Projektopgave forår 2014 – juni 2014
02324 Videregående programmering.

CDIO Del 2 – Videregående programmering
Gruppe nr.: *55.* Afleveringsfrist: mandag den 24/3 2014 Kl. 05:00
Denne rapport er afleveret via Campusnet (der skrives ikke under).

Denne rapport indeholder 19 sider inkl. denne side, samt 6 sider bilag.

S133991, Larsen, Anders *s133988, Magnussen, Malte* *s133974, Hansen, Nicolai*

Kontakt person (Projektleder)



s134010, Jensen, Lars *s133980, Liang, Jiahua* *s133970, Hansen, Kristin*



Timeregnskab

CDIO del 2 - Gruppe 55

Time-regnskab

Dato	Deltager	Design	Implementering	Test	Dokumentering	Andet	I alt
03-03-14	Anders	0	0	0	0	0	0
	Kristin	1			1		2
	Lars Peter	1			1		2
	Malte	1			1		2
	Jiahua	1			1		2
	Nicolai	0			1	1	2
10-03-14	Anders	0	0	0	0	0	0
	Kristin				1	1	2
	Lars Peter	1				1	2
	Malte				1	1	2
	Jiahua	1			1		2
	Nicolai				2		2
17-03-14	Anders	0	0	0		0	0
	Kristin		2				2
	Lars Peter		2				2
	Malte		2				2
	Jiahua		2				2
	Nicolai				2		2
23-03-14	Anders	0	1	0	5	2	8
	Kristin		8				8
	Lars Peter	2	2	2	2		8
	Malte	1	2	2	1	2	8
	Jiahua	2	2	2	2		8
	Nicolai		2	2	4		8
Sum		11	25	8	26	8	78
I alt pr. person	Anders	0	1	0	5	2	8
	Kristin	1	10	0	2	1	14
	Lars Peter	4	4	2	3	1	14
	Malte	2	4	2	3	3	14
	Jiahua	4	4	2	4	0	14
	Nicolai	0	2	2	9	1	14

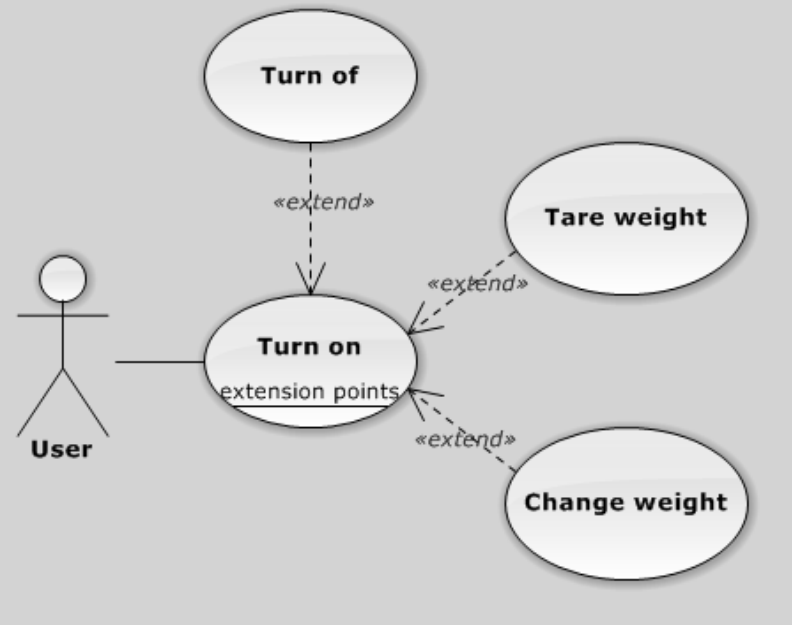
Indholdsfortegnelse

Indledning

I dette projekt ønskes der en vægtsimulator med console interface, som skal simulerer en Mettler BBK vægt. Vægten skal simulerer den virkelige vægts nødvendige kommandoer nøjagtigt, sådan så det skulle være muligt at direkte kunne erstatte simulatoren med vægten og der ingen ændringer skal fortages i nogle eksterne programmer i fremtiden. Vi har valgt at benytte os af nødvendige diagrammer til at beskrive hele/dele af programmets procedurer, samt nogle af de brugte metoder til opstilling af programmet (GRASP og 3 lags modellen).

Use Case

Diagram



Beskrivelse

Scoope: Vægtsimulator

Primary Actor: Bruger

Stakeholders and Interests:

Kunden forventer at få en simulator der efterligne en fysisk vægt.

Main Succes Scenario:

1. Brugeren tænder for vægtsimulatoren.
2. Brugeren skriver kommando til simulatoren.
   1.1. Brugeren ændrer belastningen på vægten.
   1.2. Brugeren tarerer belastningen på vægten.
   1.3. Brugeren aflæser belastningen på vægten.
   1.4. Brugeren ændrer displayteksten på vægten.
   1.5. Brugeren ændrer serverens port i simulatorens kommandolinje.

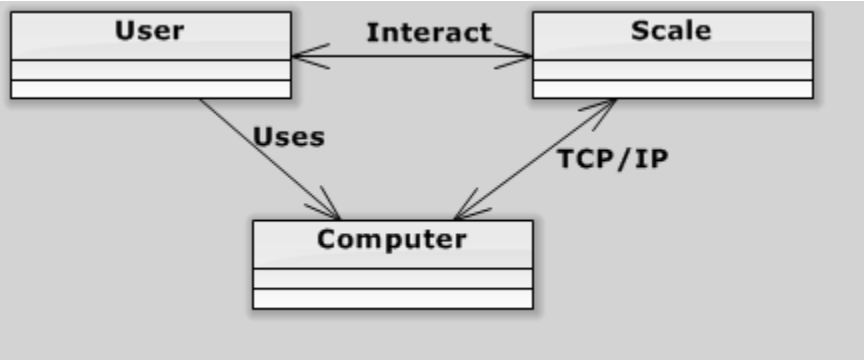
2. Brugeren slukker for vægtsimulatoren.

Extensions:

2a. Simulatoren "forstår" ikke kommandoen.

