

Systemarkitektur

RoboPlay

Semesterprojektgruppe: 10

Afleveringsdato: 19. december 2018

Version: 1.0.0

#	Stud.nr.	Navn	Initialer
1	201704441	Frank Andersen	FA
2	201705186	Kristian Bang Nielsen	KN
3	201509800	Christian Lundtoft Trebbien	CT
4	201707807	Frederik Munch-Hansen	FM
5	06923	Niels Pallisgaard Thøgersen	NT
6	201505470	Mads Skytte Nielsen	MN
7	201704714	Michael Møller-Hansen	MM
Vejleder: Martin Ansbjerg Kjær			

Tabel 1 - Medlemmer i gruppe 10

1 Indholdsfortegnelse

1 Indholdsfortegnelse	1
2 Versionshistorik	3
3 Indledning	4
4 Overordnet systemarkitektur	4
4.1 Domænemodel	4
4.2 Blok-definitions-diagram	5
4.3 Systemsekvensdiagrammer for udvalgte use cases	6
4.3.1 Systemsekvensdiagram for UC1's hovedscenarie	7
4.3.2 Systemsekvensdiagram for UC1's extension 3	8
4.3.3 Systemsekvensdiagram for UC2's hovedscenarie	8
4.3.4 Systemsekvensdiagram for UC6's hovedscenarie	9
4.3.5 Systemsekvensdiagram for UC7's hovedscenarie	10
5 Kommunikationsprotokoller	11
5.1 Protokol mellem MobilApp og RPiApp	11
5.1.1 TCP klient/server	11
5.1.2 Protokol for udførelse af bevægelser	11
5.1.3 Protokol for indstilling af robot til startposition	13
5.2 Protokol mellem RPiApp og PSoCApp	13
5.2.1 Opsætning	14
5.2.2 Besked-struktur	14
5.2.2.1 Byte #1	14
5.2.2.2 Byte #2	14
5.2.2.3 Byte #3	15
5.2.2.4 Byte #4	15
6 Hardwarearkitektur	16

6.1 Udvidet BDD	17
6.2 Internt blokdiagram	17
7 Softwarearkitektur	19
7.1 MobilApp	19
7.1.1 Applikationsmodel for UC1	21
7.1.1.1 Klassediagram	21
7.1.1.2 Sekvensdiagram for hovedscenariet	23
7.1.2 Applikationsmodel for UC2	24
7.1.3 Applikationsmodel for UC6	24
7.1.3.1 Klassediagram	24
7.1.3.2 Sekvensdiagram for hovedscenariet	26
7.1.4 Applikationsmodel for UC7	26
7.2 RPiApp	27
7.2.1 Applikationsmodel for UC1	27
7.2.2 Applikationsmodel for UC2	27
7.2.3 Applikationsmodel for UC6	27
7.2.4 Applikationsmodel for UC7	27
7.3 PSoCApp	27
7.3.1 Applikationsmodel for UC1	27
7.3.2 Applikationsmodel for UC2	27
7.3.3 Applikationsmodel for UC6	27
7.3.4 Applikationsmodel for UC7	27

2 Versionshistorik

Vers	Dato (Y-M-D)	Navn	Ændringer
v0.1.0	2018-09-17	NT	Dokument oprettet
v0.2.0	2018-09-25	MM	Forslag til struktur for dokumentet lavet Indledning til dokumentet lavet Protokol mellem MobilApp og RPiApp lavet
v0.3.0	2018-10-03	MM & FA	Opdateret BDD og tilføjelse af systemsekvensdiagrammer.
v0.4.0	2018-10-04	NT	IBD og domænemodel tilføjet
v0.4.1	2018-10-05	NT	Opdateret domænemodel og IBD. Rettet til Dansk
v1.0.0	2018-10-08	Alle	Sidste ændringer - klar til review #1

Tabel 2 - Versionshistorik

3 Indledning

I dette dokument beskrives systemarkitekturen i detaljer. Det indledes med en helt overordnet beskrivelse af systemet under brug af et BDD diagram med tilhørende blok- og signalbeskrivelser. I den overordnede beskrivelse indgår også systemsekvensdiagrammer for relevante use cases. Med "relevante use cases" menes de use cases som er beskrevet med "fully dressed" use case beskrivelser, og som der er specificeret en accepttest for.

Dette følges op af en beskrivelse af de protokoller, som systemets forskellige blokke benytter sig af for at kunne kommunikere med hinanden.

Dernæst følger et afsnit omhandlende hardwarearkitekturen. Her ses en nærmere beskrivelse af de enkelte hardwareblokke og deres indbyrdes forbindelser.

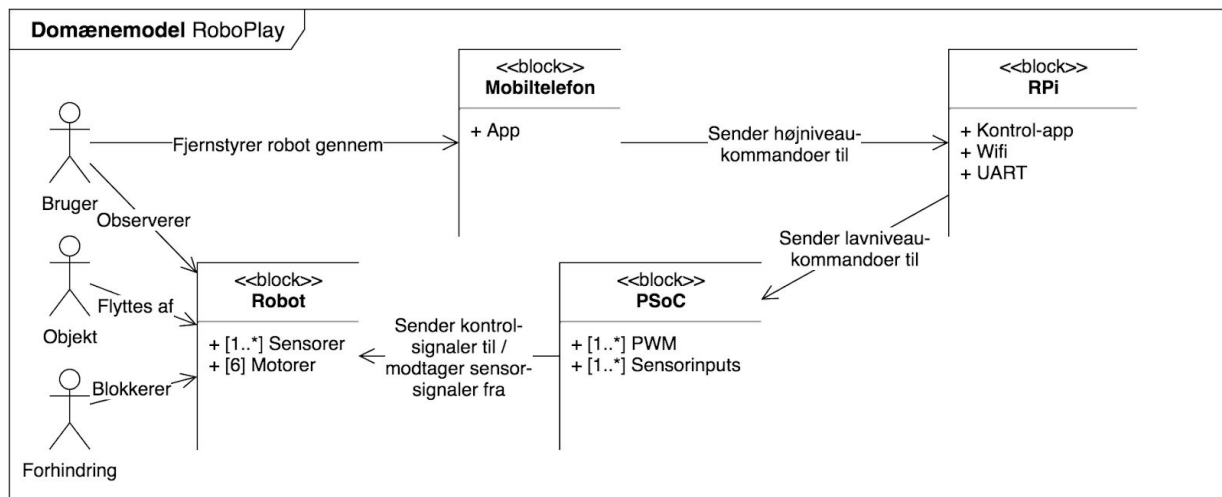
Afslutningsvis beskrives softwarearkitekturen under brug af applikationsmodeller, hvorfor der her gøres brug af sekvens-, klasse- og tilstandsdiagrammer.

4 Overordnet systemarkitektur

Dette afsnit beskriver systemarkitekturen på et overordnet plan.

4.1 Domænemodel

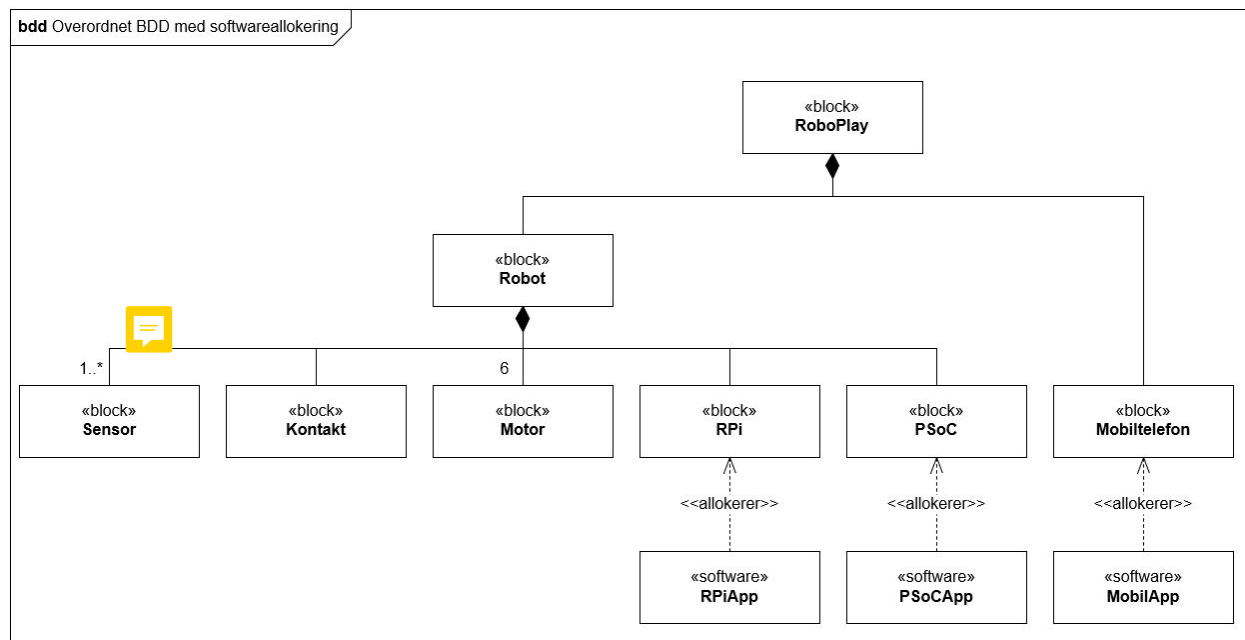
På figur 1 herunder ses en domænemodel for RoboPlay, som definerer systemets hardwareblokke og deres respektive ansvarsområder. På figuren ses også informationsflowet i systemet ved kommunikationsvejene mellem blokkene, og ligeledes også hvordan de tre aktører "Bruger", "Objekt" og "Forhindring" interagerer med systemet.



