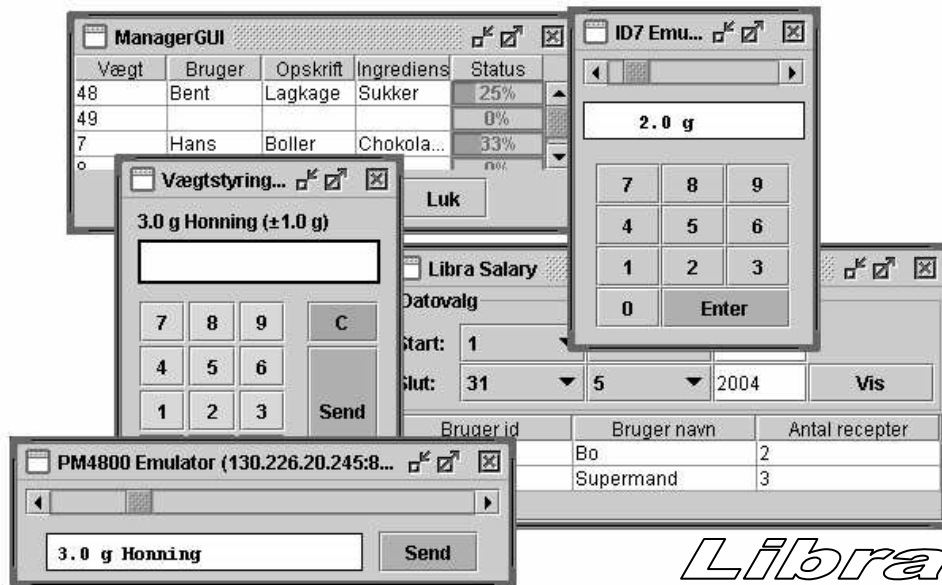


# Vægtstyrings System



Udarbejdet af: Studie-nr, Navn



s022307, Martin Atke Bentsen



s023144, Frank Ditlevsen



s022184, Michael Eiland



s020560, Amjad Majid

Afleveringsfrist: 25. Juni 2004

Denne rapport er modtaget af: \_\_\_\_\_, den \_\_\_/\_\_\_, kl: \_\_\_:\_\_\_.

Denne opgave indeholder 46 Sider.

# Indholdfortegnelse

1	Indledning .....	4
2	Projektplan .....	5
2.1	Tidsplan .....	5
2.2	Test-plan .....	5
2.3	Arbejdsfordeling .....	5
3	Foranalyse .....	6
3.1	Krav specifikation: .....	6
3.2	Use Cases: .....	7
3.2.1	Løn .....	8
3.2.2	Afvejning af recept .....	8
3.2.3	Aktuel status .....	9
3.3	Kravspecifikationsliste .....	10
4	Analyse .....	11
4.1	Use case i forhold til systemet .....	11
4.1.1	Recept afvejning .....	11
4.1.2	Håndter vægtstatus ændring .....	12
4.1.3	Se realtime status .....	13
4.1.4	Se log .....	13
4.1.5	Se lønningsdata .....	14
4.1.6	Håndter DB .....	14
4.2	Domain model .....	15
5	Design .....	16
5.1	Overblik over systemet .....	16
5.1.1	Flow diagram .....	17
5.2	Sekvensdiagrammer .....	18
5.2.1	Vægt-proceduren .....	18
5.2.2	Mediater kommunikation .....	20
5.3	Aktivitetsdiagram for afvejningsproceduren .....	21
5.4	Klassediagram .....	22
5.5	Database Design .....	23
6	Implementation .....	24
6.1	Forklaring af kode .....	24
6.1.1	WeightHandler .....	24
6.1.2	WeightDisplayAndKeypad .....	24
6.1.3	PM4800Emul og ID7Emul .....	25
6.1.4	WeightStatusChange .....	25
6.1.5	ManagerApp .....	25
6.1.6	Logger .....	25
6.1.7	Message FrameWork .....	26
6.1.8	Database implementeringen .....	27
6.1.9	dataObjects .....	28

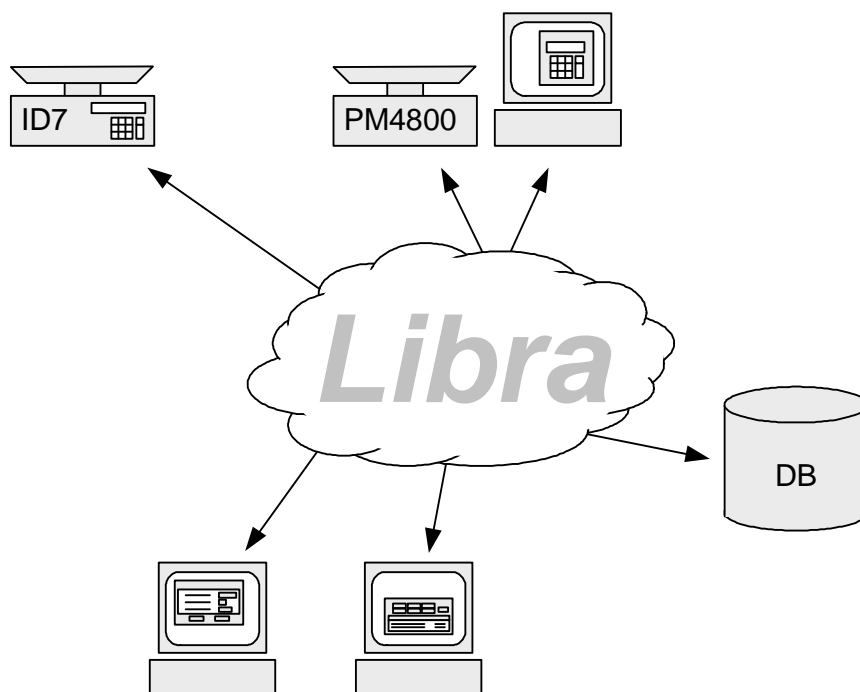
6.1.10	SalaryApp .....	28
6.2	Databasetabeller .....	29
6.2.1	weightjob .....	29
6.2.2	recepty.....	29
6.2.3	ingredience.....	29
6.2.4	user .....	29
6.3	Hvordan man tilføjer en ny vægttype.....	30
6.3.1	Følgende skal implementeres.....	30
6.4	Hvordan man tilføjer printere til systemet.....	32
7	Test.....	33
7.1	Unit-test .....	33
7.1.1	DummyObjectFactory .....	33
7.1.2	TestDataStorage .....	33
7.1.3	TestRemoteDB .....	33
7.1.4	TestMessageListener .....	34
7.1.5	DummyWeightStatusChangedGenerator og TestStatusListener .....	34
7.2	Integrations-test.....	34
7.3	Accept Test .....	36
7.4	JavaDoc .....	37
7.5	Brugervejledning.....	37
8	Konklusion.....	38
	Bilag.....	39
	Bilag A - Opgavebeskrivelse.....	39
	Bilag B - Database Oprettelse .....	42
	Bilag C – Udvalgt Kode.....	43
	interface WeightProxy .....	43
	handleWeightProcedure() .....	44

# 1 Indledning

En virksomhed skal bruge et system, der støtter deres afvejningsprocedure, på elektroniske vægte. Systemet skal fortælle operatøren hvad der skal afvejes og sikre at der bliver afvejet indenfor angivne tolerancer. Alle afvejninger skal logges, og der skal kunne vises relevante udtræk til firmaets værkfører og lønningspersonalet.

Firmaet har to typer vægte (kaldet PM4800 og ID7), som begge kan kommunikeres med via et almindeligt PC-netværk (LAN) over sockets.

Systemet er implementeret i Java, og de enkelte dele kommunikerer sammen over netværk. Der er implementeret et generelt framework til kommunikation mellem enhederne. Data opsamles i en SQL-database.

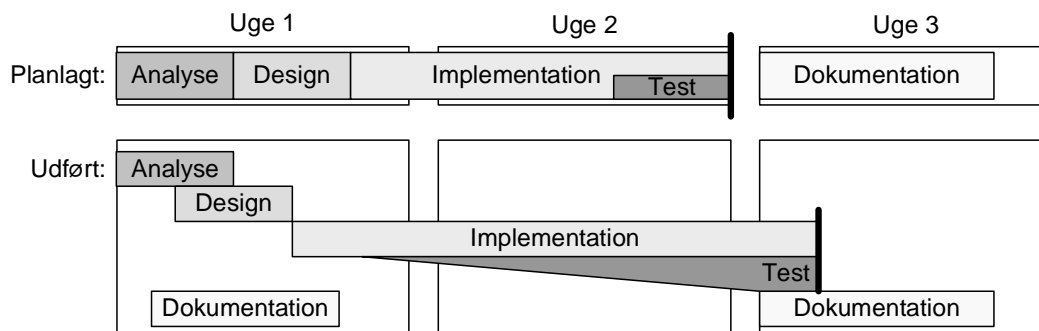


*Vi har valgt at kalde systemet Libra, efter det engelske navn for stjernevægten.*

## 2 Projektplan

### 2.1 Tidsplan

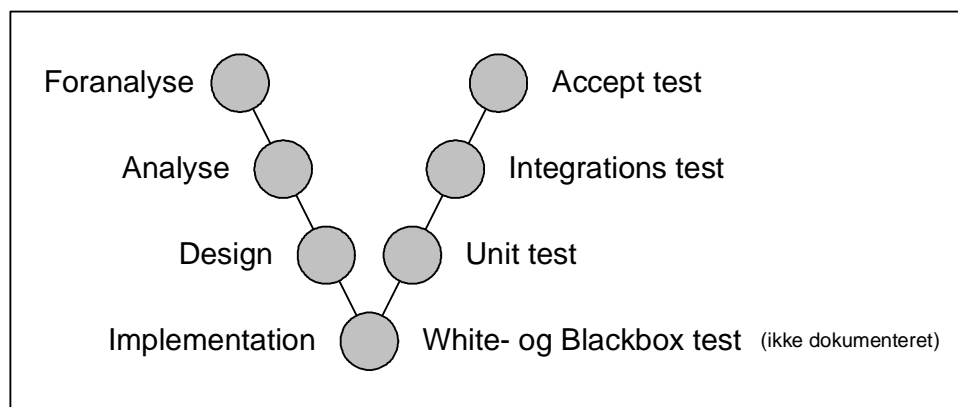
Projektet skulle udføres på 3 uger, så det var nødvendigt med en arbejdsplan baseret på en simpel vandfaldsmodel. Udførelsen er kun flyttet en smule fra planen, men ikke kritisk meget.



Der er defineret en enkelt deadline til kode-stop. Den blev vi desværre nødt til at flytte en weekend frem i tid. For at undgå inkonsistens mellem implementation og dokumentation, er den meste dokumentation placeret sidst i perioden. Det korte projektførløb gør at man ikke mister overblikket.

### 2.2 Test-plan

Der er planlagt at følge V-modellen for test. Det vil sige, testene er udført ud fra hvad der er beskrevet i de enkelte faser. Rapporten indeholder kun testbeskrivelse, ved selve udførelsen af testen.



### 2.3 Arbejdsfordeling

Der er fire medlemmer i gruppen. Under analyse og design, arbejdes der samlet ved en tavle med at tegne modeller, som efter enighed tegnes ind på computer. Under implementationen deltes systemet op i fire dele, som hver kunne arbejde med selvstændigt med hyppige møder. Dokumentationen deles ud i delkomponenter som udfærdiges af enkeltpersoner, og gennemlæses af et andet gruppemedlem.

## 3 Foranalyse

### 3.1 *Krav specifikation:*

Der skal udvikles et system til en vægtproducent som skal levere et afvejningssystem til en gammel kunde. Systemet skal kunne foretage afvejning ud fra givne recepter.

Alle data fra vejeprocessen skal opsamles elektronisk. Dette indbefatter blandt andet operatøridentitet, mængde afvejet, osv.

Kunden har allerede 20 vægte af typen PM4800 som med en RS232/Ethernet konverter kan tilsluttes til Ethernet, og sende vejeresultater med telnet protokollen. Disse har dog intet tastatur så kunden har foreslået at han blot sætter en gammel bærbar PC op ved siden af hver vægt så han kan bruge display og tastatur på denne.

Kunden har brug for flere vægte, så han har yderligere bestilt 10 stk. ID7, som er født med display og tastatur. Disse skal bruges parallelt med de eksisterende vægte og have samme indvejnings-procedure. (se bilag: Opgavebeskrivelse)

Det program som implementerer indvejningsproceduren skal udvikles på en maskine som befinder sig i kundens server rum. Afvejningsdata, operatør data osv. hentes under afvejningen fra en server i firmaets hovedsæde i Sverige.

Der skal være muligt for en værkfører at logge ind fra en vilkårlig PC i virksomheden og se den aktuelle afvejnings-log.

Lønningskontoret skal kunne trække antal afvejn timer / operatører ud for en vilkårlig tidsperiode, idet disse indgår i operatørnes akkordløn.

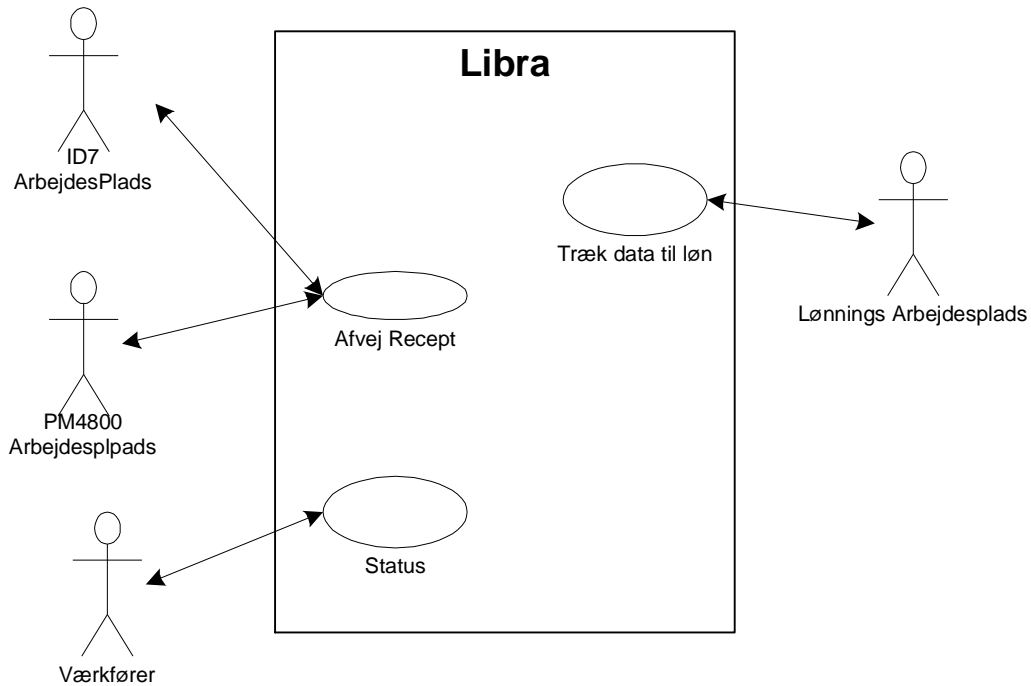
For nærmere detaljer se opgavebeskrivelsen under bilag "Opgavebeskrivelse"

Der er forud specificeret følgende del-opgaver der skal designes og implementeres.

