Tin Can Skype

RB-MRO3 - Gruppe 3

Uddannelse og semester:

Robotteknologi - 3. semester

Afleveringsdato:

18. December 2015

Vejleder:

Ib Refer (refer@mmmi.sdu.dk)

Gruppemedlemmer:

Anders Ellinge (aelli14@student.sdu.dk)
Anders Fredensborg Rasmussen (andra14@student.sdu.dk)
Daniel Holst Hviid (dahvi14@student.sdu.dk)
Mathias Elbæk Gregersen (magre14@student.sdu.dk)
Rasmus Skjerning Nielsen (rasni14@student.sdu.dk)
René Tidemand Haagensen (rehaa14@student.sdu.dk)
Sarah Darmer Rasmussen (srasm14@student.sdu.dk)

Det Tekniske Fakultet
Syddansk Universitet

1 Abstract

2 Forord

Denne rapport er udarbejdet af gruppe 3, på andet semester på Civilingenør i Robotteknologi på Syddansk Universitet. Rapporten er blevet skrevet i forbindelse med dette semesters projekt og beskriver hvordan denne gruppe har valgt at løse opgaverne i det valgte projekt, "Tin Can Skype", som er et chatprogram, der bruger DTMF-toner og indeholder bla. et simpelt log-in og historik system.
Formålet med denne rapport er, at læseren skal være i stand til at læse og forstå projektet ved blot at have grundlæggende viden om C++ og datakommunikation, og ved at læse rapporten.

I forbindelse med dette projekt, blev følgende udstyr stillet til rådighed:

- To mikrofoner
- To højtalere

Indhold

3 Indledning

3.1 Projektbeskrivelse

I dette projekt er to højtalere og to mikrofoner blevet stillet til rådighed. Formålet med dette projekt er, at kunne sende data vha. DTMF-toner.
Det valgte projekt er et chatprogram, der udvikles i C++, og skal have de primære funktioner:

- Overførsel af tekst.
- Log-in funktioner.
- Historik af chat-samtale.

Desuden er disse sekundære funktioner blevet overvejet:

- Filoverførsel
- Gruppe chat
- Spil
- Redigering af tidligere beskeder
- Video streamings funktioner
- Humørikoner

Herudover er der desuden blevet overvejet at bruge en tredje computer, som kan bruges som en server. Her vil mindst to computere altså være i stand til at kommunikere med hinanden vha. DTMF-toner.

3.1.1 Krav til produktet

Følgende krav blev stillet til projektet:

- Bærbare computere skal kommunikere med hinanden, eller evt. et embedded system, ved udveksling af lyd
- Der skal anvendes DTMF toner, og der skal designes en kommunikationsprotokol
- Der skal udvikles en distribueret applikation i C++
- Der skal anvendes en lagdelt softwarearkitektur
- Arkitekturen kunne være client/server med f.eks. tykke klienter

For at fuldføre dette projekt skal der anvendes to computere som skal være i stand til at kommunikere med hinanden ved hjælp af lyd i form af DTMF-toner. Derudover skal dette programmers i C++, her bruges klasser.

3.1.2 Metodebeskrivelse

Vi har i dette projekt valgt at benytte SCRUM, da alle gruppe medlemmer således er i stand til at arbejde med den metode der passer dem bedst. Vi har valgt at bruge brainstorm, som vores primære form for idégenererings-teknik. Desuden prøver gruppen så vidt muligt at beregne alle de ting, der kan beregnes på forhånd.

3.1.3 Afgrænsning

Her ses de emner, som gruppen har overvejet at arbejde med. De primære funktioner er de funktioner som skal løses først, mens de sekundære løses efter tidsbegrænsning.
Primære funktioner:

- Overførsel af tekst
  - Protokol
  - Karakter definition
  - Størrelse
- Historik
  - Tidspunkt
  - Størrelse
- Log-in

Sekundære funktioner:

- Fil-overførsel
  - Protokol
  - Queue
- Gruppe chat
  - Protokol
- Spil database
  - Protokol
  - Funktion
- Rediger tidligere beskeder
  - Protokol
  - Funktion
- Stream funktion
  - Protokol
  - Funktion

- Humørikoner

  Char def.

  Database

- GUI

  Agil

- Sky "server"

  Funktion

3.2 Workload (product backlog)

Der laves en product backlog i stedet for en tidplan (se figur 1).