Figur 1 - Domæne model for RoboPlay

4.2 Blok-definitions-diagram

På figur 2 ses et BDD for RoboPlay og herpå fremgår også hvilken software, der allokeres til de forskellige blokke.



Figur 2 - Overordnet BDD med softwareallokering. De steder der er tale om 1-1 relationer er disse ikke anført.

I de følgende afsnit refereres der til de forskellige blokke og deres softwareapplikationer med de navne, som er angivet på BDD'et.

En beskrivelse af de enkelte blokke ses i *tabel 3*.

Blok	Beskrivelse
RoboPlay	Det samlede system som består af de øvrige blokke.
Robot	Selve robotten som kan styres via blokken mobiltelefon.
Sensor	En eller flere sensorer, som bruges til at indstille og begrænse robottens bevægelser.
Kontakt	Kontakt til at tænde og slukke robotten.
Motor	Robotten har 6 motorer, som styrer robottens bevægelige led.
RPI	Robottens "hjerne" som er ansvarlig for at omsætte instruktioner, der er modtaget fra mobiltelefonen til bevægelser i robotten via PSoC'en.
PSoC	PSoC'en er tilsluttet robottens sensorer og styrer på baggrund af input fra disse samt kommandoer modtaget fra RPI'en robottens motorer.
Mobiltelefon	Brugeren benytter en applikation, der er installeret på mobiltelefonen, til at styre robotten.
RPIApp	Softwareapplikationen der afvikles på RPI'en.
PSoCApp	Softwareapplikationen der afvikles på PSoC'en.
MobilApp	Softwareapplikationen der afvikles på mobiltelefonen.


Tabel 3 - Blokbetegnelse for systemet.

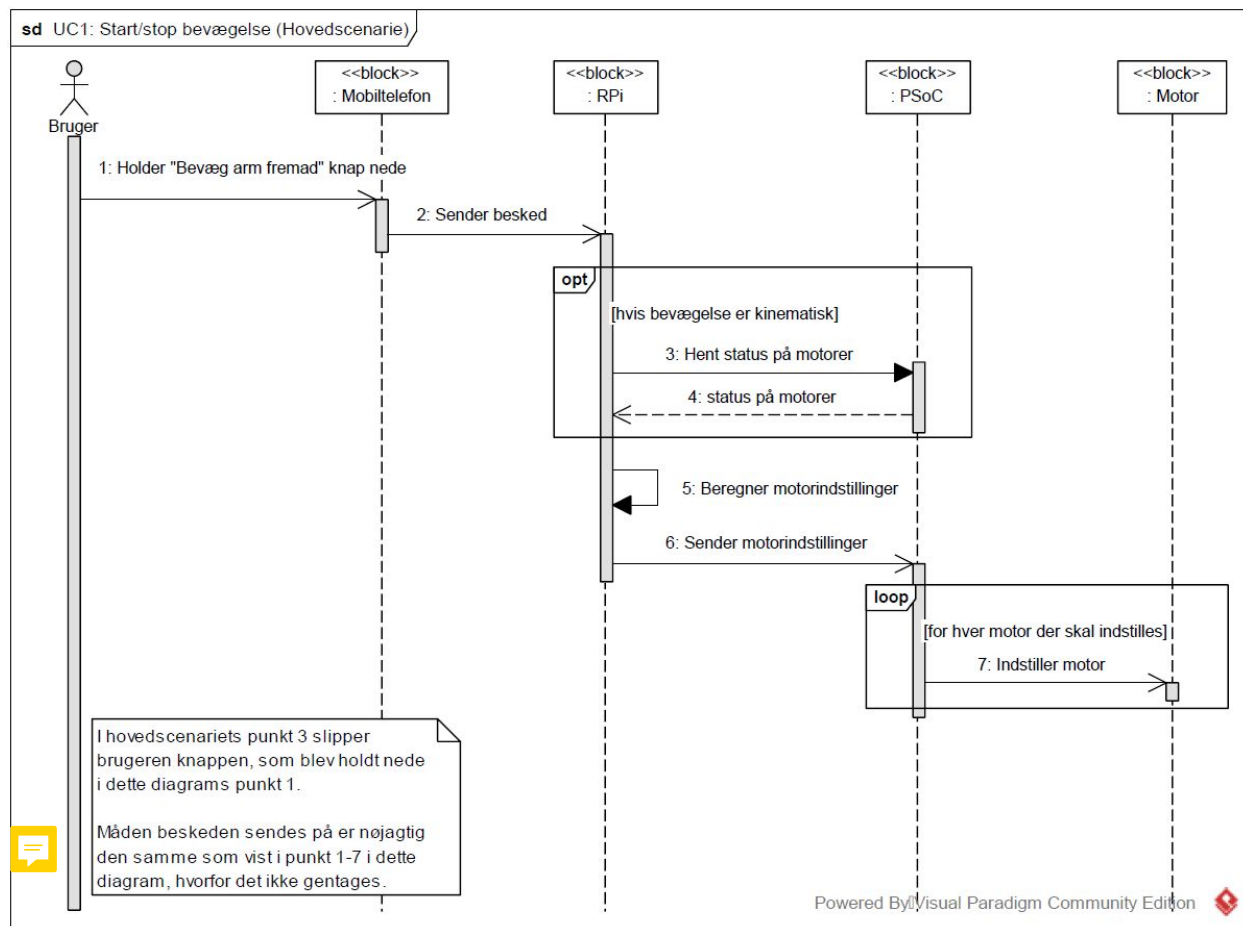
4.3 Systemsekvensdiagrammer for udvalgte use cases

I de følgende underafsnit ses en række systemsekvensdiagrammer for de vigtigste use cases. Diagrammerne giver et overblik over hvordan de forskellige blokke på det overordnede BDD (*figur 2*) taler sammen.

4.3.1 Systemsekvensdiagram for UC1's hovedscenarie

Systemsekvensdiagrammet, der ses på *figur 3*, viser samtlige bevægelser som brugeren kan starte og stoppe via mobiltelefonen - dog undtaget bevægelsen "Luk hånd" som håndteres i UC7. Extension 1 og 2 til denne use case har det samme flow som hovedscenariet, **hvorfor systemsekvensdiagrammer for disse extensions ikke vises.**

Fremgangsmåden er den samme uanset om en bevægelse startes eller stoppes. De fleste bevægelser er ikke kinematiske (fx bevægelsen "Åbn hånd"), hvorfor PSoC'en ikke inddrages. I det viste eksempel starter brugeren dog en kinematisk bevægelse. 