- Program som kan simulere ID7 terminalen. (arbejdsplads type 1)
- Program som kan simulere PM4800 / bærbar PC. (arbejdsplads type 2)
- Program som styrer afvejningsprocedure på de to afvejningssystemer.
- Program til overvågning som betjenes fra værkførers PC.
- Program som simulere databasen i Sverige.
- Program til lønningskontoret.

### 3.2 Use Cases:

Der er blevet udarbejdet en overordnet use case model for projektet. Denne Use Case model skal afdække de relevante fysiske og funktionelle behov på en overskuelig måde. Disse behov ses ud fra aktørens synsfelt. Dette gøres for at finde de områder der skal dækkes af systemet. Det er aktøren der iværksætter eller påvirker aktiviteter i Use Case diagrammet.



*Hvilke arbejdsopgaver udføres på systemet*

Beskrivelse af aktørerne:

Vægtoperatør (operatør på ID7 eller PM4800)	En medarbejder der foretager en afvejning på en vægt.
Værkfører	En person der skal følge med i status for produktionen.
Lønningspersonale	Person der har brug for oplysninger til udregning af løn.

### 3.2.1 Løn

Use case:	Løn-personalet kan se de nødvendige løn data
Aktør:	Lønningspersonale.
Mål:	Løn personalet kan se de nødvendige data til udregning af løn data.
Initiering:	Løn personalet sidder ved sin på pc og har adgang til systemet
Forløb:	Valg af periode dataene skal ses for. Visning af data til løn
Undtagelse	
Succes:	Løn personalet har fået de nødvendige løn data antal af afvejninger / Operatør.
Fejl:	Problemet med kommunikation med database.

### 3.2.2 Afvejning af recept.

Use case:	Operatøren har hentet en eller nogle recepter og skal fortage afvejninger af disse.
Aktør:	Operatør.
Mål:	Operatøren har fået foretaget sine afvejninger.
Intiering:	Operatøren skal have en eller flere recepter der skal afvejes.
Forløb:	<ol style="list-style-type: none"><li>1. Logger ind</li><li>2. Indtaster receptnr</li><li>3. Foretager af vejningen</li><li>4. Tarer</li><li>5. Evt fra punkt 3 igen</li><li>6. Evt. print af lable</li><li>7. Valg om afvejning af ny recept.</li></ol>
Undtagelser:	Der mangler en råvarer på lageret.
Succes:	En til flere recepter er blevet afvejet.
Fejl:	Bruger eksister ikke. Recept eksister ikke

### 3.2.3 Aktuel status.

Use case:	Værkføreren har brug for at få et overblik over den foregående produktion samt mulighed får at opnå en uddybende information om produktionen.
Aktør:	Værkføreren.
Mål:	Værkføreren kan følge med i den aktuelle status, for dagen.
Intiering:	Værkføreren er logget ind ved en pc og har adgang til systemet.
Forløb:	<ol style="list-style-type: none"><li>1. Værkføreren ser den aktuelle status</li><li>2. Værkføreren kan få uddybende oplysninger</li></ol>
Undtagelser:	
Succes:	Værkføreren har fået et overblik over den aktuelle produktion.
Fejl:	Problemer med kommunikation for overvågning.

### **3.3 Kravspecifikationsliste**

Her vises en liste over de målbare krave, der testes på i accept-testen.

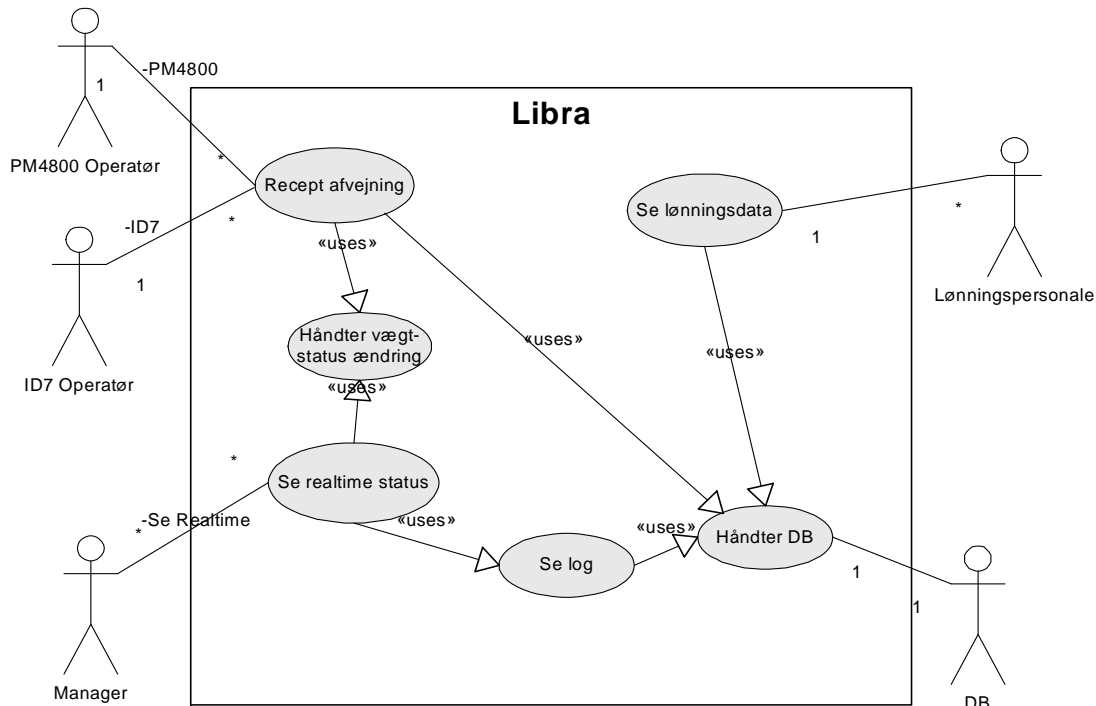
I de nedenstående punkter er det beskrevet hvem, der skal have adgang til de enkelte ting. Det forventes at virksomheden selv sikrer, at adgang til de enkelte programmer kun sker af relevant personale.

- For at beregne løn for operatørerne i valgt periode, skal lønningspersonalet kunne se id, navn og antal recepter for de enkelte operatører i den angivne periode.
- Før operatøren kan indtaste recept job nr., skal operatøren først logge ind på vægten.
- Operatør skal kunne indtaste et recept job nr. på vægten.
- Operatør skal kunne se, hvilken type ingrediens og hvor stor mængde der skal afvejes. Dette skal vises på displayet på vægten.
- Operatør skal kunne afveje en ingrediens fra en recept på en vægt.
- Vægten skal automatisk tareres efter hver afvejning.
- System skal eventuelt udskrive en label, når en afvejning er fuldført.
- Værkfører skal kunne se aktuelle aktiviteter på vægtene.
- Værkfører skal kunne se log over dagens arbejde.
- Der skal logges hvor lang tid en operatør har brugt på en afvejning, og værkfører skal kunne se det.
- Der skal være JavaDoc af Java koden.
- Der skal være en vejledning i at tilføje en eller flere nye vægt-typer.

## 4 Analyse

### 4.1 Use case i forhold til systemet.

Use case modellen udvides, så den bedre beskriver den interne del af systemet.



#### 4.1.1 Recept afvejning

Precondition	Operatøren skal være oprettet. Et recept-job skal være oprettet.
Success end Condition	Et recept-job er færdiggjort. Jobbets tid for færdiggørelse er logget.
Main succes scenario	<ol style="list-style-type: none"> <li>1. Vægten tændes</li> <li>2. Operatør id indtastes</li> <li>3. Recept job nr. indtastes</li> <li>4. Systemet tarer vægten</li> <li>5. Systemet viser hvad der skal afvejes</li> <li>6. Operatøren afvejer det angivne</li> <li>7. Der forsættes fra punkt 4 til alle ingredienser er afvejet.</li> <li>8. Valg om man vil forsætte, hvis Ja punkt 3</li> <li>9. Vægten slukkes</li> </ol>

Alternate scenario	<ol style="list-style-type: none"> <li>1. Vægten tændes</li> <li>2. Operatør indtaster forkert id</li> <li>3. Operatøren bliver spurgt om et rigtigt id</li> <li>4. indtil rigtigt id indtastes punkt 3</li> <li>5. Herefter forsættes fra punkt 3 i Main scenario</li> </ol> <ol style="list-style-type: none"> <li>1. Vægten tændes</li> <li>2. Operatør id indtastes</li> <li>3. Der indtastes et ugyldigt recept-job id.</li> <li>4. En fejlmeddelelse vises i vægten display.</li> <li>5. Operatøren bliver spurgt om et gyldigt recept-job id</li> <li>6. Herefter forsættes fra punkt 4 i Main scenario</li> </ol>
Possible errors	<p>Hardware:</p> <ul style="list-style-type: none"> <li>• Vægten kan veje forkert, man tarer den for hver afvejning.</li> <li>• Forbindelsen til vægtene kan blive afbrudt. (Dette kommer an på miljøet, som vægtene er i).</li> <li>• Forbindelsen til databasen kan være afbrudt.</li> </ul> <p>Råvare:</p> <ul style="list-style-type: none"> <li>• Mangle på en eller flere ingredienser til en recept.</li> </ul>

#### 4.1.2 Håndter vægtstatus ændring

Preecondition	WeightHandler skal være startet
Success end condition	Der er sendt et event fra WeightHandler til ManagerApp.
Main succes scenario	<ol style="list-style-type: none"> <li>1. ManagerApp registrerer sig som listener hos WeightHandler ved at sende en message.</li> <li>2. WeightHandler lægger ManagerApp'ens navn i en liste af listeners.</li> <li>3. WeightHandler sender den aktuelle status for samtlige af de kendte vægte til den nye listener, som enkelte event.</li> <li>4. Hver gang der sker noget nyt ude hos vægtene sender WeightHandler en ny event som message til samtlige der har registreret sig som listeners.</li> <li>5. Når ManagerApp lukkes, sender den en ny message for at fjerne sig fra WeightHandlerens liste af listeners.</li> </ol>
Alternate scenario	Hvis der ikke er registreret nogen til at lytte på events, sendes ingen events-beskeder.
Possible errors	

### 4.1.3 Se realtime status

Preecondition	WeightHandler'en skal køre, og der skal eventuelt være aktivitet ved vægtene.
Success end condition	Systemet har holdt værkføreren opdatere på de aktuelle afvejninger.
Main succes scenario	<ol style="list-style-type: none"><li>1. Managerprogrammet startes op</li><li>2. Managerprogrammet bliver registreret i WeightHandler.</li><li>3. WeightHandler sender aktuel status for alle kendte vægte .</li><li>4. Managerprogrammet viser vægtenes status.</li><li>5. For hver afvejning opdateres de felter som tilhør den pågældende vægt</li><li>6. Programmet lukkes</li></ol>
Alternate scenario	Aktivering af Se log
Possible errors	Der er ikke forbindelse til databasen Message-systemet er ikke startet WeightHandler er ikke startet

### 4.1.4 Se log

Preecondition	ManagerProgrammet er startet.
Success end condition	Værkfører har set loggen
Main succes scenario	<ol style="list-style-type: none"><li>1. Popup'en åbnes med dobbeltklik på et felt i ManagerProgrammet eller ved at brugen har valgt et felt ManagerProgrammet og trykket på knappen "vis"</li><li>2. data hentes fra databasen</li><li>3. data vises i popup'en</li><li>4. popup'en lukkes</li></ol>
Alternate scenario	
Possible errors	Der er ikke forbindelse til databasen

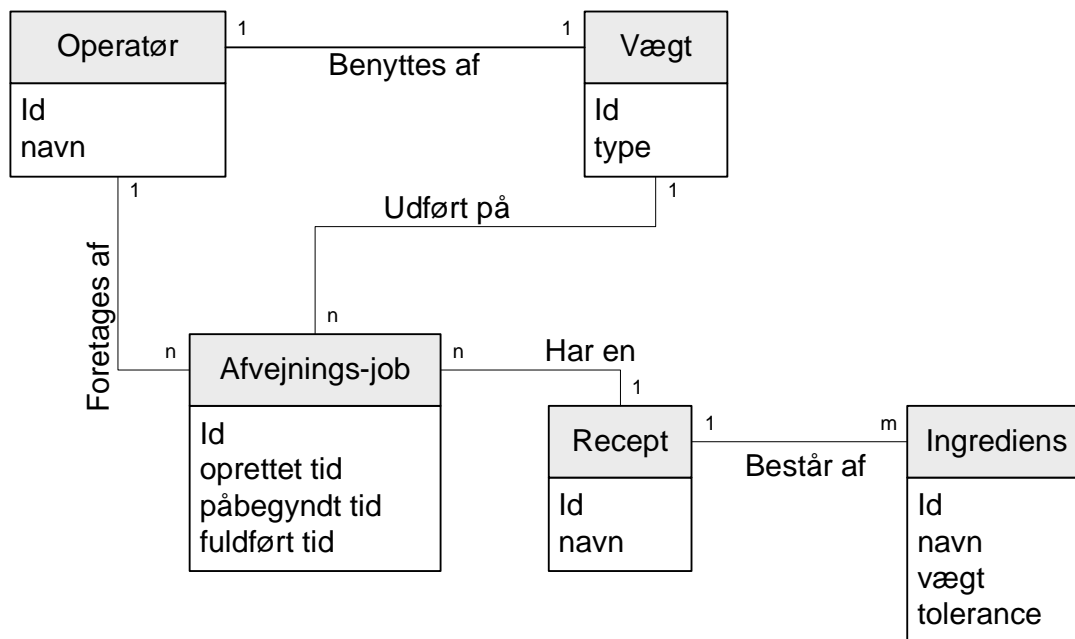
#### 4.1.5 Se lønningsdata

Preecondition	Der skal være oplysninger i databasen om hvad de forskellige medarbejdere har afvejet.
Success end condition	Systemet har vist de relevante oplysninger, så lønningspersonalet kan lave deres løn beregninger, så operatørerne kan få den rette løn.
Main succes scenario	<ol style="list-style-type: none"><li>1. Lønningsprogrammet startes op</li><li>2. Start- og sluttid vælges i GUI'en</li><li>3. Der trykkes vis.</li><li>4. Programmet viser en liste over alle medarbejdere for den angivne periode, og hvor mange afvejninger de hver især har udført.</li><li>5. Der kan fortsættes fra punkt to, hvis brugeren vil.</li><li>6. Program lukkes.</li></ol>
alternate scenario	
Possible errors	Der er ikke forbindelse til databasen Lønningsmedarbejderen vælger en slutdato, som er før startdatoen. Operatørerne har ikke lavet nogen afvejninger i det valgte tidsrum.