Requirement	Status	Priority	Estimate (Hr)	
Overførsel af tekst	Not started	1	70	
Log-in	Not started	1	40	
Historik	Not started	1	40	
Fil-overførsel	Not started	2	120	
Gui	Not started	2	70	
Rediger tidligere besked	Not started	3	40	
Gruppe chat	Not started	3	40	
Streaming	Not started	4	120	
Cloud/server (tredje computer)	Not started	4	100	
Smileys	Not started	4	30	
Spil	Not started	4	100	
		I alt	770	

Figur 1: Product backlog

En product backlog er et værktøj inden for metoden SCRUM, som viser hvor langt tid en opgaver tager i mandetimer og hvilken status opgaven har (Not started, In process og Finished).

4 Det Fysiske Lag

4.1 Fysisk Lag Ind

dsf
dkdk idkfgfdggfd

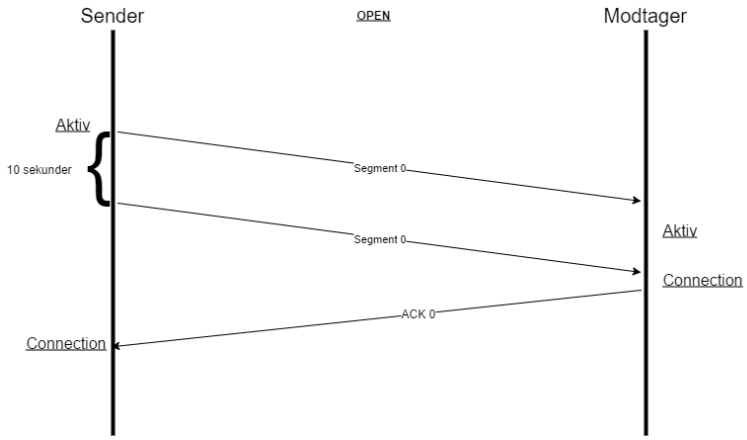
4.1.1 Teori

Send-and-Wait protokol

Stop-and-Wait er et special tilfælde af Go-Back-N. Stop-and-Wait har to sekvensnumre, mens Go-Back-N har flere
Ved hjælp af sekvensnummerering af pakkerne, er modtageren i stand til at konstatere om den modtagne pakke er den korrekte, eller om den har et forkert nummer i forhold til rækkefølgen.
Automatic repeat request (ARQ) benyttes når mistede eller fejlbehæftede pakker skal retransmitteres. Pakker retransmitteres hvis en ny ACK ikke er modtaget inden timeren udløber.

Open

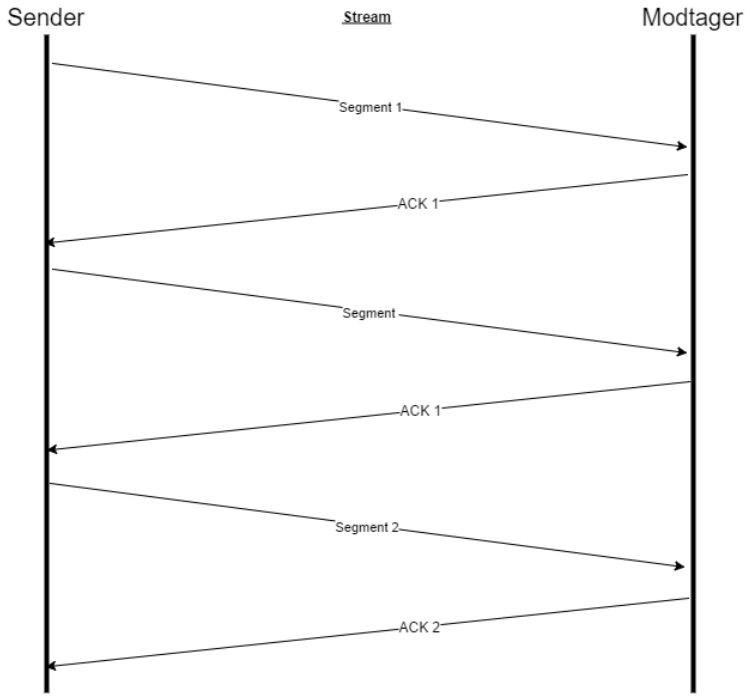
Udføres når både sender og modtager er aktive.
Som der ses i figur 2, sendes Segment 0, når Sender er aktiv. Hvis Modtager er aktiv oprettes der forbindelse og Modtager sender ACK 0 (acknowledgement 0) og Sender ved at der er forbindelse. Hvis Modtager derimod ikke er aktiv vil Sender ikke modtage ACK 0, og Sender vil derfor vente 10 sekunder før igen at sende Segment 0. Dette vil Sender gøre op til tre gange, hvis nødvendigt.