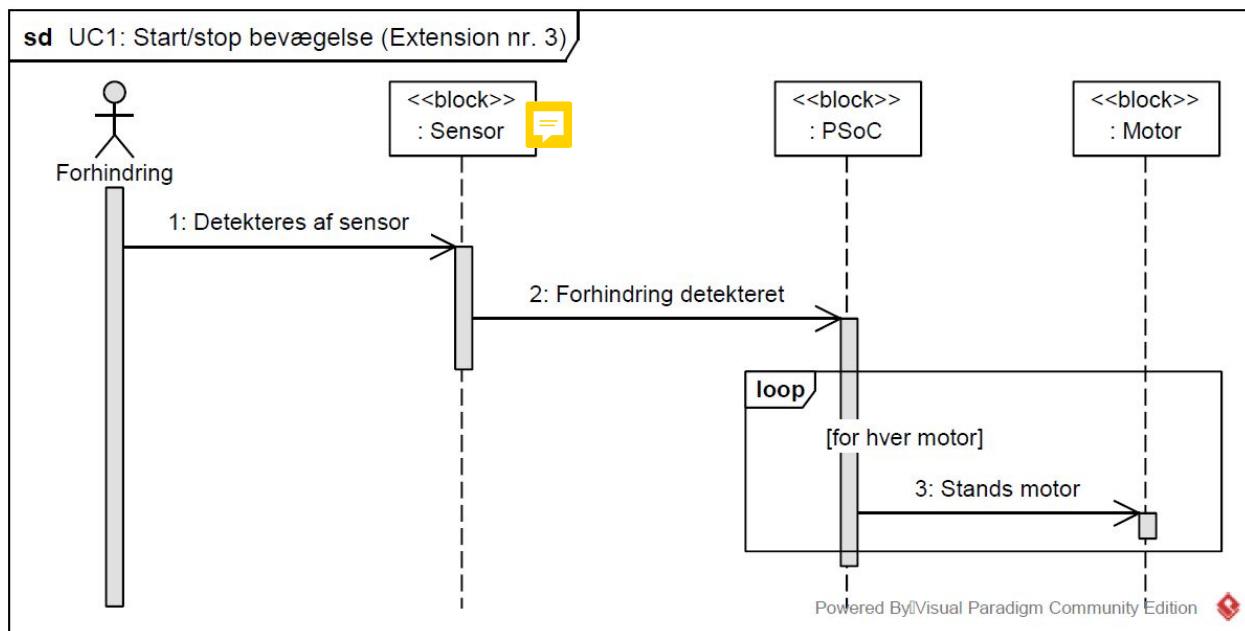


Figur 3 - Systemsekvensdiagram for UC1: Start/stop bevægelse (hovedscenarie). Brugeren holder en knap nede på mobiltelefonen, hvorefter kommandoen sendes til RPi'en. Hvis der er tale om en kinematisk bevægelse, så hentes motorernes nuværende positioner fra PSoC'en.

Uanset beregnes de nye motorindstillinger, hvorefter de sendes til PSoC'en, som indstiller de berørte motorer.

4.3.2 Systemsekvensdiagram for UC1's extension 3

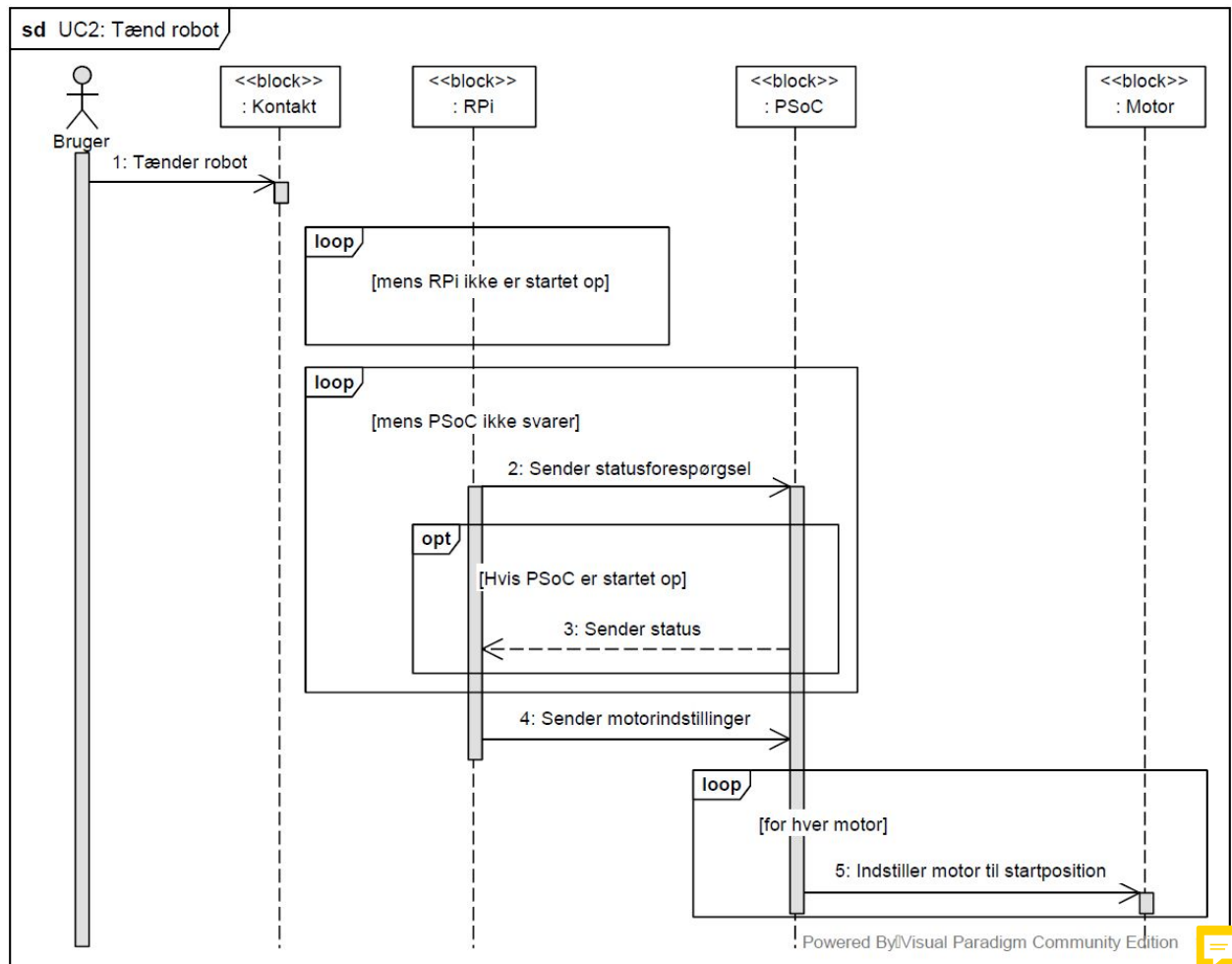
I dette systemsekvensdiagram ses situationen, hvor en bevægelse er påbegyndt som vist på *figur 3*, men hvor bevægelsen ikke afbrydes af brugeren men af en forhindring, som står i vejen for bevægelsens videre udførelse. Et eksempel på en sådan forhindring kunne være en væg.



Figur 4 - Systemsekvensdiagram for UC1's extension 3, hvor en sensor detekterer et objekt der forhindrer en bevægelses videre udførelse.

4.3.3 Systemsekvensdiagram for UC2's hovedscenarie

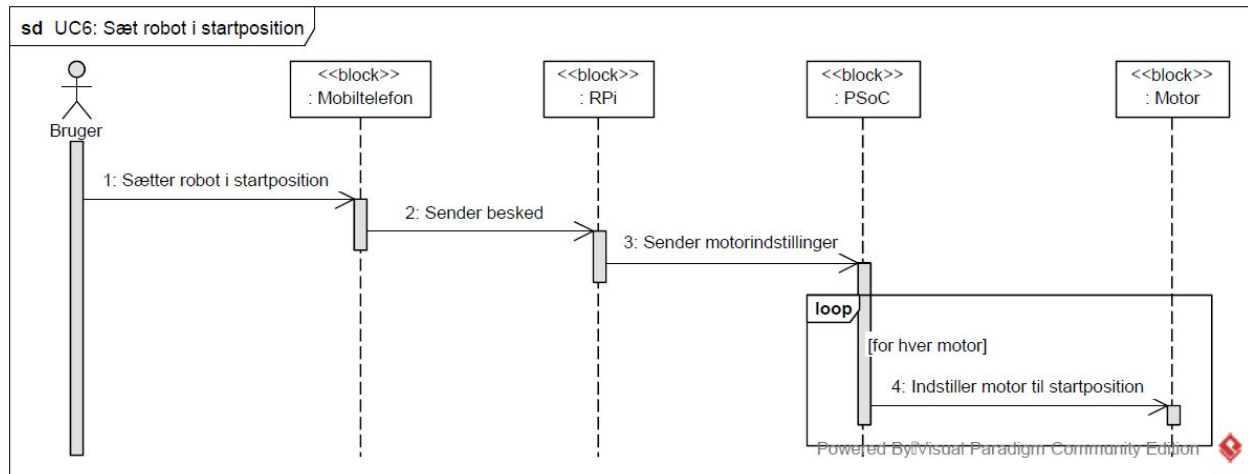
Situationen hvor brugeren tænder for robotten ved at påvirke en fysisk kontakt ses på *figur 5*.