#### 4.1.6 Håndter DB

Preecondition	Database server skal kører. Der skal kunne sendes messages til Database-serveren.
Success end condition	At der er sendt relevante data.
Main succes scenario	<ol style="list-style-type: none"><li>1. modtager et kald</li><li>2. indlæser data</li><li>3. sender data retur</li></ol>
Alternate scenario	<ol style="list-style-type: none"><li>1. Modtager kald</li><li>2. Data findes ikke databasen (ugyldigt id angivet)</li><li>3. Sender tomt svar tilbage</li></ol>
Possible errors	

## 4.2 Domain model



### Operatør

Person der håndterer afvejningsproceduren. Er kun registreret i systemet med navn og et id, som bruges til at registrere hvem der har foretaget hvilke afvejninger.

### Vægt

Fysisk enhed som operatørerne benytter til at udføre arbejdsproceduren. Denne enhed fjernstyres af en server, som igennem vægten fortæller brugeren hvad der skal afvejes. For PM4800's vedkommende, er den delt op i et separat tastatur som kører på en PC (ikke vist på denne model).

### Afvejnings-job

Et afvejningsjob kan kun udføres en enkelt gang. Når det er udført, skal tiderne registreres, hvilken operatør der har foretaget det og på hvilken vægt. Det er disse data der er basis for loggen.

### Recept

En opskrift, som er en række ingredienser som alle skal afvejes før recepten er færdig. En recept kan godt genbruges af flere afvejnings-jobs. Skal man således have afvejet ingredienser til 7 ens kager, må man oprette 7 afvejningsjob med samme recept.

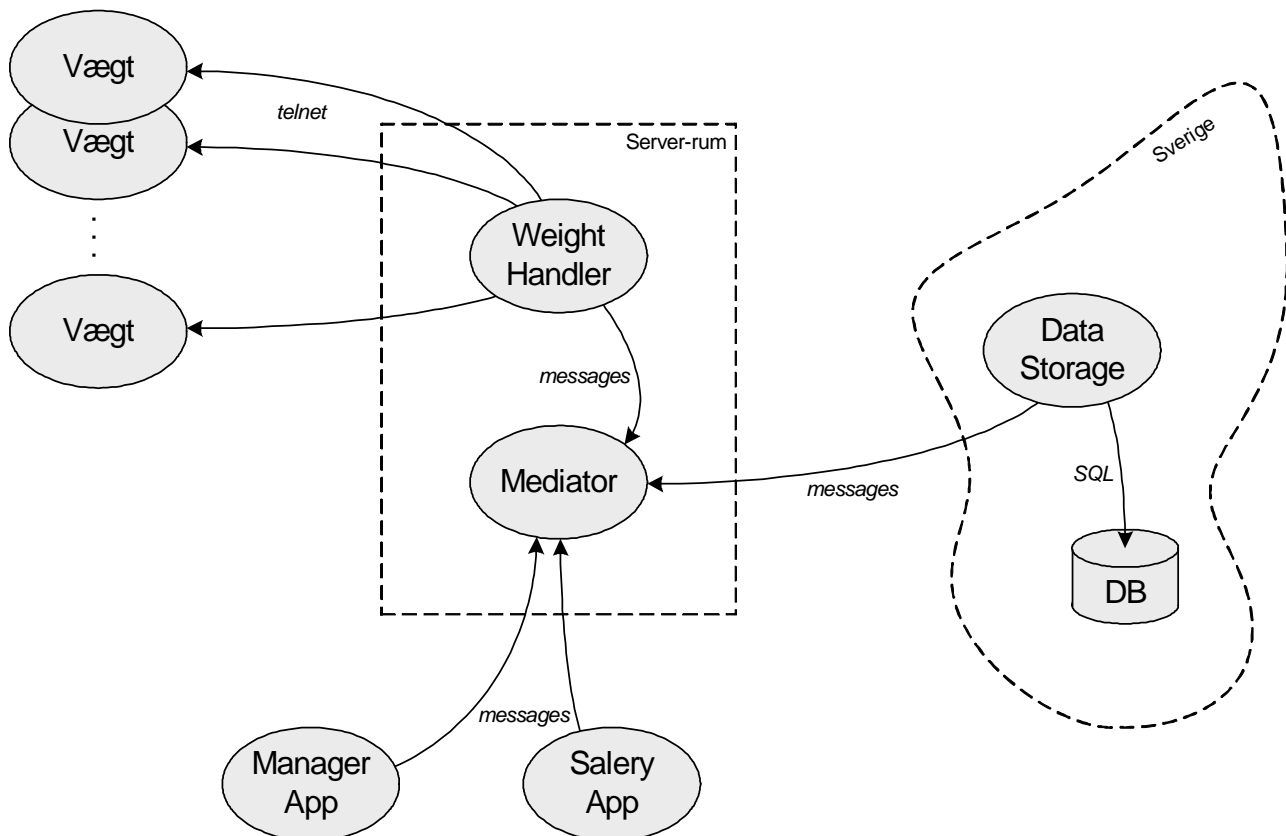
### Ingrediens

En ingrediens har et navn (ex: mel), en vægt (ex 800g) og en tolerance (ex: ±12g)

## 5 Design

### 5.1 Overblik over systemet

Systemet er spredt over forskellige lokationer, med net-forbindelser imellem. Her vises hvem der skal har direkte forbindelse til hvem:



System-overblik. Pilene viser kendskabsrelationer.

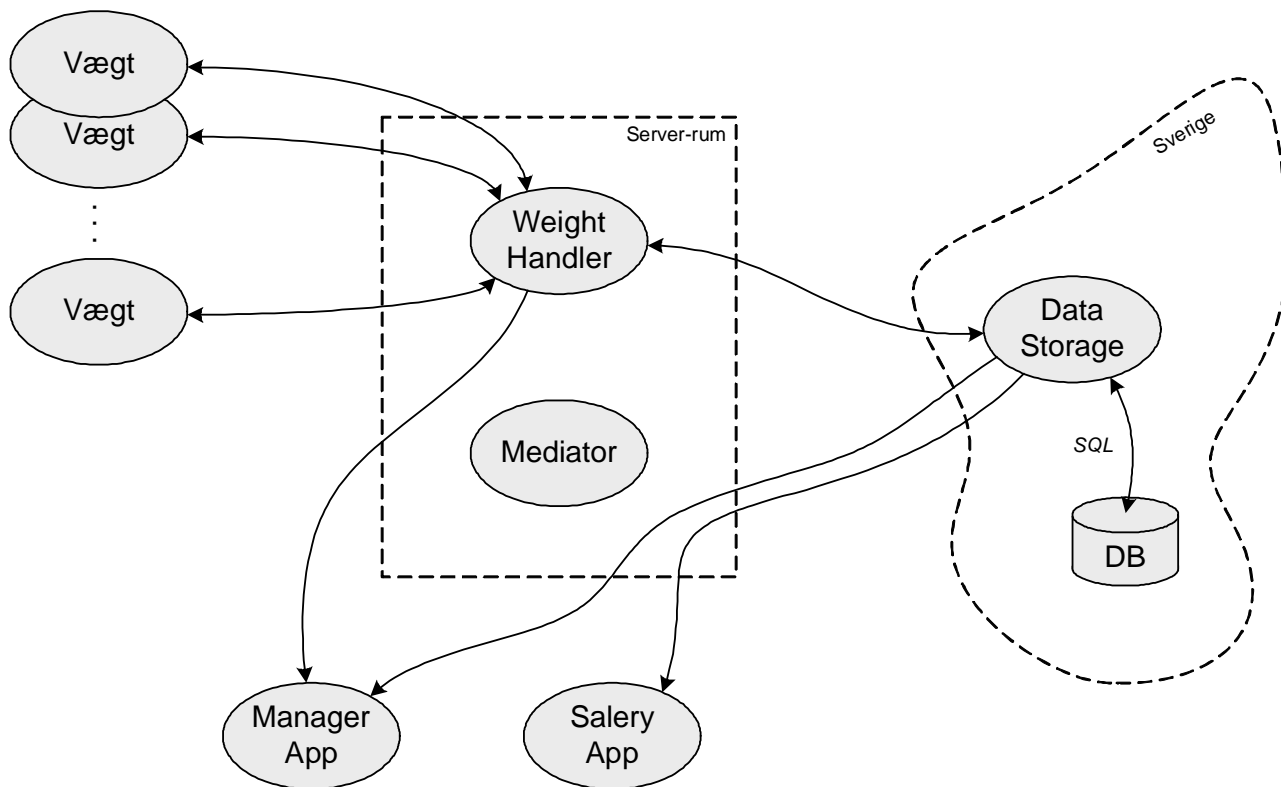
Som det ses, står mediatoren for at kende de forskellige, som derved, hver især kan sende beskeder til hinanden gennem den. Kun WeightHandler kommunikerer direkte med sine vægte udenom mediatoren.

Læg mærke til pilen fra DataStorage, hvilket betyder at serveren i Sverige skal kende serveren i Danmark for at kunne connecte. Det forudsætter at DataStorage kan registrere at forbindelsen eventuelt forsvinder, og forsøger at reconnecte. Det giver dog den sikkerhed at andre (ondsindede) ikke kan connecte direkte til DataStorage-serveren for at hente data, men at personalet i Sverige kan definere hvor der kommunikeres til.

Vægtene kender ikke WeightHandler. Der er WeightHandler der skal kende alle vægtene på forhånd ud fra en xml-fil. Beskrivelse af denne kan findes i JavaDoc under 'libra.weight.KnownWeightListReader'.

### 5.1.1 Flow diagram

For også at give et overblik over hvordan data flyttes rundt i systemet, vises her den samme illustration som ovenstående, men hvor pilene er skiftet ud, så de repræsenterer den logiske data-flow:



System-overblik. Pilene viser Data Flow.

Som det ses, er mediatoren ikke interessant i denne sammenhæng, da det eneste den gør, er at videregende data, uden at bruge eller ændre det selv.

De to ManagerApp og SalaryApp læser kun data, og man har derfor ingen mulighed for at ændre i noget data herfra.

Bemærk at ManagerApp får data fra WeightHandler. Det er for at kunne vise den aktuelle status for alle vægtene i realtime. Den får således besked fra WeightHandler hver gang der sker en ændring ude ved en af vægtene.

WeightHandler henter opgaver fra DataStorage, og styrer vægtene (og guider operatørerne) så recepterne bliver afvejet. Når de er udført registreres det af DataStorage.

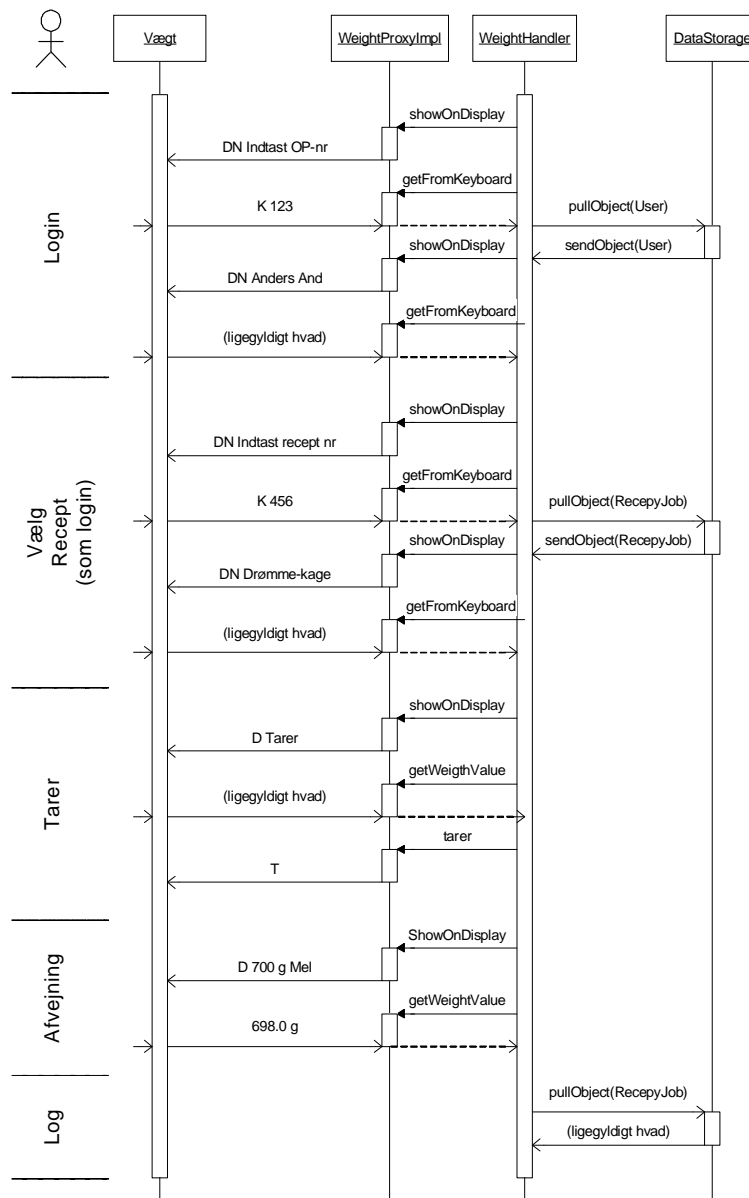
Der er ingen dele i systemet der kan oprette recepter eller opgaver i databasen.

