Udgave

1

2. SEMESTERPROJEKT

Gruppe 5

Secure´O´matic

Design

Benjamin Sørensen, 02284
Tomas Stæhr Hansen, 03539
Stefan Nielsen, 02829
Mubeen Ashraf, 9279
Hussein Kleit, 9281

SECURE´O´MATIC

Design

Dalgas Avenue 2 • 8000 Aarhus C
IKT

Indholdsfortegnelse

Kapitel

1

1. Indledning

1.1 Formål

Målet med dette dokument er at beskrive projektets designfase. Herunder beskrives valg af designstruktur og udarbejdning af færdige klassediagrammer og sekvensdiagrammer. Dokumentet har til formål at overskueliggøre produktets virkemåde.

1.2 Referencer

SPU-Håndbogen: Udgivet af Ingeniøren|bøger, ISBN 87-571-1046-8

SPU/UML-noten: Udleveret af kunden.

Design Patterns, Udgivet af Pearson Professional Education, ISBN 0201633612.

1.3 Læsevejledning

Dette designdokument indeholder en problemanalyse og en UML modelbeskrivelse.

1.4 Ordliste

ODBC: Open Database Connectivity, program til at få adgang til data fra forskellige databasesystemer.

DTLib: Database Template Library

Kapitel

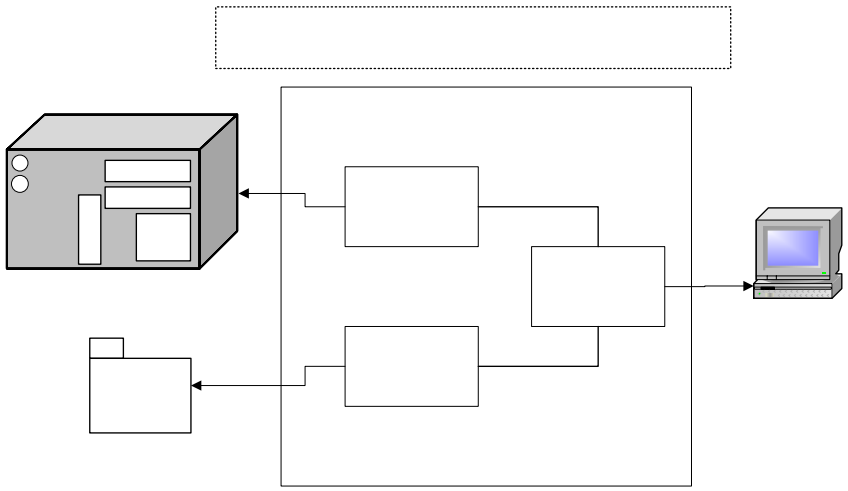
2

2. Problemanalyse

Dette afsnit indeholder indledende overvejelser ang. procesinddeling af opgaven og valg af designstruktur.

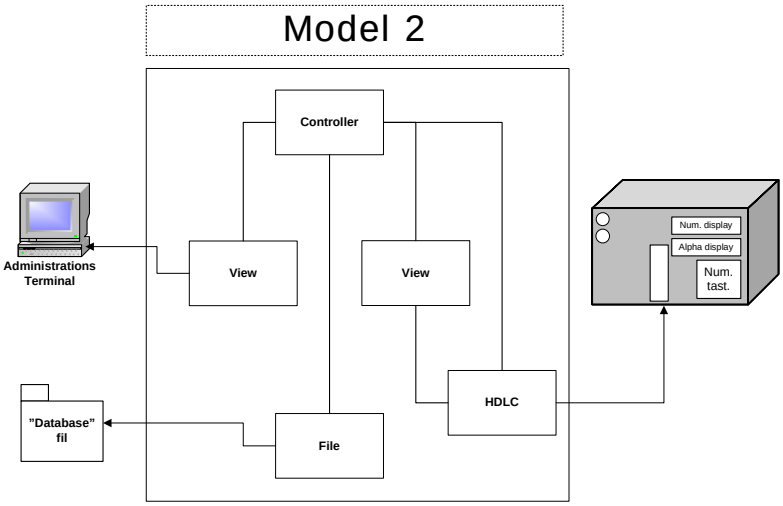
2.1 Procesinddeling

Hvis vi kigger på systemet som værende én proces, skal det totale program indeholde kommunikation og grænseflade til alle aktører der har forbindelse til systemet. I sin enkleste form kunne det se ud som vist på nedenstående figur 1.



