Aarhus School of Engineering

Electronic Engineering

E4PRJ

Detaljeret Software Design

Author:
Nicolai Glud
Johnny Kristensen
Rasmus Lund-Jensen
Mick Holmark
Jacob Roesen

3. december 2012

Indholdsfortegnelse

Indledning 1

Dette dokument beskriver det detaljerede SW-design for BROS, som er fastlagt ud fra dokumenterne kravspecifikation og systemarkitektur.

1.0.1 Formål

Formålet med dokumentet er:

- At fastlægge systemets detaljerede softwarestruktur udfra kravene specificeret i kravsspecifikationen. Derudover beskrivelsen af softwarekomponenterne og deres grænseflader beskrevet i systemarkitektur-dokumentet.
- At fastlægge systemets softwareklasser og deres indbyrdes interaktioner.
- At beskrive de enkelte klassers vigtigste metoder.

1.0.2 Reference dokumentation

- Kravspecifikation for projektet.
- Systemarkitektur-dokument.

KI 2

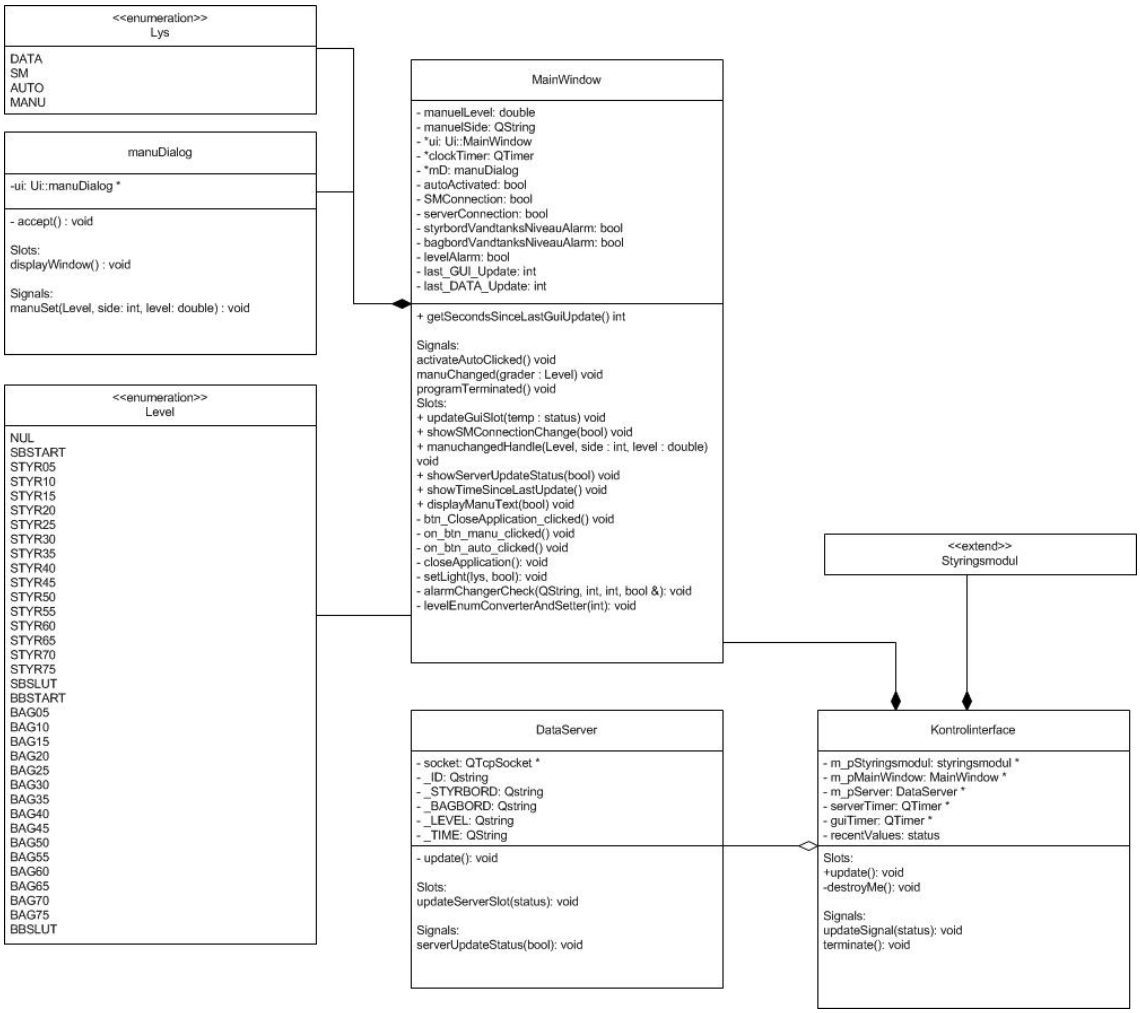
Nedenfor følger design af software til Kontrolinterfacet. Dette er lavet på baggrund af kravspecifikation og systemarkitektur.