Figur 2: Open

Stream

Stream delen, som kan ses på figur 3, er den del hvor data sendes.
Her sender Sender først Segment 1. Hvis dette er blevet korrekt modtaget vil Modtager sende ACK 1. Dette vil forstætte indtil Close.

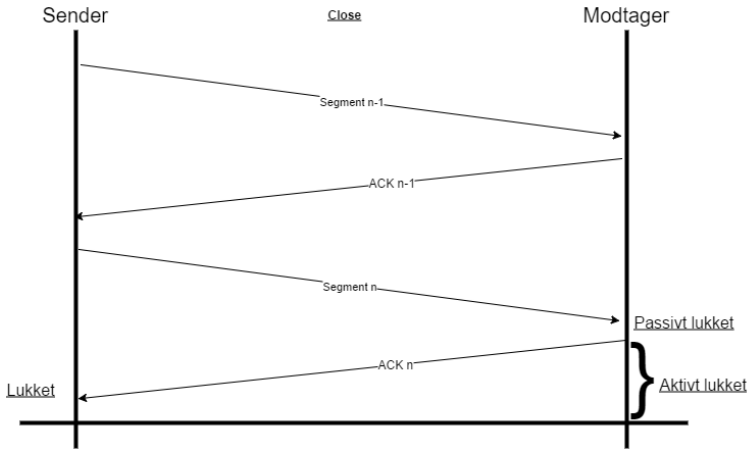


Figur 3: Stream

Close

Close delen kan ses på figur 4.

I denne del gør Sender Modtager opmærksom på at der ikke sendes mere.



Figur 4: Close

Flag

I projekt koden er der benyttet tre flag til at fortælle om beskeden er en probe, accept, eller sidste besked. Der benyttes 3 bit, til at definere flaget. 001 er probe, 010 er accept og 100 er last.

5 Data Link Laget

5.1 Teori

CRC

CRC står for Cyklisk Redundant Check, og bruges til fejl-detektering. Formålet med fejl-detektering er at gøre det muligt for modtageren at afgøre om et datagram, sendt gennem en støjet kanal, er blevet beskadiget. For at gøre dette konstruer senderen en checksum og føjer denne til datagrammet. Modtageren er i stand til at bruge CRC til at beregne checksum af det modtagne datagram og sammenligne denne med den vedlagte checksum for at se, om datagrammet er blevet modtaget korrekt.[4]

CRC er baseret på polynomiel aritmetik, primært beregning af resten når et polynomium divideres med et andet. Addition og subtraktion udføres ved hjælp af modulo-2.

Modulo-2 Binary Division

Modulo er resten af en division mellem to tal, og er oftest udtrykt som "%".

I modulo defineres nye oprationer, som ofte ligner Boolske logiske operationer, heriblandt XOR, som bruges til addition, og AND, som bruges til mulitplikation. Bitvist er er modulo-2 det samme som XOR, og her noteres modulo-2 additions operationen med en cirkel med et plus, f.eks.:

