SW01
Ubiquitious Doorman

af

6. gruppe:

Steffen Denize	Cand. IT.	s_denize@hotmail.com
Quang Lam	BSc Datalogi	qlam@mip.sdu.dk
Tino Didriksen	BSc Datalogi	t@pjj.cc
Brian M. Pedersen	Cand. IT.	brian@moellegaard.dk

17. Maj, 2004

21 sider

Table of Contents

Reflection

Projektet ligger i sin definition opad beskrivelsen af Doorman-delen af Ubiquitious Dream Team som beskrevet på de udleverede ark fra Mærsk Instituttet for Produktions Teknologi. I vores version af Ubiquitous Doorman (UbiDoor) har vi lagt vægt på et hændelsesbaseret system, frem for et informationssystem hvor brugeren selv skal opsøge informationerne. Det vil sige at UbiDoor af sig selv viderebringer informationer til slutbrugeren hvis de pårører denne. UbiDoor understøtter dog at informationer kan hentes fra systemet af tilmeldte brugere. Brugere opdeles i vores system, jvf. Visionen, i forskellige grupper.

Denne taksonomi af brugerne giver anledning til at system, udover at være den aktive informationsformidler, også er den passive informationsformidler, i.e. at gæster kan indhente informationer der opbevares i systemet.

I forbindelse med vores Use Cases var er i starten af projektet en overgang hvor UbiDoor blev beskrevet som værende et passivt informationssystem. Efter revision af Use Cases blev UbiDoor redefineret til at være et aktivt, hændelsesbaseret system.

Der har været interne problemstillinger i gruppen der til tider har hindret en normal fremdrift. Enkelte gruppemedlemmer har til tider ikke fulgt den planlagte projekttidsplan hvilket har bevirket, at resten af gruppen måtte bruge unødig tid på at bringe pågældende up-to-date. Yderligere har der så været frister der ikke har været overhold, hvilket igen har bevirket at de resterende gruppemedlemmer har måttet påtage sig ansvaret for færdiggørelsen af opgaver. Dette har igen indvirket at progressionen i projektet ikke har været optimalt.

Vores tilgang til kodningen af systemet har ikke været baseret på en færdig designmodel. Derimod har vi kodet udvalgte Use Cases og set dem i forhold til gennemgået læsning og derefter identificeret Patterns der passer på vores kode. Dette er nok lidt omvendt, men det virker meget godt. Også fordi at vi potentielt senere vil kigge mere på Patterns først og kodning derefter; altså en proces. Vi har implementeret ca. en fjerdedel af de klasser og den funktionalitet der indgår i vores design. Det er primært ukompliceret kode, med brug af ArrayList og Date.

Gruppens sammensætning har gjort at vi har anskuet problemet fra flere forskellige sider. Dette har været en stor fordel.

System Vision

Ubiquitous Doorman er et centralt styret system der holder styr på personer i bygningen, møder, aftaler, kalendere, og meget andet. UbiDoor er underbygningen for de andre dele i Ubiquitous Dream Team. Det er primært til for at lette arbejdsbyrden for administrationen og sekretærerne ved at gøre den information de normalt formidler lettere tilgængelig. Dog kan alle personer tilknyttet systemet kan indlægge egne relevante informationer. Det ville ikke være utænkeligt at systemet udvikler sig til et nøglepunkt i dagsordenen for mange ikke-privilegerede personer. Studerende kan bruge det til at aftale gruppemøder, lærere kan oplyse om aflysning eller flytning af timer, og alle kan sende beskeder til hinanden.

Systemet er ansvarlig for at holde alle personer ajour med informationer der er relevante for dem. Om en given information er relevant er det mest personens eget ansvar at gøre systemet opmærksom på idet personerne kan tilmelde sig informationsgrupper, men systemet skal også delvist kunne (kritisk) forudsige at en person har brug for den givne information. Yderligere er der serviceelementer der tillader at udvalgte brugere af systemet at automatisk aktivere/kontrollere undervisningselementer i lokaler.

UbiDoor er i første omgang indskrænket til Mærsk Instituttets bygning for afprøvning og stabilisering før det eventuelt kan bruges på SDU et al. Af den grund behøver systemet ikke at være absolut solidt fra første implementation. Der skal dog tænkes på at systemet bør kunne arbejde sammen med eksisterende systemer, og derfor skal den første implementation være gennemtænkt med hensyn til mulighed for udbyggelse og interoperabilitet. Og, idet systemet er stærkt centraliseret, skal det kunne modstå diverse ustabiliteter. Altså ikke noget med at miste skemaet for semesteret bare fordi systemet genstartes.

Sikkerhedsmæssigt deles brugere op i 3 overgrupper: Fastansatte, studerende, og gæster. Kun fastansatte og studerende har mulighed for at lægge information ind i systemet, og disse grupper skal bære deres barcode-bearer (BCB) device eller logge ind med navn og password. Gæster har adgang til at se visse informationer, såsom at lokalisere en person. Fastansatte kan ydermere blive tildelt visse privilegier, for eksempel at kunne tilføje og slette brugere eller at kunne oprette et fag.

Det hele kører over et netværk i bygningen. Der er den centrale server, og til denne er der tilsluttet en række klienter. En klient kan være alt fra en PDA, over diverse computere i

bygningen, til de fysiske UbiDoor konsoller og sensorer. Derfor skal der udvikles en række protokoller for disse client-server forhold. I sidste ende er de egentlige klienter dog brugerne, så måske det er bedre at sige client-frontend-server forhold. Noget af det fysiske UbiDoor udstyr bliver designet til opgaven (sensorer, BCB-devices), hvor andre dele kan afvikles på normale computere (konsoller, central server). Specielt derfor bliver systemets software udviklet i Java. Valget af Java hjælper også til senere videreudvikling og vedligeholdelse.

Use Case Model

EnterBuilding

En bruger ankommer til MIP, UbiDoor registrerer dette.

ExitBuilding

En bruger forlader MIP, UbiDoor registrerer dette.

EnterRoom

En bruger går ind i et lokale, UbiDoor registrerer dette.

ExitRoom

En bruger går ud af et lokale, UbiDoor registrerer dette.

EnterBuildingExpanded

En bruger kommer til indgangen af Mærsk Instituttet og skal ind af en af indgangene. Brugerens ankomst til Mærsk Instituttet bliver registreret. Brugeren får at vide at nogle timer han er tilmeldt er flyttet fra lokale 1 til lokale 2.