Figur 1: Simpel enkeltproces model

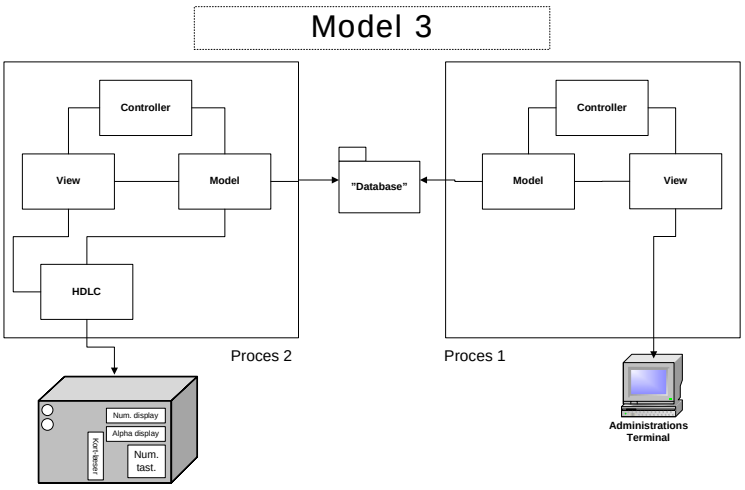
Modellen vist på figur 1 viser en kompakt måde at realisere løsningen på. Denne model vil give et "hurtigt" program med korte kommunikationsveje, men vil til gengæld nedprioritere vedligeholdelsesvenligheden og modificerbarheden. I henhold til kravspecifikationen anses vedligeholdelsesvenligheden for et meget vigtigt punkt, og vi vil derfor se nærmere på en anden mulig modelstruktur.



Figur 2: Grænsefladeorienteret model

Modelstrukturen vist på figur 2 er opbygget efter et grænsefladeorienteret princip, der inddeler processen i forskellige grænseflade-klasser (f.eks. en fil-klasse til at kommunikere med databasefilen). Denne metode giver en bedre udskiftelighed, idet en enkelt klasse kan skiftes ud og erstattes med en tilsvarende klasse til kommunikation med et nyt objekt. Fil-klassen kan skiftes ud med en klasse der kommunikerer med en SQL-database. Den nye klasse skal blot indeholde de samme medlemsfunktioner som den oprindelige.

Opgaven kan med fordel anskues som værende 2 processer. Use Case 1-3 kan ses som værende Proces 1 og Use Case 4 Proces 2. Proces 1 vil da indeholde al administration og kommunikation med aktøren Administrator hvorimod Proces 2 kommunikerer med aktøren Bruger (Se figur 3).

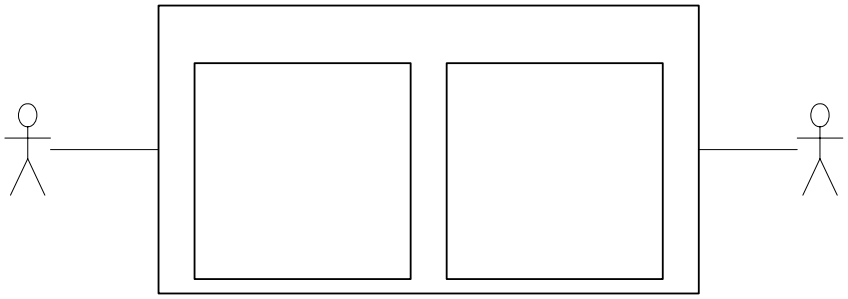


Figur 3: Procesinddelt model-view-controller struktur

Figur 3 viser den procesinddelte struktur med en yderligere forbedring af de enkelte processer. De er her opbygget efter model-view-controller strukturen. Controller-klasserne sørger for at de rigtige funktioner bliver kaldt på de rigtige tidspunkter, hvorimod model-klasserne og view-klasserne modtager kald fra controlleren og sørger for den videre kommunikation til hhv. databasen og et brugerinterface.

2.2 Valg af designstruktur

Ud fra de indledende overvejelser omkring mulige designstrukturer har vi valgt at benytte os af en model-view-controller struktur inddelt i 2-processer. Det har vi valgt med baggrund i en høj prioritering af vedligeholdelsesvenlighed og modificerbarhed.



Figur 4: Systemoversigt jf. valgt designstruktur

Adgangsk

Proces 1:
Use Case 1-3

Administrator

Kapitel

3

3. UML Modelbeskrivelse

Afsnittet giver en beskrivelse af designet udfra UML og use case principperne.

Vi har valgt at lave to selvstændige programmer. I det efterfølgende vil program 1 blive kaldt Admin og program 2 for Secured.

3.1 Admin

Denne del af programmet varetager følgende use cases.