Figur 5 - Systemsekvensdiagram for UC2: Tænd robot. Brugeren tænder robotten, hvorefter det afventes at RPi'en starter op. Derpå afventer RPi'en, at PSoC'en er klar til kommunikation og til sidst indstilles robottens motorer til deres startpositioner.

4.3.4 Systemsekvensdiagram for UC6's hovedscenarie

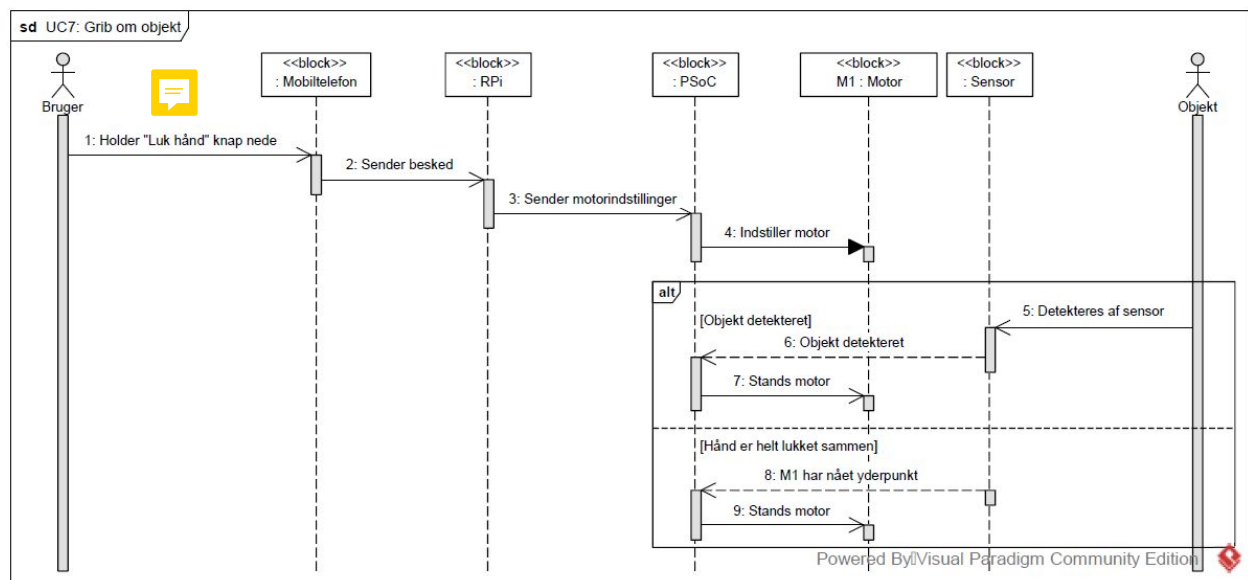
Systemsekvensdiagrammet for UC6, hvor brugeren sætter robotten i dens startposition via MobilApp'en ses på figur 6.



Figur 6 - Systemsekvensdiagram for UC6: Sæt robot i startposition. Brugeren indstiller robotten via mobiltelefonen, hvorefter RPi'en sender de nødvendige beskeder til PSoC'en, som har ansvar for at indstille robottens motorer.

4.3.5 Systemsekvensdiagram for UC7's hovedscenarie

Systemsekvensdiagrammet for UC7's hovedscenarie, hvor robottens hånd lukkes i et forsøg på at gribe om et objekt, ses på figur 7.



Figur 7 - Systemsekvensdiagram for UC7: Grib om objekt. Brugeren lukker robottens hånd for at gribe om et objekt og denne bevægelse standses enten ved at hånden klemmer om et objekt

eller hånden lukkes helt sammen. Bevægelsen kan også standses ved at brugeren slipper knappen, dette er dog ikke vist på diagrammet.

5 Kommunikationsprotokoller

I dette afsnit beskrives de protokoller, som benyttes til kommunikation mellem RoboPlay's forskellige delsystemer.

5.1 Protokol mellem MobilApp og RPiApp

Protokollen, der benyttes mellem MobilApp og RPiApp, bygger på TCP, og dataen, som sendes, består altid af 16 bit.

5.1.1 TCP klient/server

MobilApp'en opsættes som klient mens RPiApp fungerer som server. TCP protokollen understøtter tovejskommunikation, men kun envejskommunikation fra MobilApp til RPiApp implementeres i udviklingen af RoboPlay.

Forbindelsen mellem enhederne sker via et WIFI hotspot, som Raspberry Pi'en har ansvaret for at udbyde. Klienten forbinder til WIFI hotspotet, hvorefter TCP forbindelsen oprettes under følgende forudsætninger:

- Raspberry Pi'en skal have IP-adressen 10.9.8.2
- Den benyttede mobiltelefon tildeles en IP-adresse via DHCP
- RPiApp opsættes til at lytte efter klienter på port 4000

5.1.2 Protokol for udførelse af bevægelser

Jf. kravspecifikationen er det nødvendigt at styre robotten i 12 forskellige bevægelser, og hver af disse skal kunne både startes og stoppes.

Det implementeres således, at hver besked består af 16 bits. De 12 LSB'er er relateret til hver sin bevægelse således, at et 0 betyder en bevægelse skal stoppes, mens et 1-tal betyder, at en bevægelse skal påbegyndes.

I *tabel 4* ses en beskrivelse af betydningen af den enkelte bit.

Bit nr.	Beskrivelse
0 (LSB)	Starter/stopper bevægelsen "Åbn hånd".
1	Starter/stopper bevægelsen "Luk hånd".
2	Starter/stopper bevægelsen "Roter hånd med uret".
3	Starter/stopper bevægelsen "Roter hånd mod uret".
4	Starter/stopper bevægelsen "Tilt hånd op".
5	Starter/stopper bevægelsen "Tilt hånd ned".
6	Starter/stopper bevægelsen "Bevæg arm fremad".
7	Starter/stopper bevægelsen "Bevæg arm bagud".
8	Starter/stopper bevægelsen "Roter arm med uret".
9	Starter/stopper bevægelsen "Roter arm mod uret".
10	Starter/stopper bevægelsen "Bevæg arm op".
11	Starter/stopper bevægelsen "Bevæg arm ned".
12	Benyttes ikke.
13	Benyttes ikke.
14	Benyttes ikke.
15 (MSB)	Benyttes ikke.

Tabel 4 - Betydning for hver enkelt bit i protokollen mellem MobilApp og RPiApp. Et logisk "0" stopper en bevægelse, mens et logisk "1" starter en bevægelse.

I *tabel 5* ses eksempler på, hvordan bitmønstre for udvalgte beskeder ser ud.

Ønskede bevægelser	Bitmønster
Alle bevægelser stoppes	0000 0000 0000 0000
"Åbn hånd".	0000 0000 0000 0001


“Bevæg arm fremad” og “Bevæg arm op”.	0000 0010 0010 0000
---------------------------------------	---------------------

Tabel 5 - Eksempler på sammenhæng mellem den/de ønskede bevægelser og det tilhørende bitmønstre som udgør beskeden, der sendes fra MobilApp til RPiApp.

Visse kombinationer af kommandoer vil altid være ugyldige, idet det ikke giver mening fx at udføre bevægelserne “Åbn hånd” og “Luk hånd” samtidigt. Disse på hinanden modsat rettede kommandoer er grupperet i bitmønstret således, at bit nr. 0 og 1 ophæver hinanden, bit nr. 2 og 3 ophæver hinanden og så fremdeles.


Eksempler på ugyldige kommandoer ses i *tabel 6*.

Ønskede bevægelser	Bitmønstre
“Åbn hånd” og “Luk hånd”.	0000 0000 0000 0011
“Bevæg arm op” og “Bevæg arm ned”.	0000 1100 0000 0000

Tabel 6 - Eksempler på ugyldige kommandoer, som ophæver hinanden. Bemærk hvordan de ugyldige kommandoer er grupperet to og to. 