2.0.3 Modulets ansvar

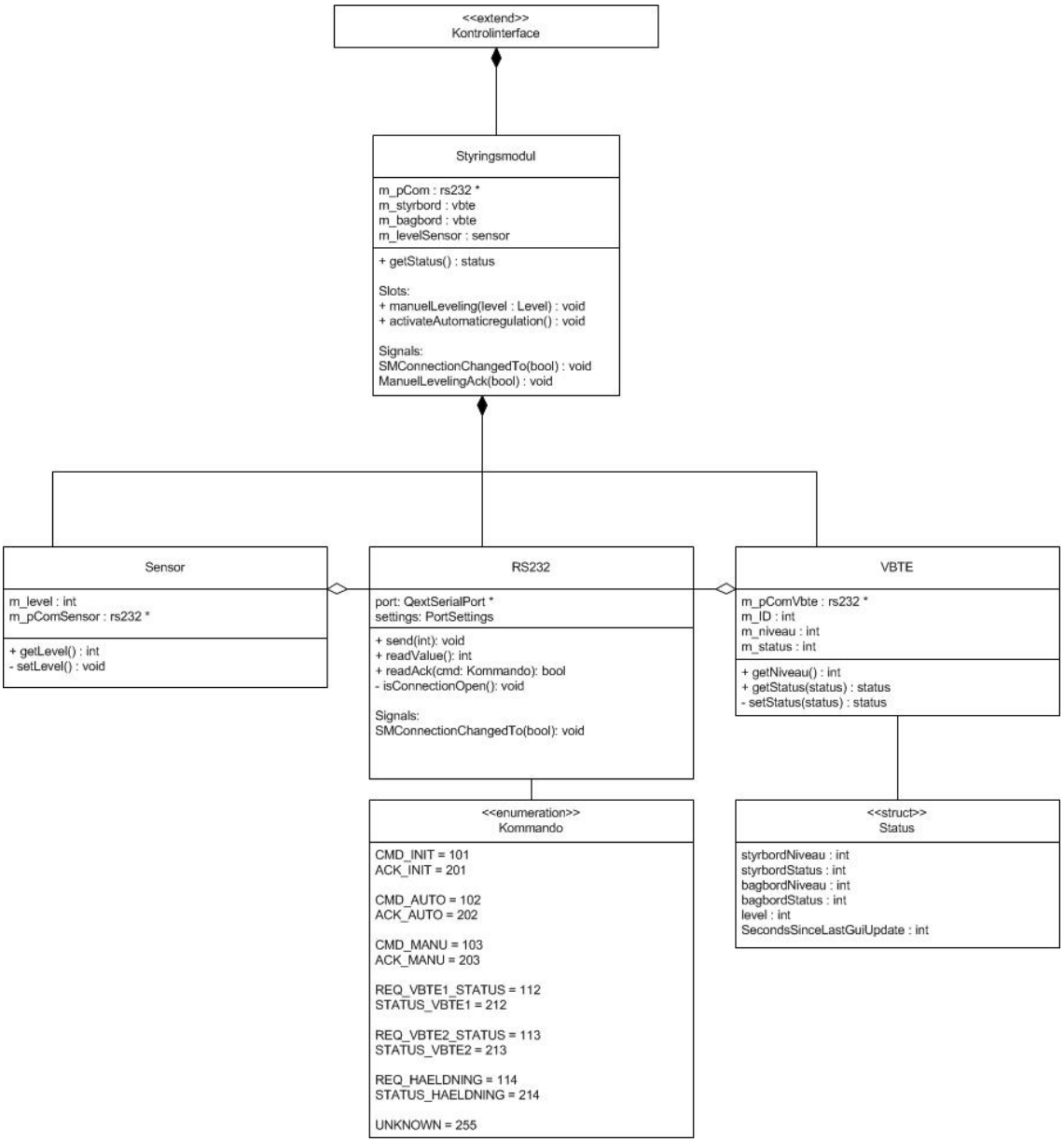
Kontrolinterfacet er brugerens primære kontaktflade til systemet. Programmet indeholder en brugergrænseflade der opfylder kravene i Kravspecifikationen. Her kan der også ses en prototype på den grafiske brugergrænseflade. Kontrolinterfacet står for at modtage inputs fra brugeren. Disse inputs sendes som kommandoer til Styringsmodulet. Det er også herfra at Kontrolinterfacet modtager de værdier, som sidenhen vises på den grafiske brugergrænseflade. Kontrolinterfacet står også for kommunikationen til den eksterne database. Her sendes en række parametre om skibet og dets status.

2.0.4 Klassediagram

Nedenfor ses klassediagrammet for Kontrolinterfacet. Bemærk at klassediagrammet er delt op i to. Skæringsstedet er mellem Kontrolinterface-klassen og Styringsmodul-klassen og er markeret med «extend».



Figur 2.1. På figuren ses klassediagrammet for KI - Kontrolinterface-delen



Figur 2.2. På figuren ses klassediagrammet for KI - Styringsmodul-delen

2.1 Metode- og klassebeskrivelser

2.1.1 MainWindow

Ansvar

Denne klasse indeholder de funktioner der er skrevet til Qt-formen MainWindow.ui hvori selve det grafiske er opbygget. Klassen indeholder de funktioner der anvendes i forbindelse med den grafiske brugergrænseflade. Det være sig når der kommer et input, eller der skal opdateres nogle værdier på skærmen.

Funktionsbeskrivelser

`int getSecondsSinceLastGuiUpdate();`

Beskrivelse:	En simpelt get-funktionen der returnerer værdien af klasseattributten *last_GUI_Update.*
Parametre:	Ingen
Returværdi:	`int secondsSinceLastGuiUpdate`

`void updateGuiSlot(status temp);`

Beskrivelse:	Bliver kaldt når GUI'en skal opdateres. Den modtager parameterent *temp* som er en struct af typen status. Ud fra denne struct hives de værdier ud, som skal vises på GUI'en. Værdierne vises ved de set-funktioner der er tilknyttet de anvendte widgets (og dermed en del af Qt-frameworket.) Når værdierne er opdateret vises det som en aktivitet i aktivitetsloggen
Parametre:	`status temp`
Returværdi:	Ingen

`void showSMConnectionChange();`

Beskrivelse:	Kaldes hvis SM-forbindelsen til styringsmodulet ændres fra forbundet til mistet forbindelse eller omvendt. Det udløser en aktivitet i aktivitetsloggen. Derudover skiftes lyset på gui'en. Parameteren state er den status som forbindelsen har ændret sig til.
Parametre:	`bool state`
Returværdi:	Ingen