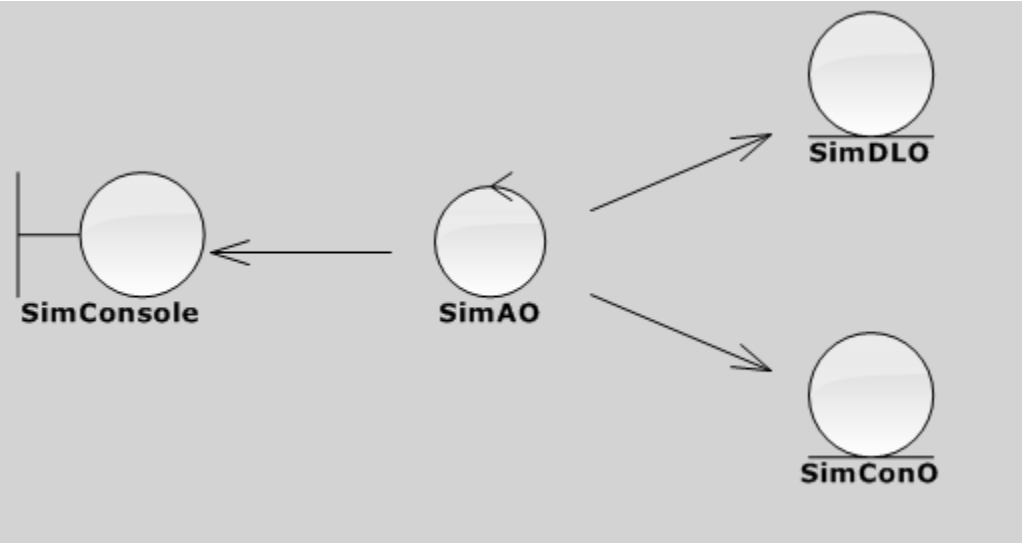
1. Kommandoen refererer ikke til nogen funktion – vægten gør intet.
2. Brugeren får mulighed for at indtaste korrekt kommando på tom linje.

Domændemodel



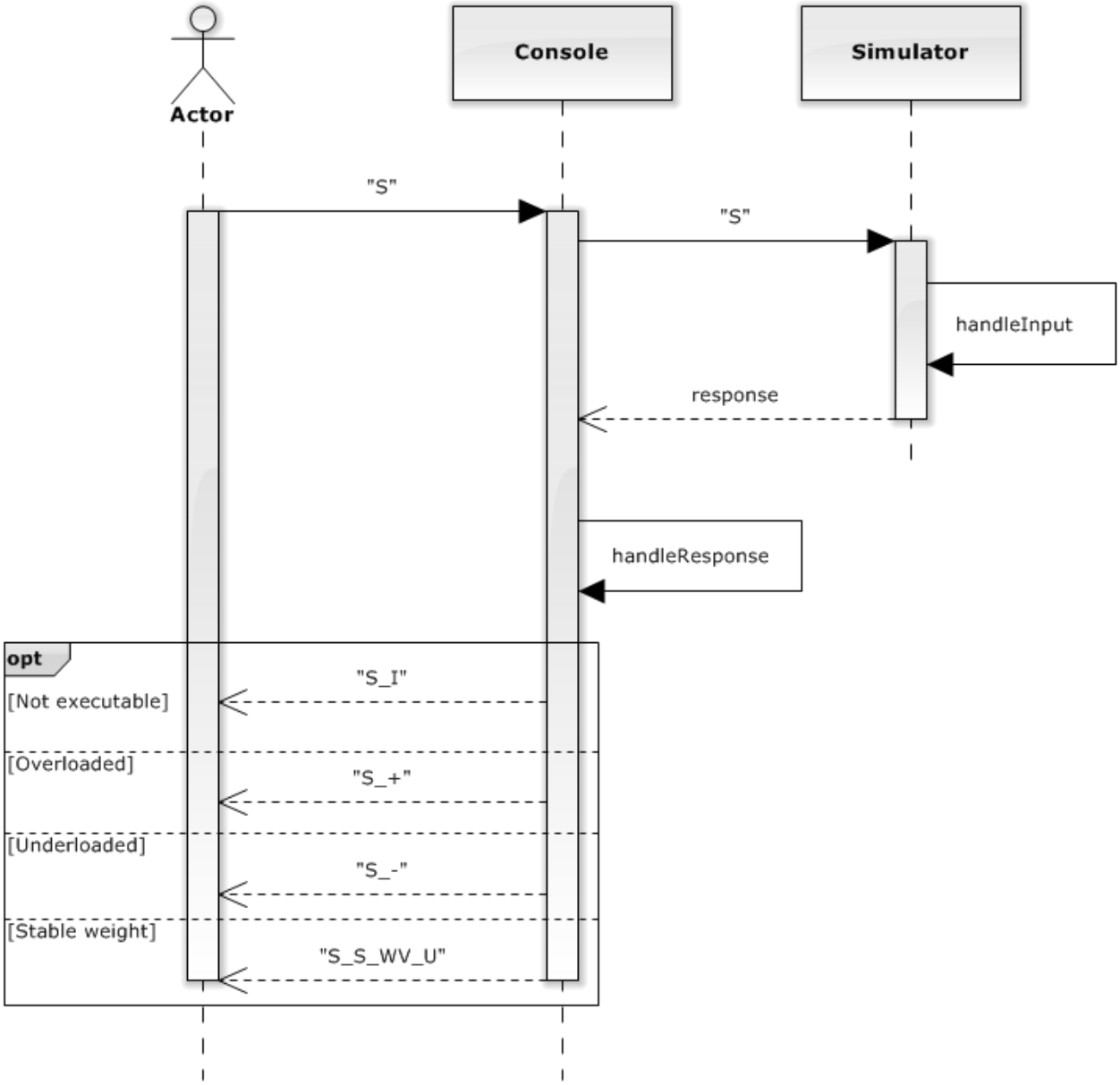
Vores domænemodel beskriver, hvordan en person ville komme i kontakt med vægten i det virkelige liv. Der er to muligheder for "user" at komme i kontakt med vægten, han kan enten stå lige foran vægten. Den anden mulighed er igennem en computer, her vil computeren connecte til vægten igennem en TCP/IP-connection. Hvis "user" bruger mulighed et, får han/hun alle outputs direkte fra vægtens skærm, og knapperne på vægten giver mulighed for inputs. Ved mulighed to skal "user" bruge et program på computeren til at give inputs, såvel som at få outputs af vægten.