## 5.2 Sekvensdiagrammer

I dette afsnit vises sekvensdiagrammer der beskriver kommunikationen over netværket på et overordnet plan - anvendt til at udvikle systemet ud fra. Der er yderligere sekvensdiagrammer på CD'en under "readme.html" -> "JavaDoc med diagrammer", der er genereret ud fra den implementerede kode.

### 5.2.1 Vægt-proceduren

Dette diagram viser hvordan vægthåndteringen kommunikerer gennem en vægt-proxy ud til selve vægten, og til DataStorage for at få data. Mediatoren er ikke tegnet med her.



## **Vægt**

Den aktuelle vægt der kommunikeres med via telnet. Telnet-beskederne er grundlæggende ens for de to vægttyper, og kommandoerne vist her er kun vejledende. Da vi skal vente på brugeren når vi vil modtage input fra vægten (operatør trykker send), skal `getFromKeyboard` og `getWeigthValue` vente indtil der kommer svar. For PM4800's vedkommende venter de to funktioner så henholdsvis på keyboardet og vægten selv.

## **WeightProxyImpl (i koden som PM4800Proxy og ID7Proxy)**

Interfacet eller adapteren om man vil der forbinder handleren med vægten. Det er denne der implementeres på ny, hvis man vil tilføje endnu en vægttype. Dens opgave er at virke som en lokal proxy for en simpel udgave af vægtene, som handleren kan bruge under afvejnings-proceduren.

## **WeightHandler**

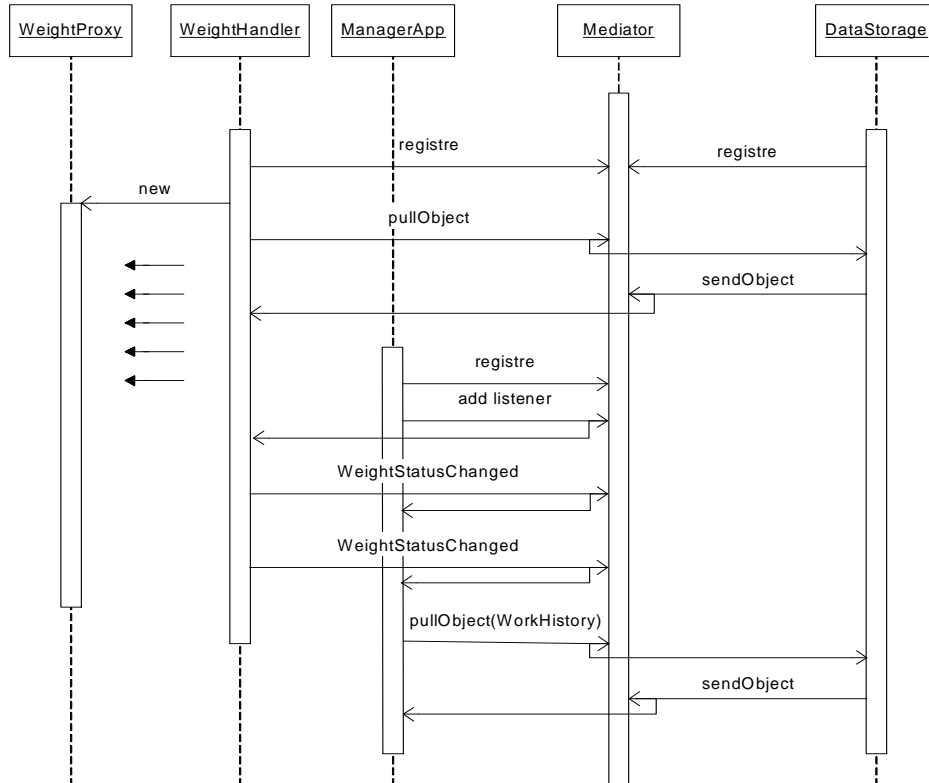
Implementerer selve afvejningsalgoritmen, og lader brugeren logge ind, vælge recept, tarere og afveje ingredienser. Den sørger desuden for at hente de relevante data fra DataStorage-systemet. Der er en `WeightHandler` for hver vægt, som kører i hver sin tråd på en server.

## **DataStorage**

Sørger for at hente fra, og skrive i databasen.

## 5.2.2 Mediator kommunikation

Mediatoren skal startes op som det første, og de andre komponenter registrerer sig i mediatoren. Alle beskeder sendes gennem mediatoren, og uændret videre til modtageren (vises med alternativ syntaks!).



### WeightProxy

Kommunikerer kun med WeightHandler, og kører på samme maskine i samme tråd.

### WeightHandler

Registrerer sig på Mediator, og forventer at DataStorage gør det samme, før egentlige afvejsninger kan foretages. Kan derefter hente data-objekter til afvejning fra DataStorage. Lytter på om nogen vil registrere sig til at modtage WeightStatusChanged-events.

### ManagerApp

Kan vise det aktuelle arbejde på vægtene, ved at modtage events fra WeightHandler. Registrerer sig med navn hos mediatoren, og sender sit navn til WeightHandler som derefter sender sine events til ManagerApp, når de kommer.

Kan desuden hente log-data fra databasen. SalaryApp henter kun data fra databasen, og virker derfor nogenlunde som ManagerApp når den henter log-data.

### Mediator

Denne forventes at være startet før alle andre collaboratører, og har til opgave at videresende beskeder. Den har en adressebog, så beskeder sendes til de korrekte ip-adresser ud fra et id-navn.

### DataStorage

Skal også registrere sig hos mediatoren. Det kan give lidt problemer med firewalls hvis den er placeret i Sverige. Når den har registreret sig, er den klar til at modtage requests som den besvarer.

### 5.3 Aktivitetsdiagram for afvejningsproceduren

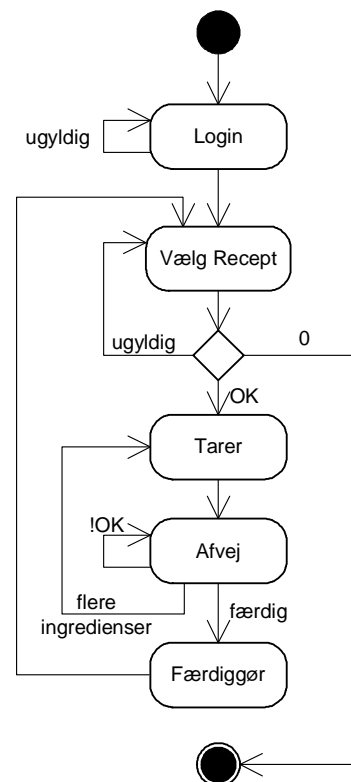
Afvejningsproceduren er den centrale del af systemet, og er implementeret så den følger beskrivelsen fra oplægget. Den er dog udvidet således at en recept kan indeholde mere end en ingrediens. Den er desuden udvidet så en operatør ikke skal angive sit ID for hver afvejning, og der er mulighed for at logge ud, så en anden operatør kan overtage arbejdsstationen.

Proceduren starter med login, og kommer ikke videre før der er indtastet et gyldigt operatør-id, som kontrolleres i databasen.

Der vælges et receptnummer. Vælges '0' logges operatøren af. Vælges et receptnummer der ikke skal afvejes, skal der indtastes et andet. Man kommer kun videre, hvis der indtastes et gyldigt recept-nummer.

Der tares, og ingredienserne afvejes en for en, indtil alle er afvejede. Hver ingrediens skal afvejes indenfor den angivne tolerance, ellers forsøges igen.

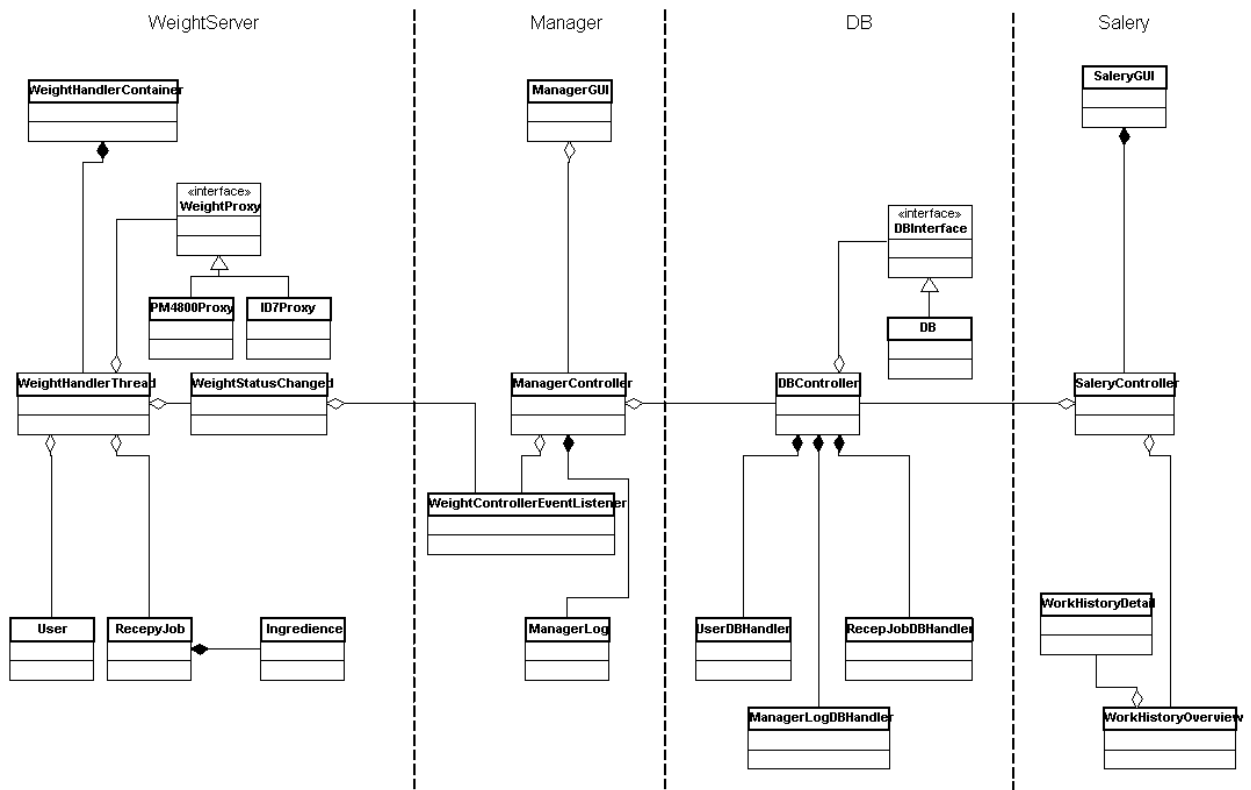
Når alt er afvejede, færdiggøres proceduren ved at afvejningen logges, og eventuelt ved at en label printes ud. Systemet vender tilbage til indtastning af recept, hvor man også kan logge ud.



Indtastning af operatør-id og recept-id implementeres som beskrevet i oplægget ved at brugeren indtaster nummer og trykker 'send'. Det valgte operatør- eller recept-id vises, og operatøren skal trykke send igen for at komme videre.

Hvis en afvejning ikke er i orden vises en fejl-besked i et sekund, hvorefter displayet igen viser hvad der skal afvejes uden at brugeren skal trykke på noget for at få fejlbeskeden væk. Det er ikke nødvendigt at vente på at fejlbeskeden forsvinder.

## 5.4 Klassediagram



Klassediagrammet fra designfasen. Ikke det komplette fra implementeringen.

Data-klasserne er vist hvor deres data læses. De bruges også indenfor DB-området. Mediatoren er ikke repræsenteret på dette niveau. Den er et system der er blevet anvendt for at implementere kommunikationen mellem del-systemerne (adskilt af stiplede linier).

Under implementeringen er det blevet tilføjet yderligere klasser. Disse kan ses på cd'en under "readme.html" -> "JavaDoc med diagrammer".

## 5.5 Database Design

For følgende domæner er det fundet relevant at gemme data i database:

Domæne	Tabel
Operatør	user
Opskrift	recepy
Ingrediens	ingridiens
Log	weightjob

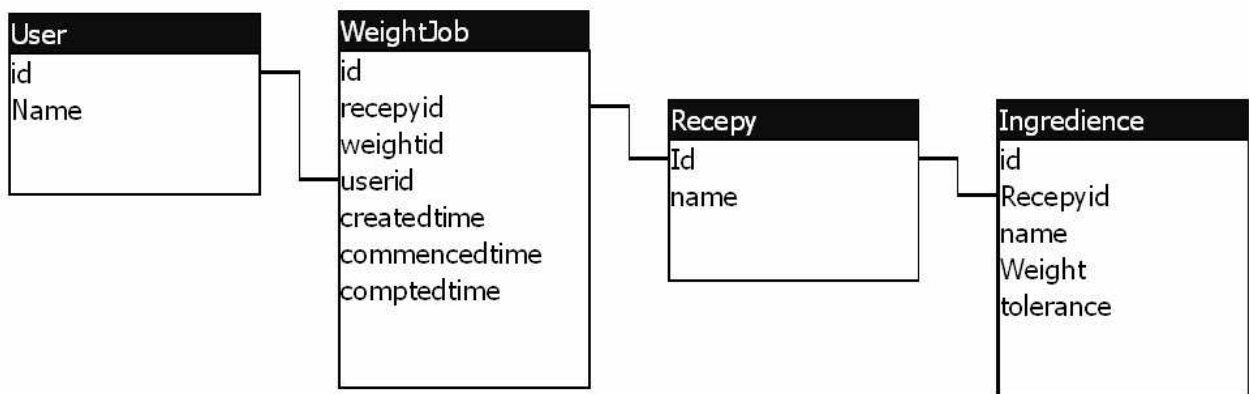
Systemet er designet så databasen nemt kan erstattes af en anden SQL database. Nærmere beskrivelse af udskiftning af databasen kan ses i JavaDoc under `libra.dataStorage.DBInterface`.