LocatePerson

En bruger, StudA, ankommer til Mærsk Instituttet. StudA skal finde en anden bruger, StudB. StudA henvender sig til en UbiDoor. StudA spørger UbiDoor hvor StudB er. UbiDoor fortæller StudA at StudB er i mødelokale 1.

NotWriteMessage

En bruger, StudA, henvender sig til en fysisk UbiDoor. StudA spørger om brugeren StudB er i bygningen, hvilket StudB ikke er. UbiDoor spørger om StudA vil skrive en besked til StudB, men det vil StudA ikke. StudA afslutter.

WriteMessage

En bruger, StudA, henvender sig til en fysisk UbiDoor. StudA skriver en besked til brugeren StudB. UbiDoor afleverer beskeden til StudB hvis StudB er indenfor UbiDoors område, ellers gemmer UbiDoor beskeden og afleverer beskeden når StudB kommer indenfor UbiDoors område.

RegisterMeeting

En bruger finder et møde der har hans interesse. Brugeren tilmelder sig dette møde. UbiDoor registrer dette. UbiDoor spørger brugeren om han vil varsles om mødet. Brugeren vælger at det vil han godt. Brugeren afslutter.

NotifyMeeting

En bruger har tilmeldt sig et møde og valgt at få varsler om dette. UbiDoor registrer at brugeren ikke er i nærheden af mødelokalet hvor mødet afholdes, men han er dog indenfor UbiDoors område. UbiDoor fortæller brugeren at mødet begynder om 5 min.

MoveMeeting

En administrator fortæller UbiDoor at studienævnsmødet er aflyst i morgen med afholdes mandag i stedet for. Alle brugere der sidder i studienævnet bliver fortalt af UbiDoor at mødet i morgen er aflyst, men at det afholdes mandag i stedet for.

RescheduleMeeting

En administrator fortæller UbiDoor at studienævnsmødet er flyttet fra mandag til fredag fremover. Alle brugere der sidder i studienævnet bliver fortalt af UbiDoor, at mødet er blevet flyttet varigt fra mandag til fredag.

CancelMeeting

En administrator fortæller UbiDoor at studienævnsmødet mandag er aflyst pga. sygdom. Alle brugere der sidder i studienævnet bliver fortalt af UbiDoor, at mødet mandag er aflyst.

MoveLecture

En underviser fortæller UbiDoor at øvelsestimerne til hans kursus er flyttet fra onsdag til fredag i denne uge. Når en studerende der er tilmeldt kurset er indenfor UbiDoors område, fortæller UbiDoor den studerende at øvelsestimerne er flyttet til fredag.

CancelLecture

En undervisers undervisning er blevet aflyst. Underviseren fortæller UbiDoor at undervisningen er aflyst i lokale 3 fra 12-14. UbiDoor registrerer dette. En studerende ankommer til MIP og skal have undervisning i lokale 3 fra 12-14. UbiDoor fortæller den studerende af undervisningen er aflyst.

CancelClass

En undervisers kursus er blevet aflyst fordi der var for få tilmeldte. Underviseren fortæller UbiDoor at kurset er aflyst dette semester. UbiDoor registrerer dette. Når en studerende der havde valgt kurset ankommer til MIP, vil UbiDoor fortælle den studerende at kurset er aflyst.

ReverveRoom

En underviser vil reservere et lokale til ekstra undervisning en onsdag kl. 12. Underviseren henvender sig til en UbiDoor. Underviseren finder oversigten over lokaler der er ubenyttede i tidsrummet onsdag 12-14. Underviseren finder lokalet og reserverer det.

InformClass

En underviser finder et foredrag der har relevans for hans undervisning. Underviseren giver UbiDoor informationerne om foredraget og fortæller at de studerende der følger hans undervisning skal have informationerne næste gang de er indenfor UbiDoor. Når en studerende der er tilmeldt undervisningen kommer indenfor UbiDoors område, vil UbiDoor fortælle den studerende om foredraget.

Interesting Use Cases in Casual Format

InterruptWriteMessage

Primært succes scenarie: Vi har 2 brugere, StudA og StudB. StudA spørger UbiDoor hvor StudB er, men StudB er ikke i bygningen. StudA vælger at lægge en besked til StudB. Imens StudA skriver beskeden kommer StudB ind i bygningen. UbiDoor fortæller da StudA at StudB nu er her, så han behøver ikke skrive videre (men må godt hvis han vil), og fortæller StudB at StudA leder efter hende.

Alternativt scenarie: StudA vælger at skrive videre på beskeden selvom StudB ankommer imens.

DeferNotifyMeeting

Primært succes scenarie: En bruger har tilmeldt sig et møde og har valgt at få varsler om dette. Brugeren ankommer til det rigtige mødelokale mere end 5 min. før mødets start. UbiDoor sender derfor ikke varsel til brugeren.

Alternativt scenarie: Brugeren ankommer til mødelokalet mere end 5 minutter før, men forlader lokalet igen og kommer ikke tilbage inden 5-minutters fristen. Derfor sender UbiDoor alligevel en besked.

RescheduleMeeting

Primært succes scenarie: En administrator fortæller UbiDoor at studienævnsmødet er flyttet fra mandag til fredag fremover. Alle personer der sidder i studienævnet bliver fortalt af UbiDoor, at mødet er blevet flyttet varigt fra mandag til fredag.

Alternativt scenarie: Administratoren omlægger mødet, men UbiDoor ser at mere end 25% der skal deltage har andre møder på det givne tidspunkt. UbiDoor melder dette tilbage til Administratoren.

CancelRevervedRoom

Primært succes scenarie: En underviser vil afmelde reserveringen af et lokale onsdag kl. 12. Underviseren henvender sig til en UbiDoor. Underviseren finder lokalet og afmelder det.

Alternativt scenarie: Underviseren afmelder reserveringen. UbiDoor ser at nogen der har prøvet at reservere lokalet for ikke så lang tid siden. UbiDoor sender en besked til disse personer om at lokalet nu er ledigt.

FrontDoor

Primære succes scenarie: En person kommer til MIP efter lukketid. Hoveddøren er låst. Personen har sin BCB med sig, og døren åbnes automatisk da han nærmer sig. Døren lukkes og låses når han er inde i bygningen.

Alternativt scenarie: Personen har ikke sin BCB med sig, men han logger ind ved en UbiDoor terminal i stedet for.

Alternativt scenarie: Personen har ikke nogen BCB, og ikke noget login. Han kan ikke komme ind.

