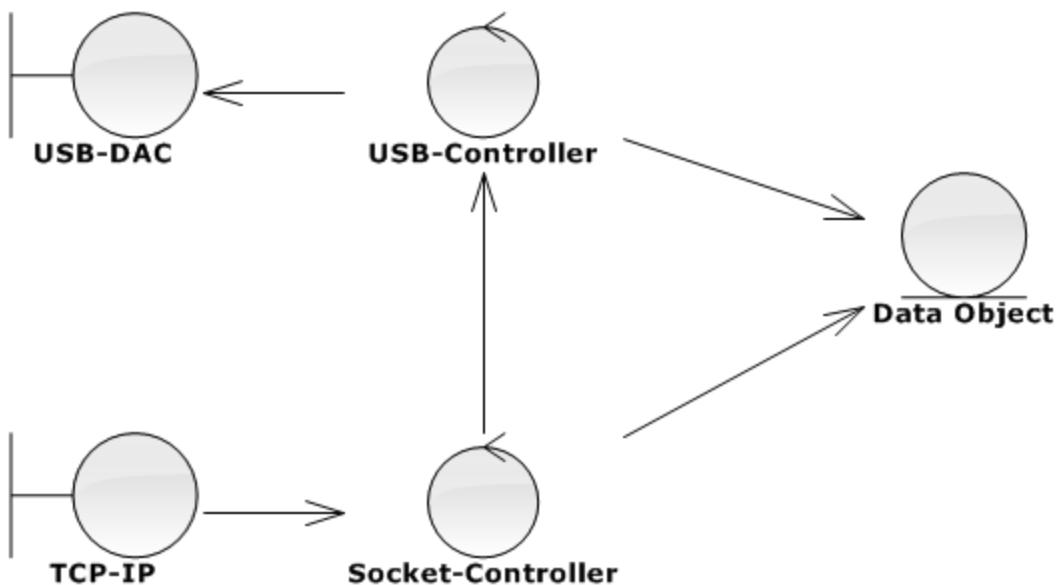
- 2a) Forskellige brugerdefinerede rapporter/udtræk

BCE-diagrammer

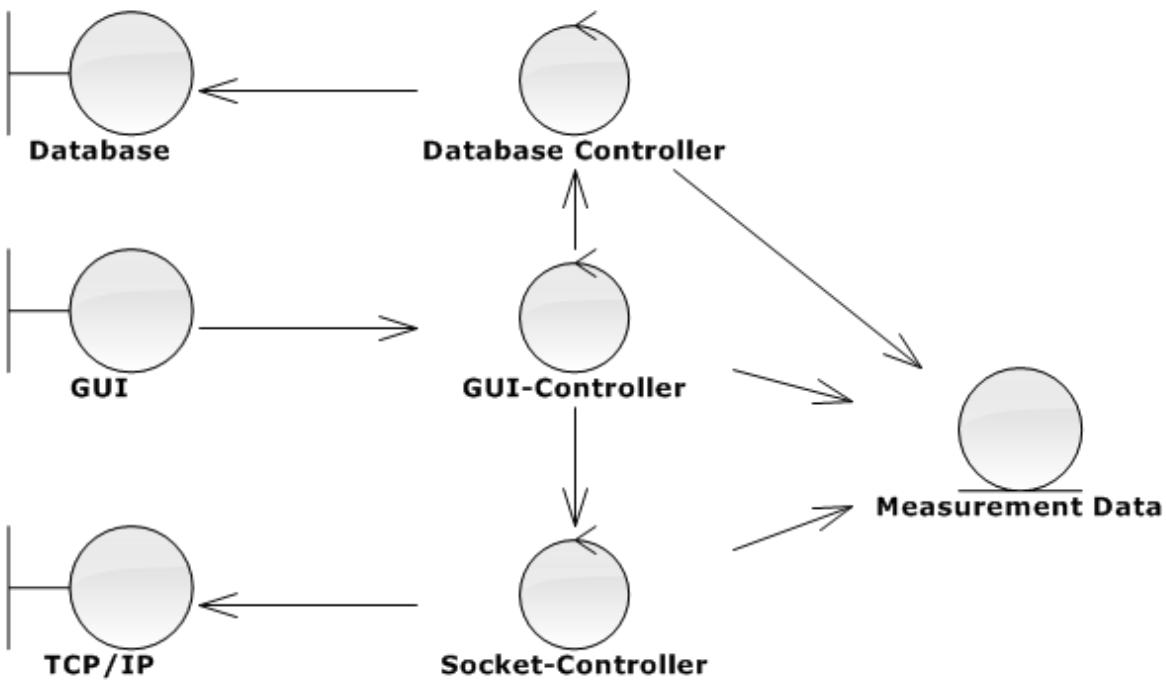
Vi har indtil videre sporet os ind på en løsning med tre delkomponenter

1. C# komponent til dataopsamlingen
2. Java komponent til styring af dataopsamlingskomponenten og logning af data i database
3. Web-komponent til hentning af data og generering af rapport.