BCE – model



Her kan man se vores BCE model, vi har Simconsole som en boundry da det er den der har ”kontakt” med brugeren. SimAO er vores controler, da den instantiere de andre klasser i sig selv, og derfor er det den der ”styrer” programmet. SimDLO og SimConO er vores to entities, de bliver instatantieret inde i SimAO, og er derfor entities da de bliver brugt i SimAO som er vores controler.

Systemsekvensdiagram



Figur 1 – Systemsekvensdiagram. Processen er ens for samtlige inputs, mens der selvfølgelig er forskellige responses for hvert input

Der er udviklet et generelt systemsekvensdiagram, der gerne skulle give et billede af, hvordan aktører og systemer interagerer med hinanden (rent overfladisk.)

Der er ikke indtegnet for samtlige kommandoer, da flowet er ens for samtlige kommandoer. De eneste forskelle findes i ”opt”-vinduet, der har forskellige guard-conditions for diverse svar. Disse guard-conditions er simulatorens

Det er vigtigt at forstå, at ”Actor” både kan være et eksternt system, der opretter forbindelse til simulatoren med TCP/IP, ligesom det kan være en fysisk person, der skriver direkte i konsolvinduet for simulatoren.

Der er en undtagelseskommando, der indstiller vægten (belaster vægten med det indtastede i kg.) Dette er nødvendigt, da der ellers aldrig ville være andet end 0.00 kg på vægten. Dette er selvfølgelig en kommando, der ikke eksisterer på den fysiske vægt.

Kommandoer og svar

Tabel over tilgængelige kommandoer, samt deres respektive svarmuligheder og beskrivelser.

CMD	Beskrivelse	Svarmuligheder
I0	Tilgængelige kommandoer	I0_B_x1_"1. Command" I0_B_x1_"2. Command" : : I0_A_x1_"Last command"
S	Send nuværende stabile vægt	**S_S_WeightValue_Unit**, *f.eks. "S_S_100.00_kg". Der skal noteres, at der altid returneres det samme antal karakterer efter "S_S_" på 7, efterfulgt af et komma og to decimaler (10 karakterer i alt.)* **S_I**, *kommandoen kan ikke eksekveres, men er modtaget. Dette kan f.eks. skyldes, at der i øjeblikket er en anden kommando, der er ved at blive udført.* **S_+**, vægten er for stor til at kunne blive læst. **S_-**, *vægten er for lille til at kunne blive registreret.*
SI	Send nuværende vægt omgående	**S_S_WeightValue_Unit**, se svar for "S"-kommando **S_D_WeightValue_Unit**, *"D" indikerer, at den returnerede vægt ikke er den stabile vægt.* **S_I**, *se svar for "S"-kommando.* **S_+**, *se svar for "S"-kommando.* **S_-**, *se svar for "S"-kommando.*
T	Tarering	T_S_WeightValue_Unit T_I T_+ T_-

TI	Omgående tarering	TI_S_WeightValue_Unit TI_D_WeightValue_Unit TI_I TI_L TI_+ TI_-
TA	Send nuværende tareringsværdi	TA_A_TareWeightValue_Unit TA_I
Z	Nulstil	Z_A, vægt er blevet nulsillet. Dvs. brutto = netto + tara. Z_I, kommandoen kan ikke eksekveres, men er modtaget. Dette kan f.eks. skyldes, at der i øjeblikket er en anden kommando, der er ved at blive udført. Z_+ Z_-
ZI	Nulstil omgående	ZI_D ZI_S ZI_I ZI_+ ZI_-
D	Displaytekst	D_A D_R D_I D_L
DW	Vis nuværende vægt (reelt set sættes vægten til at gå over på "vægttilstand", så den er i stand til at læse genstandes vægt. Da	DW_A DW_I

	simulatoren ikke understøtter andre tilstande, vil denne kommando blot udskrive en standard ”vægtvindue”.	
M21	Send/indstil vægtens enhed	M21_B_Des_Unit M21_B_... M21_A_Des_Unit

Vedr. tabel, kan flg. beskrivelser være behjælpelige:

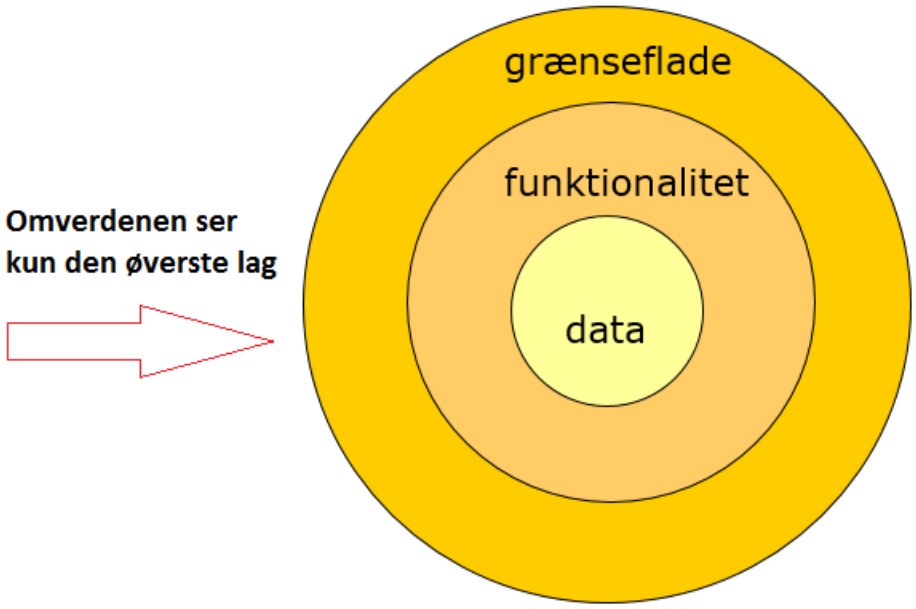
Underscore (_) skal læses som et mellemrum. Alle svar kommer som en samlet streng – nogle svar indeholder escaped anførselstegn. Skal der sendes en streng til simulatoren, skal anførselstegn medtages, og skal manuelt escapes (f.eks. ”D \”test\”” er den korrekte syntaks, hvis der ønskes ”test” som displaytekst.