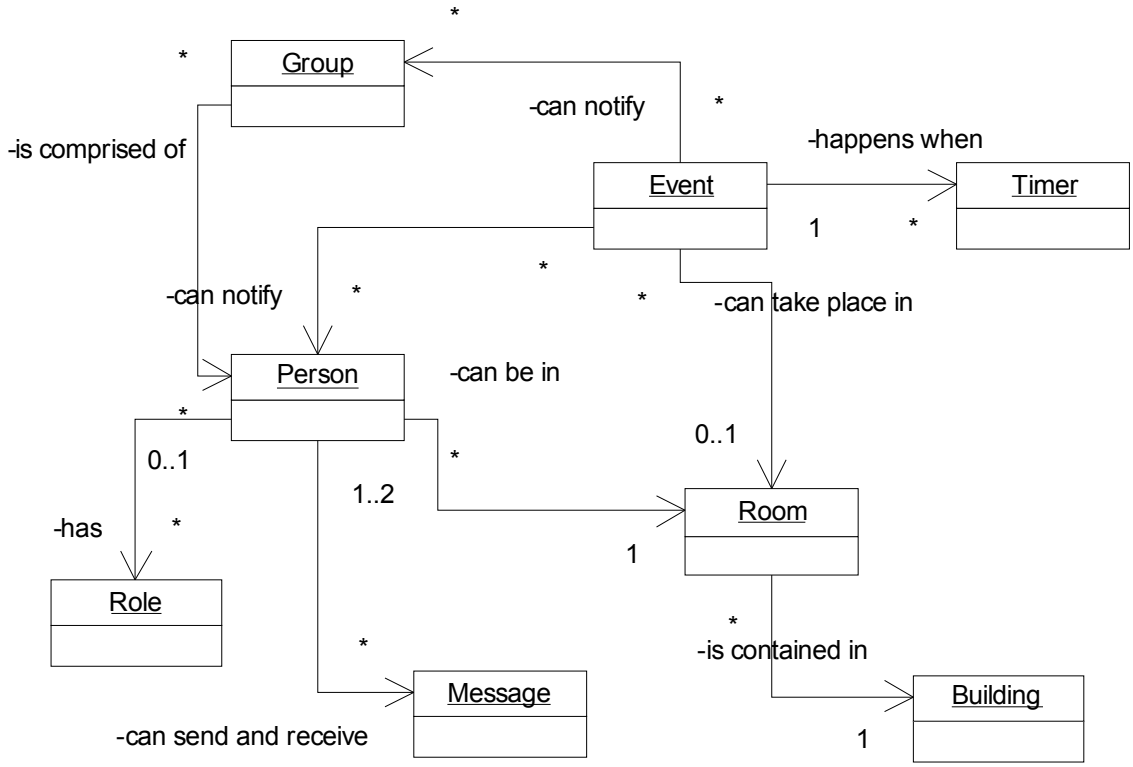
SecureRoom

Primære succes scenarie: Nogle rum er altid låst ekstra godt af (såsom server-rum) og kræver at brugeren har både BCB og login. En administrator med BCB ankommer til rummet, og indtaster sin kode for at låse rummet op.

Alternativt scenarie: Administratoren har ikke sin BCB med, men login er ikke nok så han kan ikke komme ind.

Alternativt scenarie: Rummet kræver enten BCB eller login (ikke begge), og administratoren kan logge ind og låse op.

Domain Model



Figur 1: Den samlede domain model

Person: En bruger af systemet.

Role: En måde at tildele brugere rettigheder, såsom administratorer eller undervisere.

Group: En gruppe af personer. En Group kan være alt lige fra en egenlig arbejdsgruppe til at være alle personer der er tilmeldt en studieretning.

Event: Dette kan være en forelæsning, over et møde, til arrangementer, eller bare når klokken er 10. En eventuel kalender ville være opbygget af Events.

Timer: Et enkeltstående eller et løbende (f.eks. ugentligt) tidspunkt. Grunden til at denne ikke bare er en del af Event, er at Event kan have flere tidspunkter som kan have specielle forudsætninger, og som skal kunne redigeres individuelt og præcist.

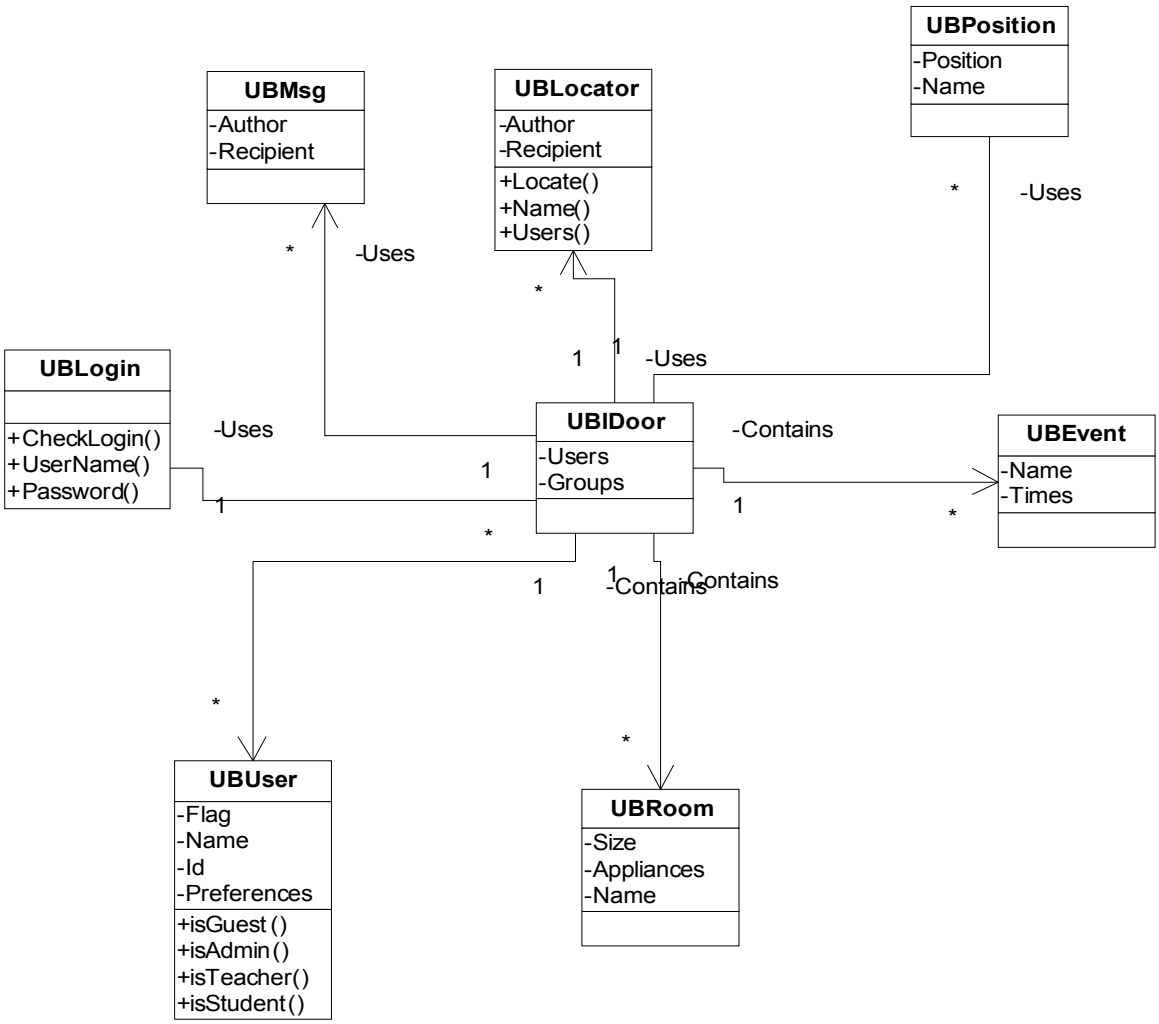
Message: En besked sendt til en person, fra en anden person eller som følge af en Event.