Use case 1: Opret bruger

Use case 2: Slet bruger.

Use case 3: Rediger bruger

Disse use cases aktiveres af administratoren. Grænsefladen til administratoren er skærmen og tastaturet.

Admin har en ekstern grænseflade til en SQL database.
Valget af database er yderligere beskrevet i kapitel 4.

Vi har valgt at bruge Model-View-Controller (MVC) designpatternet[1]. Det giver en høj modularitet og en god fragmentering af programmet. Da vi har lagt vægt på vedligeholdelsesvenlighed i kravspecifikationen er modularitet en nødvendighed. Ideen bag MVC er at adskille koden i 3 klasser, som har hver deres funktion:

Modellen

Modellen står for selve databehandlingen. Modellen leverer data til view'et uden at bekymre sig om, hvordan data skal repræsenteres. Data fra Modellen er display-neutralt, så det kan interfaces til mange views uden at der opstår redundant kode. Dette gør det nemmere at vedligeholde koden, og nedbringer antallet af fejl. Modellen svarer på forespørgsler fra Controlleren, behandler data og giver besked til aktive views om at opdatere.

Controller:

Controlleren er ansvarlig for styringen og interaktion fra brugeren i form af mus og tastatur. Interaktion fra brugeren trigger de events som skal ændre Modellen, og Controlleren fortæller registrerede View at de skal opdatere deres display.

View:

Viewdelen står for den grafiske data repræsentation. View'et er isoleret fra komplekse data operationer, eg. database adgang eller tilgang til hardware.

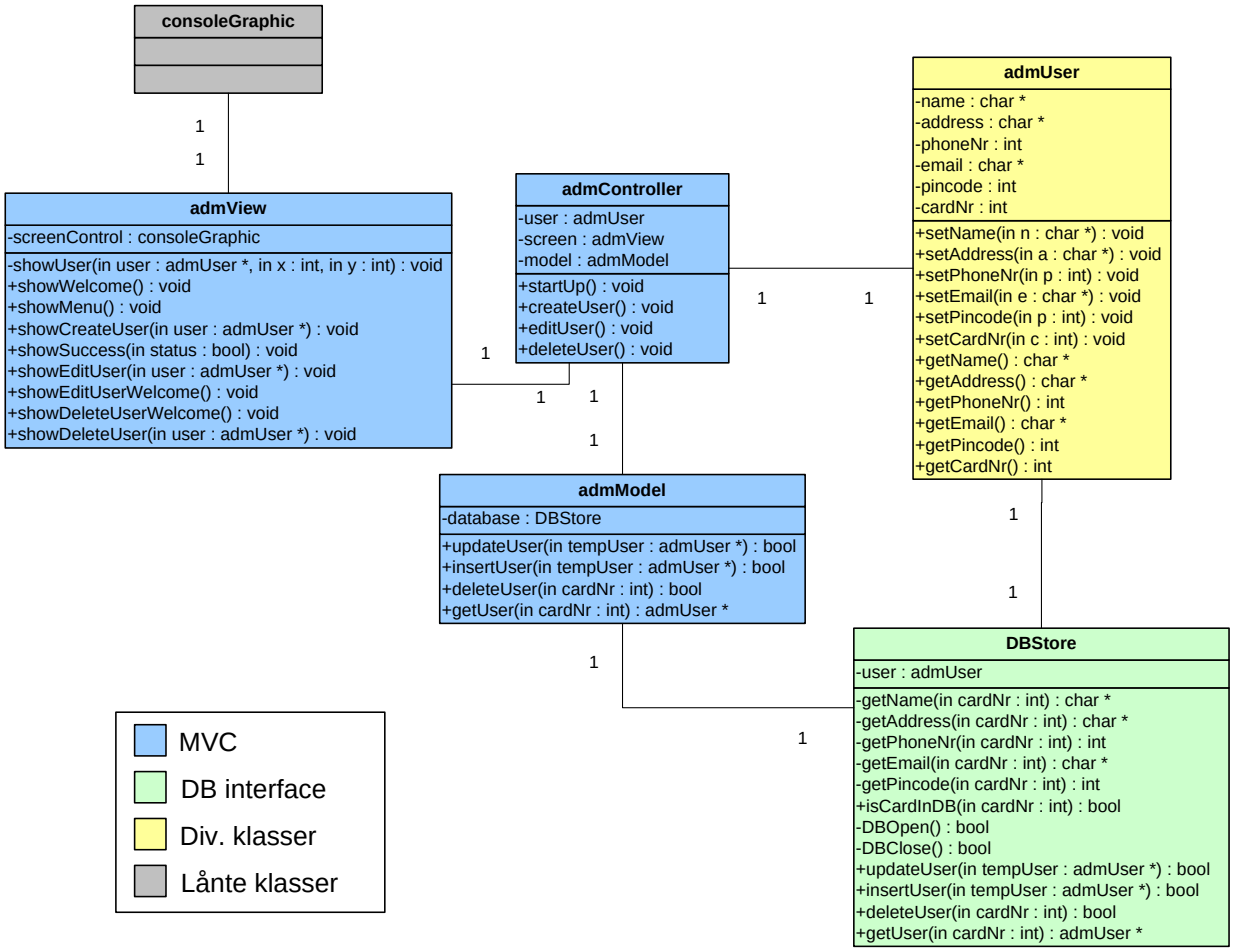
[1] MVC er bla. Gennemgået i bogen Design Patterns.

