SW01, Objektorienteret analyse og design
Syddansk Universitet Odense

Rapport

Bjørk Boye Busch, 050457, åben uddannelse <bjbu@tietgen.dk>
Finn Nørgaard, 230565, datalogi/matematik <finn.norgaard@galnet.dk>
Adam Sierakowski, 300380, anvendt matematik <adam@imada.sdu.dk>
Niels Kjeldsen, 030983, datalogi via datateknologi <tnf@mail.dk>
Preben Hagh Strunge Holm, 280483, datateknologi <preben@cyberserver.dk>

Gruppe 7
Projektperiode: Forår 2004
Afleveres 17. maj 2004
Forside og indholdsfortegnelse: 3 sider
Rapport: 29 sider
Bilag: 5 sider

Indhold

Kapitel 1

Reflektion

1.1 Introduktion

Ubiquitous Doorman betyder “Allestedsnærværende dørmand”. Man kan så spørge sig selv, hvordan en dørmand kan vide, hvor en person befinder sig - han står jo blot i en dør. Men idet denne dørmand netop er allestedsnærværende, er han overalt - dvs. stående ved alle døre. Vedkommende kan derfor se, hvem der er gået gennem en given dør og i hvilken retning personen gik. Dørmanden ved præcis på hvilket tidspunkt vedkommende er gået igennem.
Ved at forespørge denne allestedsnærværende dørmand, kan man altså få oplyst, hvor en person befinder sig i en given bygning.

Der kan forekomme problemer for dørmanden, da han ikke i alle situationer vil vide, hvor en person er. Hvis f.eks. en person er hoppet ud af et vindue vil dørmanden tro, at personen stadig er i det lokale vedkommende gik ind i sidst. Dog ved dørmanden, at personen på en eller anden måde er kommet ud, hvis vedkommende bevæger sig ind ad en dør igen, uden at være gået ud af den.

Nærmere fortalt går dette projekt altså ud på at udvikle et system, der kan overtage jobbet, som værende denne fiktive allestedsnærværende dørmand.

1.2 Implementation-status

Vi har i projektet valgt at implementere brugsmønstrene som omhandler ankomst, skiftning af område samt mødeplanlægning. I anden iteration har vi tilføjet usecases “send besked”, “er jeg efterspurgt”, “anmod besked” samt “laas op”. Vi har dog ikke implementeret “lås op”.

Vi har valgt disse brugsmønstre, da de er de mest centrale i vores projekt, og samtidig har en vis kompleksitet, hvilket vi til dels har forenklet ved at undgå implementation af visse specialtilfælde. Hermed tænkes der mest på mødeplanlægningen.

Vi har igennem brugen af disse brugsmønstre kommet igennem det meste af den centrale model.

Vi har endvidere realiseret ovenstående ved implementation i Java. De centrale klasser i denne model omhandler Person, Omraade, Moede, ForladtOmraade (log) samt Efterspoergsel, som også afspejles i vores domænemodel.

Fra starten af projektet var vi mest interesseret i et adgangssystem og samtidig et informationssystem, som gav mulighed for implementation af bl.a. adgangskontrol og mødeplanlægning. I løbet af projektet valgte vi at implementere kommunikationsdelene af systemet, hvor vi fandt

gode muligheder for anvende et Strategi- og Adapter-mønster. Med tiden fandt vi dog ud af, at vi ikke havde tid til også at implementere kontrolsystemet, da vi havde valgt nogle ret omfattende usecases. Samtidig skulle vi i anden iteration fokusere på nye usecases, hvilket yderligere gav anledning til, at fokus blev flyttet til informations/kommunikationsdelen. Dette vil selvfølgelig afspejles kraftigt i system vision-dokumentet, hvor man ser fokus på adgang, som altså ikke har fået samme vægt i designet.

I implementationen har vi fremstillet en "Kerne"-klasse, som fremstår som et samlingsobjekt, der fungerer som udgangspunkt for hele systemet. I første iteration var Kerne-klassen facade for de tilstandsløse usecases. I anden iteration udskilte vi alle usecases til selvstændige facadeobjekter. Nogle af Kerne-klassens metoder blev flyttet til Doermand-klassen. Kernen har nu ansvaret for samlingen af personer, møder og områder. Desuden står Kernen for oprettelse af personer, områder og møder, hvilket er i overensstemmelse med mønstret Expert og Creator. Kontrollen af, om objekter eksisterer udfra et givent ID, er nu flyttet til facade-pakken og altså væk fra kernen.

Vi har valgt at implementere Kernen som Singleton, da der kun må eksistere en instans af den. Det samme gør sig gældende for BeskedSender'en. Vi mindsker iøvrigt koblingen, da andre objekter ikke behøver at vide om et andet objekt (eks. Møde) skal bruge metoder fra Kernen (vi behøver ikke medsende Kernen som parameter til metoderne).

Vi har ydermere lavet selvstændige klasser til dannelse af testdata, til afprøvning af brugsmønstre samt til håndtering af mødeplanlægning.

1.3 Brug af Java

Vi implementerer systemet ved hjælp af Java2. Vi har brugt Collection framework'et fra java.util, herunder brug af maps som HashMap og Collections. Endvidere har vi tilstræbt at mindske koblingen gennem brug af referencer som Collection og tilgået gennemløb af data ved brug af Iterator.

1.4 Arbejdsmetode

Vi har arbejdet som en samlet gruppe, hvor der under arbejdet har deltaget mindst tre personer ad gangen. Det har bevirket, at vores model blev konstrueret inkrementelt, dvs. den er hele tiden blevet udbygget under processen. Vi har undgået at arbejde fragmenteret for derefter at skulle samle såvel model (både domæne og designmodel) som programkode. Dette har også givet et godt overblik over det samlede system under hele forløbet.

Kapitel 2

System Vision

2.1 Ubiquitous Doorman

I Maersk-bygningen på Syddansk Universitet Odense ønskes der installeret et antal "dørmænd", som skal registrere, hvem der går ind og ud af bygningen. Desuden skal "dørmændene" registrere tidspunktet for, hvornår personen passerer. "Dørmanden" skal endvidere give besked til andre personer, der har efterspurgt den ankomne. Ydermere skal "dørmændene" kunne videregive beskeder til personer.
"Dørmændene" er ansvarlig for hvilke personer, der lukkes ind i bygningen alt efter hvilket tidspunkt personen ankommer på. Bestemte steder i bygningen vil der være "dørmænd", således at de i samarbejde kan indsamle information, så en person kan positioneres i bygningen. Derudover kan der i fremtiden installeres en "dørmand" for hvert område[1] og alt efter personens identitet vil vedkommende få lov til at gå ind i dette område. I hovedbygningen har gæster som hovedregel altid adgang i et bestemt tidsrum i hverdagene. Anderledes forholder det sig med studerende, undervisere, andre ansatte, vagter o.lign. Alt efter deres stilling vil de have adgang til forskellige områder og kontorer. Nogle områder er spærret helt af, hvor kun specielle personer har adgang. Andre områder er tilgængelige for mange personer, der er inddelt i nogle grupper, således at hver gruppe har adgang i bestemte områder.

Figur 2.1 viser et fiktivt eksempel, hvor gæsterne har adgang til de (røde) områder og de (blå) prikker angiver "dørmændene".

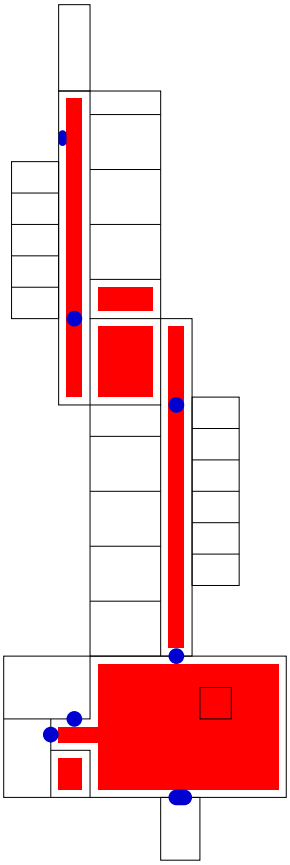
"Dørmændene" har adgang til en områdekalender med oplysninger om hvor og hvornår, der afholdes møder i det pågældende område, samt hvem der deltager i mødet, således at mødet ikke forstyrres af uvedkommende.

Enhver ansat på Maersk instituttet har sin personlige kalender. Denne er tilgængelig for både sekræteren og personen selv.

2.1.1 Systemet

Systemet skal kunne angive i hvilket område af bygningen en person befinder sig. Endvidere skal systemet håndtere en persons status — sidder personen i møde, optaget af arbejde, tilgængelig eller måske gået til frokost. Systemet skal kunne håndtere beskeder til personer.
Yderligere skal systemet have indbygget en område-administration med oplysninger om hvilke områder, der er optaget på bestemte tidspunkter.
Systemet skal have oplysninger om hvilke personer, der har adgang til de bestemte områder og

[1] Område anvendes som en generalisering for bl.a. et lokale



Figur 2.1: Maersk-bygning. (røde) områder: offentlig adgang, (blå) prikker: dørmænd

på hvilket tidspunkt systemet tillader disse personer at gå ind i bygningen. Personer skal kunne grupperes.

Yderligere ligger der i projektet, at man som bruger af systemet skal have mulighed for at søge på en person og dennes status — hvor befinder personen sig, og hvad vedkommende laver. Personer registreret i systemet skal have mulighed for at planlægge møder.

Alt efter adgangsniveau skal det være muligt at søge og reservere områder i bygningen til eventuelle møder, herunder undervisning.
Administrationen skal tilbydes at kunne administrere adgangen i systemet og bestemme, hvem der har adgang til bestemte områder.