Room: Et rum…ikke så meget at sige om det.

Building: Tja, en bygning som indeholder rum. Gee.

Architecture

Design Model



Figur 2: Samlet design model for systemet

UBIDoor: Dette er den centrale class. Alt sker gennem UBIDoor, og UBIDoor holder styr på resten af systemet. Dette gør desværre UBIDoor stor og noget rodet (high-coupling, low-cohesion), men de andre classes bliver tilsvarende meget mindre rodede (low-coupling, high-cohesion). Dette er et valg sådan at en hvilkensomhelst ændring af systemet kun ville udløse ændringer i UBIDoor og *én* anden class. UBIDoor er således både Adapter, Facade, og Controller for alle andre classes og handlinger.

UBUser: Her er det ikke en association til en UBRole, som man eller kunne have forventet. Rent designmæssigt bør der være en UBRole, men som kode ville det være ineffektivt.

UBRoom: Her 'mangler' en UBBuilding, men da en UBBuilding ikke er noget specielt udover en samling UBRoom's, så er den ikke taget med.

Non-Functional Qualities

Non-Runtime

Nogle af de ting vi har tager i betragtning i designet af vores system er at det skulle kodes i et sprog, der gør det muligt at anvende uafhængigt af platform. Derfor har vi valgt Java. Platformuafhængigheden er en stor fordel, i det det muliggør at anvende UbiDoor i større organisationer hvor der anvendes forskellige systemer. Fx kunne det tænkes at UbiDoor vil anvendes både på MIP med også på hovedcampus på SDU. Hvis de to forskellige institutioner bruger forskellige platforme, vil UbiDoor, når det er kodet i Java, kunne anvendes begge steder.

Mere generelt kan det siges om de ikke-funktionelle krav til UbiDoor at sikkerheden i systemet skal være optimal, da UbiDoor kan håndtere formidling af private oplysninger og informationer. Den ekstensive brug af database tilgodeser persistensen af informationerne, gendannelsesmulighederne ved strømsvigt og log af transaktioner af informationer, både de brugerne selv har lagt ind i UbiDoor, men også de informationer der gemmes om den enkelte bruger ved dennes almen færden i bygningen.

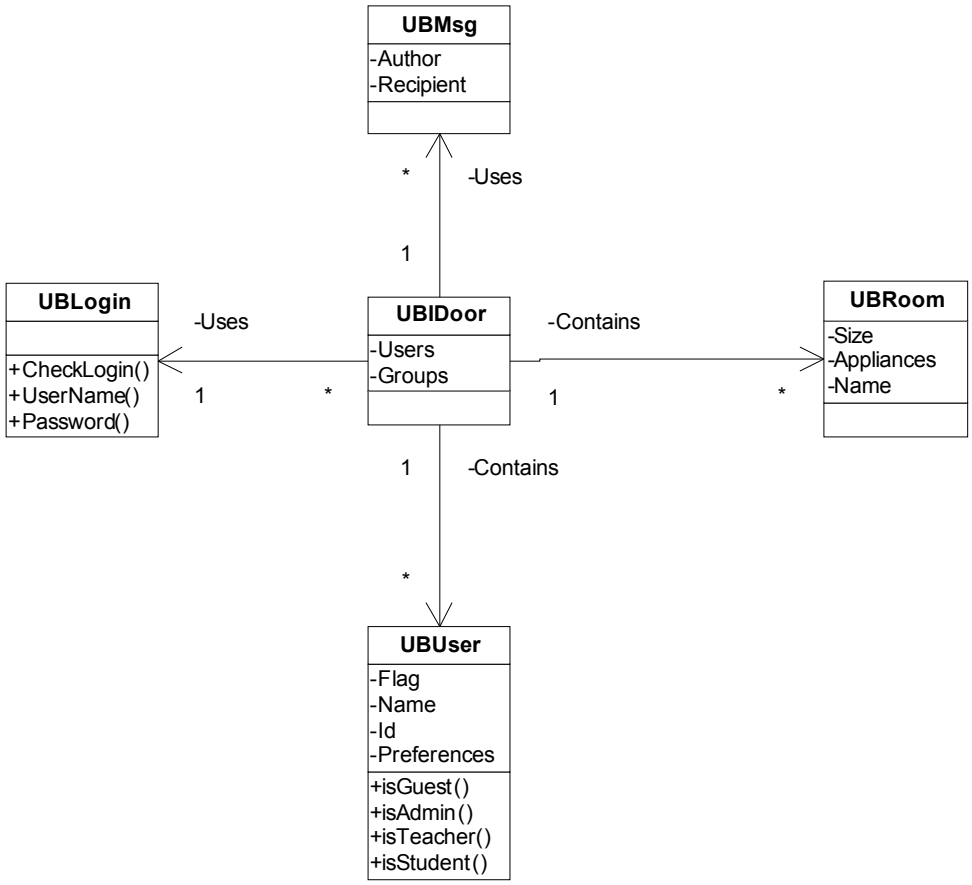
UbiDoor har yderligere den egenskab at det kan tilpasses den enkelte brugers ønsker. Hvis en bruger fx ikke ønsker at kunne få beskeder når han/hun sidder i møde, er det muligt for denne at slå den pågældende funktion fra.