View'et modtager de rå data fra Modellen og foretager den grafiske formatering af disse data. Systemets interface til interaktion med brugeren kan også ligge i View'et.

Admin er udvidet med et interface til databasen og en admUser klasse. admUser klassen er tilføjet for at give løsningen en klar objektbaseret løsning. Vi arbejder således på et "bruger-objekt" i stedet for brugerdata. Det letter overskueligheden og simplificerer dataflowet.

Grænseflader til administratoren ligger i viewklassen. Grænsefladen til databasen ligger i en database interfaceklasse, se afsnit 4.2.

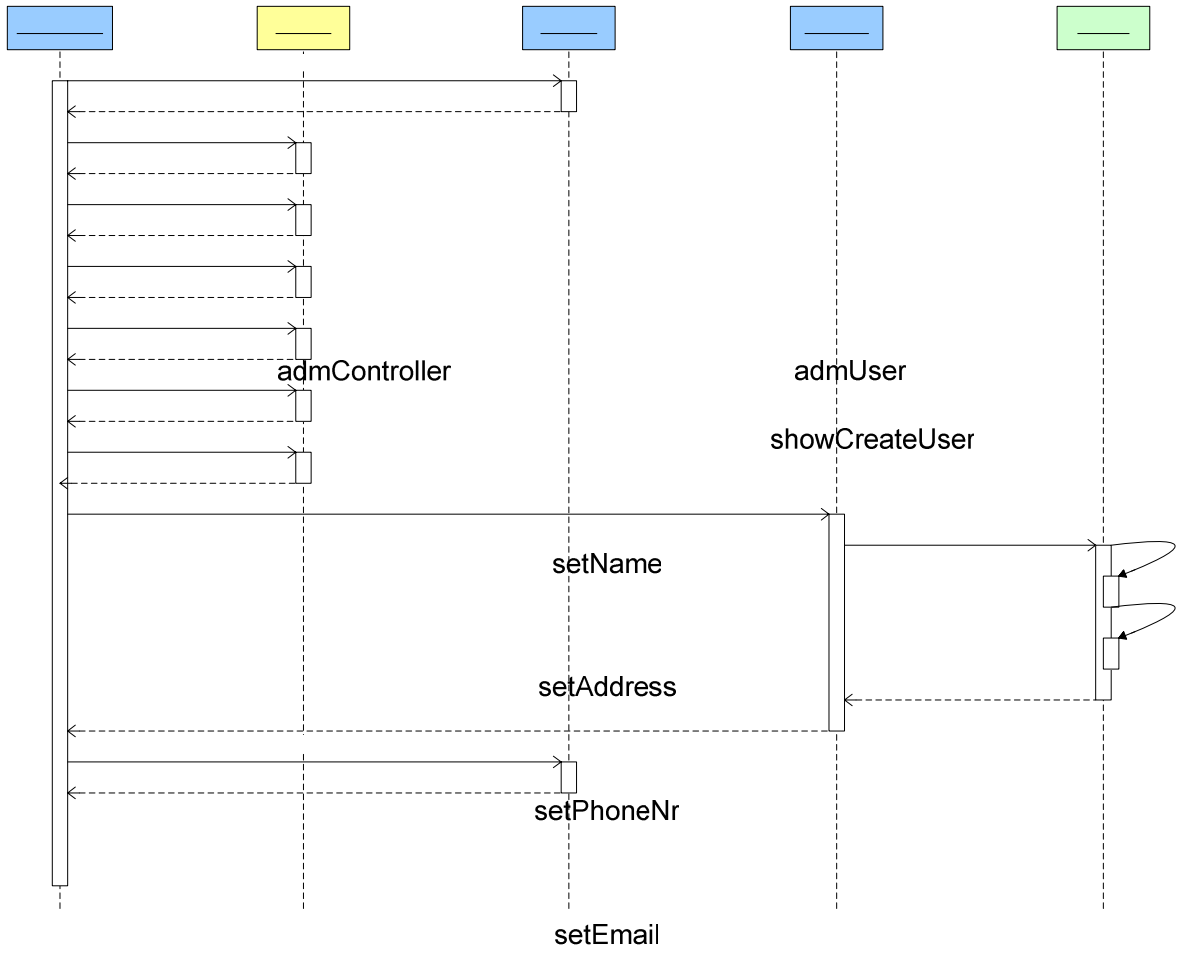
Klasserne beskrives i dette klassediagram.