2.1.2 Hvad kan vi og hvad skal vi bruge?

Af teknologi er vi ret bundne, og har valgt at udvikle på almindelig PC-platform med Java development kit. Hvad angår stabilitet og responstid i programmel vil vi udelukkende anvende de muligheder, som java-platformen giver os, og såfremt det er muligt udnytte denne platforms stabilitet. Anvendelsen af Java medfører dog også en vis platformsuafhængighed så valget mellem PC, PowerPC, Sparc osv. vil være os ligegyldigt. Yderligere har man idag også muligheden for at anvende Java på et embedded system. Dog vælger vi ikke at tage højde for strømsvigt eller lignende *force majeure*, da sådanne nedbrud kun vil kunne afhjælpes ved brug af mere hardware i form af UPS[2]-løsninger.

[2]Uninterrupted Power Supply

2.1.3 Elementer i systemet

Som udgangspunkt fandt vi frem til, at systemet består af grund-elementerne

- Dørkontrol
- Områder
- Studerende, undervisere/ansatte og gæster
- Vagt
- Personlig kalender
- Områdekalender

Kapitel 3

Usecasemodel

3.1 Use case oversigt

Vi er kommet frem til følgende brugsmønstre i projektet efter en brainstorming.

- Byd velkommen
- Skift område
- Lås op
- Skift adgangskode
- Efterspørg person
- Send besked
- Anmod besked
- Er jeg efterspurgt?
- Giv mig ny info
- Reserver lokale
- Admin. lokaleopl.
- Angiv egen status
- Admin. egne opl.
- Admin. personopl.
- Forespørg møde
- Planlæg møde
- Ændre møde

3.2 Aktører

Vi har valgt at følge de enkelte personer som værende aktører i systemet, vel vidende at vi kun kan følge en "sporingsenhed" (BCB). Dette betyder, at vi udefra betragter systemet som værende en simpel brugergrænseflade, hvor man i praksis eksempelvis kan indtaste sin ankomst i en terminal ved f.eks. en dør. Vi har altså ikke fastlagt en bestemt teknologi til registrering af ankomst, skift af område m.m.

Herunder specificeres de enkelte aktører, vi er kommet frem til.

3.2.1 Sekretær

Sekretæren er den daglige administrative person, som skal tage sig af brugeroprettelser, svare på spørgsmål fra udenforstående, eksempelvis gæster. Dette kunne f.eks. være spørgsmål om, hvor folk befinder sig.

3.2.2 Administrator

Administratoren skal vedligeholde systemet og har som hovedregel derfor ansvar for, at brugerne kan anvende systemet og kan afhjælpe brugernes problemer.

3.2.3 Underviser

Underviseren har de samme muligheder som sekræteren bortset fra, at underviseren ikke har mulighed for at ændre oplysninger om lokaler samt administrere personoplysninger.

3.2.4 Student

Den studerende er i forhold til underviseren yderligere begrænset ved ikke at kunne reservere lokaler.

3.2.5 Gæst

Gæsten er aktør på systemet, men er til forskel fra de andre personer ikke identificerbar overfor systemet, og indgår derfor ikke i problemdomænet (gæsten er aktør, men kræver ikke administration af systemet). Dog skal enhver udefrakommende som gæst have mulighed for at interagere begrænset med systemet. Gæsten skal kunne få information om, hvor en person befinder sig og mødeinformation. Ydermere skal det for gæsten være muligt at sende en besked til en registreret person i systemet.

3.2.6 Eksternt system

Det eksterne system, vi har defineret som aktør i vores usecases, skal ses som en mulighed for et system på f.eks. et andet institut som IMADA, at kunne hente information om, hvor eventuelle personer befinder sig. Det kunne også være et system med en helt anden funktionalitet end det nævnte. Vi har generelt valgt, at ethvert autoriseret system, der opererer på vores, skal tilbydes de nævnte muligheder.

3.2.7 Infotavle

Infotavlen er en form for eksternt system, men har den specielle egenskab, at den kun vil hente information og ikke opdatere information i systemet. Dvs. der skal kunne forespørges på møder og personer. Infotavlen er tænkt som en elektronisk tavle, der ved f.eks. Maersk-bygningens indgang præsenterer dagens møder, hvor der endvidere er mulighed for at forespørge på en bestemt person f.eks. ved brug af en touch-screen. Denne præsentation skal dog stå åben, og vi har derfor valgt dette som værende en form for eksternt system, vi ikke ønsker at implementere.

3.2.8 Dørkontrol

Vi har ved projektets begyndelse overvejet, at der skal bruges hardware til håndtering af dørkontrol. Dette giver ekstra aktører til systemet. Vi valgte at betragte denne hardware som en brugergrænseflade i stil med BCB'en.

3.3 Use cases

I det følgende, hvor ordet person optræder menes en person registreret i systemet.

Tabellen viser de forskellige use cases og de tilhørende aktører.

Use Case	Sekræter	Admin.	Under-viser	Student	Gæst	Extern system	Info tavle
Adgang							
byd velkommen	+	+	+	+	+	-	-
skift område	+	+	+	+	-	-	-
lås op	+	+	+	+	-	-	-
skift adgangskode	+	+	+	+	-	-	-
Komunikation							
efterspørg person	+	+	+	+	+	+	+
send besked	+	+	+	+	+	+	-
anmod besked	+	+	+	+	-	-	-
er jeg efterspurgt	+	+	+	+	-	-	-
giv mig ny info	+	+	+	+	-	-	-
Lokaler							
reserver lokale	+	+	+	-	-	-	-
admin. lokaleopl.	+	+	-	-	-	-	-
Personoplysninger							
angiv egen status	+	+	+	+	-	-	-
admin egne opl.	+	+	+	+	-	-	-
admin. personopl.	+	+	-	-	-	-	-
Møder							
forespørg på møde	+	+	+	+	-	+	+
planlæg møde	+	+	+	+	-	-	-
ændre møde	+	+	+	+	-	-	-

Brugsmønstre markeret med kursiv, er beskrevet i det følgende, og er dem vi har valgt at implementere.

Selvom Sekretær og Administrator har samme adgang til systemet, er de forskellige personer, som har forskellige primære arbejdsopgaver i dagligdagen, hvorfor at brugergrænsefladen hensigtsmæssigt kunne være tilpasset den enkelte aktør.

3.3.1 Byd velkommen - casual

En person ankommer til Maersk instituttet og verificeres af første "dørmand". Døren åbnes og personen bydes velkommen. Systemet registrerer, at personen er gået ind ad døren. Systemet meddeler ankomsten til de personer, som har efterspurgt den ankomne.

Alternative senarier

Personen er en gæst og systemet kan ikke vide, hvem personen er.

Hvis området er aflåst, hoppes til scenariet "lås op".

3.3.2 Skift område - brief

En person forlader et område og identificeres af systemet. Systemet registrerer, at personen er gået fra et område ind i et andet område.

3.3.3 Efterspørg person - brief

En aktør henvender sig til systemet. Aktøren spørger, hvor en bestemt person befinder sig, hvilket oplyses af systemet.

3.3.4 Planlæg møde - brief

En bruger henvender sig til systemet og identificeres heraf. Brugeren definerer den mødeansvarlige. Systemet oplyser brugeren de registrerede personer. Brugeren angiver herefter hvilke mødedeltagere (registrerede personer), der skal deltage, samt det totale antal deltagere. Brugeren angiver det ønskede mødetidspunkt og mødets varighed. Systemet meddeler, på baggrund af deltagernes personlige kalender, om der er eventuelle konflikter for mødedeltagerne. Brugeren vælger om mødetidspunktet skal laves om eller ej. Systemet foreslår de tilgængelige områder, hvor det totale antal deltagere kan være. Til sidst vælger brugeren et mødeområde.

Postbetingelse

I det ovenstående brugsmønster antages, at mødet er gemt og relateret til relevante personer og et område.

3.3.5 Send besked - brief

En aktør angiver modtagerens personID og besked. Systemet sender beskeden til modtagerens foretrukne medie.

3.3.6 Er jeg efterspurgt - casual

Brugeren spørger systemet om vedkommende er efterspurgt. Systemet viser, en ad gangen, de efterspørgsler, brugeren ikke har set. Hver gang spørger systemet om de skal slettes. Brugeren svarer.

Alternative senarier

Ved gennemsyn af efterspørgsler har brugeren til "enhver tid"mulighed for at afbryde.

Brugeren vælger at fortsætte interaktion. Systemet viser, en efter en, de tidligere sete efterspørgsler og efterfølgende spørger brugeren om de skal slettes. Brugeren svarer og til sidst afsluttes der.

Brugeren vælger at fortsætte interaktion. Systemet spørger om alle de sete efterspørgsler skal stettes. Brugeren svarer.

3.3.7 Anmod besked - casual

En bruger anmoder systemet om at blive informeret, når en bestemt person X ankommer.

Alternativt senarie

X er allerede i bygningen. Systemet fortæller hvor X er og spørger brugeren om vedkommende virkelig vil informeres næste gang X ankommer. Brugeren svarer systemet og der afsluttes.

3.3.8 Præbetingelser

I de førnævnte use-cases antages, at man ved oplysning af personer og områder refererer til et allerede eksisterende objekt.

Kapitel 4

Domænemodel

4.1 Domænemodel

Domænemodellen ses illustreret i figur 4.1

Klassen “Person” i modellen er uafhængig af aktøren. Vi har endnu ikke taget hensyn til de nævnte aktører beskrevet i usecase/aktør tabellen, hvilket kan få betydning, da de fleste aktører er sammenfaldende med domænet.

Vi har en association mellem Person og Møde således, at en person kan deltage i et antal møder. Omvendt kan et møde også omfatte flere deltagere, hvorfor der ses en “mange-til-mange” association.
Ydermere har vi valgt, at der skal være netop nul eller en ansvarlig for et møde. Om det er hensigtsmæssigt eller ej, at man ikke kan have flere ansvarlige, har vi ikke taget stilling til.

Et møde kan kun afholdes i et bestemt område, men et område kan sagtens benyttes til flere forskellige møder, på forskellige tidspunkter.

Yderligere ses, i diagrammet, en association mellem “Person” og “Område”, da en person altid vil være i et bestemt område. Dette betyder samtidigt, at man bliver nødt til at definere, at området udenfor Maersk bygningen, også opfattes som et område.