”WeightValue” returneres altid som 10 karakterer, som også inkluderer komma og to decimaler. Dvs. en hvilken som helst vægt over 9.999.999,99 ikke kan vises (vægten modtager slet ikke denne størrelse, ligesom den heller ikke modtager 1.000.000,00, så dette er absolut ikke noget problem.) Mens der her i eksemplet bruges komma-separator, bruger vægten punktum som separator. Returnerer vægten negativ (med ”-” foran), betragtes

Unit kan indstilles til de bekendte præfiks – mg, g, kg. Der kan også indstilles en brugerdefineret unit med M21, men dette er ikke implementeret i simulatoren.

Det skal understreges, at der for MT-vægte, findes langt flere kommandoer. Kommandoerne er dog ikke blevet implementeret, da de ikke er fundet nødvendige som eksempel. Kommandoerne kan selvfølgelig læses i MT-SICS referencemanualen for den brugte vægt.

3 Lags modellen



3 lags modellen består er følgende områder, grænseflade, funktionalitet og data. Fordelen ved benyttelse af 3 lags modellen er at skabe et overblik over de forskellige funktioner i det pågældende program, ved bl.a. at skabe lav kobling, let vedligeholdelse og let udskiftning samt ændringer i de forskellige lag.
Ved at inddele sit program på denne måde så gør man arbejdet mere overskueligt derved nemmere for de fremtidige programmører der skal arbejde på projektet. Der gives også mulighed for eks. At skifte sin grænsefalde ud men beholde sin funktionalitet samt data i programmet, hvis det nu var ønsket et andet interface.

Grænseflade – Er betegnet som det yderste lag af modellen, da det er den del af programmet som forbrugerne kommer til at opleve. Det er i bund og grund det vindue der kan ses når programmet køres, samt alt dens visuelle indhold. Grænsefladelaget har ingen funktion for nogen beregninger eller lagring af data, den for forespørgsler fra brugeren, og henter de nødvendige oplysninger fra funktionalitetslaget og fremviser den efterspurgte viden til brugeren.

Data – Er det inderste lag af modellen, som indeholder alt viden, deraf navnet datalaget. Data laget kender kan ikke komme med forespørgsler til de andre lag i programmet, men kan modtage nogle der beder om noget data eller ændring samt tilføjelser til data.

Funktionalitet – Er mellemlaget i modellen. Den modtager forespørgsler af grænsefladelaget og kan selv stille forespørgsler til datalaget. Funktionalitets laget kan anses som 'samarbejdet' imellem grænseflade- og datalaget.

Eks. Et program over et firmas medarbejder: Forbrugeren vil vide hvor mange penge der bliver lønnet månedligt til all medarbejdere. Der trykkes på en knappen ”Samlet måneds løn ” i programmet, grænsefladen sender en forespørgsel til funktionalitets laget på ”Hvad er den samlet måneds løn for all med arbejdere? ”, funktionalitets laget beder data laget om hver enkelt medarbejders løn og beregner så selv den samlede månedsløn som bliver sendt tilbage til grænsefladelaget.

Kode & test

Koden led af, at der blev brugt lang tid på at forsøge, at implementere en GUI, som aldrig kom til at virke. Selv da der blev gået på kompromis med GUI, og der i stedet blev forsøgt at udvikle et helt almindeligt konsolvindue, viste der sig at være store problemer med, at få vinduet til at tale sammen med selve server- og clientsocket.

Der blev som nød brugt store dele af det kodeeksempel, der var blevet lagt op i opgavebeskrivelsen i stedet. I forhold til de nødvendige kommandoer, er RM 20 ikke blevet implementeret.

Der er blevet implementeret mulighed for, at indstille serversocket via kommandolinjen (skal angives med en bindestreg og port, efterfulgt af selve porten – f.eks. ”-port 12345”.

Start

Der blev oprindeligt brugt to separate tråde, til hhv. serversocket og GUI (dette mistænkes for at være problemet med de tidligere nævnte komplikationer.) Derfor blev der holdt fast i, at serversocket skulle køre på en tråd, som bliver oprettet via SimMain.

```
11 private static ISimulatorDAO simDao;
12 private static ISimulatorDLO simDlo;
13 private static Thread simSrv;
```

```
44 try {
45 simDlo = new SimulatorDLO();
46 simDao = new SimulatorDAO(simDlo);
47 simSrv = new SimServer(simDao, port);
48 }
49 catch (IOException e) {
50 e.printStackTrace();
51 }
52
53 simSrv.start();
```

Linje 53 kan selvfølgelig kaldes, da SimServer extender Thread, og der senere angives en public run-metode.