`void manuChangedHandle(Level samlet, int side, double level);`

Beskrivelse:	Kaldes når brugeren har ændret i indstillingen til den manuelle hældning. Som parametre modtages hvilken side man ønsker at skibet skal hælde til (int side), hvor meget det skal hælde (double level) samt de to informationer samlet i en enum, Level samlet. Funktionen emitter signalet "manuChanged(Level temp). Det sætter også klasseattributerne manuelSide, manuelLevel samt autoActivated til deres rette værdier. Til sidst kaldes funktionen displayManuText(true)

Parametre:	`Level samlet`
	`int side`
	`double level`

Returværdi:	Ingen

`void showServerUpdateStatus(bool state);`

Beskrivelse:	Kaldes hver gang signalet DataServer::serverUpdateStatus() udsendes. Funktionen undersøger om forbindelsen har ændret sig ved at sammenligne med attributen serverConnection. Hvis forbindelsen har ændret sig udsendes dette som en aktivitet. Lyset ændres også således at det passer ved hjælp af setLight(DATA, serverConnection. Hvis vi modtager "true"vil last_DATA_Update opdateres til således til den nuværende værdi af sekunder siden epoch.
Parametre:	Ingen
Returværdi:	Ingen

`void showTimeSinceLastUpdate();`

Beskrivelse:	Kaldes hvert sekund. Funktionen opdaterer antallet af sekunder siden sidste overførelse af data til serveren eller til SM. Når tiden er længere end tiden mellem hver opdatering vil dette tal skifte til rødt.
Parametre:	Ingen
Returværdi:	Ingen

`void displayManuText(bool show);`

Beskrivelse:	Viser eller skjuler teksten med indstillingen af manuel hældning afhængig af parameteret show.
Parametre:	Ingen
Returværdi:	Ingen

`void activateAutoClicked();`

Beskrivelse:	udsendes når der er blevet trykket på knappen *activateAutoClicked*
Parametre:	Ingen
Returværdi:	Ingen

`void activateAutoClicked();`

Beskrivelse:	udsendes når der er blevet trykket på knappen *activateAutoClicked*
Parametre:	Ingen
Returværdi:	Ingen

`void manuChanged(Level grader);`

Beskrivelse:	Udsendes når der er blevet ændret en manuel indstilling.
Parametre:	`Level grader`
Returværdi:	Ingen

`void programTerminated();`

Beskrivelse:	Udsendes når programmet er blevet lukket ned.
Parametre:	Ingen
Returværdi:	Ingen

`void btn_CloseApplication_clicked();`

Beskrivelse:	Kaldes når luk-knappen på GUI'en er blevet trykket. Udsender signalet programTerminated() hvis brugeren bekræfter valget
Parametre:	Ingen
Returværdi:	Ingen

`void on_btn_manu_clicked();`

Beskrivelse:	Kaldes når der bliver trykket på knappen for manuel hældning. Viser dialogen "manuDialog".
Parametre:	Ingen
Returværdi:	Ingen

`void on_btn_auto_clicked();`

Beskrivelse:	Kaldes når der trykkes på *Automatisk Hældnings-knappen*. Aktiverer automatisk styring og deaktiverer den manuelle.
Parametre:	Ingen
Returværdi:	Ingen

`void setLight(lys id, bool state);`

Beskrivelse:	Sætter lyset i forhold til parametrene. lys er en enum der bestemmer hvilket element lyset skal ændres for. state er om lyset skal være tændt eller ej
Parametre:	`lys id` `bool state`
Returværdi:	Ingen

`void alarmChangerCheck(QString sentence, int critical_point, int value, bool &e`

Beskrivelse:	Tester om alarmerne har ændret sig. Sentence er starten af den sætning der skrives i aktivitetsloggen. critical_point er det kritiske punkt for det emne der arbejdes på. Value er den værdi den har. Earlier_state er hvilket stadie alarmen var i tidligere.

Parametre:	`QString sentence`
	`int critical_point`
	`int value`
	`bool &earlier_state`

Returværdi:	Ingen

`void levelEnumConverterAndSetter(int level);`

Beskrivelse:	Konverterer en integer baseret på Level-enumeratoren om til en side og en vinkel.
Parametre:	`int level`
Returværdi:	Ingen

2.1.2 Kontrolinterface

Ansvar

Dette er hovedklassen hvori selve programmet lever. Oprettelsen af et objekt af denne klasse er derfor også det eneste der sker i main.

Funktionsbeskrivelser

2.1.3 DataServer

Ansvar

Klassen står for al kommunikation med serveren via en TCP-forbindelse. Forbindelse oprettes og nedlægges hver gang der tages kontakt. DataServer-objektet opdaterer serveren når den får ordre om det fra Kontrolinterface-klassen.

Funktionsbeskrivelser

2.1.4 manuDialog

Ansvar

manuDialog-klassen står for håndtering af det vindue der åbnes ved tryk på *Manuel Hældningsregulering-knappen.*

Funktionsbeskrivelser

2.1.5 Styringsmodul

Ansvar

Klassen har samme rolle som det fysiske styringsmodul har i systemet. Det giver kontrolinterfacet adgang til sensor-værdier og vandstandsniveauer igennem sine underklasser, VBTE og Sensor.

Funktionsbeskrivelser

2.1.6 VBTE

Ansvar

Håndterer værdierne for hver sin vandballasttankenhed. Kommunikerer til det fysiske styringsmodul ved hjælp af RS232-klassen.

Funktionsbeskrivelser

2.1.7 Sensor

Ansvar

Håndterer værdierne for hældningssensoreren. Kommunikerer til det fysiske styringsmodul ved hjælp af RS232-klassen.

Funktionsbeskrivelser

2.1.8 RS232

Ansvar

Håndterer kommunikationen til det fysiske styringsmodul ved protokollen der ses i enumeratoren "Kommando".