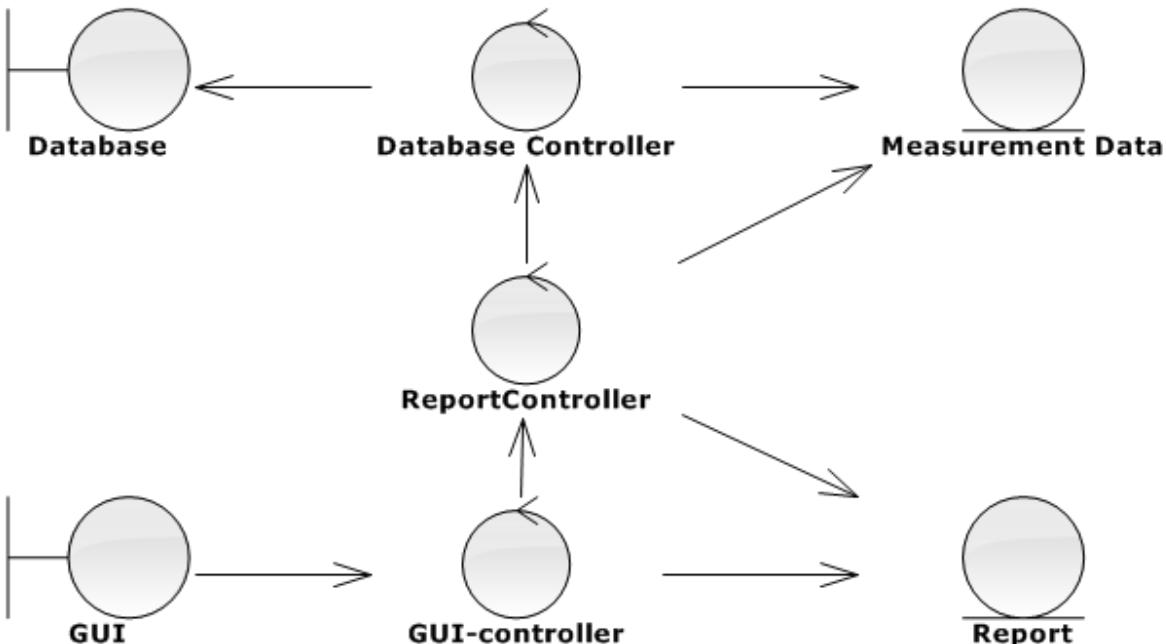
En foreløbig udgave af strukturen af delløsningerne ser ud som følger



Dataopsamlingsmodul. Henter data fra USB-device og lagrer det i et data object, som socket controlleren videresender over socket forbindelsen til Java komponenten.



Brugermodul til kontrol af dataopsamlingen. En GUI interagerer med GUI-controlleren, der igangsætter opsamlingen af data fra Dataopsamlingsmodulet over TCP/IP og lagrer data i databasen.



Rapportmodul. GUI interagerer med GUI-controller, der beder report controlleren om at generere en report. Reportcontrolleren beder database controlleren om at hente relevante data.

Udviklingsplan

Kritisk use case:

Ud fra use cases har vi identificeret vores kritiske use case til at være ‘opsamle data’. Vi har et såkaldt Data Aquisition Device (DAQ), som tilkobles PC’en med en USB. Dette DAQ har et antal indgange, som kan tilsluttes vores måleinstrumenter og konvertere nogle analoge signaler til fra instrumenter til digitale signaler til computeren. Dataopsamlingen er det væsentligste i hele projektet og her er også det mest usikre element, idet kommunikationen med måleapparater gennem USB-DAQ’et foregår gennem C# drivers, som vi ikke før har forsøgt at interface med. Vi er blevet præsenteret for et par muligheder for at løse problemet, og som nu er hælder vi mest til at lave et lille program i C#, og som vi forbinder med via TCP/IP, til et java program, som står for brugergrænseflade og dataopsamling. Altså fungerer C# programmet kun som et interface mellem USB-DAQ’en og det egentlige program. Dette vil give os fleksibilitet da C# kodebasen og Java kodebasen kan udskiftes uafhængigt af hinanden, således at et skift til en anden USB-DAQ kan foregå uden for store ændringer i kodebasen og et evt. skift af dataopsamlingen kan foregå.