5.1.3 Protokol for indstilling af robot til startposition

For at nulstille robotten som beskrevet i UC6 benyttes kommandoen, som ses i *tabel 7*.

Ønskede bevægelser	Bitmønstre 
Alle robottens led nulstilles	1111 1111 1111 1111

Tabel 7 - Kommandoen som nulstiller alle robottens led.

5.2 Protokol mellem RPiApp og PSoCApp

Protokollen mellem RPiApp og PSoCApp bygger på UART  og er 2-vejs kommunikation. Den består altid af 32 bits, hvoraf den første byte fungerer som en start-byte til identifikation af starten på en besked, men også som angivelse af om beskeden er for at læse eller skrive.

Det er kun RPiApp som kan iværksætte kommunikationen. RPiApp kan sende en skrive-besked til PSoCApp, hvorved denne eksekveres, men intet svar returneres til RPiApp.

RPiApp kan også sende en læse-besked til PSoCApp, som ikke er til eksekvering, men hvor PSoCApp returnerer med svar til RPiApp.

Når der i denne protokol beskrives en bytes indhold af bits er det fra MSB mod LSB, hvor MSB er bit nr. 7 og LSB er bit nr. 0.

5.2.1 Opsætning

Både RPiApp og PSoCApp skal opsættes til at benytte UART med følgende indstillinger:

- Databits: 8
- Stopbits: 1
- Parity: Nej
- Baud rate: 115200 bps




5.2.2 Besked-struktur

Hver besked mellem RPiApp og PSoC består af 32 bits fordelt på 4 bytes, hvoraf den første er en start-byte, som bestemmer om det er en besked som skal læse fra modtageren eller skrive til modtageren. De sidste 3 bytes er data-bytes.

5.2.2.1 Byte #1

Den første byte i hver besked er en unik start-byte, som betegner starten på en besked for modtageren. Den første byte kan være én af to muligheder, som afhænger af om der ønskes at læse fra modtageren eller skrive til modtageren, som det ses af *tabel 8*.


Binært	Decimal 	Handling
11111110	254	Læse fra modtager
11111111	255	Skrive til modtager

Tabel 8 - Oversigt over første byte i protokollens besked

5.2.2.2 Byte #2

Den anden byte er en data-byte indeholdende antal grader for anden akse eller et bitmønster, der anmoder anden akse om at standse sin bevægelse.

Graderne angives i heltal af typen unsigned int fra 0 til 180. Det variable gradantal er anført som teksten *grader tabel 9*.

Binært	Decimal	Handling
11000000	192	Stop bevægelse
<i>grader</i>	<i>grader</i>	Antal grader for første akse 

Tabel 9 - Oversigt over anden byte i protokollens besked

5.2.2.3 Byte #3

Den anden byte er en data-byte indeholdende antal grader for tredje akse eller et bitmønster, der anmoder tredje akse om at standse sin bevægelse.

Graderne angives i heltal af typen unsigned int fra 0 til 180. Det variable gradantal er anført som teksten *grader tabel 10*.

Binært	Decimal	Handling
11000000	192	Stop bevægelse
<i>grader</i>	<i>grader</i>	Drej til angivne grader

Tabel 10 - Oversigt over tredje byte i protokollens besked

5.2.2.4 Byte #4

Den fjerde og sidste byte er en data-byte, som indeholder bitmønstre til første, fjerde, femte og sjette akse.

Bit 7 og 6 vedrører første akse og bitmønstrene er som følger:

Binært	Handling
00	Stop bevægelse
01	Drej til venstre
10	Drej til højre

Tabel 11 - Bitmønstre for første akse

Bit 5 og 4 vedrører fjerde akse og bitmønstrene er som følger:

Binært	Handling
00	Stop bevægelse
01	Tilt nedad
10	Tilt opad

Tabel 12 - Bitmønstre for fjerde akse

Bit 3 og 2 vedrører femte akse og bitmønstrene er som følger:

Binært	Handling
00	Stop bevægelse
01	Roter højre
10	Roter venstre

Tabel 13 - Bitmønstre for femte akse

Bit 1 og 0 vedrører sjette akse og bitmønstrene er som følger:

Binært	Handling
00	Stop bevægelse
01	Åbn
10	Luk

Tabel 14 - Bitmønstre for sjette akse

6 Hardwarearkitektur

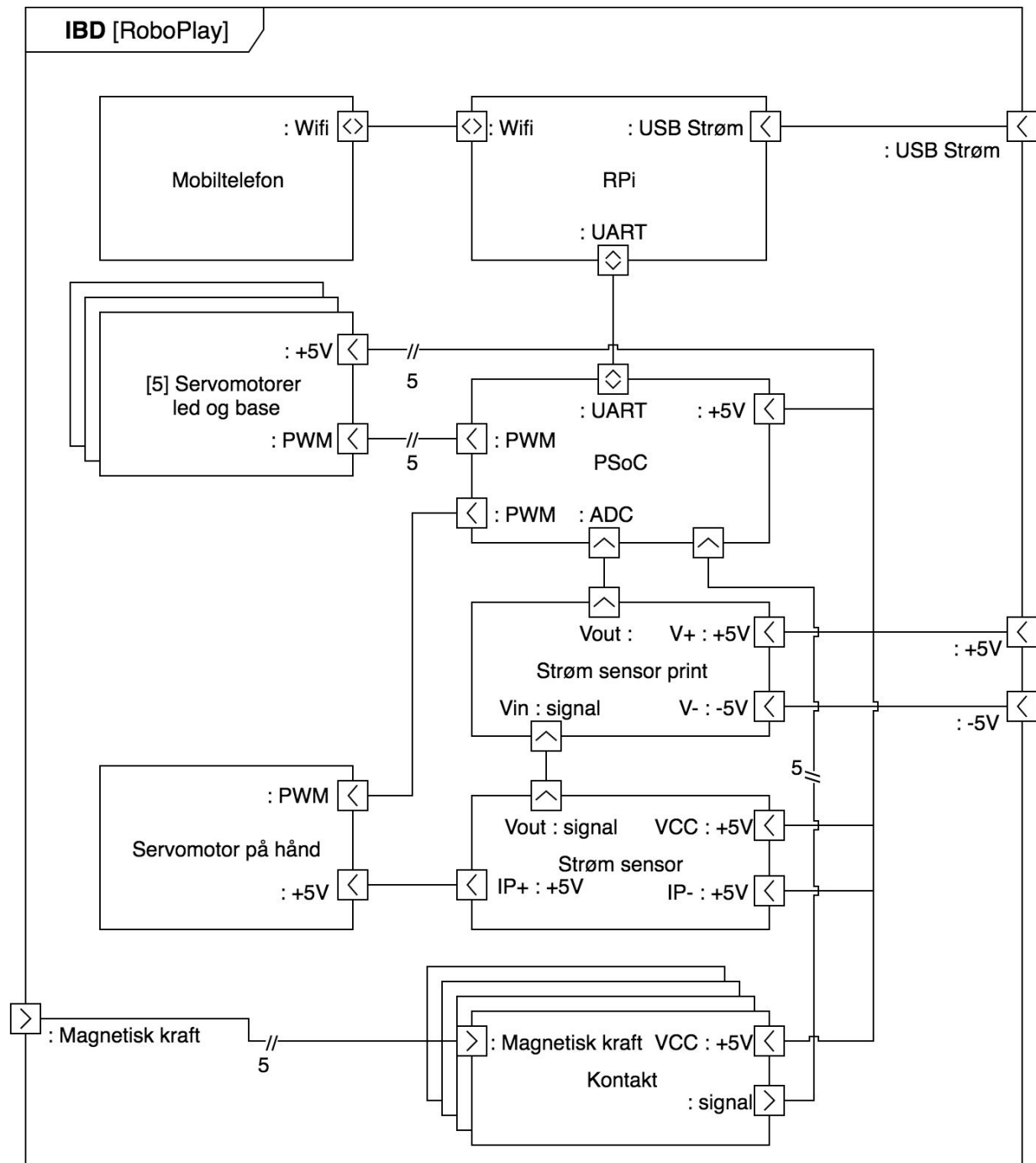
TODO

6.1 Udvidet BDD

TODO

6.2 Internt blokdiagram

Herefter, i *figur 8* ses et internt blokdiagram for RoboPlay. Dette er et midlertidigt diagram med de i skrivende stund kendte informationer, og vil blive ændret i fremtiden som gruppen lærer mere om produktet. Stelforbindelser er underforståede.