$$1 \oplus 0 = 1$$

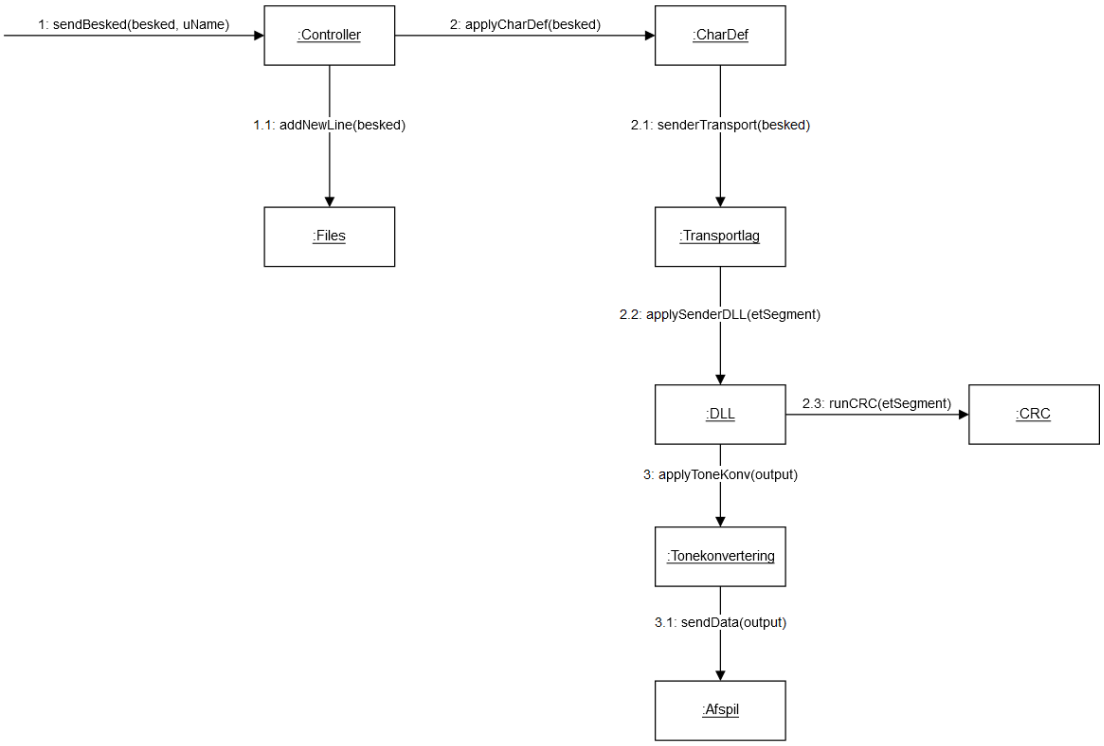
Stuffing

Bitstuffing er nødvendig for datagrammer der ikke overholder de størrelsesmæssige krav. I dette tilfælde tilføres ekstra bits, uden betydning, til datagrammet, indtil dette opfylder de nødvendige størrelsesmæssige krav.