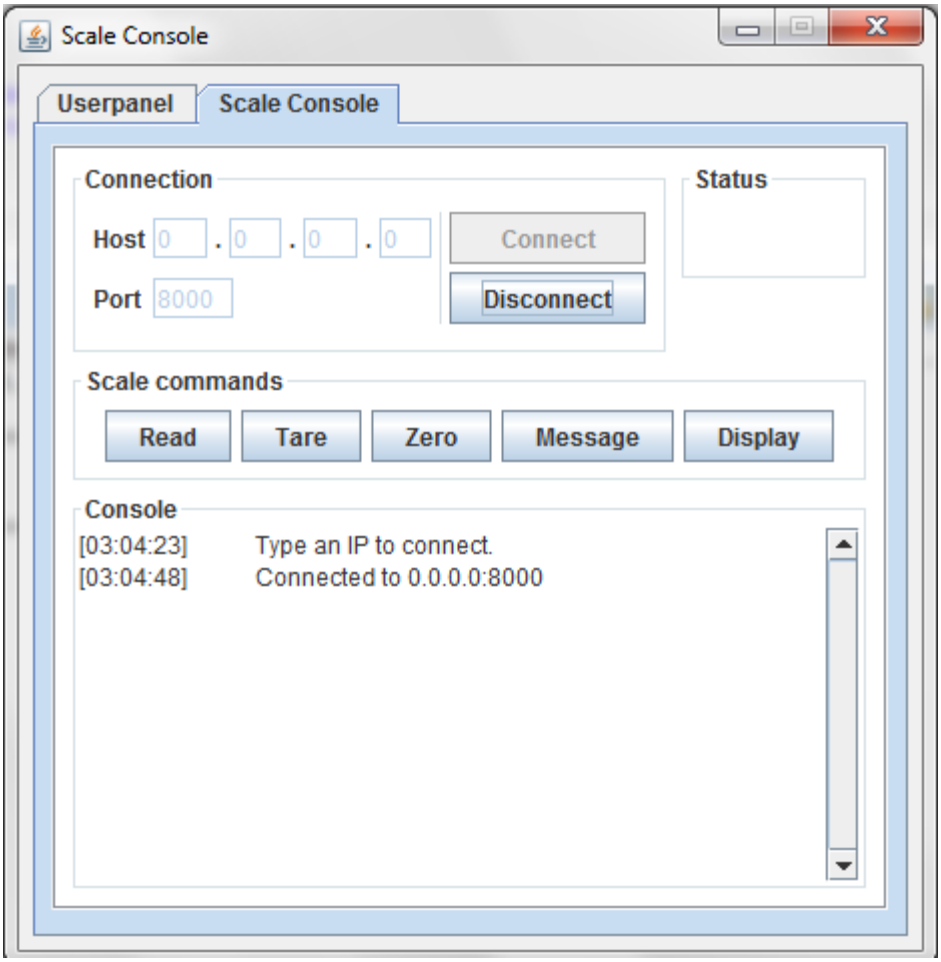
```
6 public class SimServer extends Thread {
```

```
27 public void run() {
28 updateScreen();
29
30 try {
31 while ((input……
```

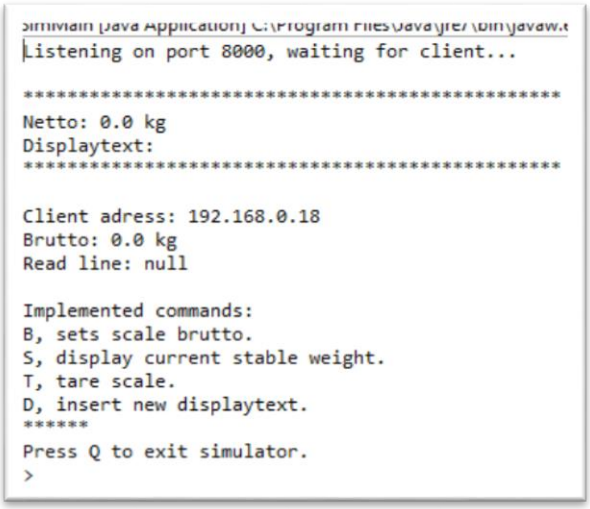
Når simSrv instantieres på linje 47, venter serveren på en klient skal forbinde.

```
SimMain [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (24/03/2014 02.53.35)
Listening on port 8000, waiting for client...
```

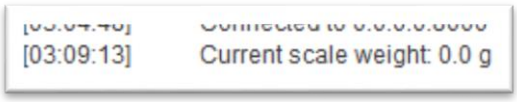
Fra Datakommunikation-kurset, er der blevet udviklet en mindre konsol, som kan forbinde til en server. Der er blevet forsøgt at få de to programmer til at tale med hinanden, og det er også gået nogenlunde, om end manglerne på serversiden, får de fleste funktioner i den lille konsol, til ikke rigtigt at virke. Serveren modtager dog kommandoerne, og læser dem også korrekt.



Der oprettes forbindelse til serveren, og serveren reagerer, ved at printe data om nuværende status ud i Javas konsolvindue.



Når der er blevet oprettet forbindelse mellem klient og server, kan der fra klienten blive forsøgt at blive kaldt nogle funktioner – f.eks. ”Read”.