Figur 8 - IBD for RoboPlay

7 Softwarearkitektur

I dette afsnit beskrives arkitekturen for den software, der kører på hvert enkelt delsystem. Dette gøres med såkaldte applikationsmodeller, hvorfor der i det følgende anvendes klasse- sekvens- og tilstandsdiagrammer i det omfang, det er nødvendigt.

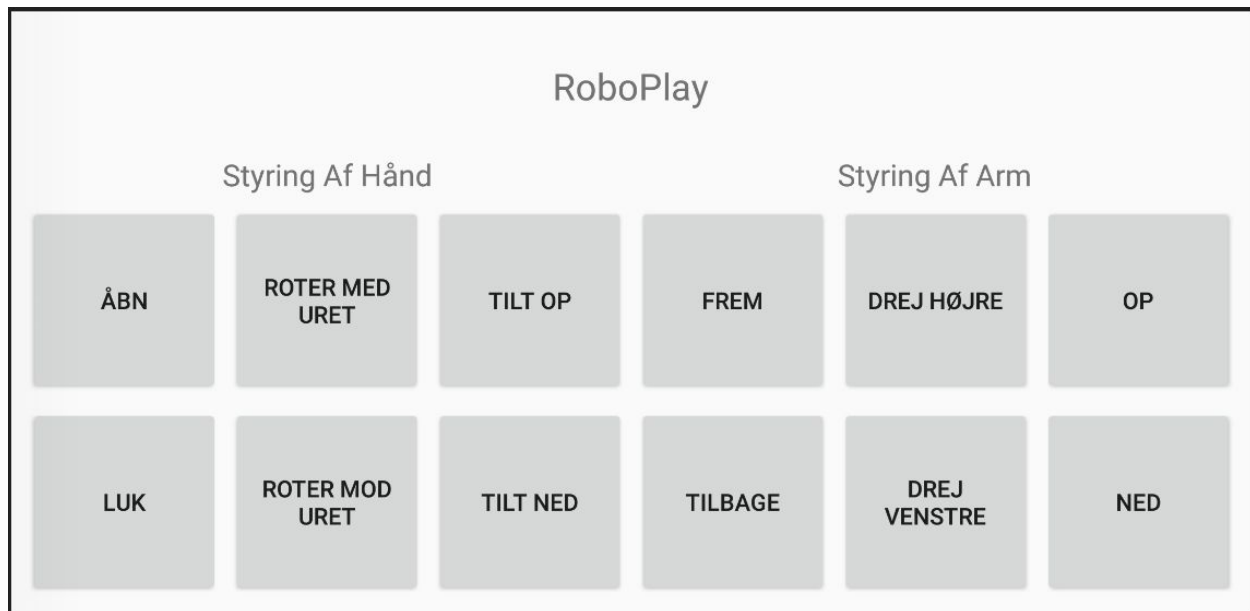
7.1 MobilApp

Applikationsmodellerne for MobilApp er lavet under den antagelse, at applikationen laves under brug af teknologien Xamarin og dermed programmeres i sproget C#. Som en naturlig følge heraf er den anvendte arkitektur MVVM (Model-View-ViewModel).

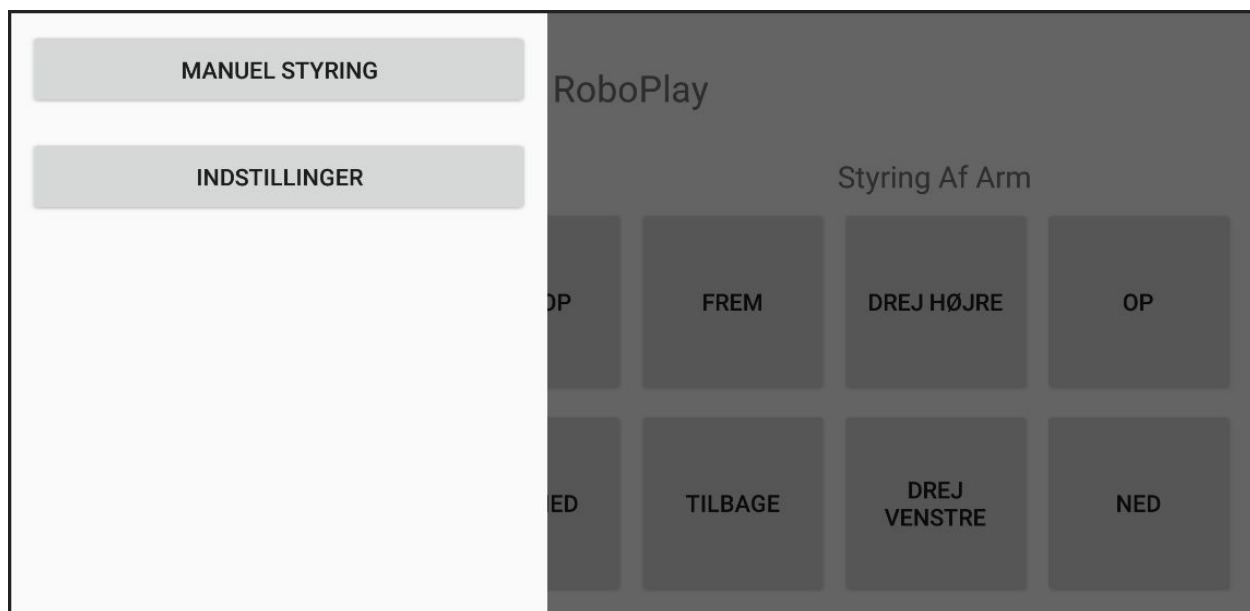
MobilApp designes med et klassisk MasterDetailView. Med dette menes at applikationen vil have følgende to former for views:

- En skuffemenu som kan trækkes ud fra venstre side med en enkelt finger. I denne menu kan applikationens forskellige views vælges. Der refereres til denne skuffemenue som “master-viewet”.
- Når et view vælges i master-viewet så præsenteres det valgte view på skærmen. Dette kaldes også at sætte “detail-viewet” til det valgte view.

Herunder ses screenshots af MobilApp, som den tænkes designet, og disse illustrerer hvordan MasterDetailViewet fungerer.

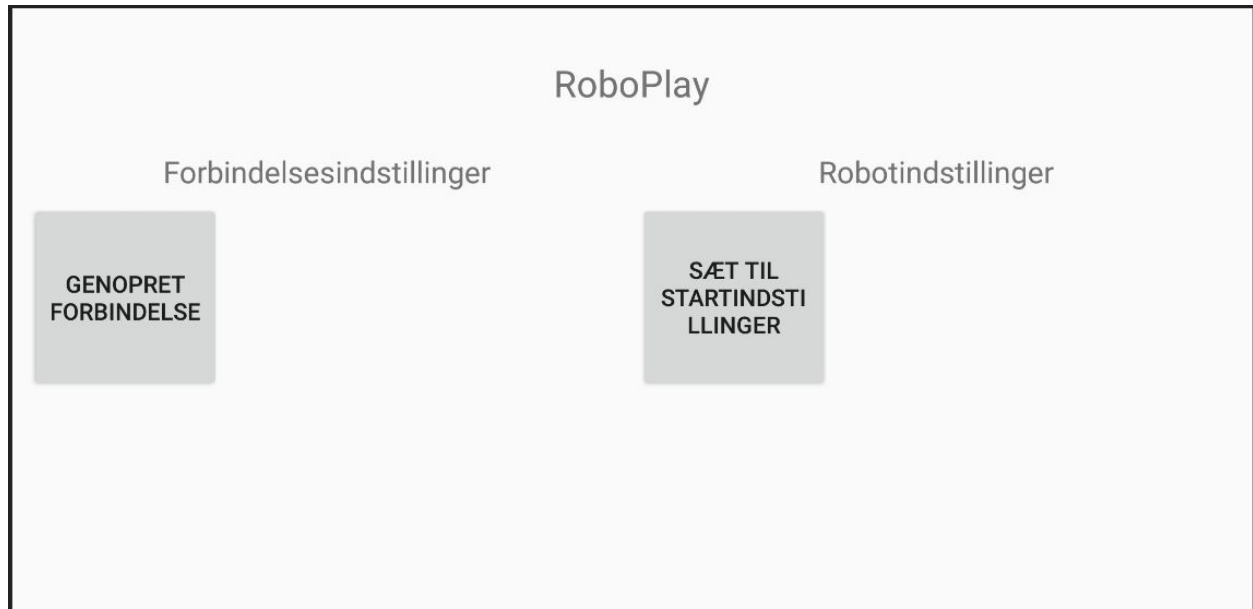


Figur 9 - Her ses MobilApp's opstartsskærm, som er detail-viewet, der er udfyldt med skærm billedet for manuel styring af robotten. Mens en knap holdes nede udføres bevægelsen - ellers er bevægelsen stoppet.