Runtime

Da systemet skal anvendes af de fastansatte samt de studerende der opholder sig ved MIP er der et stort krav til anvendeligheden af systemet. Det skal være muligt for de enkelte faste brugere af systemet at kunne anvende de mest anvendte funktioner efter blot få fremvisninger, fx LocatePerson – se dettes Use Case for beskrivelse af funktionen. Funktionaliteten skal også være så gennemskueligt at de brugere der ikke til daglig anvender systemet skal kunne finde de mest almindelige funktioner uden formel læring.

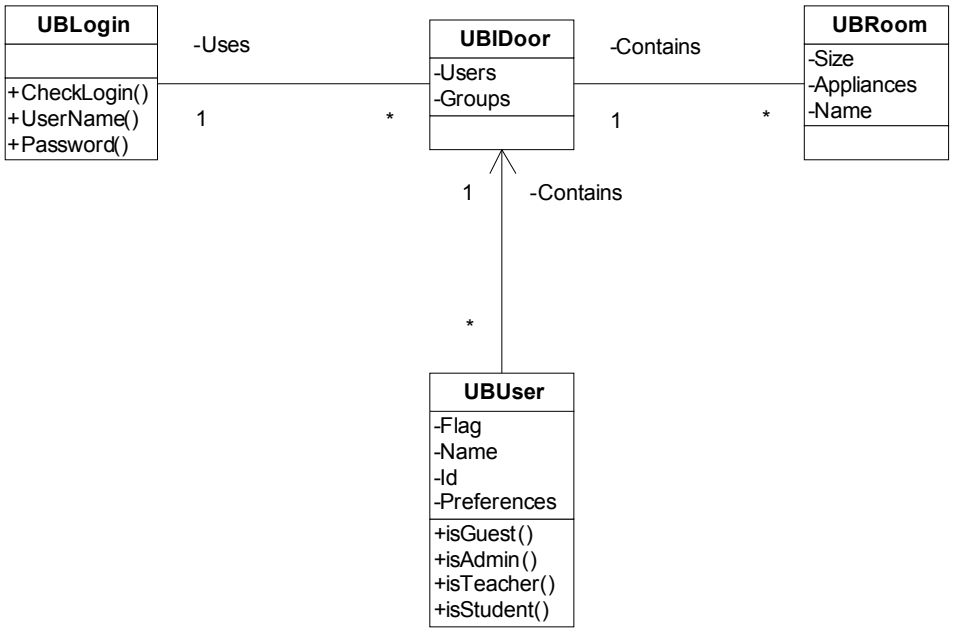
Udover dette meget centrale krav om funktionalitet og anvendeligheden er kravet om 'performance'. Det nytter ikke noget at UbiDoor er yderst funktionelt hvis det tager for lagt tiden inden man finder de informationer man leder efter, eller at der kommer en besked det fortæller at mødet er flyttet to timer frem, tre timer for sent.

Detailed Design



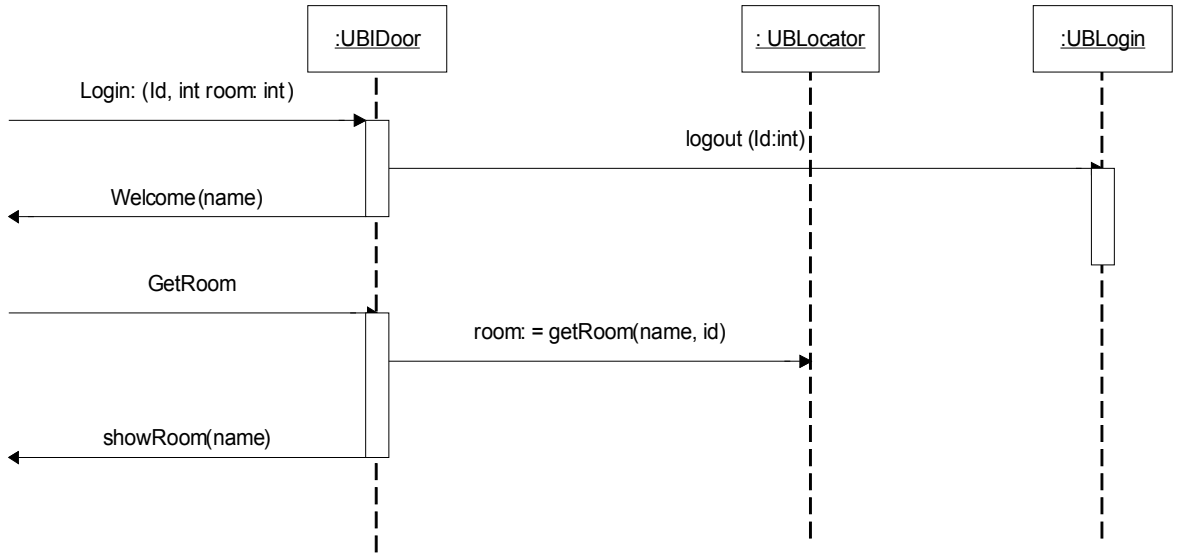
Figur 3: Design model for Use Cases: EnterBuilding og ExitBuilding.

Ikke så meget at forklare om disse. Vi henviser til den samlede Design Model i Architecture afsnittet.



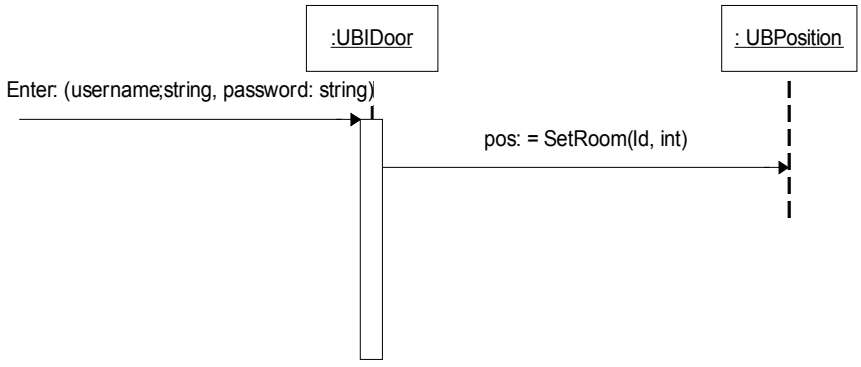
Figur 4: Design model for Use Cases: EnterRoom, ExitRoom, SecureRoom.