6 Kontrolfunktioner

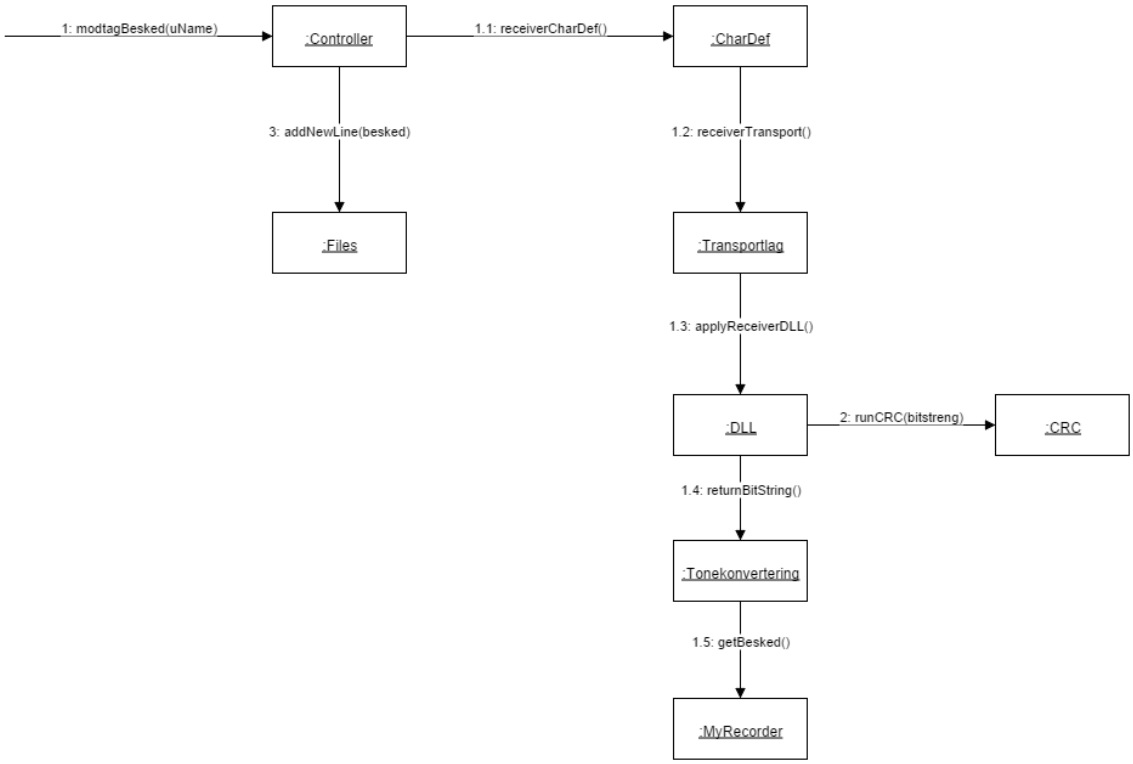
Controller klassen

Der blev valgt at samle programmets vigtigste funktioner i klassen `Controller`. Dette blev gjort for at user interface kun skulle i kontakt med én klasse, og fordi det ville blive nemmere i en senere iteration at implementere et Grafisk User Interface (GUI).
En af `Controller` klassens metoder er `sendBesked(besked, uName)`. Samarbejdsdiagrammet for `sendBesked()` er vist i figur 5.



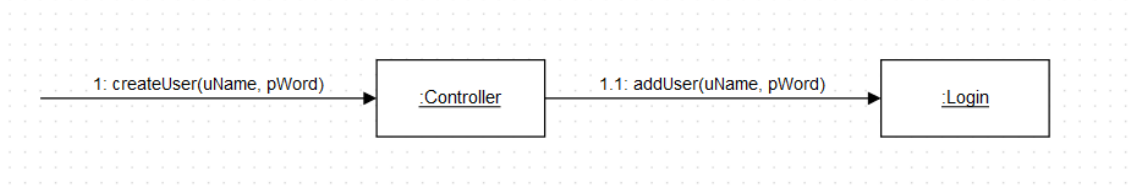
Figur 5: `sendBesked()`

Denne metode tager beskeden der skal sendes, samt navnet på den der har sendt den og samler det i en besked. Denne besked bliver sendt videre til `Chardefinition` klassen, som laver teksten om til en binær streng. Den bliver efterfølgende sendt til transportlaget, der kan dele beskeden op i mindre segmenter og sørge for at sendingerne foregår som de skal. Pakkerne vil kommere videre til Data Link Laget, hvor der vil blive tilføjet CRC og stuffing til bitstrengen. I `Tonekonvertering` bliver den binære streng omdannet til tal mellem 0 og 15, som er det antal toner der er til rådighed ved DTMF. Til sidst bliver tonerne afspillet med klassen `Afspil`. Når en besked er sendt, bliver den gemt i en fil med klassen `Files`.
Klassen har desuden en metode, der hedder `modtagBesked(uName)`. Samarbejdsdiagrammet for `modtagBesked()` er vist i figur 6.
Denne går igennem de samme klasser som `sendBesked`, dog i stedet for at sende en besked afsted, sendes der en request om at modtage en besked. I klassen `MyRecorder` modtages beskeden, som



Figur 6: `modtagBesked()`

returneres som toner af tal mellem 0 og 15 til `Tonekonvertering`'en, der omdanner tonerne til en bitstreng, som returneres til Data link laget. Ved `DLL` tjekkes for CRC og stuffing fjernes inden det returneres til transportlaget, hvor beskeden sættes sammen igen, hvis beskeden var delt op i segmenter. Den kommer retur til `CharDef` og bliver lavet fra binær streng til karakterer, som derefter returneres til `Controller`, der viser beskeden på skærmen og samtidig bruger `Files` til at gemme beskeden i en historik. Til dette bruges `uName` i metoden for at gemme i den rigtige historik.
`testLogin(uName, pWord)` er metoden der kan teste om et brugernavn og password er korrekt. Dette gør den ved at bruge `Login` klassen.
`createUser(uName, pWord)` bruger igen `Login` klassen, denne gang til at oprette en bruger. Samarbejddiagrammet er vist i figur 7.



Figur 7: `createUser()`

Der er også metoder der kan vise historikken. De bruger alle `Files` klassen og kan enten vise

hele historikken, sidste besked eller en der kan definere hvor mange linjer der skal hentes og vises.