Figur 5: Admin Klassediagram

På klassediagrammet ses MVC klasserne, userobjektklassen og database interfacet.

Kommunikationen mellem klasserne beskrives i en række sekvensdiagrammer. Her vises "solskinsscenariet" for use case 1. Bilag 1 indeholder sekvensdiagrammer for use case 1-3.



Figur 6: Admin Sekvensdiagram

Login til administration

Når administrationsprogrammet startes op, vil der blive spurgt efter brugernavn og password. De indtastede værdier sammenlignes med det angivne brugernavn og password i koden. Der gives adgang hvis de er ens. Det er ikke muligt for kunden at angive et brugernavn og password til programmet, ligesom der heller ikke er mulighed for at oprette flere administrator-logins. Denne måde er valgt for at frigive ressourcer til arbejde på de andre funktioner.

Hvis der på et senere tidspunkt ønskes mulighed for at kunden selv kan ændre disse værdier og evt. tilføje flere logins, er det oplagt at oprette en ekstra tabel i databasen, der indeholder brugernavne og passwords.

3.2 Secured

Denne del af programmet varetager følgende use cases.

Use case 4: Godkendelse af bruger

Use casen aktiveres af brugeren. Interaktionen til brugeren foregår gennem DV9802 konsollen.

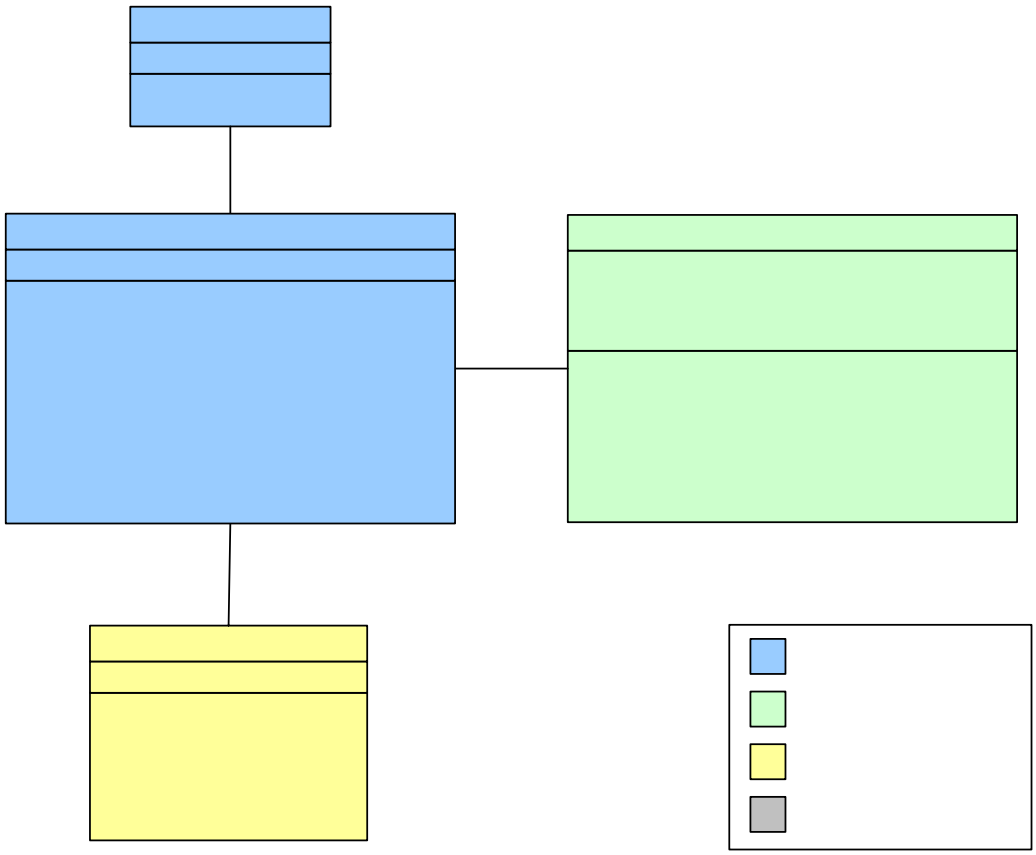
Deamonprocessen har en grænseflade til SQL databasen og en grænseflade til DV9802.