Figur 10 - Her ses master-viewet efter det er trukket ind over skærmen, hvilket sker ved at placere en finger ved skærmens venstre kant, hvorefter fingeren trækkes mod højre. Læg

mærke til, at skærbilledet for manuel styring af robotten ses bag master-viewet. Ved tryk på menupunkterne kollapsedes master-viewet og det nye detail-view vises.



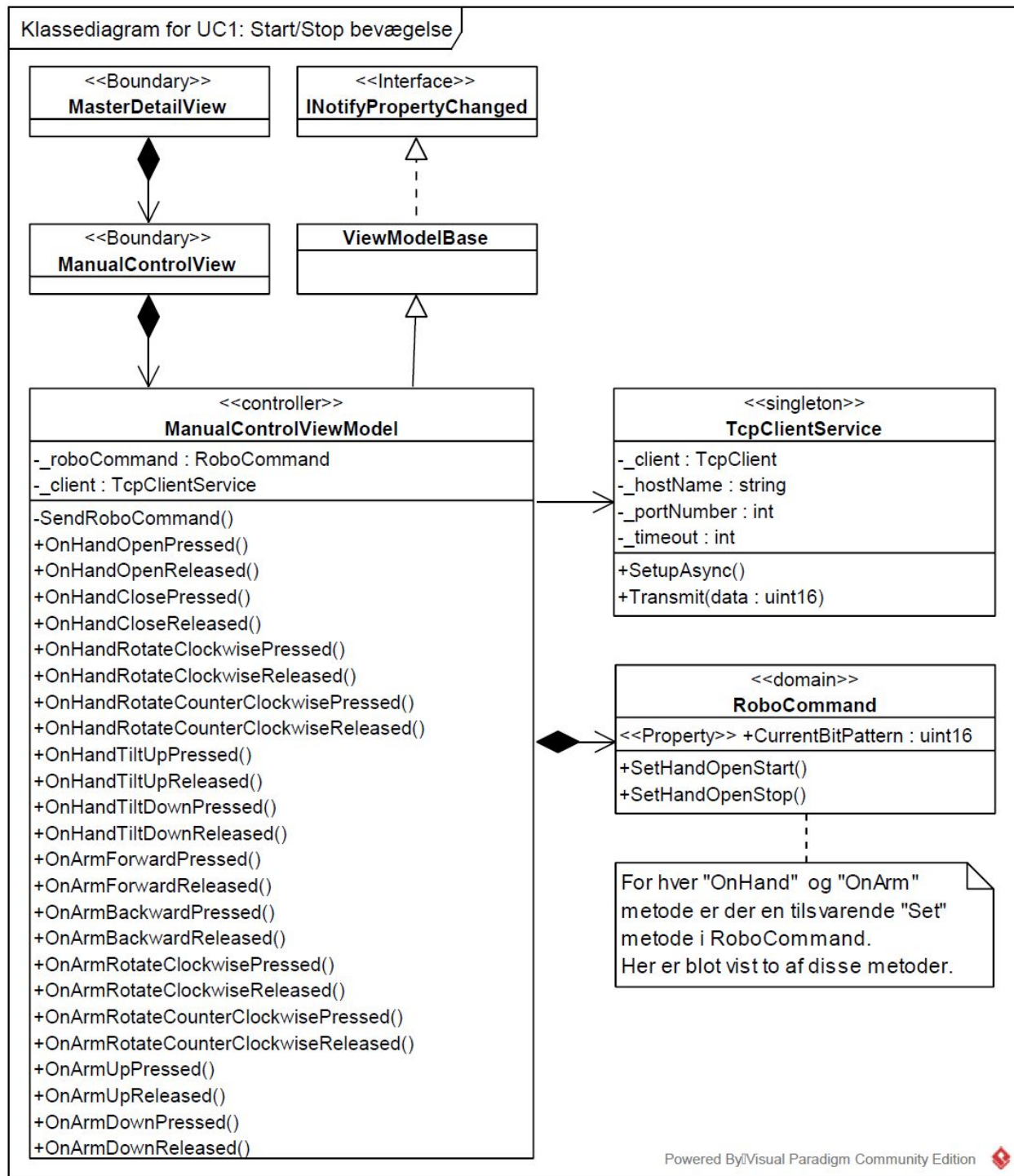
Figur 11 - Her ses detail-viewet, hvor skærbilledet for indstillinger er valgt.

7.1.1 Applikationsmodel for UC1

Denne applikationsmodel viser den underliggende arkitektur for skærbilledet, som ses på figur 9.

7.1.1.1 Klassediagram

Klassediagrammet for UC1 ses på figur 12. En kort forklaring af de vigtigste klasser ses i forlængelse af klassediagrammet.



Figur 12 - Klassediagrammet for UC1's applikationsmodel. Bemærk at klassen RoboCommand har en række yderligere metoder, som er relateret til metoderne i ManualControlViewModel.

MasterDetailView

Er selve MasterDetailViewet, som er beskrevet i det tidligere afsnit.

ManualControlView

Er selve viewet som ses på *figur 9*. Viewet er knyttet til en view model i form af ManualControlViewModel.

ManualControlViewModel

Fungerer som controller for use casen. Alt logik omhandlende de forskellige knapper, der ses i ManualControlView, håndteres altså her. Nedarver fra klassen ViewModelBase, som implementerer interfacet INotifyPropertyChanged. Herved notificeres view og model automatisk om ændringer i applikationen.

RoboCommand

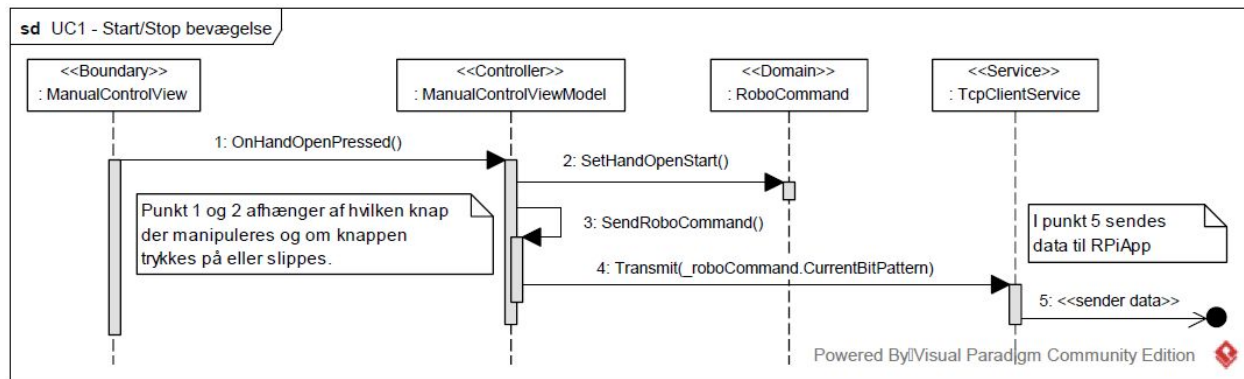
Har en property kaldet "CurrentBitPattern" som repræsenterer et bitmønster, der kan sættes til de forskellige tilstande, som protokollen mellem MobilApp og RPiApp understøtter. Har desuden en række metoder til at sætte bitmønstret afhængig af hvilke(n) kommando der ønskes fremsendt.

TcpClientService

Håndterer kommunikationen med RPiApp over TCP protokollen via et objekt af .NET klassen TcpClient.

7.1.1.2 Sekvensdiagram for hovedscenariet

På *figur 13* ses sekvensdiagrammet for UC1's hovedscenarie. Sekvensdiagrammet er meget simpelt, da der ikke er to-vejskommunikation mellem MobilApp og RPiApp.



Figur 13 - Sekvensdiagram for UC1's hovedscenarie.