System Sequence Diagrams



Figur 5: SSD for Use Case: EnterBuilding + LocatePerson

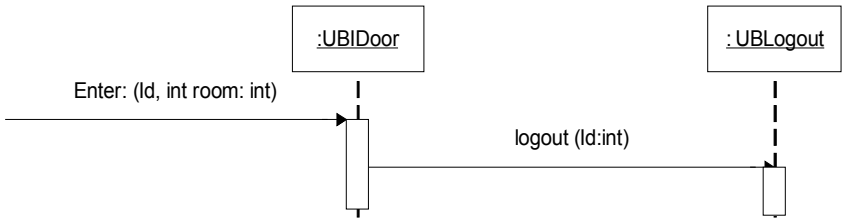
Dette SSD viser også lidt af hvordan den centrale UBIDoor class fungerer i sin Facade funktionalitet.



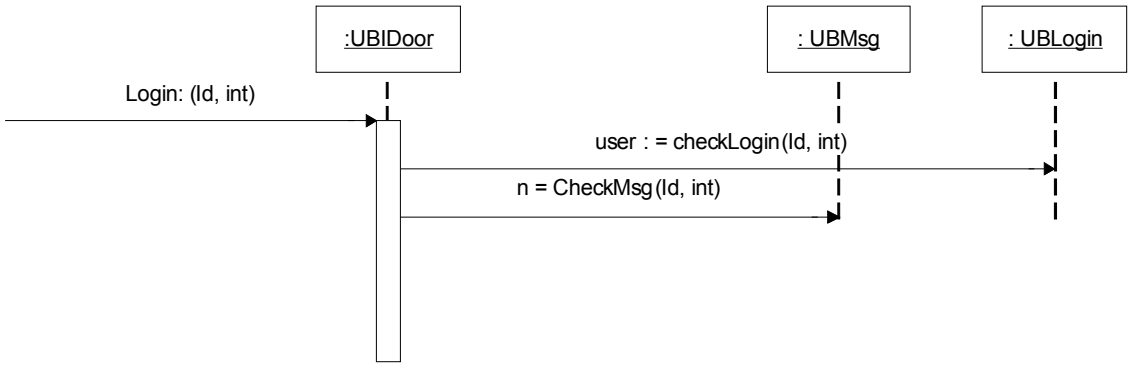
Figur 6: SSD for Use Case: EnterRoom/ExitRoom

Meget simpel SSD, men der skal ikke mere til at håndtere dette Use Case. Grunden til at SSD'en passer på både EnterRoom og ExitRoom er at man altid er i et og kun et rum. Hvis man forlader et rum træder man nødvendigvis ind i et andet (gangen er også et rum).

Specialtilfældet er dækket i næste SSD af ExitBuilding, hvor man helt forlader UbiDoos's aktivitetsområde.



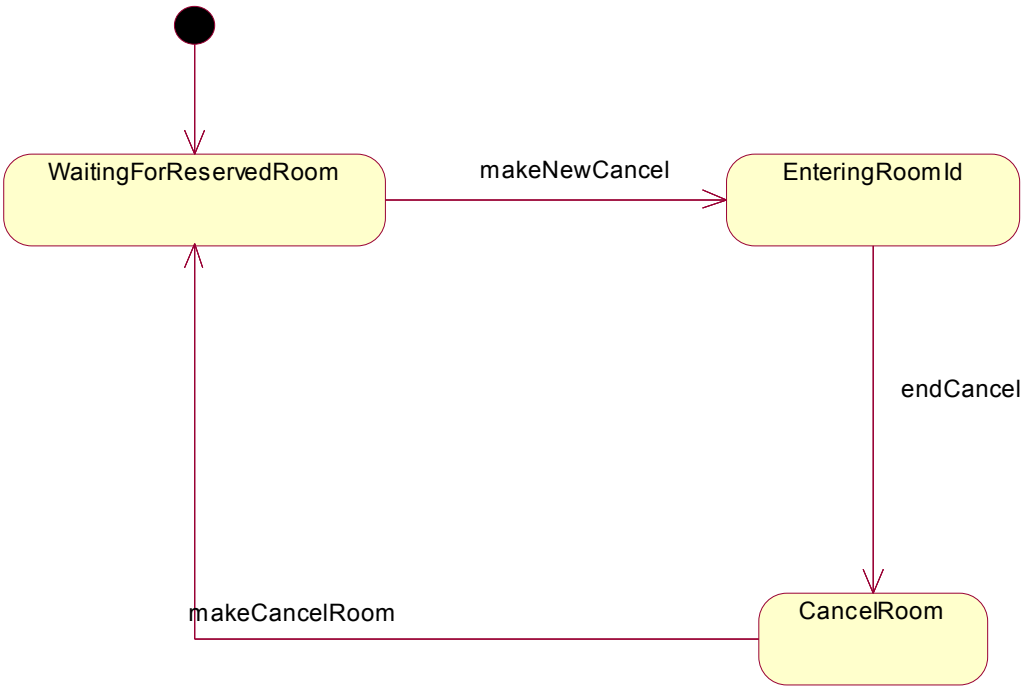
Figur 7: SSD for Use Case: ExitBuilding



Figur 8: SSD for Use Case: EnterBuilding

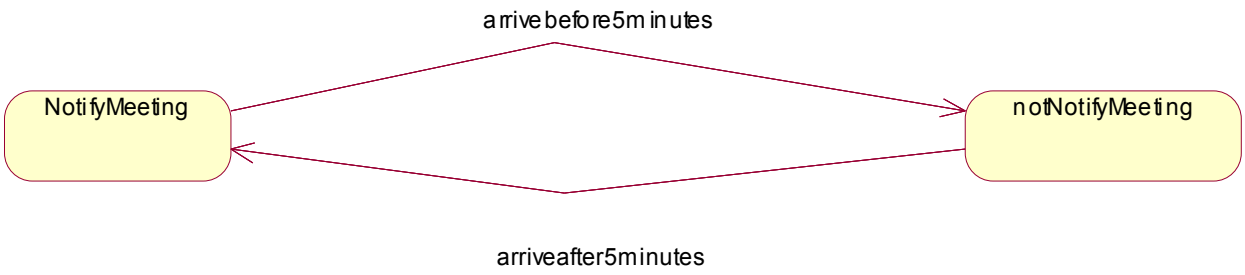
Lidt mere handling i dette SSD. Når man kommer indenfor UbiDoor's aktivitetsområde checker UbiDoor om der er nogle Events man skal have besked om, og om der er nogle Messages man ikke har set endnu. Hvis det er tilfældet, får man da at vide hvor mange beskeder man har.