Funktionsbeskrivelser

Databasen 3

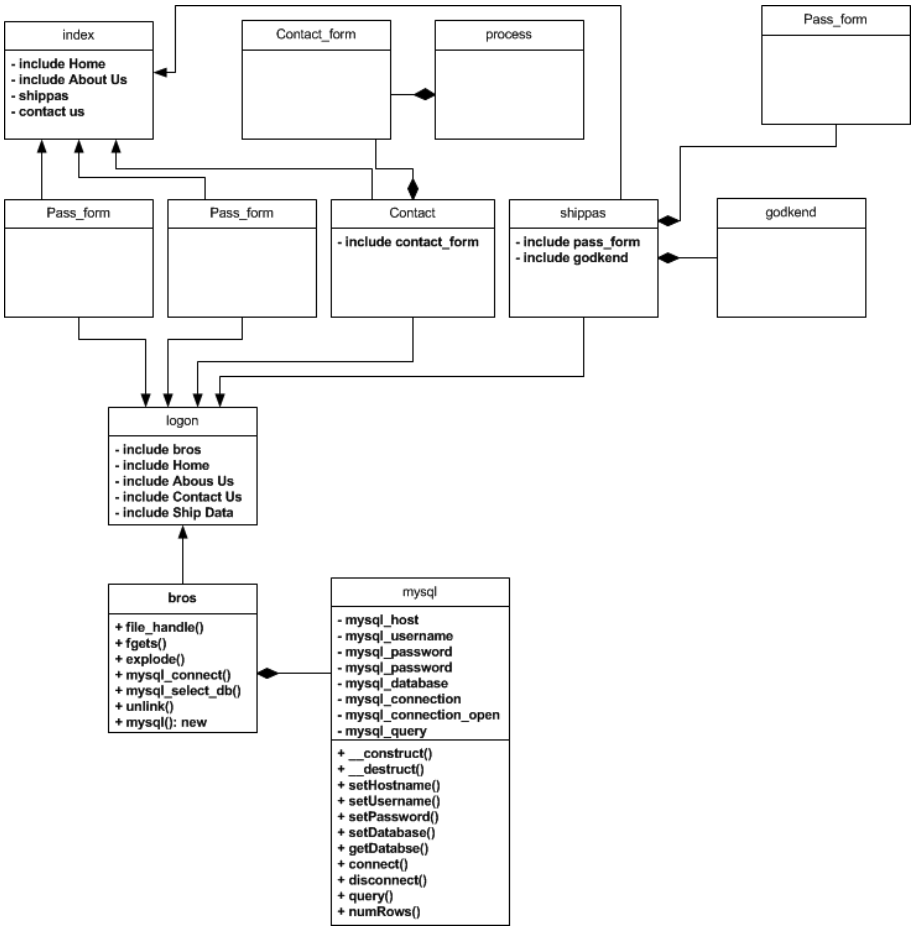
Nedenfor følger design af software til databasen og dens interface. Dette er lavet på baggrund af kravspecifikation og systemarkitektur.

3.0.9 Modultes Ansvar

Databasen er her hvor havne terminalens personale kan aflæse data fra skibet. Disse data er sendt fra KI. Programmerne indeholder brugergrænseflader der opfylder kravene, beskrevet i kravspecifikationen. Her kan der også ses en prototyper på brugergrænsefladen. Database modulet har 3 dele. Severen, Web-siden og en mySQL database.
Severen står for kommunikationen immellem KI og Databasen. Severen modtager data fra KI og lager disse i en tekst fil
Web-siden giver brugeren mulighed for at se info om BROS samt at logge sig ind i BROS database hvorfra at data om skibe der er tilsluttet systemet kan aflæses. Web-sidens 3 vigtigste funktioner er at gemme ny data til mySQL databasen, slette den tekst fil som severen lavede og vise data for brugeren. Til at håndtere web-siden benyttes en apache server. **mySQL databasen** er en database som er installeret på computeren. Alle data som er sendt fra KI er lageret i mySQL databasen.

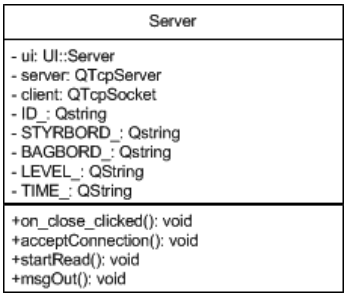
3.0.10 Klassediagrammer

Nedenfor ses klassediagrammerne for databasen. Bemærk database modulet er lavet som en server del og en web del



Figur 3.1. Klassedigram for databasens severer

tilføj dato()



Figur 3.2. Klassedigram for databasens severer

3.0.11 Globale variabler

3.0.12 Funktionsbeskrivelser

Server

Denne header har til ansvar at starte GUI og håndtere TCP forbindelse

void onCloseClicked( );

Beskrivelse:	Ved klik forespørges brugeren om denne ønsker at lukke severen
Parametre:	
Returværdi:	

void acceptConnection( );

Beskrivelse:	Står for at acceptere forbindelse fra KI og connecte.
Parametre:	QTcpServer server
	QTcpSocket* client
Returværdi:	

void startRead( );

Beskrivelse:	Læser data fra socket. Data til fil og GUI
Parametre:	QTcpSocket* client
Returværdi:	

Wep-side

Web-siden står for at fremvise skibs data grafisk for terminal personalet. Desuden står den for at lager data og loade data fra mySQL databasen.

mysql

setHostName( );

Beskrivelse:	Læser data fra socket. Data til fil og GUI
Parametre:	QTcpSocket* client
Returværdi:	

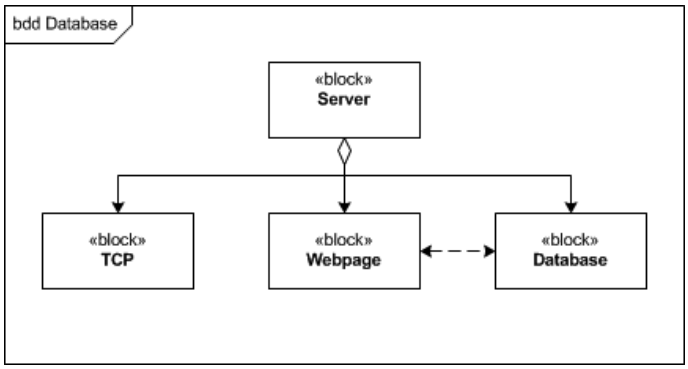
3.0.13 TCP-forbindelse

3.1 Design

3.1.1 Server

Databasen er en server som har de 3 underblokke TCP, Database og Web-page som illustreret på figur 3.3