De næste to klasser er ForladtOmråde og Efterspørgsel. ForladtOmråde–klassen skal ses som et objekt, der bliver til, hver gang en person forlader et område og samtidig ændres associationen til området vedkommende er i.

En efterspørgsel kan være oprettet af en person, og man kan som person også have en efterspørgsel. Det er højst sandsynligt aldrig den samme, hvilket derfor ses som to forskellige associationer. En person kan have mange efterspørgsler, men en efterspørgsel kan aldrig være associeret til mere end een person. Samtidig gælder det, at en efterspørgsel heller ikke kan være oprettet af flere personer, men en person kan sagtens have oprettet flere efterspørgsler.

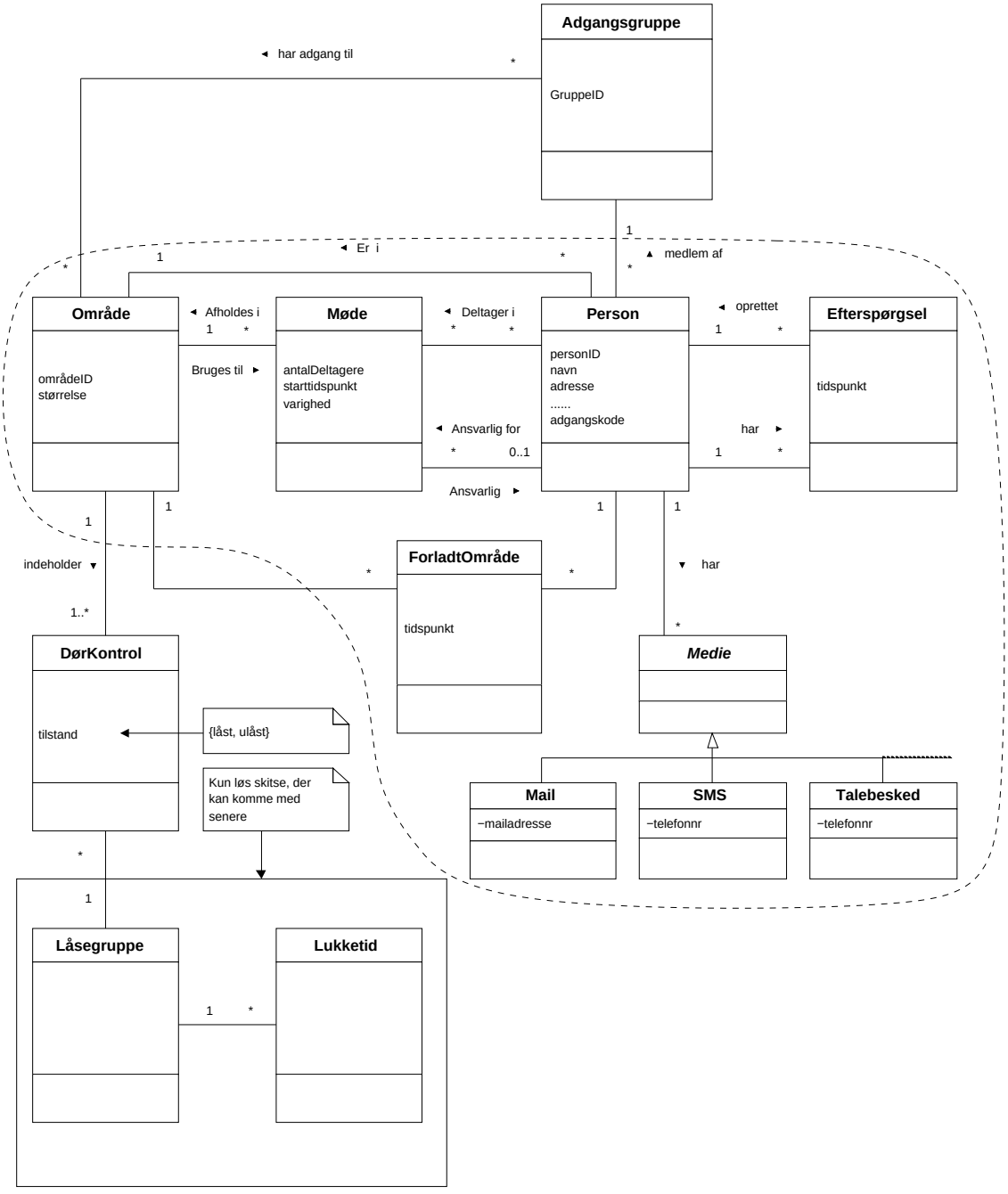
Adgangsgruppen skal bruges til at styre, hvem der har adgang til hvilke områder. Dørkontrol er klassen, som varetager tilgangen til et område. Denne klasse afspejler f.eks. en terminal. Låsegruppe skal gøre det muligt at samle Dørkontroller, så de kan låses og låses op samlet. I forlængelse heraf kan Lukketid bruges til at specificere en plan/kalender over bestemte dage og tidspunkter, hvor døre skal være låst.

De sidste klasser er Medie, Mail, SMS og Talebesked. Disse klasser er tilføjet i anden iteration, hvor vi i forbindelse med brugsmønsteret omkring afsendelse af beskeder fandt frem til, at der var mange forskellige medie-typer (herunder fremtidige medie-typer). En person kan foretrække at blive informeret om hændelser/beskeder vha. et bestemt medie. Vi har derfor valgt at lave en

generel medie-klasse, hvor man kan specialisere de medier, der kan komme på tale. Vi har i første omgang forestillet os Mail, SMS og Talebeskeder værende mulige medier. Vi forestiller os, at disse medier er prioriterede efter den rækkefølge brugeren ønsker, at de skal benyttes.

4.1.1 Viderebearbejdning

Vi har valgt at arbejde videre indenfor den del af domænet, som er indenfor det stiblede markering i figur 4.1. De udvalgte usecases har ikke givet anledning til at inddrage flere klasser fra domænemodellen.



Figur 4.1: Domænemodel af systemet

Kapitel 5

Arkitektur

5.1 Generel arkitektur

Figur 5.1(a) viser Ubiquitous Doorman-projektets arkitektur. Det er tydeligt set, at arkitekturen er lagdelt i 5 lag. De lag, vi anvender, er et user interface (ui) lag, et usecase-facade lag, et kerne lag og et lag indeholdende tekniske services samt et lag indeholdende datatyper. Denne opdeling anvendes blandt andet for at øge læselighed, mindske omkostninger ved vedligeholdelse og forbedre dokumentations mulighederne. På engelsk anvendes begrebet "-ilities", som henviser til ord som maintainability, reuseability, flexibility og readability. UsecaseFacade-laget anvendes af eventuelle brugergrænseflader (både grafiske, tekstbaserede og eventuelle lydbaserede).

Kerne-laget modellerer selve problemområdet og det er ved brug af dette lag, at facade objekterne kan stille services til rådighed. Det næstnederste lag (som er med i denne model) er tekniske services. De tekniske services har herunder en pakke kaldet beskeder, som står for beskedafsendelse.

Det nederste lag indeholder datatyper som f.eks. medietyper, hvilket også ses illustreret i en pakke for sig selv.

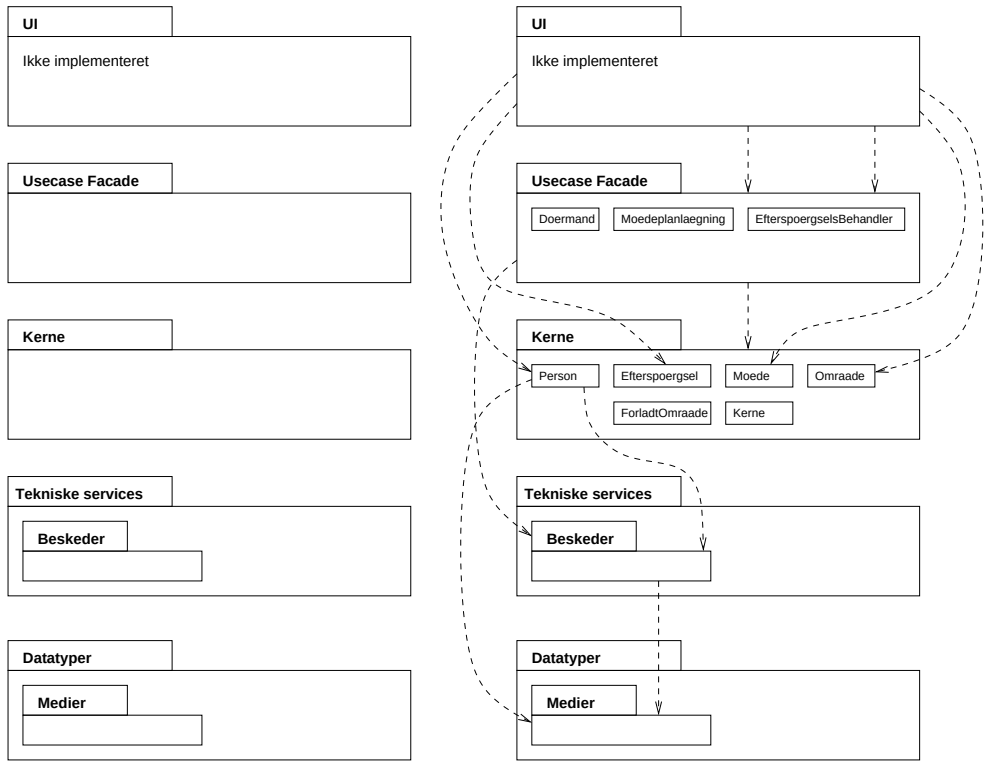
Hvis vi følger Craig Larmans begreber kan vi sætte lighedstegn mellem user interface laget og presentation laget. Derudover beskriver Larman application laget, hvilket i vores model ses illustreret som usecase-facade laget. Domain laget er i vores model Kerne-pakken. De tekniske services siger sig selv, men datatyperne vil i Larman være set som Foundation laget.