State Charts

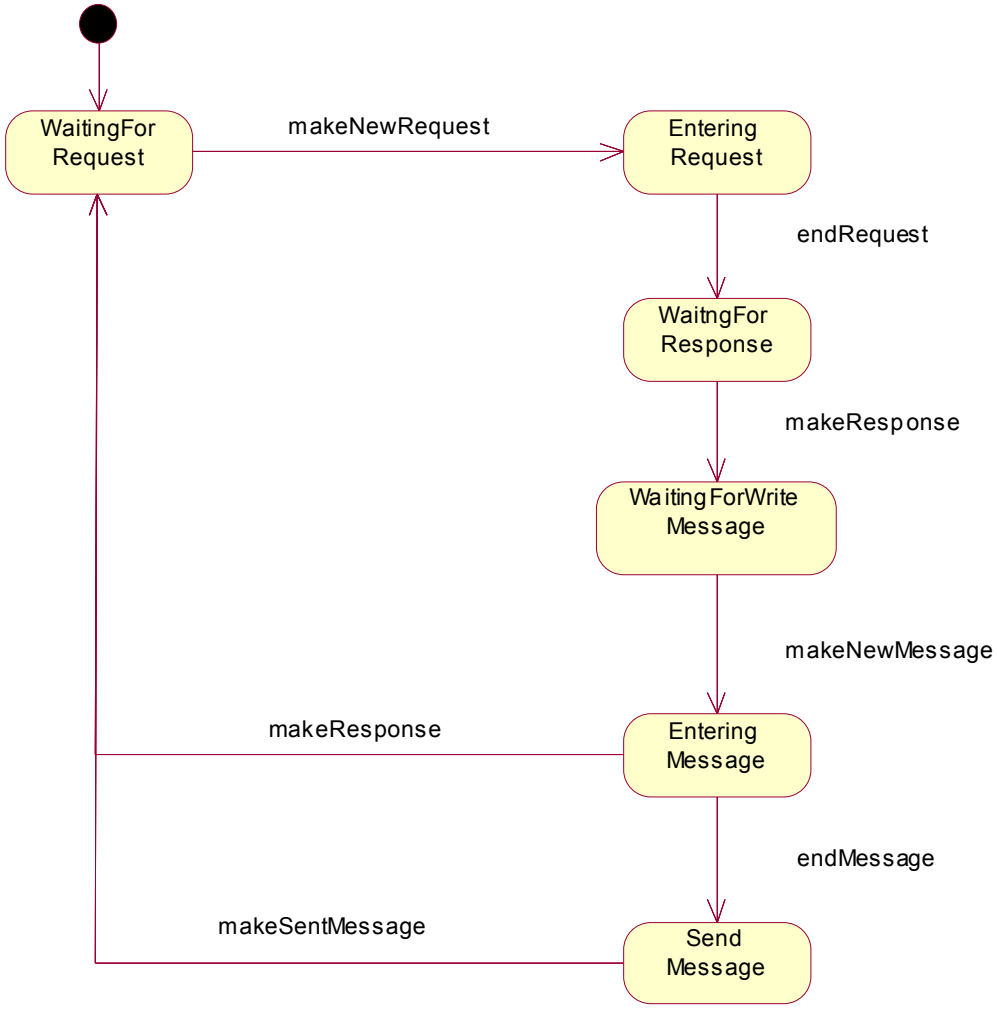


Figur 9: State-chart for Use Case: CancelReservedRoom

State charts er generelt mindre sigende end System Sequence Diagrams. Dette er fordi systemet ikke som sådan venter på at noget sker (med undtagelse af de tids-styrede Events). Alle handlinger udført af brugere bliver håndteret enkelvist. Også mindre sigende fordi i nogle tilfælde eksisterer der ikke et handler object før det skal bruges.