Der implementeres læsning og skrivning af relevante data-objekter. Oprettelse af nye vægt job i databasen er ikke med i den implementerede del af systemet.

SQL til oprettelse af tabeller er vedlagt som bilag (Database Oprettelse).

Forbindelseegenskaberne til MySQL databasen kan ændres i "config/settings.conf".

Yderlige oplysninger om installation og opsætning af MySQL kan findes på [www.mysql.com](http://www.mysql.com)



## 6 Implementation

### 6.1 Forklaring af kode

#### 6.1.1 WeightHandler

Implementeret med en container (WeightHandlerContainer ) der startes om, som indeholder en forbindelse til Mediatoren, og en række tråde (WeightHandlerThread). En for hver vægt der kører og skal fjernstyres. Listen fyldes ud fra data i en xml-fil af en klasse der er lavet kun til det (KnownWeightListReader).

Selve implementeringen af kommunikationen med vægtene er implementeret i implementationerne af WeightProxy-interfacet (PM4800Proxy og ID7Proxy). De bruger begge den samme hjælpeklasse (InOut) til telnet-kommunikationen.

Endelig er der en hjælpeklasse (RemoteObjectFiller) der skjuler det lidt besværlige system med at man skal have en event-listener når man vil trække objekter fra databasen over MediatorFrameworket.

WeightHandleren skal køre som en server altid, og kan startes op sammen med en Mediator med klassen: WeightServer.

#### 6.1.2 WeightDisplayAndKeypad

Dette program skal køres på hver PC der står ved en PM4800 arbejdsstation. Den er implementeret med en Swing-GUI der kan skaleres op til at fylde hele skærmen. Indtastninger kan foretages med musen på de synlige knapper, eller på PC'ens almindelige tastatur. Layoutet er lavet så det ligner tastaturet på ID7, så det er nemmere for medarbejderne at gå fra den ene til den anden. Knapperne '.' og '+/-' bruges dog ikke af afvejningsproceduren.

Opsætningen af GUI-komponenterne forgår i en metode der er baseret på autogenerated kode fra NetBeans. Det er private inner-class til håndtering af events (MyActionListener, MyComponentListener og MyKeyListener), og en private inner-class (MyControlLogic) der håndterer logikken i forbindelse med kommunikationen med vægt-handleren.

Beskederne den kan modtage over telnet ligner til forveksling dem til og fra ID7-vægten.

Indkomne beskeder der håndteres:

```
"D xxx\r\n"  
"DN xxx\r\n"
```

Udgående beskeder:

```
"K xxx\r\n" - kun hvis der har været modtaget en "DN xxx\r\n"
```

### 6.1.3 PM4800Emul og ID7Emul

Disse to testklasser blev implementeret før projektet startede, og krævede kun meget lidt til retning. De minder meget om hinanden i koden, og flere dele er direkte kopieret over. GUI'en kan på ingen af dem skaleres som `WeightDisplayAndKeypad`, hvilket ikke betyder noget da de ikke skal bruges i det endelige system. De er kun til test under implementationen.

De sætter komponenterne op, og går i en uendelig løkke for at lytte på indkomne beskeder fra telnet. Der er ikke oprettet en separat tråd til denne opgave, da det er det eneste programmet gør, og kan derfor bruge sin eget tråd til det.

De håndterer beskeder ind og ud som beskrevet i oplægget!

### 6.1.4 WeightStatusChange

Dette data-object har til formål at fortælle en værktøjer med sin manager-app om hvad der sker på alle vægtene i realtime. De sendes fra `WeightHandler`'en hver gang der sker noget ved en vægt til alle de programmer der har registreret sig til at lytte. Til `managerGUI`'en er der et interface (`WeightControllerEventListener`) der kan implementeres til at modtage eventsene med, men der skal også implementeres noget der trækker dem ud af mediator-collaborator systemet.

`WeightStatusChange`-objektet indeholder alle relevante oplysninger om tilstandsændringer for en vægt, som `manager-gui`'en så kan præsentere for værktøjer. Det er ikke disse objekter der bruges til at logge udført arbejde i databasen.

### 6.1.5 ManagerApp

Implementeret med en `JTable`, som bruger en `DefaultTableModel` til at vise data med. `WeightStatusChange`-events indsættes så hver vægt kun vises på sin egen linie, som løbende opdateres. De seneste indkomne events gemmes for hver vægt i en separat vector, så id'erne er til rådighed når der skal hentes log-data fra databasen. Selve `JTable`'n viser kun navnene.

For at vise `JProgressBar`'er i kolonne 4, har den fået vores egen implementation af en `CellRenderer`, der i sin ene metode returnerer at feltets indhold er et `Component`.

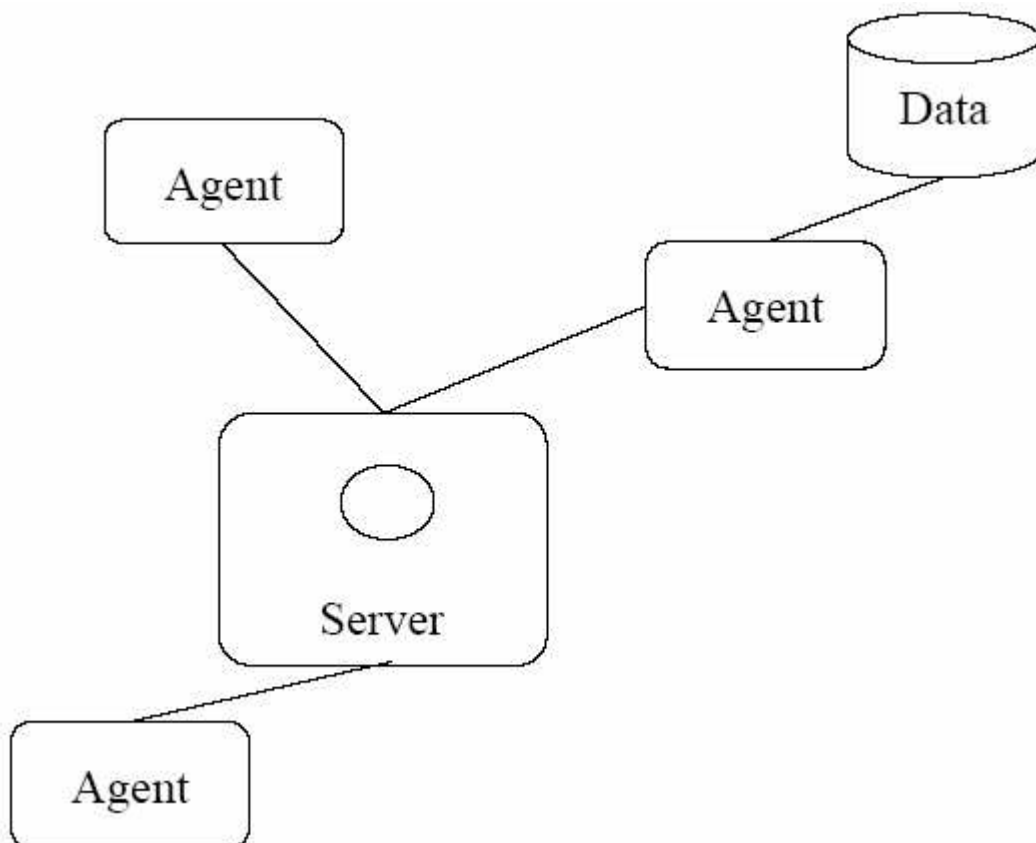
Dobbelklikker man på et felt i den primære `JTable` åbnes et nyt vindue, som henter log-data for det valgte felt fra databasen over mediatoren. Der kan ikke vælges noget i popup-vinduet, og dets data opdateres ikke løbende som hovedvinduet.

Kommunikationen med de andre dele af systemet er implementeret i en selvstændig klasse (`ManagerCon`), der ikke er bundet til GUI'en.

### 6.1.6 Logger

Til at logge har vi valgt at brug en logger fra Apache Log4j (<http://logging.apache.org/log4j/>) `LibraLogger` er en klasse med en static metode der returnerer en apache logger til brug i vores program. Under udviklingen var den sat til at udskrive i konsollen, ved aflevering at projekt skrives der kun i log filerne. Logfilerne lægges i "logs/"

## 6.1.7 Message FrameWork

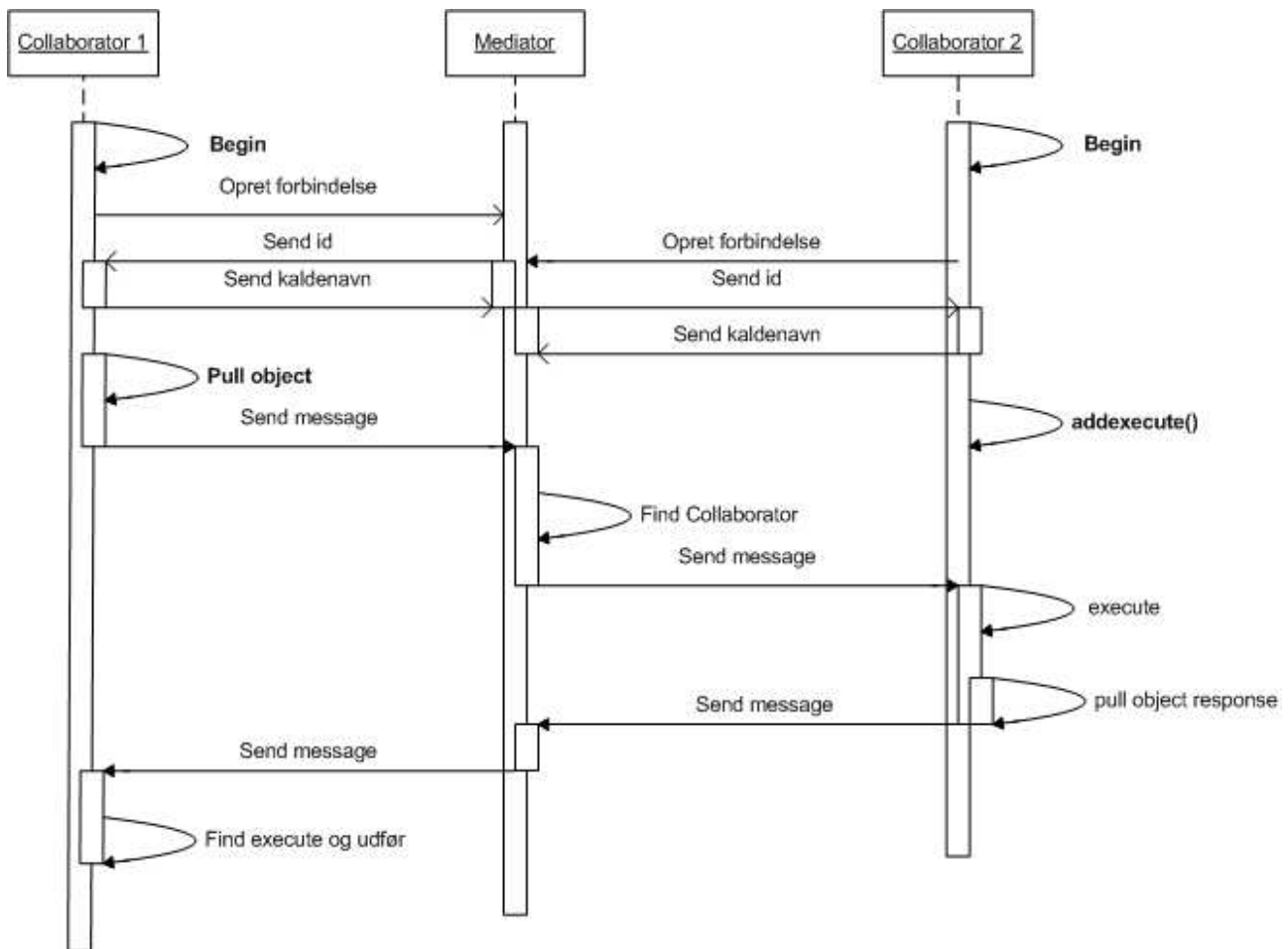


Vi valgte at bruge et message framework til kommunikation mellem enhederne. Der er blevet brugt et eksisterende framework, der er blevet videreudviklet, så det passer til vores formål.

Den grundlæggende tankegang i vores framework, er at hver klient (collaborator) forbinder til en server (Mediator). Mediatoren holder styr på forbindelserne og hvilket kaldenavn den enkelte collaborator har. For at opnå denne funktionalitet i frameworket, er det blevet udvidet med en adressebogsfunktion på mediatoren. Collaboratoren sender sit id og sit kaldenavn til serveren, med et tag om at den gerne vil tilføjes til adressebogen.

Der kan være flere brugere i adressebogen med det samme navn, det vil resultere i at de alle modtager beskederne sendt til dette navn. Der er også ændret i oprettelse af new id, så den kan håndtere at collaboratorne logger af og på.

Der er blevet implementeret et ListenerInterface, der har til formål at gøre det nemt at tilføje en ny listener, hvis man vil implementere en kryptering på forbindelsen. Serveren kan køre med flere forskellige listener, så der kun er data kryptering med nogle af collaboratorne. Et eksempel på dette, der næsten fungerer, er LisnerSSL, der er implementeret i projektet, denne mangler noget til styring af certifikaterne.



Forsimpleret sekvensdiagram over kommunikationen mellem mediator og collaborators

For at lette brugen af frameworket, er der implementeret MediatorUse og CollaboratorUse. Disse to er simple indgange til brug af frameworket. MediatorUse er lavet til at køre i en tråd, uden nogen form for bruger indblanding. CollaboratorUse er lavet med en pullObject(), addExecute() og en begin(). Begin() opretter forbindelsen til mediatorsen, og fortæller mediatorsen sit kaldenavn. AddExecute() bruges til at tilføje en lytter. Lytteren bruges til at svare på et pullObject fra en anden collaborator via mediatorsen. Efter alle lyttere på collaboratorsen er blevet gennemført sendes objektet retur, som et pullResponse. Når man kalder pullObject(), skal der også et objekt der implementer handlerInterface (definerer execute()) med som argument. Denne funktion bliver kaldt når pullResponse() kommer.