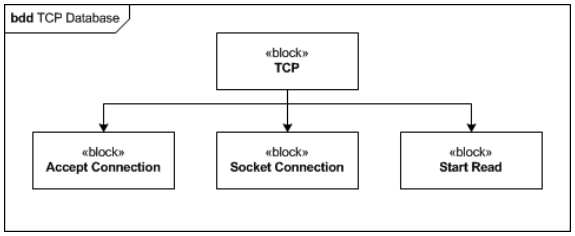
TCP er en dataforbindelse (Transmission Control Protocol). Denne protokol er benyttes til at sende data fra KI til serveren. Severen vil modtage data og lagere dette i en midlertidig backup fil. TCP-forbindelsen er kodet i C++. For TCP-forbindelsen benyttes TCP - protocollen som tilbyder sikker data overførelse fra BROS. Databasen er en mySQL database som frit kan downloades og installeres på en Linux, Windows eller Mac. Man skal installere en server del og en client del. Server delen er den del der gør det muligt



Figur 3.3. BDD server

at håndtere og lagre data. Denne ligger under client delen og er nødvendig for at client kan fungere. mySQL client gør det muligt at en bruger kan tilgår og læse fra databsen uden at gør ændringer i denne. Dette benytte i web interfacet. mySQL kan tilgås direkte fra terminalen og giver mulighed for forskellige opsætninger af databaser og tabeller samt forskellige bruger rettigheder. **mulighede med mySQL** Web-pagen er udviklet i php som giver gode muligheder for kommunikation til og fra mySQL databasen. Web-pagen er implementeret ved hjælp af en apache server som er en web server. Web-pagen har en general information omkring BROS og et login til at tilgå databasen. Ved at anvende et web-interface gives der mulighe for at data kan aflæses fra andre pladser end fra den direkte server. Ved at kende ip eller host navn er det muligt at tilgå web-siden. Den web baserede database loader mySQL databasen og fremviser denne grafisk for brugeren.

3.1.2 TCP server



Figur 3.4. BDD TCP server(ikke færdig)

TCP protocollen er en af kerneprotokollerne på nutidens internet. Gennem TCP kan forskellige værtsmaskine igennem f.eks. internet ethernet og trådet forbindelse oprette forbindelse til hinanden og udveksle datepakker. Protokollen giver programmelt på værtsmaksine nogle vitale garantier for at disse datapakker afsendes og modtages ved:

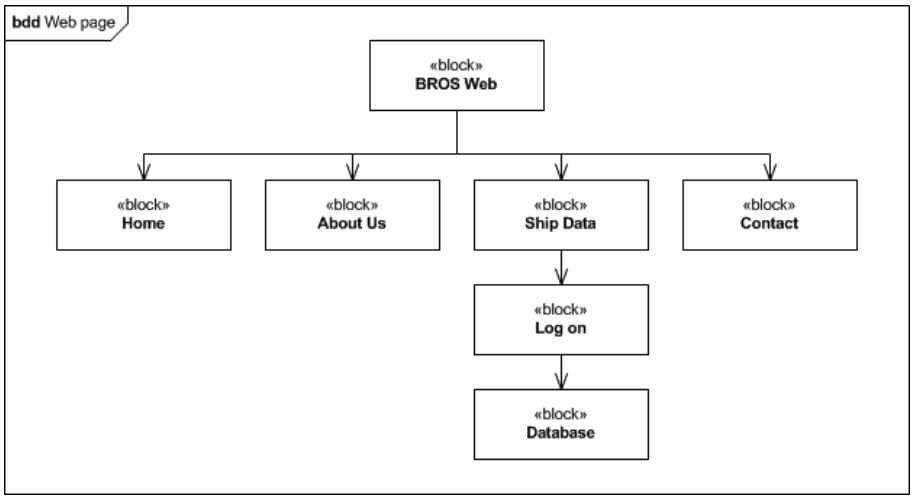
- Stabilitet: En pakke der går tabt bliver forsøgt afsendt igen
- Ordnet levering: En pakke kommer frem til modtageren i samme rækkefølge som de blev afsendt

Der ud over benytter TCP sig af forskellige port numre. Forskellige portnumrer gør dte muligt etablere flere forskellige datstrømme til og fra samme værtsmaskine.

Selve programmet er kodet op over socket programmering. Under opstart initialiseres socket, ip og porte. Når disse er succesfuldt initialiseret afvendter TCP serveren at KI connecter. Efter connection modtager TCP serveren datapakken fra KI og gemmer denne i en textfil kaldt ship.txt. Denne fil bruges bruges som en midlertidig sikkerhed for at data fra KI sikkert bliver inført i mySQL databasen.

sequuens diagram

3.1.3 Web-side



Figur 3.5. BDD Web-page BROS

For at web siden kan køre kræves der at der på serveren er installeret en web - server. Der er på denne server installeret apache som web - server.
Ved opstart af serveren bliver denne automatisk startet og start siden er **BROS**. Web siden fungere som bruger interfacet for havne terminalen. Web siden er opbygget som et dynamisk web page og kodet i php. Dette sikre minmal loading time ved hjælp af ajax. Alle styles på siden er styret af css. Siden har 4 forskellige under sider Home, About Us, Ship Data og Contact. Ved klik på Ship Data vil man blive bedt om at taste sit password som sikre at kun autoriserede personer får adgang til systemet.
Siden der håndtere skibs data starter med at connecte til mySQL databasen om ikke der kan connectes til databasen vil der blive udskrevet en error og siden vil igen forsøge at conencte til databasen. Efter connection til databasen vil den gemte fil fra TCP-serveren blive loadet ind i mySQL databasen og filen vil blive slettet. efter loading vil siden loade alle data i mySQL databasen og vise denne for brugeren. Data der kan vises for brugeren er:

- Skibs ID
- Højre tanks vandniveau
- Venstre tanks vandniveau
- Hældningsniveau
- Forbindelse til KI

Siden checker hvert 5 sekund om der er nu data. Hved ny data vil denne blvie placeret øverst på siden. Når brugeren er færdig med at benytte BROS databasen kan brugeren trykke log of i øverste højre hjørne.