Som overordnet struktur for deamonprocessen har vi igen valgt MVC modellen. Deamonprocessen er udvidet med et HDLC interface og et databaseinterface. Fordelene ved MVC er, som nævnt, fragmentering og høj modularitet.

Vi har valgt at fjerne View-delen fra MVC. Dette valg bygger på, at det ikke er nødvendigt med en separat viewklasse for deamonprocessen. Al udskrift på DV9802 konsollen foregår gennem HDLC interfacet, og skal derfor igennem Modellen. Viewklassen ville bestå af funktioner der alle blot skal kalde tilsvarende funktioner i Modellen. Derfor kan vi undlade viewklassen.

Valget stemmer overens med vores kvalitetsfaktorer fra kravspecifikationen, hvor der er lagt vægt på effektivitet. Det vil nedsætte effektiviteten at indsætte et viewmodul fordi den kun vil tilføje unødvendige funktionskald.

Klasserne beskrives i dette klassediagram.



Figur 7: Secured Klassediagram

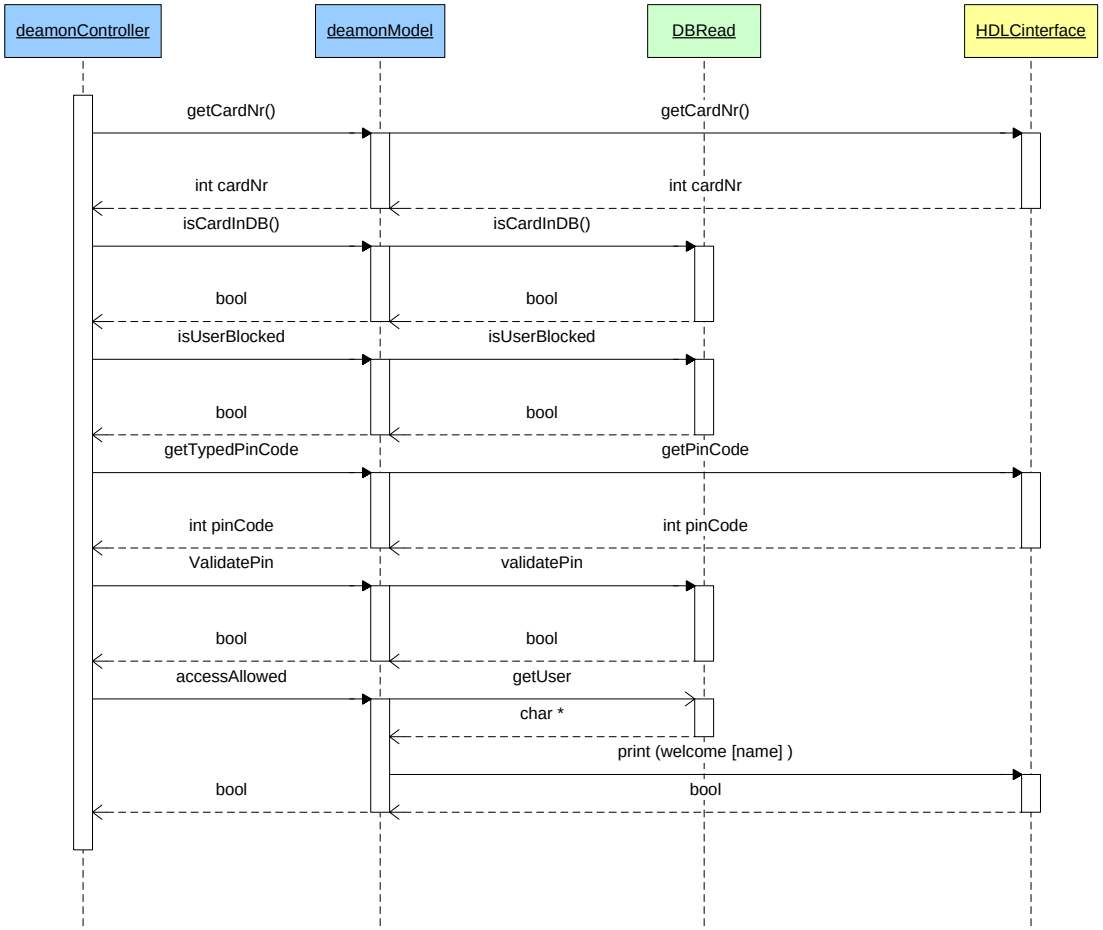
deamonController

-cardNr : int

+idleLoop() : void

Kommunikationen til databasen foregår igennem databaseinterfaceklassen, DBRead. Se afsnit 4.2.