Tidsplan

Første iteration:

1. Et ‘proof of concept’ hvor vi forsøger at logge data fra USB-DAQ. (High Risk/High Priority)
2. Videre kravspecificering, herunder (Low Risk)
 - a. BCE-diagrammer for de 3 hovedkomponenter
3. Udvikling af
 - a. tidsplan specificering af milestones. (evt. backlog)
 - b. Use case beskrivelser
 - c. Use case modeller
 - d. BCE diagrammer.
4. Test af C# USB DAQ interaktion
5. Dokumentation
 - a. Timeregistrering

Mål: Fungerende dataopsamling fra USB-DAQ. Bedre forståelse af kundens vigtigste visioner.

Slut: 2/3 - 14

Anden iteration:

1. Videreudvikling af kravspecifikation.
 - a. Mockup af brugerinterface i samarbejde med bruger (High priority)
 - b. Specificere krav til html-rapport generering.¹
 - c. Identifikation af yderligere use cases.
 - d. Indhentning af specifikationer på måleapparatur og

¹ Sideløbende med relevant undervisning.

- e. Afklaring af dataopsamlingsmodul - enkelte opsamlinger/ flere samples over tid.
 - f. Billede og udspecifcierung af testopstilling.
 - g. Kopier af sample data dokumenter til udvikling af datadesign.
2. Design
- a. Specifikation af interface - low level socket server?/Web server?
 - b. Specifikation af kommunikationsprotokol
 - c. Udvikling af klassediagram for Java-del
 - d. Udvikling af sekvensdiagram for Java-del
3. Kode
- a. Implementering af fungerende interface mellem C# komponent og Java del (High Risk, high priority)
 - b. Udvikling af Java komponent til dataopsamling.
 - i. Begynde på GUI-programmering (bundet af 2a)
 - ii. Implementere threaded datakommunikation med C# komponent (low priority)
 - c. Begynde på HTML del
4. Test
- a. C# komponent testing
 - i. testing mod dummyapparatur
 - ii. evt. test mod real life apparatur (high risk)
 - b. Interface testing

Mål: Fungerende interfacing mellem Java og C# komponent

Slut: 23/3 - 14

Tredje iteration:

1. Kravspecifikation
 - a. Evt. definering af brugerhåndtering
2. Design
 - a. Konstruktion af database
3. Kode
 - a. Interface til Database
 - b. Implementering af SQL/JSP i HTML/CSS grænseflade
4. Test
 - a. Test af dataopsamling mod real-life apparatur (high risk)

Mål: Fungerende dataopsamling og webinterface

Slut: 27/4

Sidste iteration:

Bugfixing, optimering og evt. udvidelse af implementering

1. Optimering af kodebase/GUI
 - a. Test af grænsesættilfælde

- b. Inputvalidering
- 2. 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

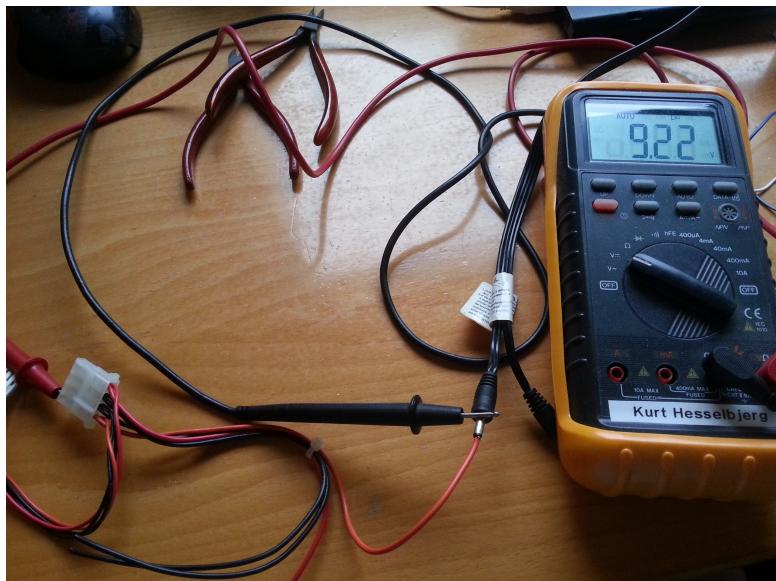
Mål: Optimering af løsning, evt. implementering af flere use cases

Slut: 11/5

Projektstatus

Indtil videre har vi arbejdet med implementering af en C#-komponent til opsamling af data og har opnået 'proof of concept' - det er lykkedes at opsamle måledata fra en dummy (en 9V strømforsyning) via USB-DAQ'en. Dermed har vi fundamentet for vores main use case på plads og kan begynde at videreudvikle resten af konceptet.