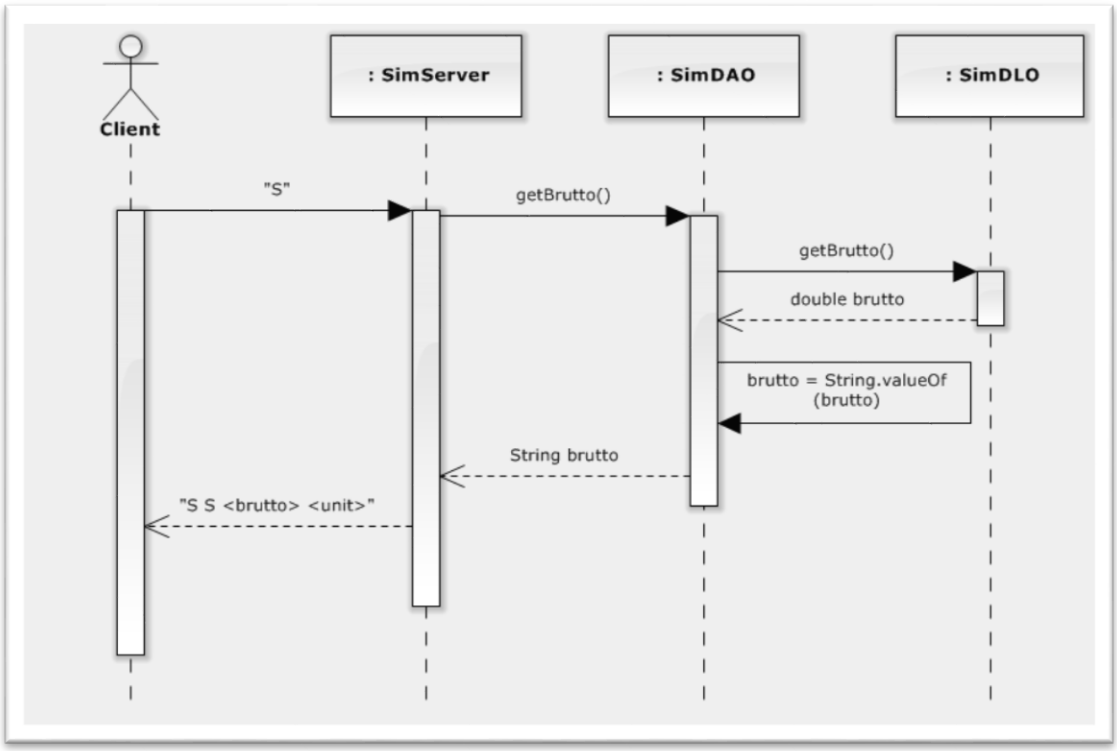


Mens klienten modtager den korrekt vægt, læser serveren også kommandoen og reagerer, ved at udskrive nyt statusvindue, hvor der kan ses, at den har modtaget kommandoen ”S”.

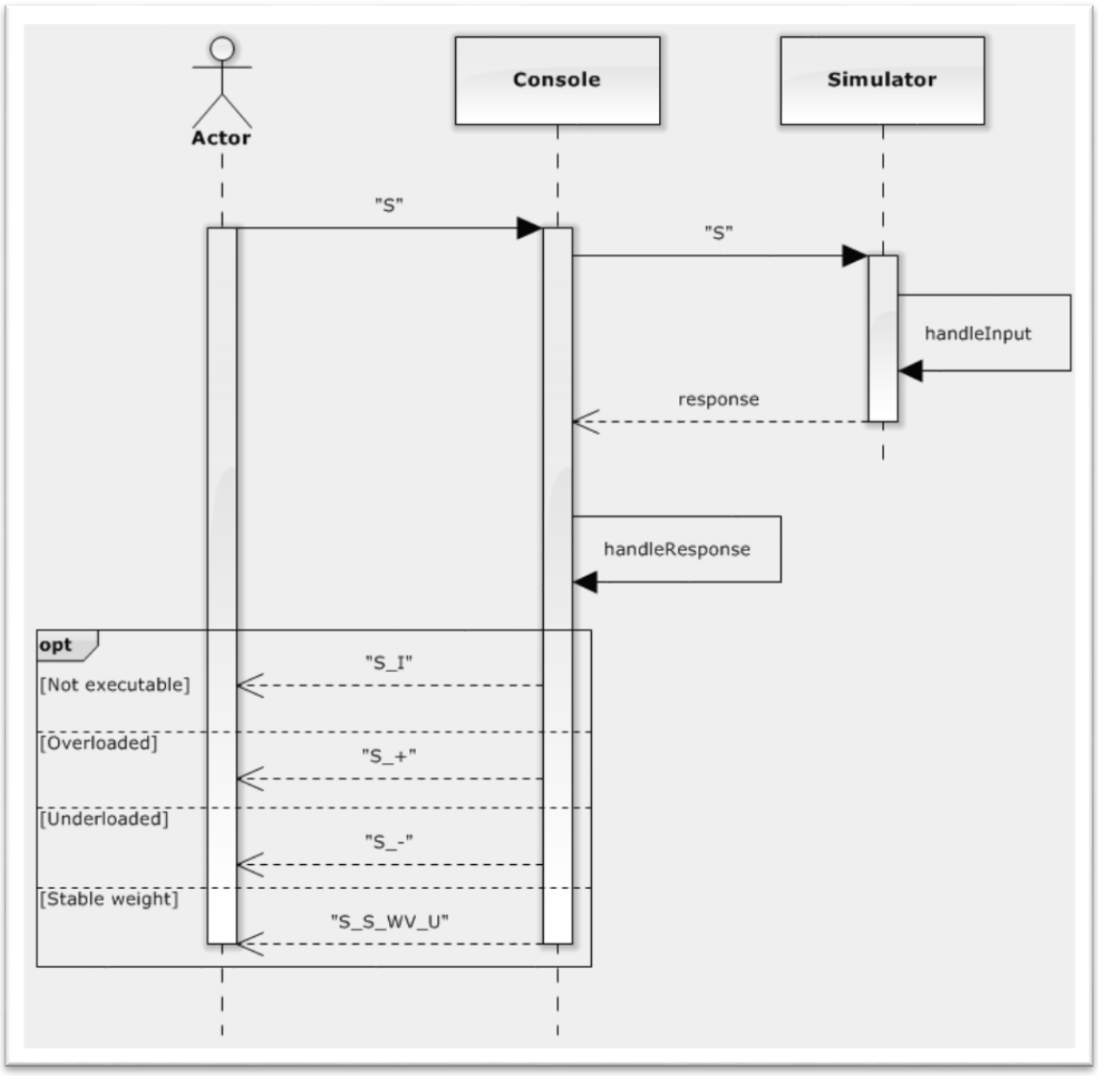

```
**********
Netto: 0.0 kg
Displaytext:
**********

Client adress: 192.168.0.18
Brutto: 0.0 kg
Read line: S

Implemented commands:
B, sets scale brutto.
```

Designsekvensdiagram

De fleste af kommandoerne, har den samme gennemgang gennem systemet, i forhold til kald og des lige. Som et led i forståelsen af programmet, er et eksempel givet med dette designsekvensdiagram.