Den interne kommunikation mellem klasserne er beskrevet vha. sekvensdiagrammer. "Solskinsscenariet" for use case 4 er vist her, undtagelsesscenarierne er vedlagt i bilag 2.



Figur 8: Deamon Sekvensdiagram "Solskinsscenariet"

Kapitel

4

4. Eksterne grænseflader

Vi har forsøgt at minimere de eksterne grænseflader. Det er gjort ved at isolere grænsefladerne i interfaceklasser. Dette afsnit behandler de eksterne ressourcer og gennemgår interfacene til dem.

4.1 Eksterne ressourcer

Der er to eksterne ressourcer: DV9802 konsollen og SQL databasen. DV9802 konsollen er beskrevet i projektoplægget. Vi kommunikerer med DV9802 konsollen igennem en HDLC lignende protokol, se bilag 3.

Vi har valgt at bruge en SQL database til at gemme brugerdata. Alternativt kunne man vælge at gemme data i en fil på disken. Fordelene ved at anvende en database er:

1. Vi skal ikke bekymre os om problemer med at læse og skrive samtidigt.
2. Vi slipper for at tænke på backup og persistens af data.
3. Det giver en række muligheder for udvidelser.

Da vi i kravspecifikationen har et webinterface som en udvidelsesmulighed, er det en stor fordel at anvende en database. Der findes utallige webinterfaces til en SQL database.

Ulemper ved at anvende en SQL database:

1. Unødigt at kræve en "full-blown" databaseserver.
2. Unødigt kompliceret kode.

Det er et stort krav at kunden har en SQL database kørende. Vi har en løsning på dette problem, men vi har ingen planer om at implementere den (se bilag 4). At arbejde med en database i stedet for en simpel fil komplicerer koden. Vi har valgt at låne et library der afhjælper problemet.

SQL databasen kontaktes igennem en ODBC driver. Der anvendes DTL[2] til kommunikationen. DTL giver en række handles til databasen. Da ingen af os kender DTL bliver det først beskrevet i implementeringsdokumentationen. DTL inkluderes fra DBInterface, der beskrives i afsnit 4.2.

Users	
PK	**cardNr**
	Name **Address** **email** **phone** **pincode** **locked**

Figur 9: database table

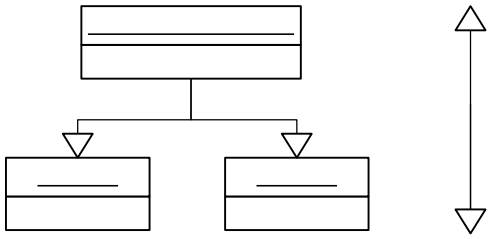
Databasen har følgende tabel vist på figur 9.

Information om ODBC string, username og password til databasen er hardcoded i DBInterface klassen. For at ændre disse data skal koden altså genkompileres.

[2] Se http://dtemplatelib.sourceforge.net/

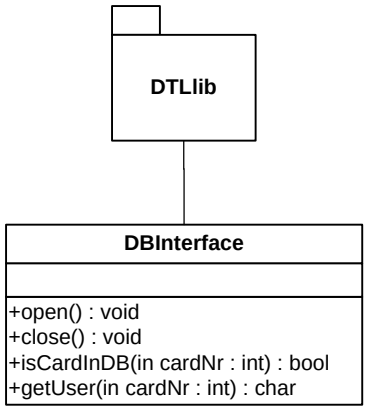
4.2 Database Interfaces

Der indgår to interfaces til databasen. Et for deamonprocessen og et for administrationsprocessen. Disse to interfaces ligner hinanden og indeholder flere ens funktioner. Det er derfor oplagt at anvende nedarvning. Derved opnår vi genbrugelig og ægte objektorienteret kode. En anden mulighed er at lave to forskellige klasser, der indeholder stortset den samme kode. Vi har valgt den objektorienterede løsning, tildels fordi vi ønsker, at bruge så meget fra PRG2 kurset som muligt.



Figur 10: Nedarvningshierarki

De to klasser DBRead og DBStore er specialiserede klasser af den abstrakte klasse DBInterface. DBInterface beskrives ud fra dette klassediagram.