### 6.1.8 Database implementeringen

DBcontrolleren opretter en adgang til databasen via et DBinterface, så denne kan udskiftes. DBcontrolleren tilføjer en listener med addExecute(), så den kan modtage messages den ikke selv har pull'et. Når listener-handler-metoden bliver kaldt, finder den ud af, om det er et objekt den skal behandle. Hvis dette er tilfældet, kaldes en DBHandler med database-interfacet og den modtagne objekt som argumenter. Objektet returneres til afsenderen med indlæste data fra databasen.

### 6.1.9 dataObjects

- WorkHistoryOverview
- Ingredience
- User
- RecepyJob
- ManagerLog
- ManagerLogElement.

Klasserne i denne package (libra.dataObjects) indeholder kun data. De bruges forskellige steder i delsystemerne.

### 6.1.10 SaleryApp

Består af to dele. En del som er 2 gange 2 dropdownmenuer og to textfelter. De to første dropdownmenuer og det første textfelt er til at vælge starttidspunkt, man vælger første dag så måned til sidst skriver man startåret ind i feltet. Herefter gentages det med sluttidspunkt.

Når disse datoer er valgt, kan man tryk på knappen ”vis”.

Når der trykkes på knappen ”vis”, sendes der en message gennem mediatoren til DataStorageServeren, om hvilke data der skal sendes retur. De modtagne data bliver herefter vist på GUI.

## 6.2 Databasetabeller

### 6.2.1 weightjob

Kolonne	Type	Default	Beskrivelse
PK id	int(11)	NOT NULL, auto_increment	Id for et vægt job
receptid	int(11)	NOT NULL, '0'	Id for den recept der skal afvejes
weightid	int(11)	NOT NULL, '0'	Id for den vægt hvor afvejningen er foretaget
userid	int(11)	NOT NULL, '0'	Brugeren der har foretaget afvejningen
createdtime	datetime	'0000-00-00 00:00:00'	Hvornår jobbet er oprettet
commencedtime	datetime	'0000-00-00 00:00:00'	Hvornår jobbet er påbegyndt
completedtime	datetime	'0000-00-00 00:00:00'	Hvornår jobbet er fuldført

### 6.2.2 recepty

Kolonne	Type	Default	Beskrivelse
PK id	int(11)	NOT NULL, auto_increment	Id for en recept
name	varchar(100)	NOT NULL, ''	Navnet på recepten

### 6.2.3 ingredience

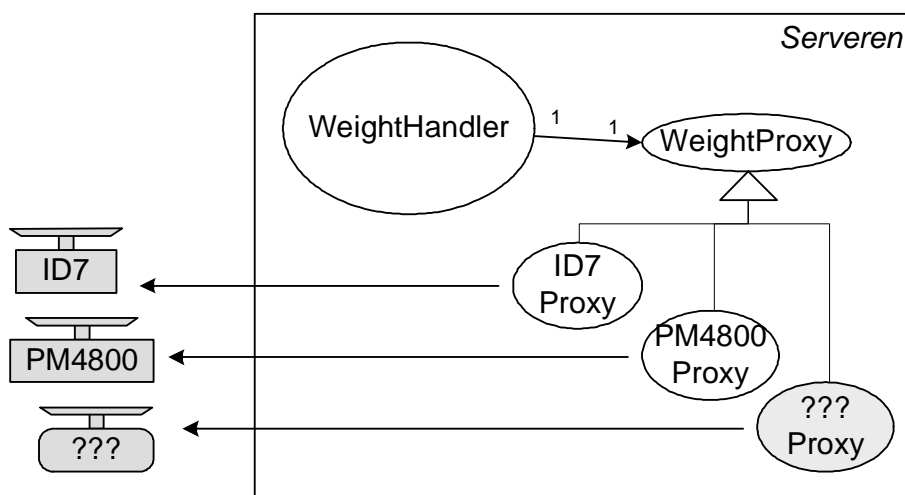
Kolonne	Type	Default	Beskrivelse
PK id	int(11)	NOT NULL, auto_increment	Id for en ingrediens
receptid	int(11)	NOT NULL, '0'	Id for den recept den tilhører
name	varchar(100)	NOT NULL default ''	Navnet på ingrediensen
weight	double	NOT NULL default '0'	Vægten der skal afvejes i gram
tolerance	double	NOT NULL default '0'	Tolerancen i gram

### 6.2.4 user

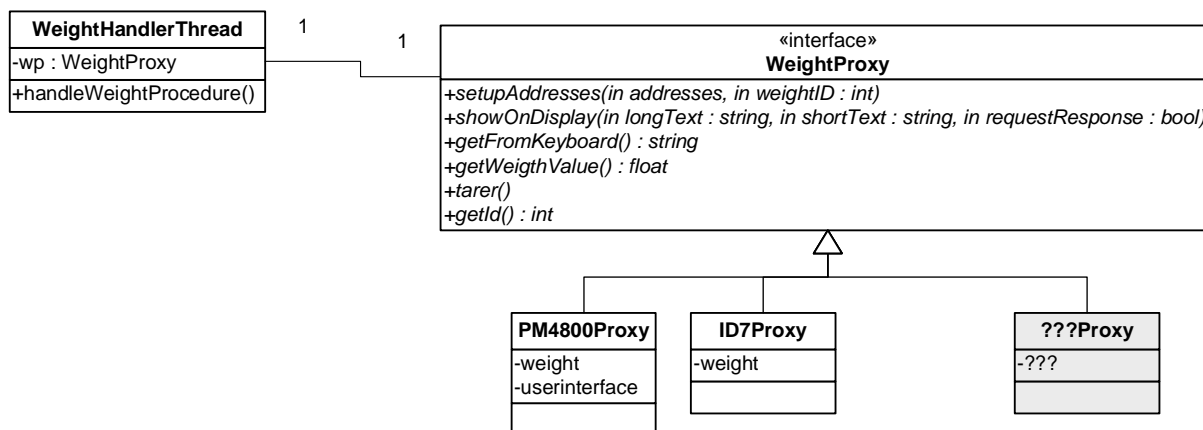
Kolonne	Type	Default	Beskrivelse
PK id	int(11)	NOT NULL, auto_increment	Id for et
name	varchar(100)	NOT NULL, default ''	

## 6.3 Hvordan man tilføjer en ny vægttype.

Systemet skulle designes, så man nemt kan tilføje en ny vægt-type og genbruge afvejningsproceduren. Det er løst det ved at lave et interface, `WeightProxy`, som kører på `WeightHandlerServeren`. Dette implementeres for hver type vægt. `WeightHandler`en ser så kun `Proxy`'en som en lokal enhed der kan udføres funktioner på.



Herunder, det samme vist som klassediagram. (de implementerende klasser implementerer naturligvis metoderne i interface, selvom det ikke vises direkte)



Skal der tilføjes nye vægttyper, skal man altså implementere en ny `Proxy`, illustreret ovenfor som `???Proxy`. Det står beskrevet i `JavaDoc`'en til `Libra.weight.WeightProxy`, detaljeret hvordan de enkelte metoder skal implementeres. Her er et overblik:

### 6.3.1 Følgende skal implementeres

#### Tom default constructor.

Der er vigtigt at implementationen har en constructor der ikke tager nogen argumenter (kan desværre ikke defineres i et interface). Når `WeightHandler`'en starter læser den en `xml`-fil med de kendte vægte, og ud fra navnet på klassen i form af en `String`, instantieres denne korrekte

WeightProxy-implementation, uden at give argumenter til constructoren. Hvis man slet ikke definerer nogen constructor i sin implementation, oprettes en default-constructor af javac der ikke tager nogen argumenter (anbefales).

#### **setupAddresses(...)**

Denne kaldes af WeightHandler efter en instans er oprettet. Formålet er at initialisere vægt-proxy'en med de adresser der er nødvendige for den videre kommunikation med selve vægten. Vægtens id gives også med, som proxy'en skal gemme og returnere med getId().

#### **showOnDisplay(...)**

Kaldes hver gang afvejnings-proceduren vil vise noget på vægtens display. Der angives med en boolean i kaldet om vægten skal returnere et svar fra keyboardet næste gang der trykkes på send af brugeren. Teksten der skal skrives på display'et gives med både i lang og i kompakt form. Hvis vægten har et lille antal karakterer i displayet, bør implementationen bruge den kompakte form. I implementationen af PM4800 vises den kompakte tekst på vægten, mens den lange form vises på displayet på PC-en ved siden af.

#### **getFromKeyboard()**

Forsøger at læse en linie tekst fra vægtens keyboard. Hvis der skal sendes en speciel kommando til vægten for at aktivere keyboardet, skal det implementeres i showOnDisplay(...).

#### **tarer()**

Sender besked til vægten om at den skal tarere, når beskeden bliver sendt. Ikke noget med at denne metode skal vente på at brugeren trykker på en knap. Det er implementeret i vægt-håndteringen således at den kalder getFromKeyboard() før den tarerer.

#### **getId()**

Returnerer ganske enkelt det id der er sat med setupAddresses(...)

De to færdige implementationer bruger begge telnet til at kommunikere med vægtene, og til det formål får de hjælp af klassen libra.weight.InOut. Skal man implementere en ny vægt-proxy der kommunikerer med vægte via telnet, kan man med fordel genbruge den. Den hjælper med at håndtere oprettelse af socket, streams og hjælper med at opdage om forbindelsen mistes.

Der er ikke noget i vejen for at en proxy implementeres så den benytter andet end telnet til kommunikationen med sin vægt.

## 6.4 Hvordan man tilføjer printere til systemet.

Vi har valgt ikke at implementere udskrivning af label efter endt vejning. Strukturen i systemet er dog således, at det kan implementeres på en pæn måde, som her beskrives.

Hver afvejningsstation skal have sin egen printer, og hver type afvejningsstation har sin egen type printer. Det vil derfor være logisk at udskrivningen styres af WeightProxy-implementationerne. Hvis der er forskellige typer printer til ens arbejdsstationer, bør man lave et LabelPrinterProxy-system tilsvarende WeightProxy-systemet. En WeightProxy-instans får så tilknyttet en LabelPrinterProxy-instans.

Konfigurationen af printerne til arbejdsstationerne kan skrives i samme 'config/knownweights.xml' fil, som i forvejen beskriver de enkelte arbejdsstationer. Klassen der indlæser xml-filen skal kun tilrettes hvis man ikke kan nøjes med at tilføje flere address-tags i xml-en.

Selve kaldet til udskrivning af label er forberedt i afvejningsprocedure-rutinen, som kalder en metode (printLabel()) som dog ikke har noget indhold den skal rettes til så den kalder WeightProxy'ens printLabel(...).

Til PM4800 skulle udskrivningen ske fra parallel-porten på PC'en ved siden af, som kører libra.emul.WeightDisplayAndKeypad. Den kan med fordel tage sig af udskrift også, så det skal også opdateres.

Der skal altså ændres i følgende hvis der er en type printer til en type arbejdsstation:

- libra.weight.WeightProxy - tilføj printLabel(...)
- libra.weight.PM4800Proxy - implementer printLabel(...)
- libra.weight.ID7Proxy - implementer printLabel(...)
- libra.weight.WeightHandlerThread - tilret printLabel(...)
- config/knownweights.xml - tilføj info om printere
- libra.emul.WeightDisplayAndKeypad - tilføj print til prn:

## 7 Test

### 7.1 Unit-test

Til test af de enkelte enheder har vi lavet en række små test-klasser. Der er ikke lavet testklasser for alt, men kun hvor der har været relevant under implementationen. De er alle lagt i sin egen under-package, ved navn `'libra.test'`. Her beskrives kort deres formål, og test-output:

#### 7.1.1 DummyObjectFactory

Lavet for at kunne implementere vægthåndteringen inden database-system var færdigt. Der er ingen main, og intet test-output.

#### 7.1.2 TestDataStorage

Tester om der kan hentes data ud fra databasen, uafhængigt af andre dele. Ikke alle data der modtages vises. Tester også hvad der sker, hvis man forsøger at hente noget med et ugyldigt id. Output:

```
Tester: User...
User 2: Hans

Tester: User (ivalid id)...
User 2000: null

Tester: RecepyJob...
RecepyJob 1: Kage
- ingr: vand, 5.0, 1.0
- ingr: Sukker, 4.0, 2.0
- ingr: Mel, 7.0, 1.0

Tester: ManagerLog...
- row: 00:00:00, Hans, Boller
- row: 16:24:32, Supermand, Boller
- row: 11:44:41, Supermand, Kage
```

#### 7.1.3 TestRemoteDB

Tester om man kan hente data fra DataStorage over Mediator-frameworket, som skal være startet, før denne test køres. DataStorageServer skal også være startet i forvejen. Se i koden hvordan det fungerer.

```
TestRemoteDB - Start...
Connecter til Mediator - OK
TestMessageListener - Done (waiting for messages)...!
Received User: 2 - 'Hans'
Received RecepyJob: 2 - 'Boller'
  ingr: Chokolade, 8.0, 6.0
  ingr: Pop, 3.0, 1.0
  ingr: Pip, 6.0, 2.0
Received User: 200 - 'null'
```

### 7.1.4 TestMessageListener

Tester om man kan modtage de events der sendes fra WeightHandler, som ManagerApp skal modtage. Vil man køre testen skal både WeightServer og DataStorage og nogle vægt-emulatorer startes op og betjenes. I eksemplet er der logget en bruger ind på den ene vægt, når test-listeners registrerer sig, og modtager status for alle de kendte vægte.

```
TestMessageListener - Start...
TestMessageListener - Done (waiting for messages)...!
48 [0] '' : -> (0 of 0)
49 [0] '' : -> (0 of 0)
7 [2] 'Supermand' : -> (0 of 0)
8 [0] '' : -> (0 of 0)
7 [3] 'Supermand' : Kage -> vand (0 of 3)
7 [3] 'Supermand' : Kage -> vand (1 of 3)
7 [3] 'Supermand' : Kage -> Sukker (2 of 3)
8 [2] 'Julemanden' : -> (0 of 0)
8 [3] 'Julemanden' : Lagkage -> Sukker (0 of 4)
7 [3] 'Supermand' : Kage -> Mel (3 of 3)
8 [3] 'Julemanden' : Lagkage -> Sukker (1 of 4)
8 [3] 'Julemanden' : Lagkage -> Honning (2 of 4)
7 [3] 'Supermand' : Boller -> Chokolade (0 of 3)
```