Af diagrammet ses følgende:

- I punkt 1 holder brugeren en knap nede eller har sluppet en knap. I eksemplet ovenfor er knappen "Åbn hånd" holdt nede.
- I punkt 2 sætter controlleren bitmønstret jf. den vedtagne protokol, som er beskrevet tidligere i dette dokument.
- I punkt 3 og 4 kaldes metoden Transmit(...) med bitmønstret som parameter på TcpClientService-objektet.
- I punkt 5 sendes dataen til RPiApp.

7.1.2 Applikationsmodel for UC2

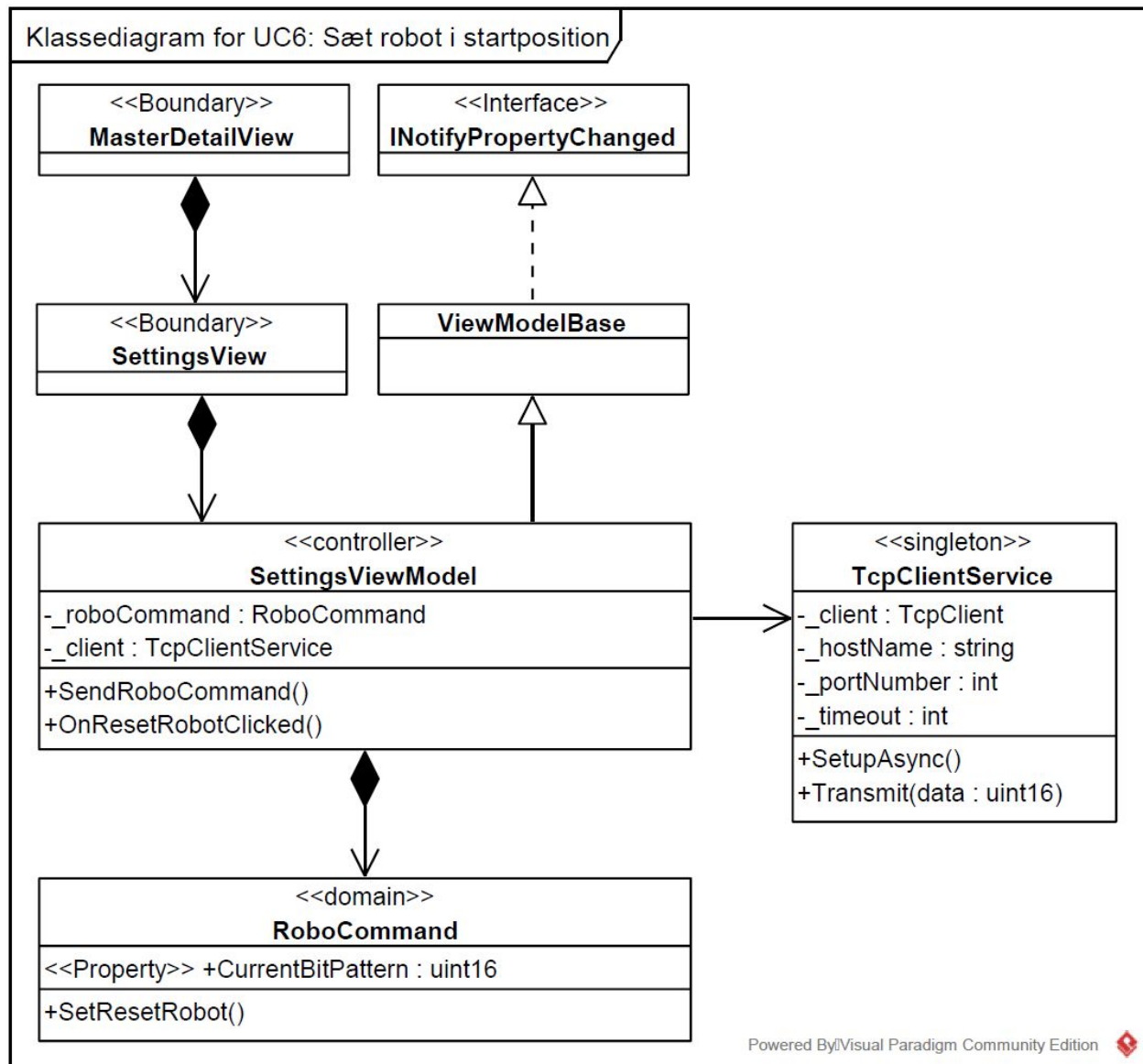
En applikationsmodel er ikke relevant, idet denne use case ikke involverer MobilApp.

7.1.3 Applikationsmodel for UC6

Denne applikationsmodel viser den underliggende arkitektur for skærbilledet, som ses på figur 11.

7.1.3.1 Klassediagram

Klassediagrammet for UC6 ses på figur 14. En kort forklaring af de klasser, som ikke er forklaret i applikationsmodellen for UC1, følger efter diagrammet.



Figur 14 - Klassediagrammet for UC6's applikationsmodel.

SettingsView

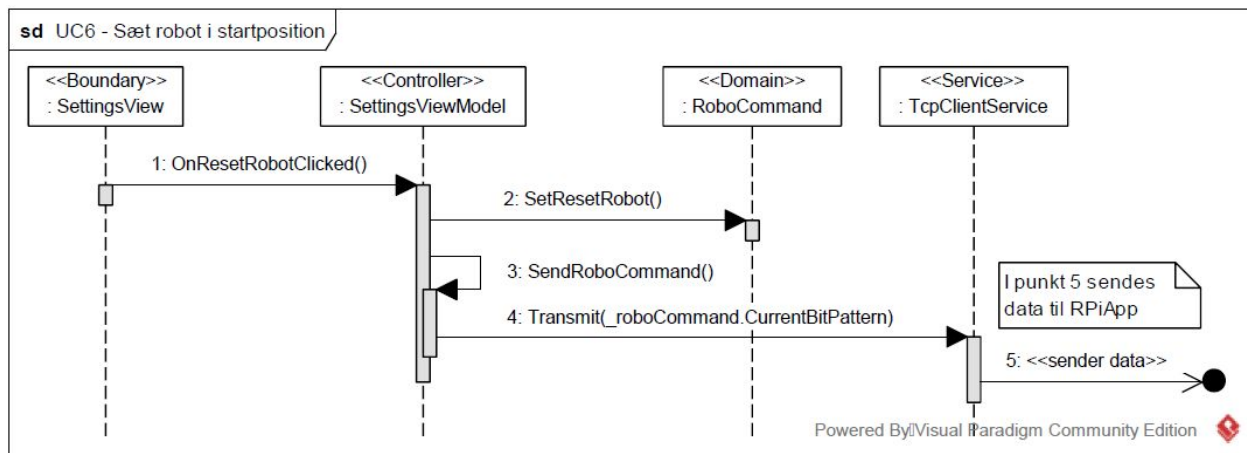
Er selve viewet som ses på figur 11. Viewet er knyttet til en view model i form af SettingsViewModel.

SettingsViewModel

Fungerer som controller for use casen. Alt logik omhandlende de forskellige knapper, der ses i SettingsView, håndteres altså her. Nedarver fra klassen ViewModelBase, som implementerer interfacet INotifyPropertyChanged. Herved notificeres view og model automatisk om ændringer i applikationen.

7.1.3.2 Sekvensdiagram for hovedscenariet

På figur 15 ses sekvensdiagrammet for UC6's hovedscenarie. Sekvensdiagrammet adskiller sig fra UC1's på den måde, at det ses, at et andet view og en anden view model er i brug. Metodekaldene i punkt 1 og 2 er også forskellige, men det er i princippet det samme der sker.



Figur 15 - Sekvensdiagram for UC6's hovedscenarie.

7.1.4 Applikationsmodel for UC7

For MobilApp er denne use case set fra et implementeringsmæssigt perspektiv fuldstændig identisk med UC1. Forskellene i de to use cases opstår først i RPiApp, hvorfor der her blot kan henvises til figur 13.

7.2 RPiApp

7.2.1 Applikationsmodel for UC1

7.2.2 Applikationsmodel for UC2

7.2.3 Applikationsmodel for UC6

7.2.4 Applikationsmodel for UC7

7.3 PSoCApp

7.3.1 Applikationsmodel for UC1

7.3.2 Applikationsmodel for UC2

7.3.3 Applikationsmodel for UC6

7.3.4 Applikationsmodel for UC7