Medierne er oplysninger som beskriver et medie. Det kunne f.eks. være et telefonnummer, en mail-adresse eller noget lign. for et måske fremtidigt medie. Ved at lagre medie-oplysninger som datatyper med et generelt interface, kan man uden problemer tilføje nye medier, efterhånden som disse fremkommer.

Der er på denne måde mulighed for at bruge et medie meget generelt, hvilket giver en generel genbrugelighed i både vores, men eventuelt også andre systemer.

De tekniske services indeholder BeskedSender, hvilken vi også har valgt at udskille, så den kan genbruges direkte i andre systemer. Besked-Senderen bliver endvidere nemmere at vedligeholde, når den er udskilt i en selvstændig pakke, da den kun afhænger af de forskellige medietyper og sin egen grænseflade. Ikke mindst pga. brugen af Adapter-mønsteret (interfacet Sender-Adapter - se figur 6.9). Endvidere ligger dette lag over Medie-laget, da man ved anvendelsen af BeskedSender skal bruge netop denne datatype.

Som det fremgår af det ovenstående er de primære ikke-funktionelle krav, vi har taget hånd om vedligeholdelse, fleksibilitet og genbrugelighed. Dette fremgår ikke af System Vision dokumentet, men har ligget i overvejelserne hele vejen igennem som et naturligt krav for os.



(a) Generel figur over arkitekturen (b) Arkitektur med afhængigheder

Figur 5.1: Arkitektur vist hhv. uden og med afhængigheder

For at optimere responstider, har vi i et par enkelte tilfælde lavet en ekstra kobling mellem to klasser. Vi ser dette illustreret i designmodellen (figur 6.9), hvor vi har ladet en Person have en direkte association til det område vedkommen er i. Dette giver konstanttids-information om personens lokalisering. Yderligere har vi ladet Område have direkte tilknytning til ForladtOmråde, da man på denne måde, i lineær tid, kan finde frem til alle personer, der har forladt området. Derudover er det endt med, at vi har lavet en kobling mellem Kernen og alle møder, hvorved vi kan få en hurtig tilgang til alle møder. Vi havde oprindeligt overvejet, at der altid skulle være en mødeansvarlig, hvorfor denne havde ejerskab over mødet. Dette ville i sig selv kun have givet mulighed for direkte tilgang til en enkelt persons møder. Hvis man ønskede tilgang til alle møder, skulle man iterere over alle personer og finde alle møder, de var ansvarlige for.

Pakkerne tekniske services og datatyper er udeladt af designmodellen, da der kun er en teknisk service (i en selvstændig pakke) og en datatype (i en selvstændig pakke).

5.1.1 Afhængigheder

Afhængigheder mellem de forskellige lag i vores model, ses illustreret på figur 5.1(b). I designmodellen (figur 6.9) har vi ikke vist alle afhængigheder fra UsecaseFacade-laget, idet dette ville fjerne overblikket over diagrammet.

Vi arbejder, som vist på figur 5.1(b), næsten komplet lagdelt. Et lag “snakker” kun sammen med de to underliggende lag, og der arbejdes kun nedad i arkitekturen. Dette giver stor fleksibilitet, og holder man en ensartet snitflade for hvert lag, kan man modificere metoder i de enkelte lag, uden det kommer til at betyde noget for de andre lag. Dernæst har det den store fordel, at hvis der ændres snitflade for et lag, påvirker det kun de lag, som ligger over dette lag. Dette er også en stor fordel, da man ved modifikationer i for eksempel usecaseFacade laget, højest vil få ændringer i brugerfladen.

5.1.2 Åben arkitektur

Brugerfladen, som vi ikke implementerer, forestiller vi os, skal have adgang til objekterne i Kernen, da detail-data kan aflæses, uden at vi skal lave ekstra metoder i usecaseFacade’n, som brugerfladen direkte er forbundet til. Dette betyder, at det bliver nemmere at præsentere detail-data, men også at vi får en åben arkitektur.

Et specielt problem vi løber ind i er, at hvis metoder i en package skal være tilgængelige udenfor denne, skal de være public. Hermed kan vi ikke sikre, at et lag kun bruger de metoder, som er ønskelige. Dette skyldes, at Java ikke har mulighed for, at angive hvilke klasser, der specifikt skal have adgang til bestemte metoder i en klasse, men kun om metoderne er tilgængelige indenfor og udenfor en hel package.

5.1.3 Test-pakke

Vi har lavet en test-pakke, som indeholder en klasse, der kan generere en model med testdata og en klasse med metoder, som kan udskrive forskelligt indhold i modellen.

Dernæst ligger der også en klasse, der kan teste de forskellige usecases.

Denne pakke er ikke taget med i diagrammerne, da den ikke indgår i det endelige system.

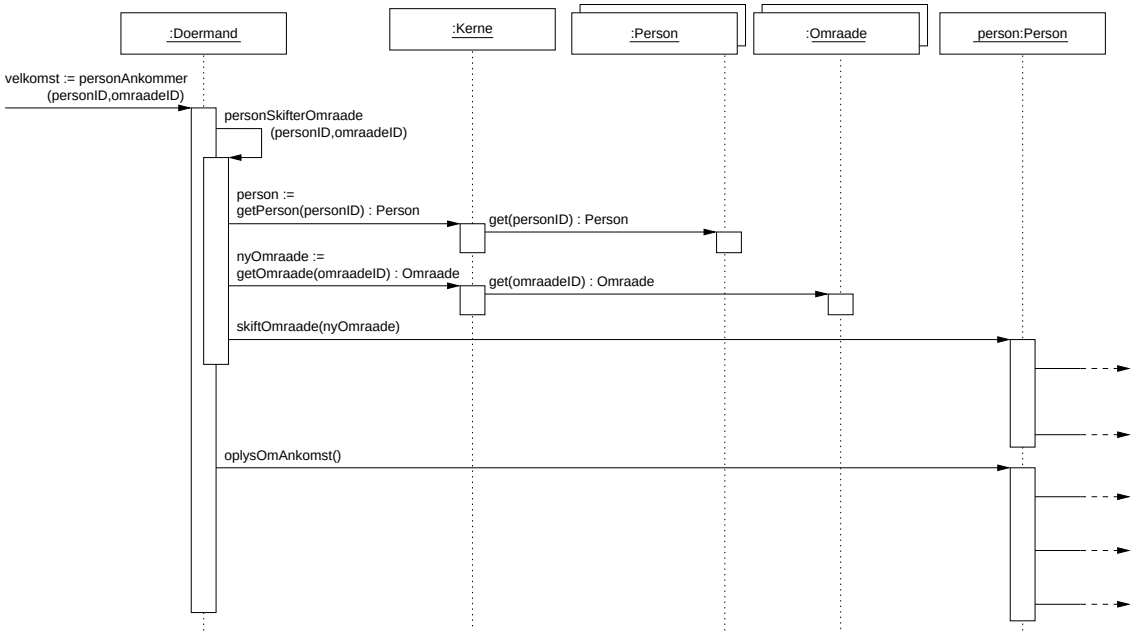
Kapitel 6

Usecase realisering

Vi vender i denne rapport nu tilbage til brugsmønstrene fra kapitel 3 og beskriver dem vha. interaktionsdiagrammer og samarbejdsdiagrammer.

6.1 Byd velkommen

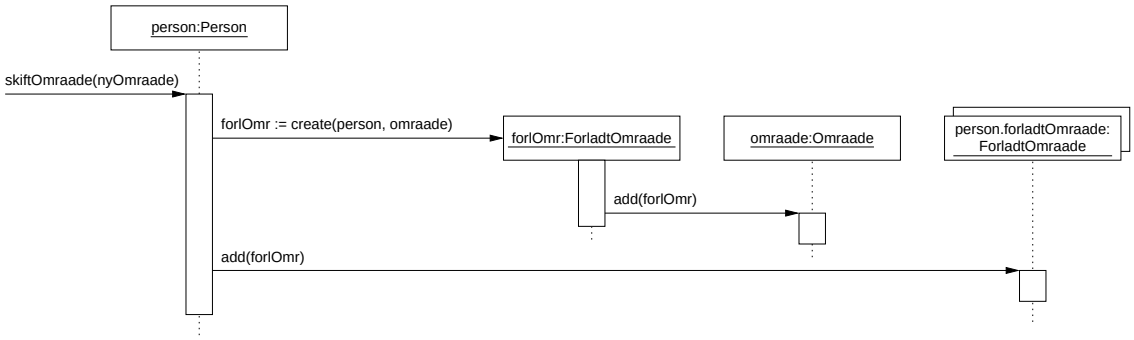
Byd velkommen scenariet ses beskrevet i figur 6.1



Figur 6.1: Byd velkommen scenariet

Scenariet starter med et kald til systemets Dørmand, hvor der bliver kaldt en metode, personSkifterOmraade(personID,omraadeID). Vi har valgt denne metode, som værende selvstændig, idet den så kan anvendes i brugsmønsteret “skift område”.

Metoden personSkifterOmraade starter med at tilgå collections af personer og områder, for at finde de objekter, som er defineret udfra de givne ID’er (ID’er identificerer unikt en person eller et område). Dette gøres vha. Kernen, da denne er Information Expert. Herefter kaldes metoden



Figur 6.2: Interaktionsdiagram for metoden skiftOmraade på Person-klassen

skiftOmraade, som kaldes på den person, som netop er ankommet. personSkifterOmraade er således færdig med sit gøremål, hvorefter der returneres fra denne metode og oplysOmAnkomst kaldes til samme person.

Vi kigger nærmere på skiftOmraade-metoden og oplysOmAnkomst metoden herunder.