Login klassen

Klassen `Login` bruges til at oprette og teste brugernavn og password. Metoden `addUser(uName, Pword)` opretter nye brugere og gemmer brugernavn og password i et txt dokument.
Klassen har også metoden `testLogin(uName, pWord)`, som bruger metoden `validateLogin(uName, pWord)`. Den modtager et brugernavn og login, som bliver lavet om til en string. Denne string bliver efterfølgende sammenlignet med alle de brugernavne og passwords der er gemt i txt filen. Hvis der ikke findes et match bliver der returneret true og brugeren logges på.

Files klassen

`Files` klassen bruges til at gemme historik, når der bliver skrevet til hinanden med chat programmet. Files virker ved at når der oprettes et objekt at klassen, med en parameter, som er brugernavnet, bliver der oprettet et txt dokument med brugerens navn, som historikken kan gemmes i.
Funktionen `addNewLine(besked)` tilføjer en besked til tekstdokumentet. `updateVector()` bruges til at lave en vektor af linjerne fra historikken, som efterfølgende evt. kan printes ud på en skærm så brugeren kan se indholdet. Der er en metode `clearText()`, der kan slette alt fra historikken. Der er også metoden `printVector()`, der kan printe hver linje ud, som er i vektoren skabt med `updateVector()`. `printLatest()` printer den sidste besked ud og `printLines(startN, endM)` kan printe de valgte linjer ud fra linje n til m. Til sidst er der en metoden flipVector(), den bruges til vende vectoren, så fx den sidste modtagne besked kommer til at ligge øverst i vektoren, altså på plads 0.

CharDefinition klassen

`CharDefinition` klassens opgave er at omdanne forskellige karakterer om til binærer tal og tilbage igen. Hvis der oprettes et objekt af klassen, bliver der skabt en string af alle de tal, karakterer og andre tegn der tænkes at skulle kunne sendes. Den string kan bruges i metoderne. Der er en metode som hedder `applyCharDef(input)`, der bruger en anden metode `charToBinary`, som kan omdanne karakterer i en besked til binær, inden den sendes videre til Tranportlaget.
`charToBinary(input)` metoden tager inputtet som er en string af karakterer og laver det om til en binær string. Det gør den ved at kigge på de karakterer, der skal sendes og sammenligne dem med de definerede karakterer og tegn. Hvis der findes et match, bruges nummeret på placeringen af karakteren i den definerede string, som den binære værdi. Den binære værdi bliver 8 bits lang. Fx karakteren b, som har placeringen 12, får binær værdien 00001100.
`receiverCharDef()` er metoden der bruges når der returneres beskeder fra transportlaget og laver beskeden om fra binær til karakterer. Hvis beskeden der modtages er en fejlbesked sende den videre og vises på skærmen. Ellers bruges metoden `binaryToChar(messageReceived)` der omdanner de otte bits om til et decimaltal, som igen kan bruges til at finde placeringen af karakteren, der er modtaget og derefter sætte det på en string. Det gøres indtil hele beskeden er omdannet.

UI source

Til chat programmet er der valgt at lave et simpelt user interface. Det blev lavet ved at bruge konsolvinduet i Visual Studio. Et flowdiagram af systemet er vist på figur 8. Programmet er bygget op ved at der gives nogle valgmuligheder. På første skærm kan der vælges mellem login eller opret bruger. Der vælges ved at indtaste et tal, her 1 eller 2. Tallet sammenlignes med de muligheder der er og programmet går til næste skærm. Når der oprettes ny bruger, indtastes brugernavn og password som efterfølgende bruges i `Controller` klassen i metoden `createUser(uName, pWord)`. Der vil efterfølgende gås videre til loginskærmen, hvor brugernavn og password indtastes igen og bruges i metoden `testLogin(uName, pWord)`. Hvis det indtastede er forkert kan der prøves igen ellers kommer interface skærmen. Her er mulighederne: Send besked, modtag besked, se historik og log ud. Ved send besked skærmen kan der skrives en besked, som efterfølgende sendes med metoden `sendBesked(besked, uName)`. Ved modtagBesked skærmen afventes der en besked. Den modtages med metoden `modtagBesked()` og bliver efterfølgende vist på skærmen. Hvis der vælges historik, kommer en skærm med 3 muligheder igen. Her kan der vælges at se hele historikken eller seneste besked som vises med metoderne `getHeleHistory()` og `getSenesteHistory()`. Den sidste valgmulighed er retur til interface skærmen.

7 Konklusion

8 Perspektivering

9 Litteraturliste

9.1 Bøger

[1] John W. Dower *Readings compiled for History 21.479.* 1991.