3.2 Metodebeskrivelse

3.2.1 TCP KI

Void msg(string, string, string, string, string);
Håndtere data der skal sendes via tcp.
Void socket();
Opretter socket forbindelse for ctp client.
Void connection();
Kalder ip adresse og portnummer for TCP server. Overføre data.

3.2.2 TCP database

Void socketConnect();
Void msgServer();

SM 4

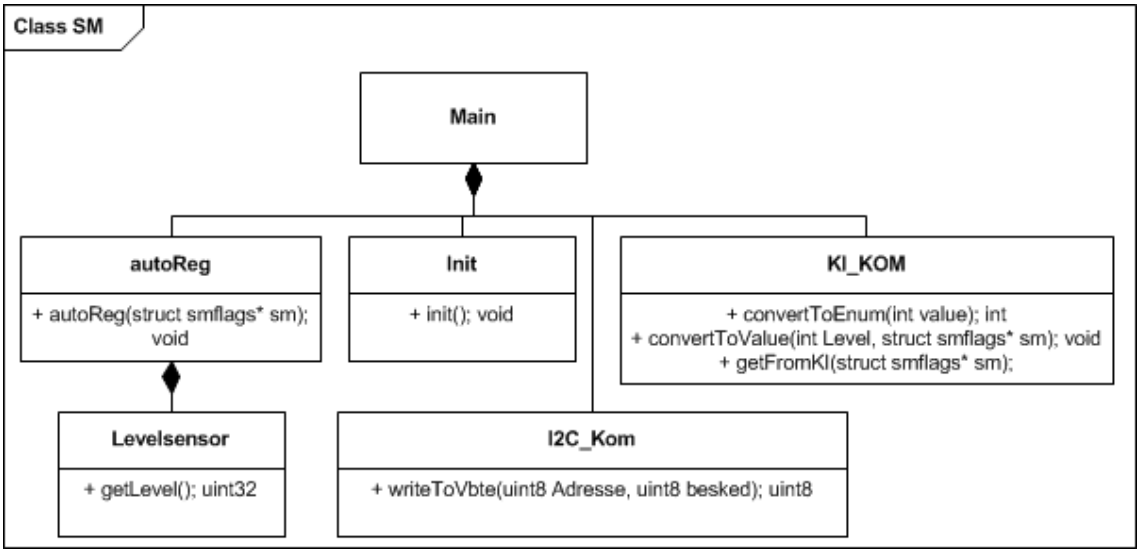
Dette afsnit beskriver designet af styringsmodulet, SM.

4.1 Klassens ansvar

Styringsmodulet har til ansvar at holde styr på levelsensoren og værdierne fra VBTE. Den kommunikere med KI og VBTE med indbyggede API'er fra Cypress PSoC 5 biblioteker.

4.2 Klassediagram

Nedenfor ses klassediagrammet for SM. Bemærk at koden dog er i C men for overblikket er der lavet klassediagram.



Figur 4.1. På figuren ses klassediagrammet for SM

4.3 Funktioner

bla bla

4.4 Variabler

Variabel	Beskrivelse
`autoflag`	Denne variable er et flag der holder styr på automatisk regulering .
`manuflag`	Et flag til at holde styr på manuel regulering.
`levelVal`	En variable med vores level værdi.
`VBTE1Niveau og VBTE2Niveau`	Holder styr på vandniveauet i ballasttanke i %.
`VBTE1Status og VBTE2Status`	Holder styr op tilgængelighed for VBTE1 og 2.
`vinkelVal`	Indeholder værdien for manuel regulering.

Alle variabler er indkapslet i en struct navngivet "smflags".

4.5 Funktionsbeskrivelser

4.5.1 Init

Ansvar

Denne header har til ansvar at sørge for alle komponenter opretter og initieret. `void init( void);`

Beskrivelse:	Funktionen anvender API'et fra Cypress componenter og står for at initiere og starte vores PSoC hardware. Den sætter også et register tilhørende vores Accelerometer.
Parametre:	ingen
Returværdi:	ingen

4.5.2 Levelsensor

Ansvar

Denne header har til ansvar at hente levelværdien ind fra vores accelerometer. `uint32 getLevel( void);`

Beskrivelse:	Funktionen anvender API'et fra Cypress componenter og venter på at vores ADC henter convertere det analoge signal. Funktionskald for ADC ses i PSoC databladet.
Parametre:	ingen
Returværdi:	`uint32 levelVal`

4.5.3 autoReg

Ansvar

Denne header har til ansvar at styre automatisk regulering. `void autoReg( struct smflags* sm);`

Beskrivelse:	autoReg anvender værdier fra VBTE moduler samt KI til at holde systemet i et bestemt level. Funktionen starter med at checke på automatisk og manuel styrings flagene. Derefter kalder den getLevel agere ud fra niveauet. Funktionen vil altid tømme fra en tank før den begynder at fylde en anden.
Parametre:	`struct smflags* sm`
Returværdi:	ingen

4.5.4 I2C_Kom

Ansvar

Denne header har til ansvar at kommunikere med VBTE modulerne. `uint8 writeToVbte( uint8 Adresse, uint8 besked);`