6.1.1 skiftOmraade

Metoden skiftOmraade() på klassen Person ses illustreret i figur 6.2

Det første, der sker, når en Person kaldes med metoden skiftOmraade, er, at der oprettes et objekt af typen ForladtOmraade. ForladtOmraade oprettes med parametrene person og omraade, dvs. vi fortæller objektet, at en bestemt person har forladt det bestemte område. Vi har ladet Person-klassen uddelegere opgaven med at tilføje objektet forlOmr:ForladtOmraade på området til ForladtOmråde-klassen (Grasp - lav kobling). Ved at give ForladtOmråde dette ansvar, bliver koblingen mellem Person og Omraade mindre, da Person-klassen ikke behøver at kende til, at forbindelsen er tovejs. Ved f.eks. implementation af andre typer objekter, i stil med Person, kan disse lettere implementeres. Dette kræver et interface til objekter, der kan skifte områder (dette er ikke fremstillet). Samtidig kræver det, at ForladtOmråde ikke kender til Personen, men istedet blot et objekt af typen, der kan skifte område. Eksempelvis kunne det være registrering af dyrebare ting som f.eks. malerier, projekterer o.lign.[1]

Vi har lavet den beskrevne tovejs-forbindelse, for at kunne se hvilke personer, som har forladt det pågældende område, hvilket vi regner med bliver en del af områdeadministrations-brugsmønsteret.

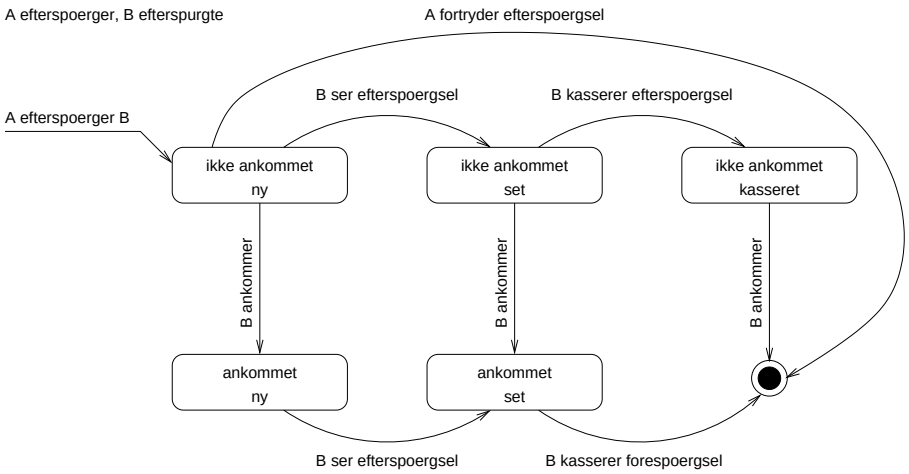
Til sidst skal Person-klassen registrere, at vedkommende har forladt området ved at tilføje det oprettede objekt til person.forladtOmraade-collectionen.

6.1.2 oplysOmAnkomst

I byd velkommen-scenariet kaldes ligeledes metoden oplysOmAnkomst på personen. Denne metode ses illustreret på interaktionsdiagrammet i figur 6.4.

For at kunne forstå oplysning om ankomst, skal bemærkes, at en person kan have et vis antal efterspørgsler. Visse efterspørgsler vil allerede være behandlet, og der skal derfor ikke gives besked til den efterspørgende. De forskellige tilstande for en efterspørgsel ses på figur 6.3.

[1] Et lignende system ønskedes implementeret for at lokalisere dyrebare bøger o.lign. materiale. Teknologien anvendt var WLAN med positionering ned til 1 meter.



Figur 6.3: Tilstandsdiagram over efterspørgsel

Når en person A efterspørger en person B oprettes en efterspørgsel. I starten er efterspørgslen i tilstanden "ikke ankommet" og "ny", svarende til, at B ikke er ankommet siden oprettelsen af efterspørgslen, og at B ikke er informeret om at have fået en efterspørgsel. Man ser, at når B ankommer, så ændres tilstanden af efterspørgsel-objektet fra "ikke ankommet" til "ankommet".

Når B ser (første gang) A's efterspørgsel, så ændrer den tilstanden fra "ny" til "set". Endelig kan B slette en "set" efterspørgsel, hvormed den ændrer status fra "set" til "kasseret".

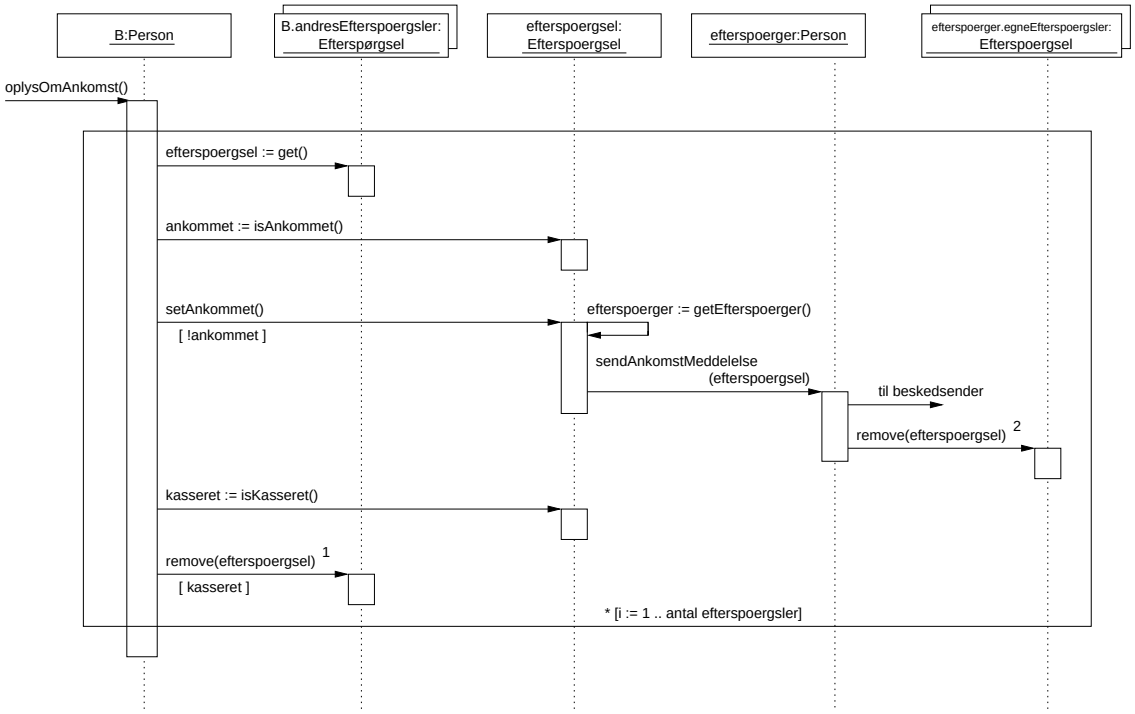
Bemærk, at en efterspørgsel bliver først rigtig slettet, når den får tilstanden "ankommet" og "kasseret". Denne tilstandens-model giver en mulighed for, at B kan se og kassere A's efterspørgsler på B, selv om B ikke er ankommet til Mærsk Instuttet (hvilket derfor må ske via et externt system).

Oplysning ved ankomst starter med, at der itereres over alle efterspørgsler på personen B, dvs. collection B.andresEfterspørgsler gennemløbes. Her udnyttes forbindelsen fra Person til Efterspørgsel.

Når en efterspørgsel er i tilstanden "ikke ankommet" kaldes setAnkommet() på efterspørgsel-objektet. Metoden setAnkommet(), liggende i klassen Efterspørgsel, sørger dels for at ændre tilstanden for efterspørgslen til "ankommet", og dels for at informere efterspørgeren om ankomsten ved at kalde sendAnkomstMeddelelse(efterspoergsel) på selve efterspørgeren. Her udnyttes forbindelsen fra Eftespørgsel til Person, her efterspørger. Vi ser ikke yderligere på sendAnkomstMeddelelse(efterspoergsel), da denne "blot" sender en besked, se bilag A.1.

Sidst bemærkes, at tilstanden for hvert efterspørgsel-objekt nu er "ankommet" og "ny/set/kasseret". Tilstanden "kasseret og ankommet" svarer til, at efterspørgsel-objekt skal slettes (se tilstandsdiagram), hvilket gøres ved at kalde et betinget remove(efterspørgsel)$^{(1)}$ på collection B.andresEfterspørgsler nævnt før. Når A (efterspørger) informeres om ankomsten af B skal A's collection egneEfterspørgsler opdateres, hvilket gøres med remove(efterspørgsel)$^{(2)}$. Dette afslutter kaldet oplysOmAnkomst.

Bemærk, at en evaluering af remove(efterspørgsel)$^{(1)}$ uanset om setAnkommet() kaldes eller ej giver en mere effektiv oprydning. Husk på, at en efterspørgsel i tilstanden "ankommet og informeret" kan komme i tilstanden "ankommet og kasseret" ved, at B kasserer efterspørgslen. Denne bliver så rigtigt slettet næste gang B ankommer.



Figur 6.4: Interaktionsdiagram over scenariet oplysOmAnkomst.

6.2 Skift område

Tilfældet, hvor en person skifter område er dækket af dette usecase. Brugsmønsteret er næsten magen til tilfældet med "Byd velkommen", med den forskel, at der her ikke skal sendes besked til personer, der har efterspurgt vedkommende. Dette kan derfor klares med et direkte kald, personSkifterOmraade(nyOmraade), på Doermand-klassen. Det medfører, at man også kan oprette en efterspørgsel selvom personen faktisk befinder sig i bygningen. På den måde vil man have mulighed for at få meddelelse om, næste gang personen ankommer.