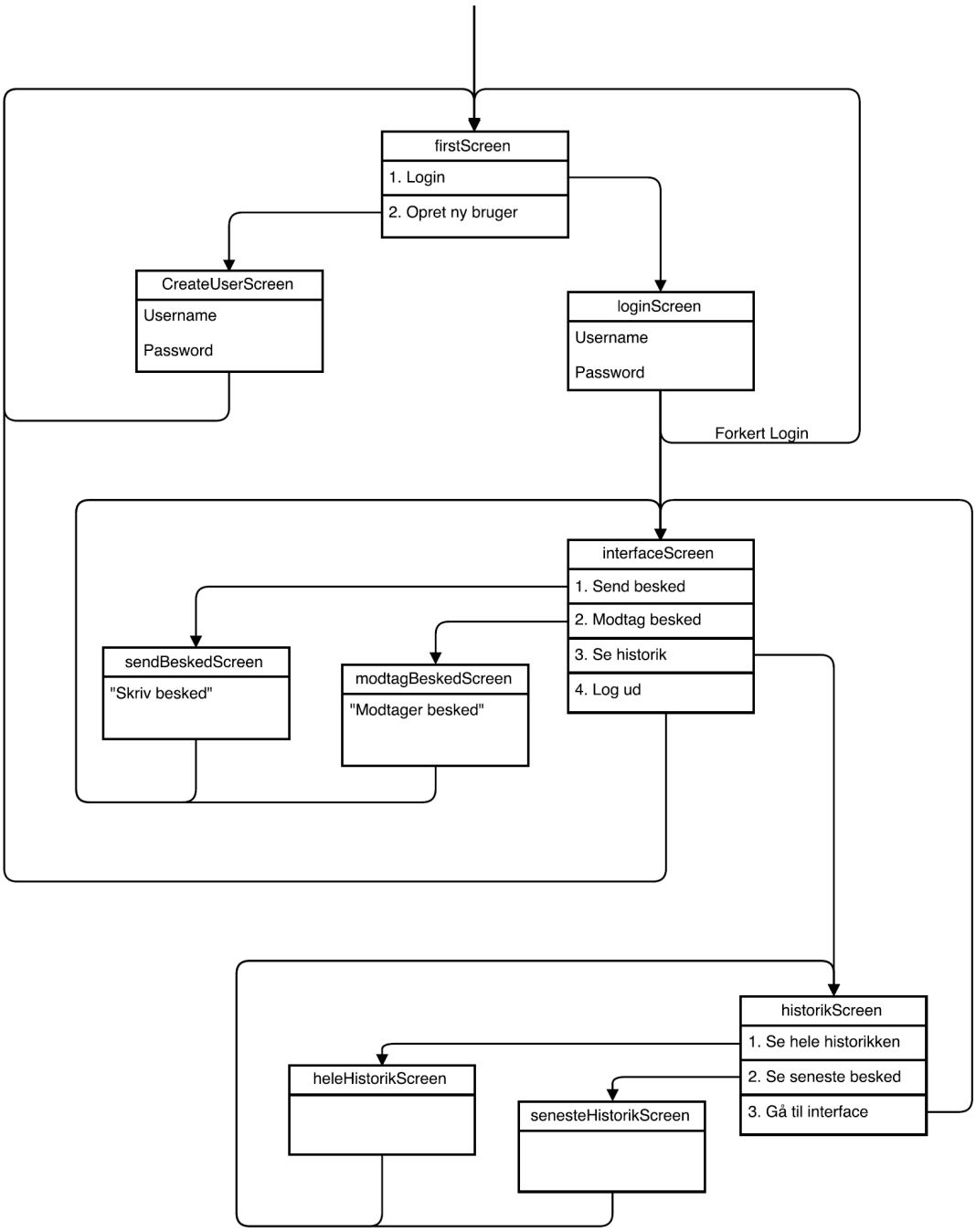
[2] The Japan Reader *Imperial Japan 1800-1945* 1973: Random House, N.Y.

9.2 Hjemmesider

[3] http://einstein.informatik.uni-oldenburg.de/papers/CRC-BitfilterEng.pdf

[4] http://www.ross.net/crc/crcpaper.html

[5] http://www.hackersdelight.org/crc.pdf



Figur 8: `UI flowdiagram`