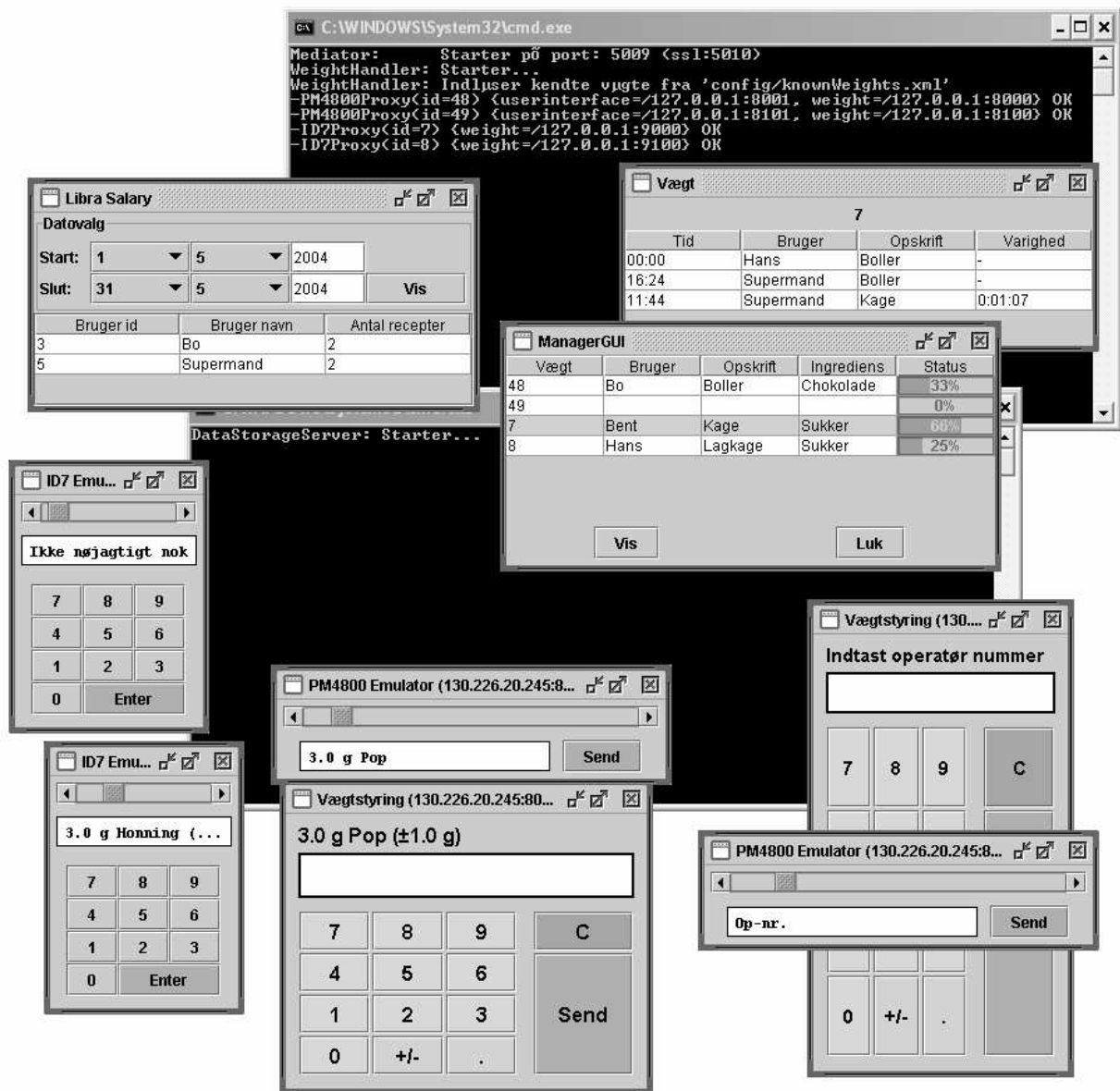
### 7.1.5 DummyWeightStatusChangedGenerator og TestStatusListener

Implementeret for at give ManagerApp noget at implementere op imod, inden vægthåndterings-systemet var færdigt. Det er ikke relevant at vise output her:

## 7.2 Integrations-test.

Her testes om alle delsystemerne kan kommunikere sammen. Der er ikke testet op imod de faktiske vægte, men en test er også foretaget succesfuldt med den udleverede ID7-simulator. De emulatorer vi selv har implementeret, følger den opgivne beskrivelse af kommunikationen.

WeightHandler skal kunne kommunikere DataStorage over Mediatoren.	OK
ManagerApp skal kunne modtager events fra WeightHandler over Mediatoren	OK
ManagerApp skal kunne hente data fra DataStorage over Mediatoren	OK
SalaryApp skal kunne hente data fra DataStorage over Mediatoren	OK



et screenshot af alle systemerne som kører på samme maskine

På screenshot'et ses de to servere (WeightServer og DataStorage) i tekst-konsoller. I bunden er vist to instanser af hver vægttype for at vise at der kan køre mere end en. Til venstre i toppen er SalaryApp, der har hentet antallet af afvejninger for en måned. I højre øvre hjørne er ManagerApp, der viser realtime hvad der sker på vægtene, og et eksempel på et udtræk af loggen for vægt 7.

Systemet er også testet, med komponenterne spredt på to maskiner:

Den ene maskine	Den anden maskine	Resultat
ManagerApp SalaryApp	WeightServer (inkl. Mediator) DataStorage	OK
DataStorage ManagerApp SalaryApp	WeightServer (inkl. Mediator) ManagerApp SalaryApp	OK

## 7.3 Accept Test

Ud fra de opstillede krav i kravspecifikationen er der testet om de opfyldes.

Krav	System test	Accept test
Lønningspersonalet skal kunne se ID, navn, antal recepter i en given periode	Dette er muligt i Salery applikationen	Ok
Før en operatør kan indtaste et recept job nummer, skal han logge ind.	Dette sker ude ved vægtene, og er en grundlæggende del af kommunikationen mellem vægtene og systemet	Ok
Operatør skal kunne indtaste et recept job nummer på vægten	Dette er muligt ved ID7-vægten på dens eget display, og ved PM4800 vægten på computeren ved siden af der kører libra tastatur-applikation.	Ok
Operatør skal kunne se, hvilken ingrediens og antal gram der skal afvejedes på displayet på vægten.	Det vises på ID7-vægtens diplay, og for PM4800 på både libra-tastaturapplikationen, og på vægtens eget lille display.	Ok
Vægten skal automatisk tareres efter hver afvejning	Dette er også en del af den afvejningsprocedure der er implementeret til fjernkontrol af vægtene.	Ok
Værkføreren skal kunne se aktuelle aktiviteter	Dette er muligt i Manager applikationen.	Ok
Værkføreren skal kunne se en log over dagens arbejde	Dette er muligt i Manager applikationens popup.	Ok
Vise tid for hvor længe en operatør har brugt på en afvejning. Denne tid skal vises i minutter og sekunder.	Dette er muligt i Manager applikationen, ved at vælge den ønskede operatør.	Ok
Der skal være JavaDoc af Java koden	Der ligger en JavaDoc, der beskriver koden på CD'en	Ok
Der skal være en vejledning i at tilføje en eller flere nye vægte	Beskrivelsen forefindes i denne rapport	Beskrivelse findes.
System skal udskrive en label, når en afvejning er afsluttet.	Denne funktionalitet er ikke blevet implementeret , men der er en forklaring, på hvordan det skal implementeres i punkt 6.4 i denne rapport.	Ikke implementeret

## 7.4 JavaDoc

JavaDoc ligger på CD'en. Der er vedlagt to slags doc. Den ene er standart javadoc, den anden er genereret i Borland Together. Sidstnævnte indeholder også en del diagrammer der giver et andet overblik over koden. Se vedlagt CD "readme.html"

## 7.5 Brugervejledning

Dette system består af nogle Java-applikationer, og de vægte som er på virksomheden.

### Salery

Salery systemet, er til brug for lønningskontoret. For at få de forskellige data ud, vælges et starttidspunkt, dette vælges ved hjælp af de to dropdown menuer. Det ønskede årstal indtastes i feltet. Det ønskede sluttidspunkt vælges på samme måde. Det antages at brugeren ikke vælger et sluttidspunkt som er før starttidspunktet.

### Manager

Når programmet startes, vises en liste af de vægte der er kendt for systemet.

Hver gang en operatør, fortager sig noget ved en vægt opdateres vægtens linie i tabellen med vægtens id, operatørens navn, den opskrift der arbejdes på og den ingrediens som afvejes. Desuden vises en status, for hvor langt operatøren er nået i opskriften.

Hvis man dobbeltklikker på et af felterne vises en popup med log-data for det valgte felt.

### WeightDisplayAndKeypad

Dette er et numerisk keyboard, til brug ved afvejninger på PM4800 vægten. Fungere som et almindeligt numerisk keyboard. Kan betjenes med mus og tastatur. Programmet kan desuden vise de beskeder der skal vises til operatøren.

### WeightServer

Dette program startes i en kommandolinie, med en bat fil. Den indeholder en Mediator, og WeightHandler serveren.

Mediatoren sørger for kommunikationen mellem de forskellige enheder i systemet.

WeightHandleren sørger for kommunikationen til vægtene. Programmet skal bare startes op, og vil ikke vise noget under normal kørsel.

Når programmet er startet står der følgende:

```
Mediator: Starter på port: 5009 (ssl:5010)
```

```
WeightHandler: Starter...
```

```
WeightHandler: Indlæser kendte vægte fra 'config/knownWeights.xml'
```

```
-PM4800Proxy(id=48) {userinterface=/127.0.0.1:8001, weight=/127.0.0.1:8000} OK
```

```
-PM4800Proxy(id=49) {userinterface=/127.0.0.1:8101, weight=/127.0.0.1:8100} OK
```

```
-ID7Proxy(id=7) {weight=/127.0.0.1:9000} OK
```

```
-ID7Proxy(id=8) {weight=/127.0.0.1:9100} OK
```

### Database

Programmet start med en kommandolinie. Med en bat fil. Når den er startet skrivers der "DataStorageServer: Starter..."

Dette fortæller at nu database delen klar til at sende og modtage data.

## 8 Konklusion

Der er blevet implementeret et system der kan styre et vilkårligt antal vægte over et netværk. Vægtene styres i en afvejningsprocedure der ikke kender vægtende direkte, men som kommunikerer via et interface der implementeres for hver vægttype. Der er beskrevet hvordan man implementerer kommunikation med en ny vægttype under punkt 6.3 i denne rapport.

Afvejningsproceduren er implementeret så der kan være mere end en ingrediens i hver recept. Hvert færdiggjort afvejningsjob logges i en database med start og sluttid samt vægt- og operatør-id.

Der er ikke implementeret udskrift af label for hvert afsluttet afvejningsjob, men det er beskrevet kort hvordan man kan tilføje det.

En værkfører og en lønmedarbejder kan fra hver deres brugerinterface se hvad der er blevet lavet. Værkføreren kan desuden se den aktuelle status for samtlige vægte, i sit interface som opdateres løbende, hver gang der er udført noget på en vægt.

Der er ikke implementeret noget til at oprette afvejningsjob i databasen, og for ikke at skulle oprette nye hver gang der testes, kan samme afvejningsjob udføres flere gange. Dette bør ændres hvis systemet skal sættes i brug.

Der er ingen dele af det bestilte system der ikke er blevet designet og implementeret.

# Bilag

## **Bilag A - Opgavebeskrivelse**

2004-06-07 Ver. 0.90

02355 Netteknologi B  
3 ugers projekt.

En vægtproducent skal levere et afvejningssystem til en gammel kunde.

Systemet skal kunne foretage afvejning ud fra givne recepter.

Alle data fra vejeprocessen skal opsamles elektronisk.

Dette indbefatter blandt andet operatøridentitet, mængde afvejet, osv. (se bilag).

Kunden har allerede 20 vægte af typen PM4800 som med en RS232/ethernet konverter kan tilsluttes til ethernet, og sende vejeresultater med telnet protokollen.

Disse har dog intet tastatur så kunden har foreslået at han blot sætter en gammel bærbar PC op ved siden af hver vægt så han kan bruge display og tastatur på denne.

Kunden har brug for flere vægte, så han har yderligere bestilt 10 stk. ID7, som er født med display og tastatur.

Disse skal bruges parallelt med de eksisterende vægte og have samme indvejningsprocedure. (se bilag)

Det program som implementerer indvejning proceduren skal afvikles på en maskine som befinder sig i kundens server rum.

Afvejningsdata, operatør data osv. hentes under afvejningen fra en server i firmaets hovedsæde i Sverige.

Det er endvidere et krav at en værkfører kan logge ind fra en vilkårlig PC i virksomheden og se den aktuelle afvejnings-log.

Lønningskontoret skal kunne trække antal afvejninger / medarbejdere ud for en vilkårlig tidsperiode, idet disse indgår i medarbejdernes akkordløn.

Systemets samlede deployment-diagram kan ses i deployment-diagram\_bilag\_1.

I skal designe og programmere følgende.

Program som kan simulere ID7 terminalen. (arbejdsplads type 1)

Program som kan simulere PM4800 / bærbar PC. (arbejdsplads type 2)

Program som styrer afvejningsprocedure på de to afvejningssystemer.

Program til overvågning som betjenes fra værkførers PC.

Program som simulere databasen i Sverige.

Program til lønningskontoret i PC.

Afvejningsprocedure se næste side.

Data for vægte og ethernet adapter se øvelse uge10 fra 13 ugers periode. (se den opdaterede version på campusnet.)

Samt bilaget Kommunikation med PM4800 sidst i dette dokument.

Generelt gælder det at i det software i fremstiller skal i anvende principperne fra OOAD og UML.

Ekstra opgave.

Arbejdsplads type 1 tilføjes en netværksprinter.

Tilsvarende tilføjes en parallelprinterport printer til arbejdsplads type 2. Denne printer er monteret på parallelprinterporten på PC.

Disse printere udskriver en label ved hver afvejning.

Afvejningsprocedure:

Display viser <Indtast operatør nummer>

Operatør indtaster operatør nummer og afslutter med enter.

Display viser navn på operatør <ANDERS AND>

(hvis operatør nr ikke er oprettet udskrives fejl meddelelse og der startes forfra)

Operatør taster enter for at komme videre.

Display viser <Indtast recept nummer>

Operatør indtaster nummer og afslutter med enter.