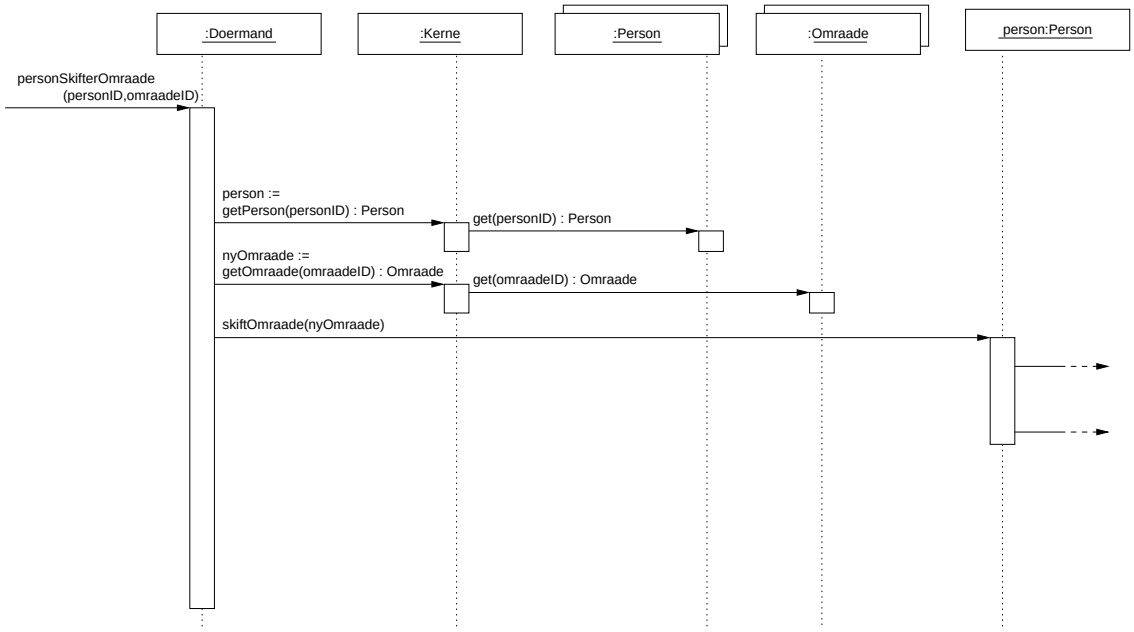
Tilfældet ses illustreret på figur 6.5

6.3 Planlæg møde

Brugsmønsteret, hvor der planlægges møde, er rent visuelt mere omfattende, hvorfor vi har illustreret dette vha. et samarbejdsdiagram. Se figur 6.6

For det første ser man, at vi har lavet en klasse Moedeplanlaegning til håndtering af hvert møde under oprettelse. Grunden er primært, at Mødeoprettelse er statefull (se afsnit om designmodel). Man ser desuden at Kernen på den måde uddelegerer et vigtigt ansvar, hvormed vi opnår lavere kobling mellem brugerflade (vi forestiller os Moedeplanlaegning som værende facade-objekt) og klasserne i systemet (mere om det i afsnit 6.6.2).

Under oprettelsen af et Moedeplanlaegnings-objekt sørger Kernen for oprettelse af et Moede-objekt (1.1.1). Den ansvarlige for mødet angives med metoden angivMoedeAnsvarlig(personID). Kernen bruges til at finde personen (2.1.1 - ifølge Expert-mønster) og mødet tilføjes til samlingen af de møder, som personen har ansvaret for (2.2). Desuden ser man, at der oprettes en reference til den mødeansvarlige i Moede-objektet (2.3).



Figur 6.5: Interaktionsdiagram over "skift område" brugsmønsteret

Efterfølgende angives hvilke personer, man ønsker skal deltage i mødet, vha. angivelse af person-ID (3*). Hver deltager får opdateret sin liste af møder (3.2) og Moede-objektets liste af deltagere opdateres (3.3). Dette foregår iterativt som vist med stjerner på figuren.

Med metoden angivDeltagerAntal(antal) (4) og angivStartSlut(start, slut) (5) opdateres simple attributter i Moede-objektet. Systemet har nu mulighed for at kunne vise de personer, som ikke kan deltage på det foreslåede tidspunkt. Dette gøres ved at iterere på deltagerlisten (6.1.1*) og se, om der opstår konflikt (6.1.2*). Output fra metoden visKonfliktPersoner() (6) er en Iterator over personer. Hvis man ikke ønsker at acceptere det angivne tidspunkt vil man have mulighed for at vælge et nyt tidspunkt - dvs. man iterere altså over (5) og (6). Denne mulighed har vi overladt til brugerfladen, eller en eventuel overliggende klasse. På denne måde sikrer vi en vis skalerbar klasse, som endnu ikke er alt for specifik.

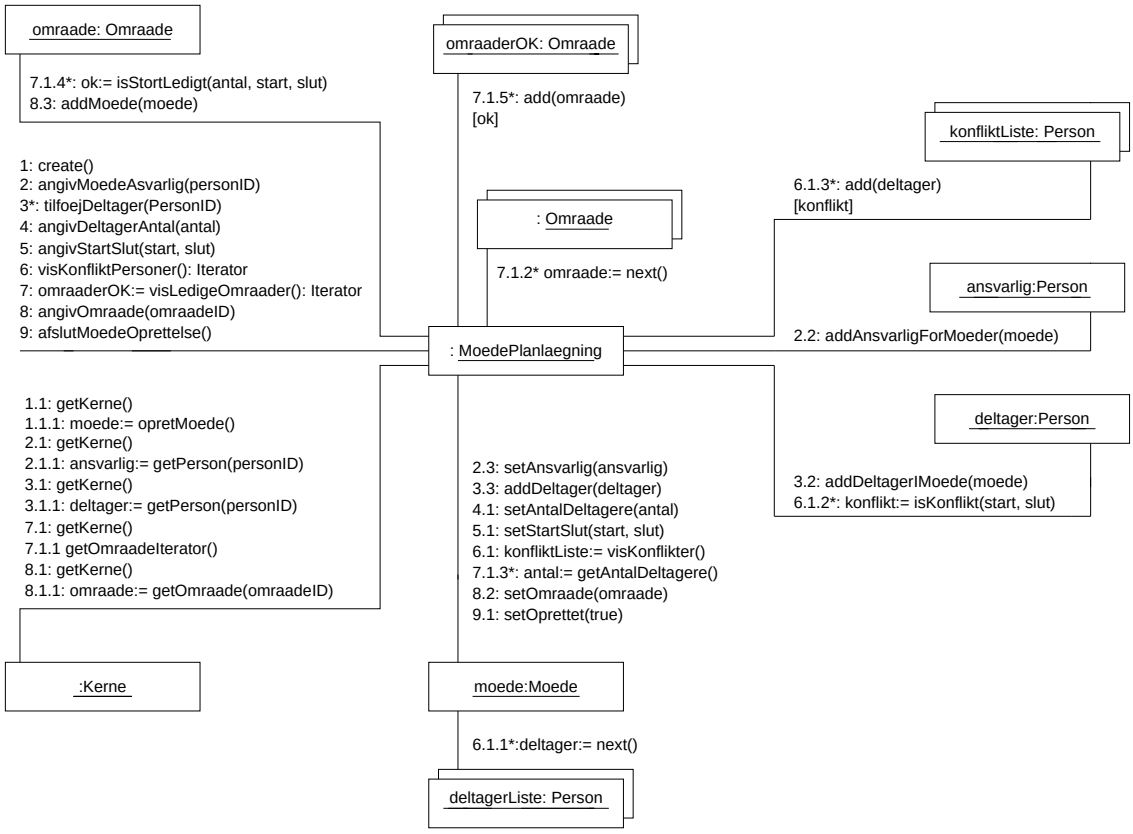
Systemet har nu de relevante oplysninger til at foreslå passende områder (7), hvor mødet kan foregå. De passende områder findes ved at gennemløbe alle områder (7.1.2*) fra en Iterator fået af kernen (7.1.1 - ifølge Expert-mønster), og se om de er passende i størrelse, og ikke optaget af andre møder (7.1.3*, 7.1.4*).

Til sidst kan det ønskede område angives (8) og mødeplanlægningen afsluttes (9). Funktionen setOprettet() (9.1) er taget med for ikke at angive området som optaget, før oprettelsen faktisk er afsluttet.

6.4 Efterspørg person

Efterspørgsel indeles i to dele.

Første del (efterspoergPerson(personID)) dækker lokalisering af den efterspurgte person, se figur 6.7. Anden del (registrerEfterspoergsel(egetID,efterspurgtesID)) dækker registrering af efterspørgsel. Dette ses hhv. illustreret på figur 6.7 og 6.8. Opdelingen skyldes et ønske om, at kunne lave en efterspørgsel uden at skulle registrere den.



Figur 6.6: Samarbejdsdiagram over mødeplanlægning

Personen lokaliseres ved først at få fat i selve Person-objektet med metoden getPerson(personID) på Kernen (iflg. Expert-mønster). Efterfølgende findes hvor personen er, ved at kalde getOmraade() på Person-objektet.

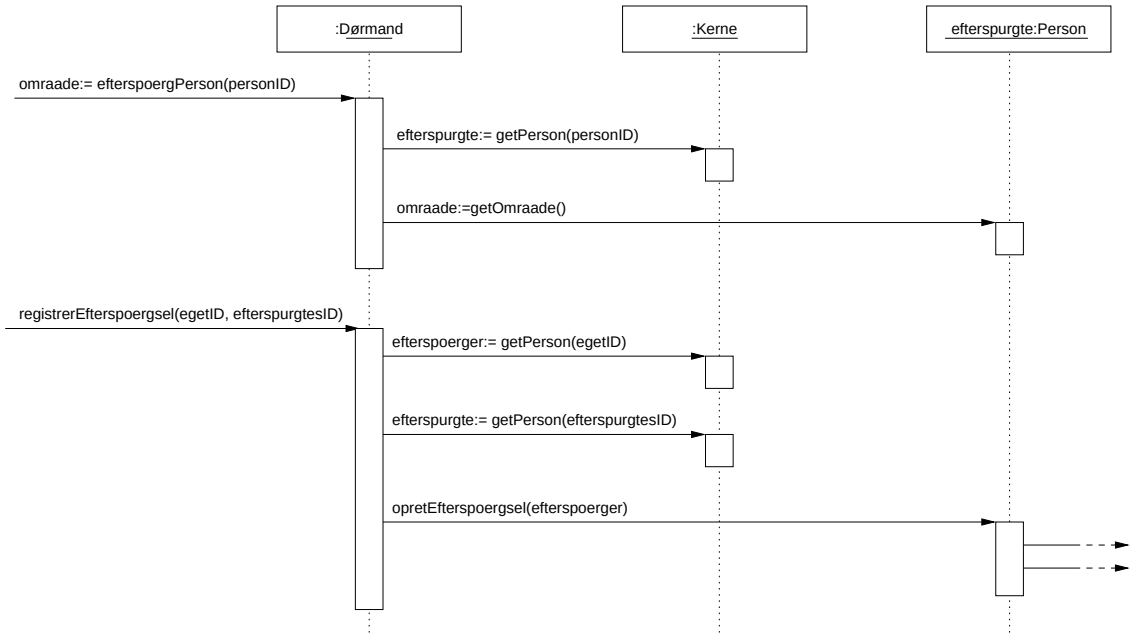
Registrering starter med, at Person-objekterne efterspørger og efterspurgte, matchende hhv. egetID og efterspurgtesID findes. Herefter kaldes metode opretEfterspørgsel(efterspørger) på den efterspurgte (se figur 6.8)