Beskrivelse:	writeToVbte anvender I2C fra Cypress PSoC 5 API til at skrive til VBTE modulerne. Den tager adressen og beskeden man skal sende og sender til pågældende enhed. Derefter venter den på svar som den så returnere.
Parametre:	`uint8 Adresse` `uint8 besked`
Returværdi:	`uint8 VbteNiveau`

4.5.5 KI_KOM

Ansvar

Denne header har til ansvar at kommunikere med KI enheden.
`int convertToEnum( int value);`

Beskrivelse:	funktionen tager en level værdi ind for så at konvertere den til en Enum(integer) som den returnere.
Parametre:	`int value`
Returværdi:	`int Enum`

`void convertToValue(int Level, struct smflags* sm);`

Beskrivelse:	Funktionen tager en enum og en pointer som den så konvertere til en level værdi og sætter i sm structen.
Parametre:	`int Level,` `struct smflags* sm`
Returværdi:	ingen

`void getFromKI( struct smflags* sm);`

Beskrivelse:	Funktionen anvender UART fra Cypress PSoC 5 API'en til at modtage en besked fra KI modulet som den så vurdere og agere på. Når den har modtaget noget sender den en ack tilbage til KI modulet. Derefter handler den og hvis det er nødvendigt sender data til KI.
Parametre:	`struct smflags* sm`
Returværdi:	ingen

4.6 Eventuelle Sekvensdiagrammer og state machines

Måske kommer de senere?

VBTE 5

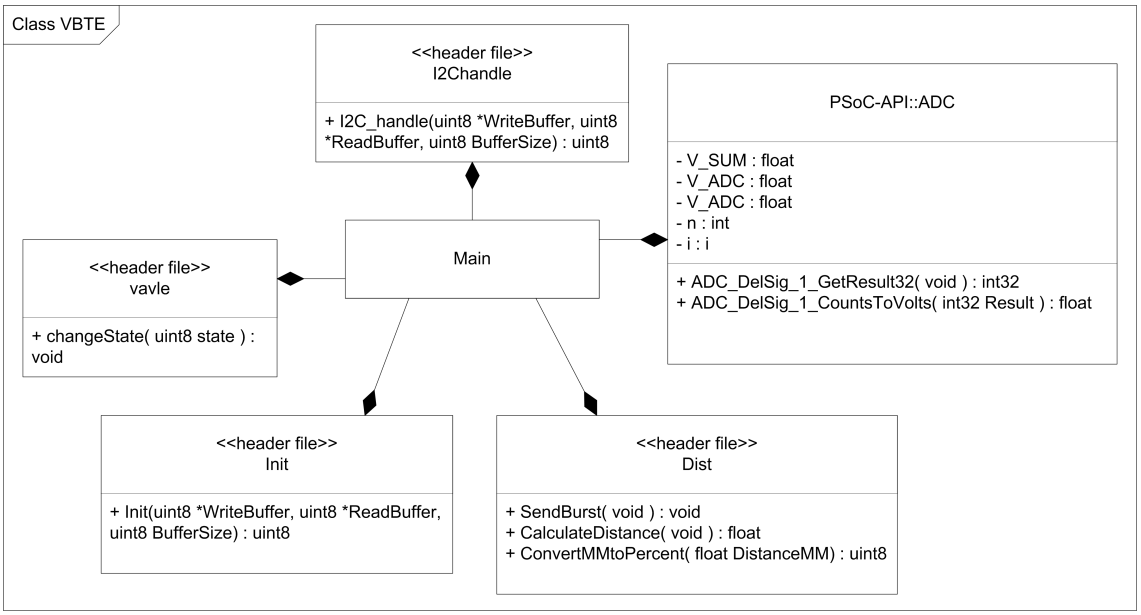
Nedenfor følger design af software til VBTE. Dette er lavet på baggrund af kravspecifikation og systemarkitektur. Bemærk der i dette design dokument blandt andet ikke er beskrevet mixer, pga osv. da deres eneste funktion er "Start();". Derudover er der en betydelig hardware del knyttet til dette modul, der refereres derfor til detaljeret hardware design for yderligere detaljer om VBTE modulet.

5.1 Modulets ansvar

Som beskrevet i systemarkitektur står VBTE'en for at måle vandniveauet i ballasttankene samt at lukke vand ind eller ud af ballasttankene. Hertil er der også en kommunikation med SM modulet indeholende instruktioner.

5.2 Klassediagram

Nedenfor ses klassediagrammet for VBTE. Bemærk at koden dog er i C men for overblikket er der lavet klassediagram.



Figur 5.1. På figuren ses klassediagrammet for VBTE **Billedet skal opdateres**

5.3 Globale variabler

Variabel	Beskrivelse
`BurstLengthVal`	Denne variabel er anvendt til at håndtere antallet af perioder burstet bliver sendt med.
`WaitBurstVar`	Bliver brugt til nonblocking delay til SendBurst funktionen.
`BurstTimerVal`	Holder på Timerens værdi når et burst er sendt.
`DistanceTimerVal`	Holder værdien på timeren når et burst er modtaget.
`CalcDistFlag`	Bliver sat når et burst er modtaget så en afstand kan blive beregnet.
`BurstFlag`	Bliver sat når et burst bliver sendt og hevet ned når et burst er modtaget. Dette sker for ikke at få flere detektioner på samme signal.

5.4 Metode- og klassebeskrivelser

5.4.1 Valve

Ansvar

Denne header har til ansvar at styre ventilerne ud fra "state-variablen modtaget fra I2C_handle. Headeren benytter PSoC-API'et til kontrol registre..

Funktionsbeskrivelser

```
void ChangeState( uint8 state );
```

Beskrivelse:	Funktionen anvender API'et fra I2C blokken i PSoC miljøet. Med disse tjekker den om der er fyldt nyt i bufferen og aflæse dette. Herfer kalder den funktionen I2C_decode(); til at afkode beskeden fra SM. Herefter klargøres readbufferen til evt. at sende vandniveau tilbage.