(hvis recept nr. ikke er oprettet udskrives fejl meddelelse og der startes forfra)

Systemet tarerer.(nulstilles.)

Display viser herefter <Placer tara>

Operatør pålægger tarabeholder og taster på enter.

Systemet registrerer taravægte.

Vægten tareres (nulstilles).

Display viser recept navn og mængde <1.234 kg MEL>

Operatør afvejer 1.234 kg og taster enter.

Vejerresultat kontrolleres og godkendes.

(hvis vejerresultat er uden for tolerance område må afvejning gå om)

Der udskrives resultatet af afvejningen i loggen.

Der udskriver eventuel etikette.

Display viser <Fjern brutto>

Operatør fjerner tarabeholder med netto indhold og taster på enter.

Systemet tareres.

Herefter startes der forfra.

Overvej selv mulige fejltilstande.

Kommunikation med PM4800

PM4800 anvender telnet protokollen på port 8000.

Vægten er meget simpel den har blot en knap SEND.

Klienten som kommunikerer med vægten håndterer ovenstående afvejningsprocedure. D.v.s. t den sender kommandoer til vægten og modtager svar fra denne.

Nedenfor er listet en del af den vigtigste kommandoer.

Kommunikation med PM4800 / EB01:

EB01 anvender telnet protokollen på port 8000.

Her er nogle af de ting I kan anvende.

Der sendes følgende til vægten:

```
"D Hello World" chr(13)chr(10)
```

Vægten udskriver på displayet "Hello World".

Der kommer intet svar.

Der sendes følgende til vægten:

```
"S" chr(32)chr(13)chr(10)
```

Vægten svarer med aktuelvejeresultatet.

```
"S 1.234 kg" chr(13)chr(10)
```

Der sendes følgende til vægten:

```
"T" chr(32)chr(13)chr(10)
```

Vægten tarerer, der kommer intet svar.

Der tages på SEND knappen:

Vægten sender:

```
" 0.05g" chr(13)chr(10)
```

Opgave ændring !!!

Kundens EDB afdeling er blevet indblandet i projektet. De har fundet ud af at forbindelsen til ID7 og PM4800/Bærbar PC skal foregå med en adapter.

De er helt hysteriske med dokumentationen til denne idet de ønsker senere at kunne udvide systemet med en helt tredje type vægt. De regner så selv med at omskrive denne adapter så den kommer til at passe til en senere nyere vægttype.

Vi har fortalt kundens edb afdeling at det er alt for sent at komme med sådan et krav.

De argumenterer med at de synes det er for dårligt at vi ikke af os selv har valgt denne løsning.

Det har vi måttet give dem ret i og lovet dem at vi vil forsøge også at få dette med i projektet.

## ***Bilag B - Database Oprettelse***

```
# Database: libra
# Table: 'weightjob'
#
CREATE TABLE `weightjob` (
  `commencedtime` datetime default '0000-00-00 00:00:00',
  `id` int(11) NOT NULL auto_increment,
  `receptid` int(11) NOT NULL default '0',
  `userid` int(11) NOT NULL default '0',
  `createdtime` datetime default '0000-00-00 00:00:00',
  `completedtime` datetime default '0000-00-00 00:00:00',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
# Database: libra
# Table: 'recepty'
#
CREATE TABLE `recepty` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(100) NOT NULL default "",
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
# Database: libra
# Table: 'ingredience'
#
CREATE TABLE `ingredience` (
  `id` int(11) NOT NULL auto_increment,
  `receptid` int(11) NOT NULL default '0',
  `name` varchar(100) NOT NULL default "",
  `weight` double NOT NULL default '0',
  `tolerance` double NOT NULL default '0',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

```
# Database: libra
# Table: 'user'
#
CREATE TABLE `user` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(100) NOT NULL default "",
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

## Bilag C – Udvalgt Kode

Vi har udvalgt enkelte dele af koden som kan være interessante at slå op og kigge i. Resten vil være at finde på den vedlagte CD, eller kan hentes på:

<http://libra.atkeland.dk>

### interface WeightProxy

Det interface der skal implementeres for hver type vægt, for at de kan bruges i afvejningsproceduren.

```
/*
 * Created on 10-06-2004
 */
package libra.weight;

import java.util.Hashtable;

/**
 * <p>Det interface der skal implementeres for at lave en ny vægt-proxy.</p>
 *
 * <p><b>BEMÆRK:</b> Den implementerende klasse SKAL have en default-constructor der IKKE tager nogen argumenter. Det kan desværre ikke defineres i et interface. Det er vigtigt, fordi systemet skal kunne oprette en WeightProxyImpl ud fra navnet alene, som står i xml-filen med kendte vægte.</p>
 *
 * <p>Man kan med fordel bruge InOut-klassen til at hjælpe implementationen af en ny vægt-proxy.</p>
 *
 * @version 1.2 (17-06-2004)
 * @see libra.weight.InOut
 */
public interface WeightProxy {

    //NOTE: Implementations must have default Constructor.
    //      Not possible to declare in interface.

    /**
     * Modtager alle relevante adresser i et hashtable, og vægtens id.
     * <p>Det er op til implementationen at opbevare disse oplysninger til senere brug.</p>
     * <p>Denne metode ville have været constructoren, hvis en sådan kunne defineres i et interface.</p>
     */
    public void setupAddresses(Hashtable addresses, int weightID) throws MissingAddressException;

    /**
     * Implementationen skal sørge for at operatøren har trykket på en send-knap før værdien læses og sendes tilbage til den der har kaldt denne metode.
     *
     * @return en float-værdi af vægten <b>i gram</b> efter der er trykket send
     * @throws NoConnectionToWeightException Hvis forbindelsen mistes
     */
    public float getWeigthValue() throws NoConnectionToWeightException;

    /**
     * Viser besked på display.
     *
     * <p>Fordi ikke alle vægte har et særligt stort display angives den samme tekst to gange når denne metode kaldes. Både i fuldt forståelig tekst, og i forkortet form, som kan læses på små displays. Det er op til implementationen at vælge om den vil vise den lange eller den korte tekst.</p>
     *
     * <p>Da de fleste vægte skal vide på forhånd om de skal returnere et input fra keyboardet, gives det med som en boolean til denne metode. Der skal typisk skrives noget på displayet, når man vil have et input fra brugeren.</p>
     *
     * @param longText beskeden i let forståelig tekst
     * @param shortText beskeden i kompakt, forkortet form.
     * @param requestResponse Om vægten skal sende et tak fra keyboardet tilbage næste gang.
     * @throws NoConnectionToWeightException Hvis forbindelsen mistes
     */
}
```

```

public void showOnDisplay(String longText, String shortText, boolean requestResponse)
    throws NoConnectionToWeightException;

/**
 * Modtager en String fra tastaturet på vægten.
 *
 * <p>Bemærk at mange vægte skal have at vide på forhånd hvornår de skal returnere
 * et svar fra tastaturet. Det sættes med <code>setupAddresses</code> som kaldes
 * umiddelbart før denne metode.</p>
 *
 * @return En tekst fra vægten, som indeholder et tal der kan parses
 * @throws NoConnectionToWeightException Hvis forbindelsen mistes
 */
public String getFromKeyboard() throws NoConnectionToWeightException;

/**
 * Nulstiller vægten på den aktuelle værdi, når metoden kaldes.
 *
 * <p>Hvis man ønsker at brugeren skal trykke send for at tarere, implementeres
 * det ved at kalde <code>showOnDisplay</code> og <code>getFromKeyboard</code>.
 * tarer-metoden tarerer bare med det samme !</p>
 *
 * @throws NoConnectionToWeightException Hvis forbindelsen mistes
 */
public void tarer() throws NoConnectionToWeightException;

/**
 * Returnerer vægtens id, som bruges når der skal logges i databasen.
 * Værdien modtages i <code>setupAddresses</code>
 * @return Vægtens id, til logning.
 */
public int getId();
}

```

## handleWeightProcedure()

Selve metoden der implementerer afvejningsproceduren. Den er her klippet ud af klassen: `libra.weight.WeightHandlerThread`.

```

/**
 * Selve vægt-håndterings-proceduren.
 *
 * Starter med at logge en operatør ind, og slutter når operatøren logger ud igen.
 *
 * Vægt-håndterings-proceduren er holdt i en enkelt metode, da vi mener det er nemmere at
 * overskue hvad der sker når det hele er samlet i en lang metode, frem for at man skal kigge
 * frem og tilbage i forskellige metoder der kalder hinanden.
 *
 * I pseudokode, ville implementeringen se således ud:
 * <pre>
 * operatør_login();
 *
 * while(handleAnotherRecepy) {
 *     vælg_recept();
 *
 *     while(hasMoreIngredienses) {
 *         tarer();
 *         afvej();
 *     }
 *
 *     log_afvejning();
 * }
 * </pre>
 * Operatøren kan sætte handleAnotherRecepy = false ved at vælge 0 som recept-id.
 */
public void handleWeightProcedure() {

    sendStatusChange(null, null, WeightStatus.NO_OPERATOR, 0);

    User user = null;

    boolean userOK = false;
    while (!userOK) {
        try {
            wp.showOnDisplay("Indtast operatør nummer", "Op-nr.", true);

```

```

String idStr = wp.getFromKeyboard();
int id = safeParseInt(idStr, 0);

//read from database
wp.showOnDisplay("Vent venligst", "Vent...", false);
user = (User) remote.fillObject(new User(id), myContainer.getDataStorageName());

if (user != null && user.getName() != null) {
    wp.showOnDisplay(user.getName(), user.getName(), true);
    wp.getFromKeyboard();
    userOK = true;
} else { // not OK
    wp.showOnDisplay("Ugyldigt operatør nummer: " + id, "Ukendt", true);
    wp.getFromKeyboard();
}

} catch (NoConnectionToWeightException e) {
    sendStatusChange(null, null, WeightStatus.UNCONNECTED, 0);
    sleepEasy(sleepMillisOnError);
}
}

//now user is OK -----
sendStatusChange(user, null, WeightStatus.NO_RECEPY, 0);

boolean handleAnotherRecepy = true;
while (handleAnotherRecepy) {

    RecepyJob recepyJob = null;

    boolean recepyOK = false;
    while (!recepyOK) {
        try {
            wp.showOnDisplay("Indtast recept nummer", "Recept-nr", true);
            String idStr = wp.getFromKeyboard();
            int id = safeParseInt(idStr, -1);

            if (id < 0) {
                wp.showOnDisplay("Ugyldigt input", "Ugyldigt", true);
                wp.getFromKeyboard();
            } else if (id == 0) {
                wp.showOnDisplay("Logget ud.", "Log ud", true);
                wp.getFromKeyboard();

                //get out of recepy-loop, and don't handle this one
                recepyOK = true;
                handleAnotherRecepy = false;
            } else {
                //read from database
                wp.showOnDisplay("Vent venligst", "Vent", false);
                recepyJob = (RecepyJob) remote.fillObject(new RecepyJob(id), myContainer.getDataStorageName());

                if (recepyJob != null && recepyJob.getName() != null) {
                    wp.showOnDisplay(recepyJob.getName(), recepyJob.getName(), true);
                    wp.getFromKeyboard();
                    recepyOK = true;
                } else { // not OK
                    wp.showOnDisplay("Ugyldigt recept nummer: " + id, "Ugyldigt", true);
                    wp.getFromKeyboard();
                }
            }
        } catch (NoConnectionToWeightException e) {
            sendStatusChange(user, null, WeightStatus.UNCONNECTED, 0);
            sleepEasy(sleepMillisOnError);
        }
    } //END while(!recepyOK)

    if (!handleAnotherRecepy) {
        break;
    }

    //now recepy is OK -----
    sendStatusChange(user, recepyJob, WeightStatus.HANDLING_RECEPY, -1);

    boolean notifyAboutTarer = true; //only first time.

    ArrayList ingrediencies = recepyJob.getIngrediencies();
    int ingrTop = ingrediencies.size();
    for (int ingrThis = 0; ingrThis < ingrTop; ingrThis++) {

```

```

    Ingrediente ingr = (Ingrediente) ingrediencies.get(ingrThis);

    boolean tarerOK = false;
    while (!tarerOK) {
        try {
            if (notifyAboutTarer) {
                wp.showOnDisplay("Placer tara", "Tarer", true);
                wp.getWeigthValue();
            }
            wp.tarer();
            tarerOK = true;
            notifyAboutTarer = false;
        } catch (NoConnectionToWeightException e) {
            sendStatusChange(user, recepyJob, WeightStatus.UNCONNECTED, 0);
            notifyAboutTarer = true;
            sleepEasy(sleepMillisOnError);
        }
    }

    //Now the display on the weight is set to 0.0 g -----

    boolean weightOK = false;
    while (!weightOK) {
        try {
            wp.showOnDisplay(
                ingr.getWeight() + " g " + ingr.getName() + " (± " + ingr.getTolerance() + " g)",
                ingr.getWeight() + " g " + ingr.getName(),
                false);

            float f = wp.getWeigthValue();

            //if within tolerance !!!
            if ((ingr.getWeight() - ingr.getTolerance() < f
                && f < ingr.getWeight() + ingr.getTolerance())) {
                weightOK = true;
            } else {
                wp.showOnDisplay("Ikke nøjagtigt nok", "Unøjagtigt", false);
                sleepEasy(1000);
            }
        } catch (NoConnectionToWeightException e) {
            sendStatusChange(user, recepyJob, WeightStatus.UNCONNECTED, ingrThis);
            sleepEasy(sleepMillisOnError);
        }
    }

    //Now an ingredience is correctly meassured -----
    sendStatusChange(user, recepyJob, WeightStatus.HANDLING_RECEPY, ingrThis);
}

//now all ingredienses are OK -----
sendStatusChange(user, recepyJob, WeightStatus.HANDLING_RECEPY, ingrTop - 1);

logResult(user, recepyJob);
printLabel(user, recepyJob);

boolean completionOK = false;
while (!completionOK) {
    try {
        wp.showOnDisplay("Fjern brutto", "Fjern brutto", true);
        wp.getWeigthValue();
        wp.tarer();
        completionOK = true;
    } catch (NoConnectionToWeightException e) {
        sendStatusChange(user, recepyJob, WeightStatus.HANDLING_RECEPY, ingrTop);
        sleepEasy(sleepMillisOnError);
    }
}

} //END while(handleAnotherRecepy)
} //END of method

```