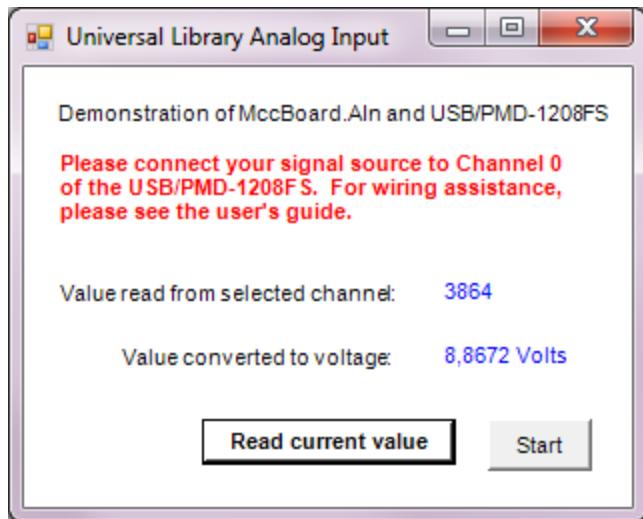
Her følger et par screenshots/billeder af en lille test af om det kunne lade sig gøre at læse data fra USB DAQ'en. Der måles på en strømforsyning, der efter specifikationen skulle give 9 v DC.



Måling med voltmeter.



Måling med USB DAQ (opstilling).



Screenshot af måling via program og USB DAQ.

Målingen passer ikke helt med den der blev fundet på voltmeteret, men det kan skyldes at voltmeteret og USB DAQ'en har forskellig impedans, eller at udstyret ikke er kalibreret korrekt.

```

private void btnRead1_Click(object sender, EventArgs e)
{
    float EngUnits;
    System.UInt16 DataValue;
    int Chan;

    // Collect the data by calling AIn member function of MccBoard object
    // Parameters:
    //   Chan      :the input channel number
    //   Range     :the Range for the board.
    //   DataValue :the name for the value collected

    //FS-1208 understøtter kun +-10 v input range
    Range = Range.Bip10Volts;           // select Bip10Volts (member of Range enumeration)
    Chan = 0;                          // set input channel

    ULStat = DaqBoard.AIn(Chan, Range, out DataValue); //afslæs værdien på den valgte kanal
    if (ULStat.Value == MccDaq.ErrorInfo.ErrorCode.BadRange)
    {
        MessageBox.Show("Change the Range argument to one supported by this board.", "Unsupported Range", MessageBoxButtons.OK);
        Application.Exit();
    }

    // Convert raw data to Volts by calling ToEngUnits (member function of MccBoard class)
    ULStat = DaqBoard.ToEngUnits(Range, DataValue, out EngUnits);

    lblShowData.Text = DataValue.ToString();           // print the counts
    lblShowVolts.Text = EngUnits.ToString("F4") + " Volts"; // print the voltage
}

```

Screenshot af den event, hvor målingen foretages.

Designmæssigt har vi i første omgang valgt en løsning hvor vi separerer komponenterne i 3 dele: 1) En C# komponent, der står for opsamling af data. 2) En Java komponent, der styrer C# komponenten over en socket forbindelse. Java komponenten beder C# komponenten om at opsamle data og modtager data, der lagres i en database. 3) Data udtrækkes fra databasen af et web-baseret rapportgenererings-modul.

Vi har udvikles BCE-diagrammer for løsningen og vil i anden iteration mappe disse til klassse-diagrammer, som vi vil begynde at implementere i Java-delen.

Plan

En højrisikoopgave i næste iteration bliver at få specificeret, hvilke data kunden gerne vil opsamle og desuden indhente specifikationer på måleapparatur. Vi skal desuden begynde på at designe en brugergrænseflade som lever op til kundens forestillinger. Kodedesignt skal videreføres til egentlige klassediagrammer og sekvensdiagrammer. Vi påbegynder kodning af Javakomponent - incl. interface med C#-komponent og GUI.