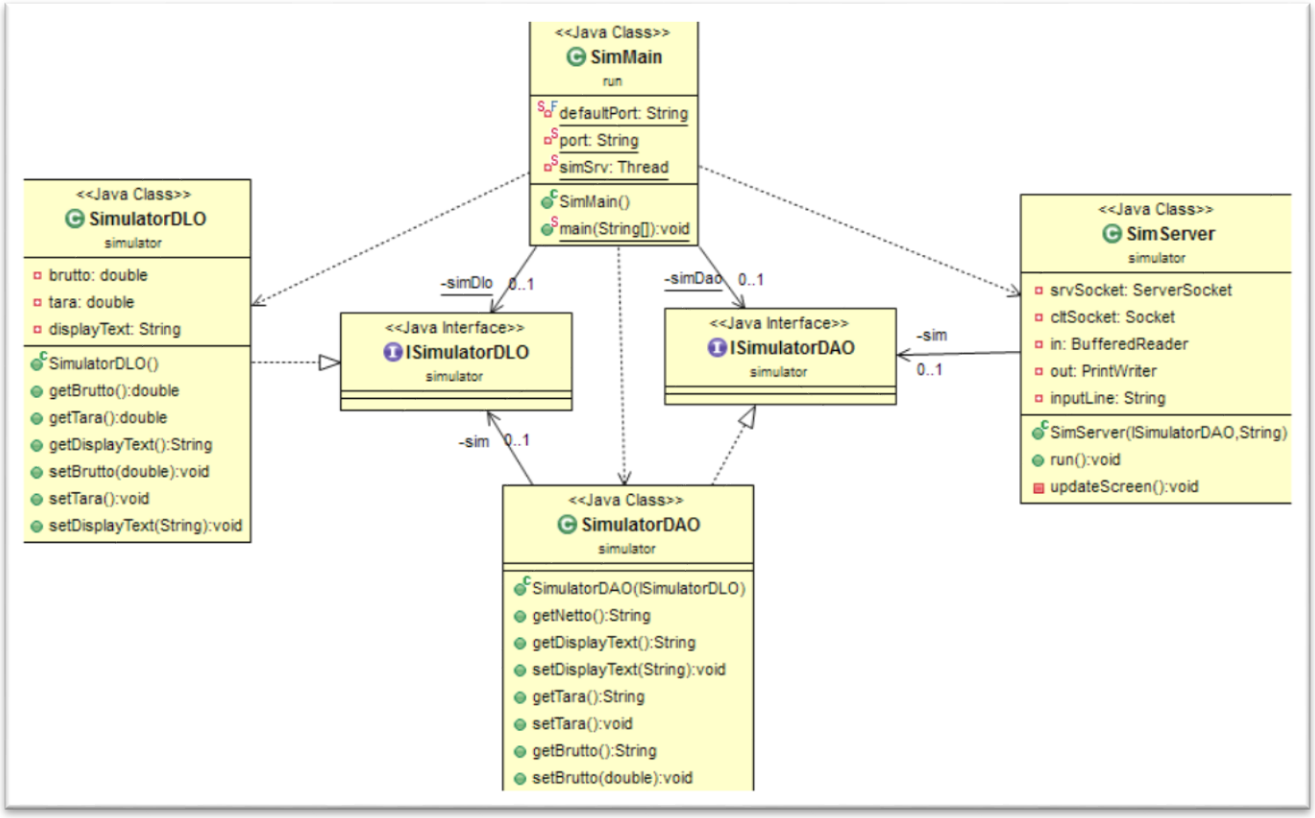
Designsekvensdiagrammet ligner i og for sig det, som der blev sat som udgangspunkt i den indledende analyse (systemsekvensdiagram), hvilket taler for, at der er blevet holdt en nogenlunde overgang fra indledende arbejde, til selve implementeringen, om end der viste sig, at være komplikationer vedr. GUI og serversocket.



Figur 2 – jf. tidligere, skal underscore læses som mellemrum. WV = WeightValue (brutto), U = Unit (kg).

Designklassediagram

Ligesom der var en synlig overgang fra den analytiske fase til designfasen. Kun små ændringer (der blev fjernet nogle klasser fra det oprindelige klassediagram, da der blev nød til at blive udviklet forfra), blev der lavet i løbet af implementeringen.



Main-filen (SimMain) instantiere de tre objekter, som hver især repræsenterer de tre lag, i den bekendte 3-lags-model,

- SimServer som grænsefladelag,
- SimulatorDAO som funktionslag,
- SimulatorDLO som datalag.

De to interfaces, bruges selvfølgelig som kontrakter mellem lagene,

- ISimulatorDAO er kontrakten mellem SimServer og SimulatorDAO,
- ISimulatorDLO er kontrakten mellem SimulatorDAO og SimulatorDLO.

Konklusion

Efter en slutspurt, hvor der blev indset, at det ikke ville kunne lade sig gøre, at lave programmet på en sådan måde, som det var ønsket fra start, blev der med panik lavet en nødløsning, som nogenlunde kom til at fungere, og stadig overholdte noget af det indledende og forberedende arbejde.

Der er dog nogle alvorlige fejl og mangler i det endelige program, som ikke gør det særligt acceptabelt, og er noget, der nødvendigvis må indhentes mod slutningen (herunder særligt i 3-ugers forløbet.)

Af fejl og mangler, er der selvfølgelig den manglende mulighed for, faktisk at indstille vægtens vægt.

Der er, jf. afsnittet om systemsekvesdiagrammer, kun blevet implementeret *meget* få af de forventede kommandoer, ligesom dem der er implementeret, sagtens kunne være blevet udviklet bedre (hvis forbehold for tidspres fjernes.)

Bilag

Kildekode

SimMain.java

```
1 package run;
2
3 import java.io.IOException;
6
7 public class SimMain {
8 private static final String defaultPort = "8000"; // SETS DEFAULT LISTENING PORT FOR
SERVER
9 private static String port;
10
11 private static ISimulatorDAO simDao;
12 private static ISimulatorDLO simDlo;
13 private static Thread simSrv;
14
15 public SimMain() {
16 if(port.isEmpty())
17 port = defaultPort;
18
19 try {
20 simDlo = new SimulatorDLO();
21 simDao = new SimulatorDAO(simDlo);
22 simSrv = new SimServer(simDao, port);
23 } catch (IOException e) {
24 e.printStackTrace();
25 }
26
27 simSrv.start();
28 }
29
30 public static void main(String[] args) {
31 if(args.length > 0) {
32 for(int i = 0; i < args.length; i++) {
33 if(args[i].equals("-port"))
34 port = args[i + 1];
35 }
36 }
37 else {
38 port = "";
39 }
40
41 if(port.isEmpty())
42 port = defaultPort;
43
44 try {
45 simDlo = new SimulatorDLO();
46 simDao = new SimulatorDAO(simDlo);
47 simSrv = new SimServer(simDao, port);
48 }
```

```
49 catch (IOException e) {
50 e.printStackTrace();
51 }
52
53 simSrv.start();
54 }
55 }
```