Figur 11: DBInterface klassediagram

abstract : DBInterface

DBRead

DBStore

DBRead og DBStore er beskrevet i

Figur 7: Secured Klassediagram (DBRead)

Figur 5: Admin Klassediagram (DBStore)

Funktionsbeskrivelserne af DBRead og DBStore findes i bilag 5.

4.3 DV9802 Interface

Secured bruger HDLCInterface til at kommunikere med DV9802 konsollen. Interfacet stiller funktioner til rådighed for deamonModel, som må kalde disse funktioner for at udføre opgaver på DV9802 konsollen. Funktionerne er beskrevet i bilag 5.

Kapitel

5

5. Funktionsbeskrivelser

Dette kapitel beskriver kort nogle af hovedfunktionerne i klasserne. For bedre overblik er kapitlet inddelt i funktioner for Ssecured og funktioner for Admin. Bilag 5 gennemgår alle funktionerne.

5.1 Admin

admController

```
/* funktionsnavn: void createUser()
 * param:
 * returtype: void
 * beskrivelse:
 * Funktionen skal starte indtastning af ny bruger ved at kalde
 * admView og oprette vedkommende i databasen ved at kalde admModel.
*/
```

admModel

```
/* funktionsnavn: bool insertUser( admUser * tempUser )
 * param:
 * admUser * tempUser, pointer til det user-objekt indeholdende
 * brugerdata, der skal oprettes i databasen.
 * returtype: bool, angiver om operationen blev udført.
 * beskrivelse:
 * Funktionen skal modtage et user-objekt og bruge det til at oprette
 * den pågældende post i databasen ved at kalde DBStore.
*/


/* funktionsnavn: admUser * getUser( int cardNr )
 * param: int cardNr, det kortnummer, der skal udtrækkes data fra.
 * returtype: admUser, pointer til det samlede user-objekt, der
 * returneres.
 * beskrivelse:
 * Funktionen skal modtage en forespørgsel på en bruger i form af et
 * kortnummer og returnere et helt user-objekt ved at kalde DBStore.
*/
```

admView

```
/* funktionsnavn: void showUser( admUser * user, int x, int y )
 * param:
 * admUser * user, pointer til det user-objekt, der skal skrives ud
 * på skærmen.
 * int x, udskrivningens x-koordinat.
 * Int y, udskrivningens y-koordinat.
 * returtype: void
 * beskrivelse:
```

```
 * Funktionen skal modtage et user-objekt og et (x,y) koordinatsæt og
 * printe brugeren ud på skærmen på den angivne position.
*/
```

DBStore

```
/* funktionsnavn: bool insertUser( admUser * tempUser )
 * param: admUser * tempUser, pointer til det user-objekt, der skal
 * oprettes.
 * returtype: bool, returnerer sandt, hvis operationen bliver
 * fuldført.
 * beskrivelse:
 * Funktionen skal modtage et user-objekt og bruge dets data til at
 * oprette en bruger i databasen.
*/

/* funktionsnavn: admUser * getUser( int cardNr )
 * param: int cardNr, det kortnummer, der skal udtrækkes data fra.
 * returtype: admUser *, pointer til det samlede user-objekt, der
 * returneres.
 * beskrivelse:
 * Funktionen skal modtage en forespørgsel på en bruger i form af et
 * kortnummer og returnere et helt user-objekt.
*/
```

5.2 Secured

deamonController

```
/* Funktionsnavn: void validateUser ()
 * Parametre: ingen
 * Returtype: void
 * Beskrivelse: så snart en bruger indlæser et kort, starter
 * validateUser funktionen, som styrer valideringsprocessen.
*/
```

deamonModel

```
/* Funktionsnavn: bool validatePin ( int cardNr, int pinCode )
 * Parametre:
 * int cardNr, nummeret på det indlæste kort.
 * int pinCode, den indtastede PIN kode.
 * Returtype: bool
 * Beskrivelse: funktionen validere kortnummeret og dets tilhørende
 * pinkode.
*/
```

HDLCInterface

```
/* Funktionsnavn: bool print ( char info )
 * Parametre: char info, tekststreng, som printes ud på det
 * alfanumeriske display.
 * Returtype: bool
 * Beskrivelse: Funktionens opgave er at printe info på det
 * alfanumeriske display.
*/
```

Kapitel

6

6. Bilagsoversigt

Bilag 1: Sekvensdiagrammer for Administrationsprocess.

Bilag 2: Sekvensdiagrammer for Deamonprocess.

Bilag 3: HDLC protocol beskrivelse.

Bilag 4: SQL/FIL løsning.

Bilag 5: Funktionsbeskrivelser