Personen, den efterspurgte, opretter en efterspørgsel. Efterspoergsel-objektet får ansvaret for at tilføje en forbindelsen mellem efterspørger og efterspørgsel. Dette sker ved at addere efterspørgslen til efterspørgerens collection'en egneEfterspørgsler. På denne måde anvender vi altså samme mønster som ved skiftOmraade()-metoden (der er det dog klasserne Person, ForladtOmraade og Omraade). Her kunne den efterspurgte sørge for at oprette associationen mellem Efterspoergsel og efterspørger, men vores løsning giver mulighed for udvidelse af systemet, så f.eks. andtre typer objekter end Person kan oprette efterspørgsler. (Dette kræver interface til objekter, der kan oprette efterspørgsler og blive efterspurgt, hvor vi har diskuteret lignende tidligere).

Sidst adderer den efterspurgte efterspørgslen til sin collection af andres efterspørgsler.

6.5 Anmod besked

Scenariet anmod besked ligner til forveksling "efterspørg person". Dog er der den forskel, at efterspurgte ikke skal kunne se "efterspørgslen". Dette gøres ved at ændre efterspørgslens tilstand



Figur 6.7: Interaktionsdiagram over efterspørgsel af person

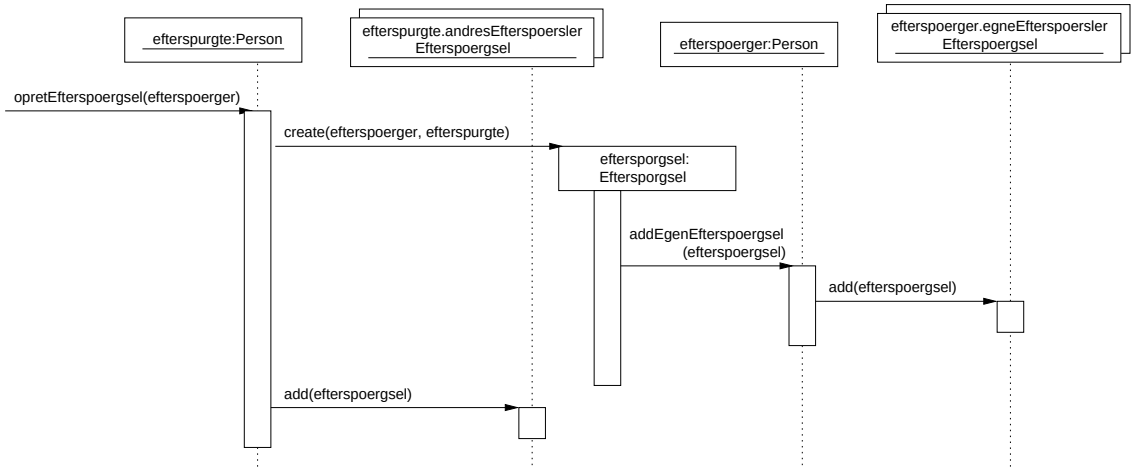
til "kasseret"[2].

I det alternative scenarie, hvor den "efterspurgte" person allerede er i bygningen, vil det være controller-objektets/brugerfladens opgave, at registrere efterspørgslen hvis den "efterspørgende" person ønsker besked ved næste ankomst.

Scenariet ses illustreret i bilag A.2.

Det kan ses, at funktionen opretAnkomstMeddelelse kaldes på den "efterspurgte" (se bilag A.2, diagram 2). Dette scenarie ligner meget opretEfterspoergsel. Der er dog den forskel, at efter objektoprettelse af efterspørgsel, skal "efterspørgslen" markeres som værende "kasseret", hvilket kan foretages med metode-kaldet "setKasseret".

[2]Efterspørgsler kan have tilstande. Se tilstandsdiagram 6.3. Vi har valgt at efterspørger ikke kan se netop de kasserede efterspørgsler



Figur 6.8: Interaktionsdiagram over oprettelse af efterspørgsel.

6.6 Designmodel

Designmodellen er illustreret i figur 6.9.

Som det ses, er der til forskel fra domænemodellen tilføjet et par klasser. Der er hhv. kommet et par facade-objekter til håndtering af use-cases. Derudover er der kommet specifikke tekniske klasser til håndtering af beskedafsendelse.

Designmodellen er endvidere uden angivelse af alle objekt/data-typer, da dette har givet tegnetekniske problemer (det kunne ikke være der i et diagram, hvis teksten skulle kunne læses) og mindsker overblikket. Vi har parallelt med fremstilling af diagrammet lavet signaturen til Java-koden. Derfor har vi ikke følt et behov for detaljeret type-angivelse i diagrammet.

De enkelte lag i arkitekturen ses i figur 5.1(a). I det følgende har vi beskrevet designet for hvert af lagene.

6.6.1 Kerne-pakke

Vi har taget de fleste elementer fra første iteration og valgt at lægge disse i en pakke for sig (med undtagelse af klassen Mødeplanlægning). Klasserne i denne pakke har stærke relationer til hinanden og danner tilsammen en “kerne” for systemet.

Kerne

Vi har valgt at implementere en klasse Kerne, som er blevet tildelt ansvaret for at indeholde oplysninger om alle personer, områder samt møder i systemet. Dette betyder, at hvis man skal finde en person udfra f.eks. et ID er det altså Kernen, der har ansvaret for at give denne information videre (information expert).

Kernen er en samlingsklasse, der samler collections af områder, møder og personer. Alternativt kunne vi have valgt at lave tre samlingsklasser, der hver især stod for samling af netop en type objekter. Vores implementation giver dog den fordel, at det er meget simpelt at lave persistens vha. Java’s serialisering, hvor vi så kan nøjes med at gemme Kerne-objektet.

UsecaseFacade-controllerne har på denne metode kun en klasse de behøver tage udgangspunkt i, for at få tilgang til objekter. Grunden til at Møder er knyttet til kernen, og ikke den mødeansvarlige, er, at dette giver mulighed for at få en nem tilgang til en oversigt over alle møder. Derudover er der nu en bestemt ansvarlig for alle møder, hvilket stemmer overens med Grasp-mønstrene “Information Expert” og “Creator”.

Hvis en usecase f.eks. skal finde en Person udfra et person-ID, er det ikke nødvendigt, at man medsender collectionen af personer som parameter til funktionen. Man kan blot spørge Kernen (ifølge Expert-mønster). For at man ikke ender op med at skulle medsende kernen som parameter (og dermed øge koblingen), har vi valgt at oprette denne som en Singleton.

6.6.2 UsecaseFacade-pakke

Pakken Usecase facade danner en pakke, der indeholder facade-klasserne, en eventuel bruger-flade kunne tilgå. Pakken indeholder klasserne Mødeplanlægning, Dørmand samt Efterspørgsels-behandler.

Mødeplanlægning

Mødeplanlægningsklassen er noget anderledes end Dørmanden da denne skal behandle en usecase, planlæg møde, der er statefull. Mødeplanlægningen er statefull, da der skal interageres med brugeren undervejs, og huskes hvor langt vi er nået.

Dette betyder, at der skal oprettes en instans af mødeplanlægningsklassen, hver gang en mødeoprettelse starter. På denne måde kan der altså være flere mødeplanlægninger i gang ad gangen. Når mødeoprettelsen er slut, er mødet oprettet og associeret til mindst en person, og referencen til objektet kan derfor kastes væk.

Dørmand

Dørmand er en facade, for usecases som "ankomst" (personAnkommer(personID,områdeID)) og "skift område" (personSkifterOmråde(personID,områdeID)). Desuden er den facade for usecases "efterspørg person" (efterspørgPerson(personID) og registrerEfterspørgsel(egetID, efterspurgtesID)) og "anmod besked" (registrerAnkomstMeddelelse(egetID, efterspurgtesID)) implementere i denne klasse. Ydermere er usecase'n "send besked" ligeledes implementeret i denne facade. Metoden sendBesked(personID,objekt) håndterer dette usecase. Det er korrekt at der ikke angives afsender, da man kan se i usecase/aktør tabellen, at gæster også skal have mulighed for at sende beskeder.

EfterspørgselsBehandler

Efterspoergselsbehandleren administrerer ligesom Mødeplanlægningsklassen kun et usecase, "er jeg efterspurgt", som også er statefull, pga. interaktionen med brugeren. Man kan starte et gennemløb af efterspørgsler, som kan være nye efterspørgsler eller sete efterspørgsler. Herefter er det muligt at spørge på, om der er flere nye/sete efterspørgsler. Derudover kan der forespørges om den næste nye/sete efterspørgsel. Dvs. EfterspørgselsBehandleren virker som en slags Iterator. Der er yderligere mulighed for at kassere hhv. en set efterspørgsel og alle sete. kasserSet() kasserer den netop sete efterspørgsel. Alle sete efterspørgsler slettes med kasserAlleSete().

6.6.3 Beskeder-pakke

Pakken beskeder er en pakke, som indeholder tekniske services.

BeskedSender

BeskedSender er en generel klasse, der står for afsendelse af beskeder af forskellige typer. Klassen har implementere metoderne getBeskedSender(), som er en metode til at få fat i instansen af klassen, da denne er implementeret som singleton.