Parametre:	`uint8* WriteBuffer`
	`uint8* ReadBuffer`
	`uint8 BufferSize`

Returværdi:	`uint8 State`

5.4.2 Dist

Ansvar

Denne header har til ansvar at sende burst, beregne afstanden samt at omregne afstanden til procent.

Funktionsbeskrivelser

```
void SendBurst( void );
```

Beskrivelse:	Denne metode aktiverer en 40kHz clock og tæller perioderne op til 10, lukker for burstet og ligger timerens værdi ind i den globale variabel BurstTimerVal
Parametre:	Ingen
Returværdi:	Ingen

```
float CalculateDistance( void );
```

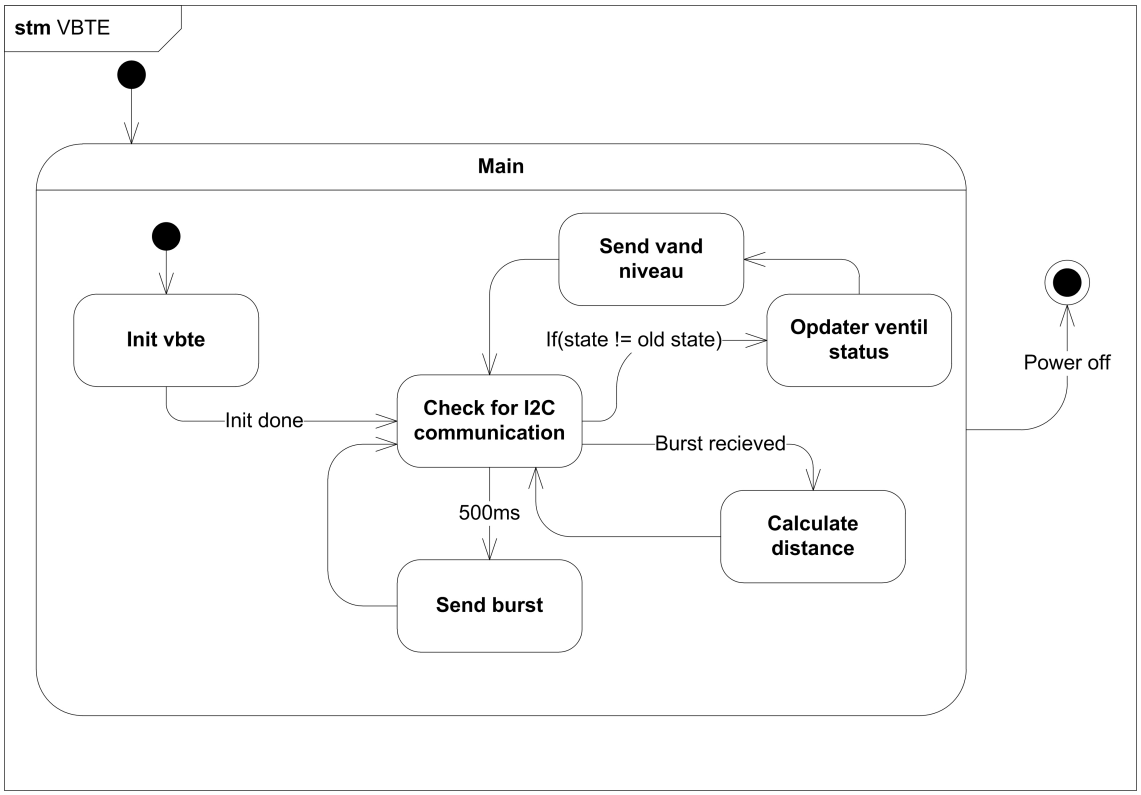
Beskrivelse:	Denne metode anvender BurstTimerVal og DistanceTimerVal til at finde ud af hvor mange clocks der er gået fra burstet er blevet sendt til det igen er blevet registreret.
Parametre:	Ingen
Returværdi:	`float DistanceMM`

```
uint8 ConvertMMtoPercent( float );
```

Beskrivelse:	Metoden modtager afstanden i millimeter og returnerer hvor mange % der er i tanken
Parametre:	`float DistanceMM`
Returværdi:	`uint8 DistancePercent`

5.4.3 State Machine

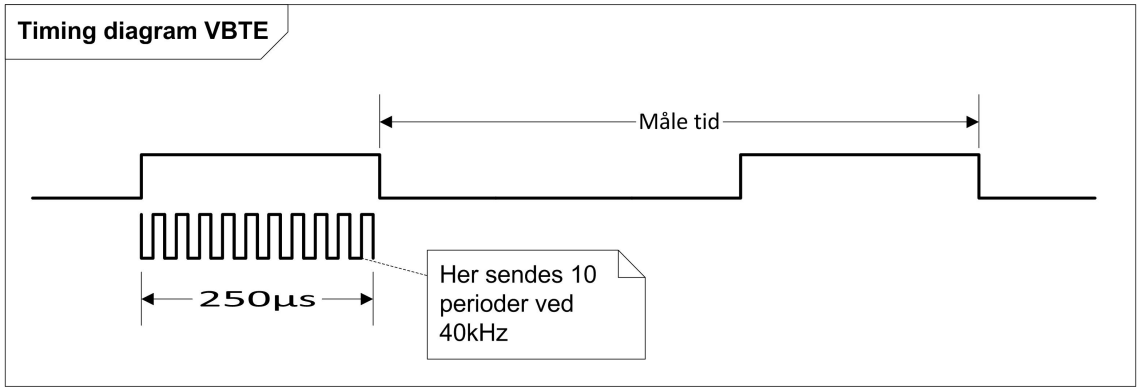
Nedenfor ses statemachine der beskriver det overordnede flow i VBTE programmet.



Figur 5.2. Statemachine for VBTE program

5.4.4 Timing Diagram (Hører til hardware)

Nedenfor ses timing diagram for en ultralydspuls til afstandsmåling



Figur 5.3. Timing diagram for VBTE ultralydspuls