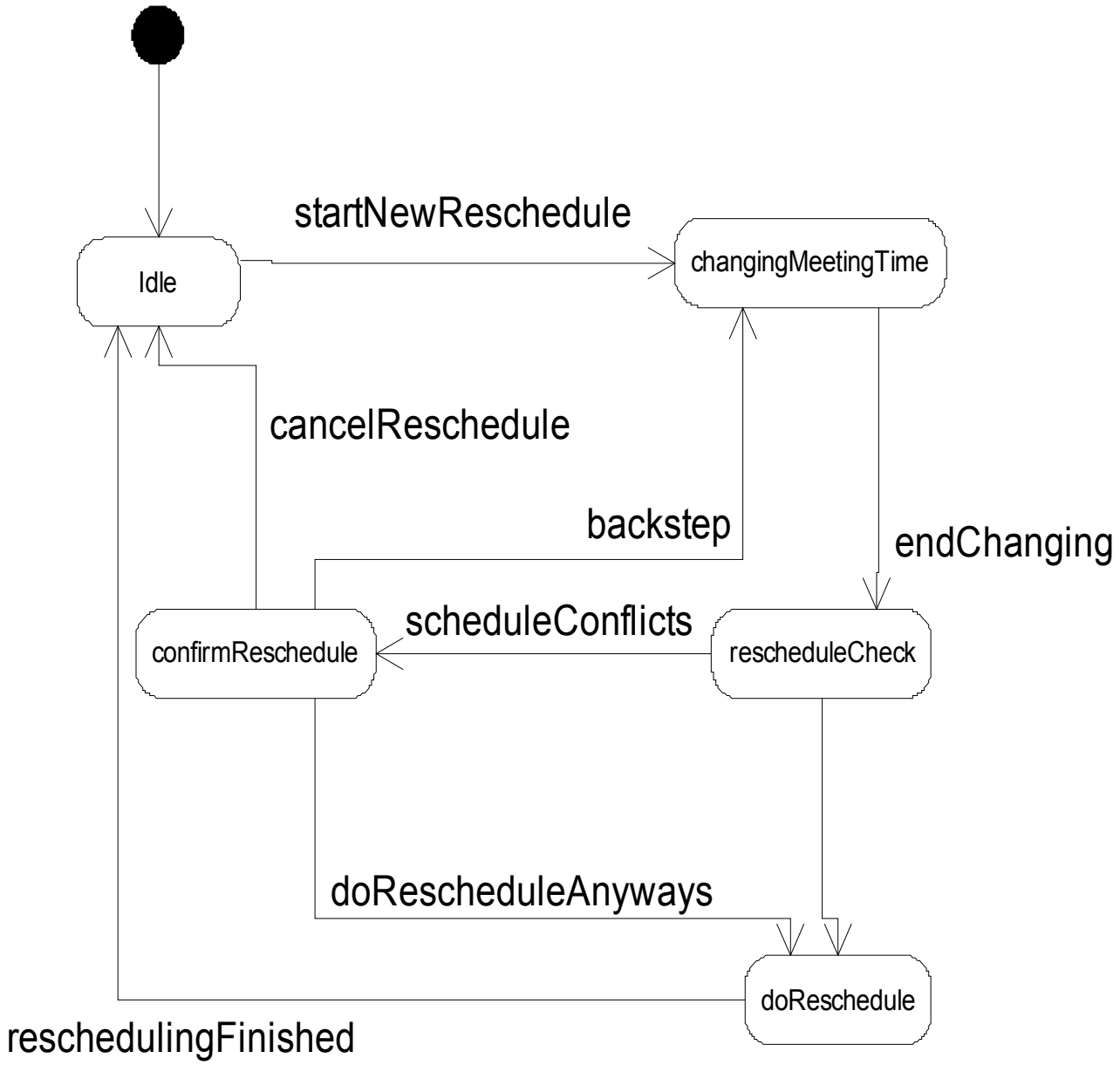


Figur 10: State-chart for Use Case: DeferNotifyMEeting



Figur 11: State-chart for Use Case: InterruptWriteMesssage

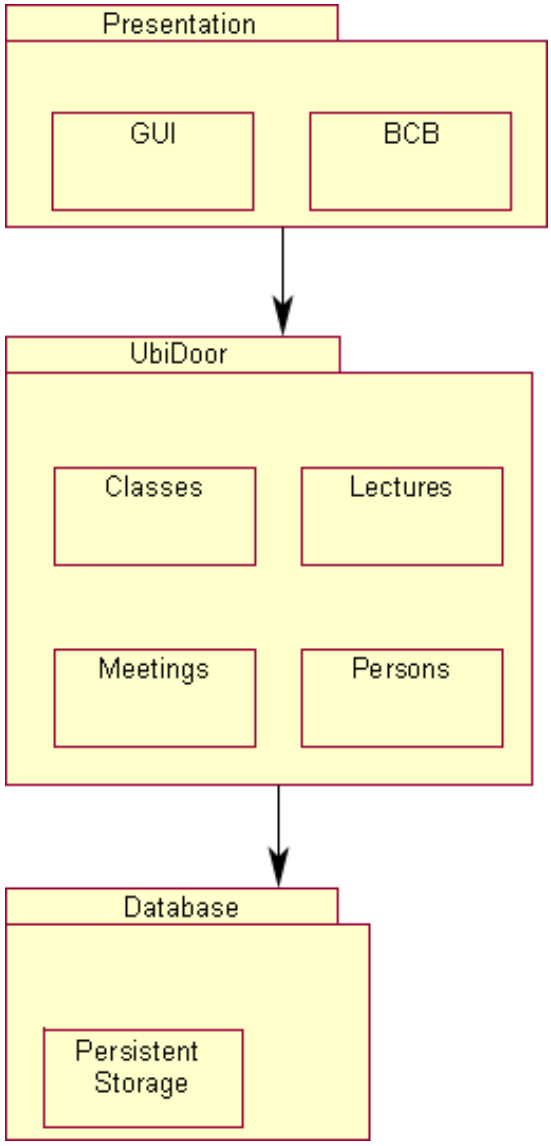
Hvor state charts bliver mere nyttige er til at vise alternative handlinger og scenarier, såsom i **Figur 10:** *InterruptWriteMessage*.



Figur 12: State-chart for Use Case: RescheduleMeeting

Dette viser meget af hvordan RescheduleMeeting fungerer. Der er 2 steder hvor der er flere udgangsresultater: rescheduleCheck, der undersøger om det nye mødetidspunkt ikke konflikter med andre møder for mere end 25% af deltagerne. Hvis der er en sådan konflikt går den til confirmReschedule der spørger brugeren hvad han nu vil gøre (cancel, backstep, eller accept).

Package and Layers



Figur 13: Simpel package og layer oversigt

Den ovenstående figur viser meget simplificeret hvordan vores packaging og layering er udtænkt, og også at der ikke er tænkt så meget over det. I denne opgave beskæftiger vi og kun med det midterste layer, idet vi blev forbudt at lave GUI og databaser. Vi har ikke opdelt vores system i yderligere interne layers, da der er meget simpelt i sig selv.

Actor Description

Gæster

En end-user af systemet. Har næsten ingen privilegier, andet end at kunne se hvornår der er offentlige foredrag, og lignende.

Studerende

En end-user af systemet. Kan sende/modtage beskeder, oprette gruppemøder, og tilmelde sig notifications.

Undervisere

Også en end-user af systemet, men disse brugere har privilegier til at lave om på deres fag, reservere lokaler.

Administratorer

Endnu en end-user af systemet, men disse brugere har alle privilegier, såsom at oprette og slette andre brugere.

Vi havde i første omgang taget UbiDoor med som aktør af systemet (altså at system er aktør af sig selv) idet i nogle Use Case, såsom NotifyMeeting, vil det være systemet der er aktivt primært. Men da disse event i sidste ende er bestilt eller oprettet af en bruger har vi fjernet systemet selv som aktør.

Table of Use Case Actors

Legend:

P Primary actor, highest priority

S Secondary actor, medium priority

T Tertiary actor, low priority

Use Case / Actor	Studenter	Undervisere	Administratorer	Gæster
EnterBuilding	P	P	P	
ExitBuilding	P	P	P	
EnterRoom	P	P	P	
ExitRoom	P	P	P	
EnterBuildingExpanded	P	P	P	
LocatePerson	P	P	P	P
NotWriteMessage	P	P	P	
WriteMessage	P	P	P	
InterruptWriteMessage	P	P	P	
RegisterMeeting	P	P	P	
NotifyMeeting	S	S	S	
DeferNotifyMeeting	S	S	S	
MoveMeeting	P	P	P	
RescheduleMeeting	P	P	P	
CancelMeeting	P	P	P	
MoveLecture	S	P		T
RescheduleLecture	S	P		T
CancelLecture	S	P		T
CancelClass	S	P		T
ReserveRoom	P	P	P	
CancelReservedRoom	P	P	P	
InformClass	S	P		
FrontDoor	P	P	P	
SecureRoom	P	P	P	

Table 1: Use Case Actors
Identificerer og markerer aktører i overensstemmelse med deres prioritet i forhold til Use Case