SimServer.java

```
1 package simulator;
2
3 import java.io.*;
5
6 public class SimServer extends Thread {
7 private ServerSocket srvSocket;
8 private Socket cltSocket;
9 private BufferedReader in;
10 private PrintWriter out;
11
12 private ISimulatorDAO sim;
13 private String inputLine;
14
15 public SimServer(ISimulatorDAO sim, String port) throws IOException {
16 this.sim = sim;
17 this.srvSocket = new ServerSocket(Integer.parseInt(port));
18
19 System.out.println("Listening on port " + port + ", waiting for client...");
20
21 cltSocket = srvSocket.accept();
22
23 out = new PrintWriter(cltSocket.getOutputStream(), true);
24 in = new BufferedReader(new InputStreamReader(cltSocket.getInputStream()));
25 }
26
27 public void run() {
28 updateScreen();
29
30 try {
31 while ((inputLine = in.readLine()) != null) {
32 if (inputLine.startsWith("DN")) {
33 // TODO
34 }
35 else if (inputLine.startsWith("D")) {
36 if (inputLine.equals("D"))
37 sim.setDisplayText("");
38 else
39 sim.setDisplayText(inputLine.substring(2, inputLine.length()));
40
41 updateScreen();
42
43 out.println("DB"+"\r\n");
44 }
```

```
45
46 else if (inputLine.startsWith("T")) {
47 out.println("T " + sim.getTara() + " kg "+"\r\n");
48
49 sim.setTara();
50
51 updateScreen();
52 }
53
54 else if (inputLine.startsWith("S")) {
55 updateScreen();
56
57 out.println("S " + sim.getNetto() + " kg " +"\r\n");
58 }
59
60 else if (inputLine.startsWith("B")) {
61 String temp = inputLine.substring(2, inputLine.length());
62
63 sim.setBrutto(Double.parseDouble(temp));
64
65 updateScreen();
66
67 out.println("DB"+"\r\n");
68 }
69
70 else if ((inputLine.startsWith("Q"))) {
71 System.out.println("");
72 System.out.println("Quitting...");
73
74 System.in.close();
75 System.out.close();
76
77 in.close();
78 out.close();
79
80 System.exit(0);
81 }
82 }
83 }
84 catch (Exception e) {
85 System.out.println("Exception: " + e.getMessage());
86 }
87 }
88
89 private void updateScreen() {
90 System.out.println(" ");
91 System.out.println("************************************************");
92 System.out.println("Netto: " + sim.getNetto() + " kg");
93 System.out.println("Displaytext: " + sim.getDisplayText());
94 System.out.println("************************************************");
95 System.out.println(" ");
96 System.out.println("Client adress: " +
```

```
cltSocket.getInetAddress().getHostAddress());
97 System.out.println("Brutto: " + sim.getBrutto() + " kg");
98 System.out.println("Read line: " + inputLine);
99 System.out.println(" ");
100 System.out.println("Implemented commands:");
101 System.out.println("B, sets scale brutto.");
102 System.out.println("S, display current stable weight.");
103 System.out.println("T, tare scale.");
104 System.out.println("D, insert new displaytext.");
105 System.out.println("******");
106 System.out.println("Press Q to exit simulator.");
107 System.out.println("> ");
108 }
109 }
```

ISimulatorDAO.java

```
1 package simulator;
2
3 public interface ISimulatorDAO {
4 String getNetto();
5 String getDisplayText();
6 void setDisplayText(String string);
7 String getTara();
8 void setTara();
9 String getBrutto();
10 void setBrutto(double parseDouble);
11 }
```

SimulatorDAO.java

```
1 package simulator;
2
3 public class SimulatorDAO implements ISimulatorDAO {
4 private ISimulatorDLO sim;
5
6 public SimulatorDAO(ISimulatorDLO sim) {
7 this.sim = sim;
8 }
9
10 public String getNetto() {
11 return String.valueOf((sim.getBrutto() - sim.getTara()));
12 }
13
14 public String getDisplayText() {
15 return sim.getDisplayText();
16 }
17
18 @Override
19 public void setDisplayText(String displayText) {
20 sim.setDisplayText(displayText);
21 }
22
```

```
23 @Override
24 public String getTara() {
25 return String.valueOf(sim.getTara());
26 }
27
28 @Override
29 public void setTara() {
30 sim.setTara();
31 }
32
33 @Override
34 public String getBrutto() {
35 return String.valueOf(sim.getBrutto());
36 }
37
38 @Override
39 public void setBrutto(double brutto) {
40 sim.setBrutto(brutto);
41 }
42 }
```

ISimulatorDLO.java

```
1 package simulator;
2
3 public interface ISimulatorDLO {
4 double getBrutto();
5 double getTara();
6 void setBrutto(double brutto);
7 void setTara();
8 String getDisplayText();
9 void setDisplayText(String displayText);
10 }
```

SimulatorDLO.java

```
1 package simulator;
2
3 public class SimulatorDLO implements ISimulatorDLO {
4 private double brutto;
5 private double tara;
6 private String displayText;
7
8 public SimulatorDLO() {
9 this.brutto = 0;
10 this.tara = 0;
11 this.displayText = "";
12 }
13
14 @Override
15 public double getBrutto() {
16 return brutto;
17 }
```

```
18
19 @Override
20 public double getTara() {
21 return tara;
22 }
23
24 @Override
25 public String getDisplayText() {
26 return displayText;
27 }
28
29 @Override
30 public void setBrutto(double brutto) {
31 this.brutto = brutto;
32 }
33
34 @Override
35 public void setTara() {
36 this.tara = this.brutto;
37 }
38
39 @Override
40 public void setDisplayText(String displayText) {
41 this.displayText = displayText;
42 }
43 }
```