Herudover findes kun en metode, nemlig send(medieIterator, besked). Metoden sørger for at sende en besked til det først mulige medie, der er muligt at sende til (Iterator givet i prioriteret rækkefølge). Selve BeskedSenderen er derfor nødsaget til at kende de forskellige medie-typer for at kunne videredelegere arbejdet til en adapter-klasse, der kan udnytte en teknisk funktionalitet enten i systemet eller som et eksternt system.

Strategi-mønstret er anvendt her, da man udvælger sendemetode på baggrund af medie-type og besked-type (tale, tekst).

Vi har angivet tre eksempler på besked-afsendelses klasser, nemlig MailSender, SMSSender og TaleSender. Man kan måske undre sig over, at TaleSender er valgt som eksempel. Der er flere forskellige muligheder for at lave talebeskeder. Det skal dog siges, at vi ikke har tænkt os at implementere andet end tekstbeskeder, hvor besked-objektet her vil være et streng-objekt. Det kan dog synes mærkeligt, at vi har valgt at give et eksempel på TaleBeskeder, hvis man i vores implementation kun vil kunne sende tekst-beskeder. Det er dog teknisk muligt at omsætte tekst til tale[3]. Det kunne f.eks. være andre tekniske services, at TaleSenderen kunne benytte.

SenderAdapter

SenderAdapter er med til at danne Adapter-mønsteret i systemet. Vi har valgt denne implementation, da den giver en større frihed i BeskedSenderen, idet snitfladen til alle de tekniske klasser som MailSender, SMSSender og TaleSender vil være ens.

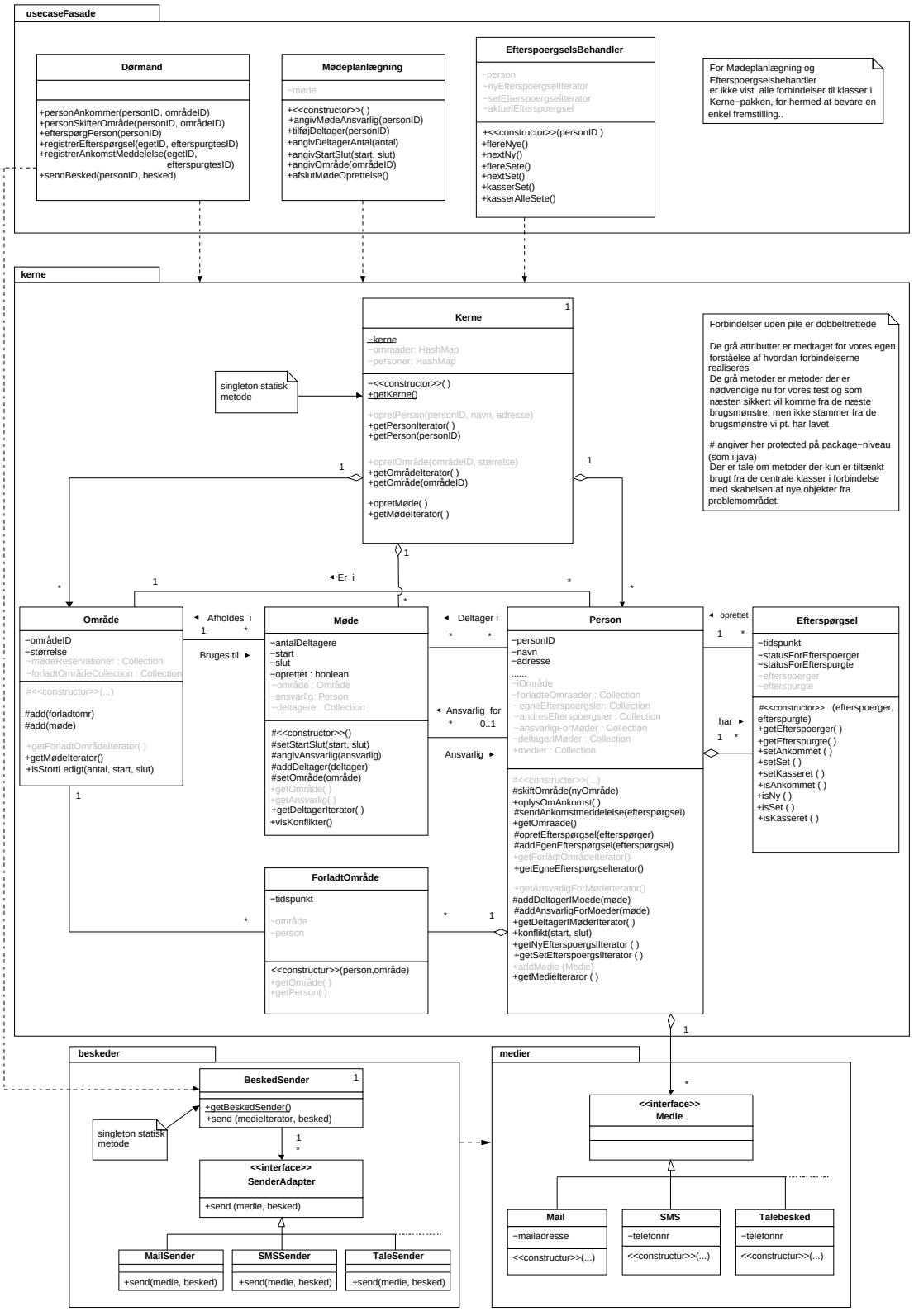
6.6.4 Medier-pakker

Vi har ligeledes oprettet en pakke til medie-beskrivelse, da alle disse medier er stærkt relaterede. Da der med tiden kan komme flere forskellige medier, har vi valgt at lægge "personers attributter" ned i disse klasser, da person-klassen på denne måde er mere stabil. Dette betyder, at hvis en person ændrer kontakt-oplysninger ændrer det ikke på person-klassen, men kun på associationer til personen. Telefonnumre, mail-adresser o.lign. gemmes altså på medie-objektet, hvilket også giver mindre kobling mellem Kerne-pakken (Person) og BeskedSenderen, idet BeskedSenderen kun skal kende til medie-typerne og ikke person-objektet, som beskeden skal sendes til.

Medie

Medie er et interface for de forskellige medier, og giver på denne måde mulighed for, at Person-klassen ikke skal kende yderligere til de forskellige medier. Alt i alt giver det mindre kobling, og ved tilføjelse af medietyper skal der altså ikke rettes andre steder i designet (udover i BeskedSender hvis mediet skal kunne benyttes til beskedafsendelse). Selvfølgelig giver det ændringer i brugerflade, da denne også skal kende til mediet. Eventuelt kan dette også gøres dynamisk ved at have en liste over medier, som er tilknyttet en beskrivelse og et navn. Dette vælger vi dog ikke at tænke på her, da det er et problem, som vi ikke skal varetage i dette projekt, hvor vi ikke må designe brugergrænseflade.

[3] TDC har services som kan omsætte SMS til talebeskeder, og som er simpel at udnytte ved at skrive 2 foran telefonnummeret når beskeden afsendes.



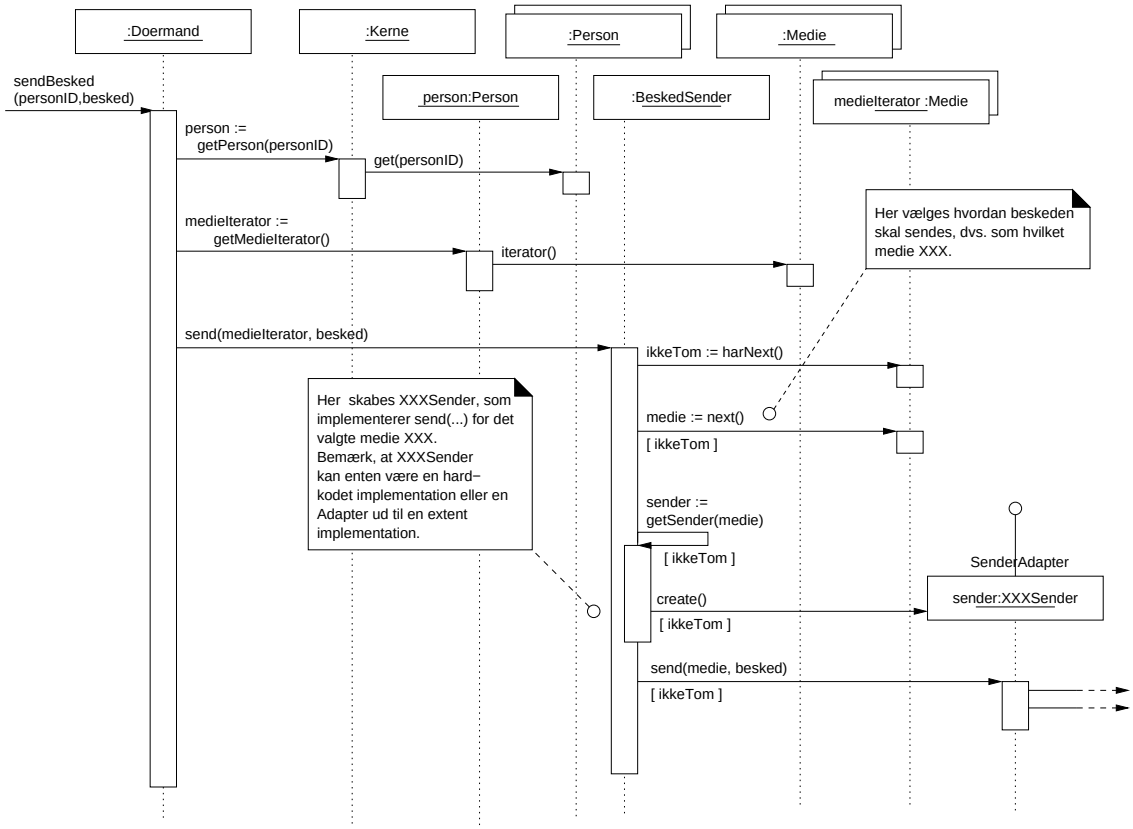
Figur 6.9: Designmodel implementeret

Figurer

Bilag A

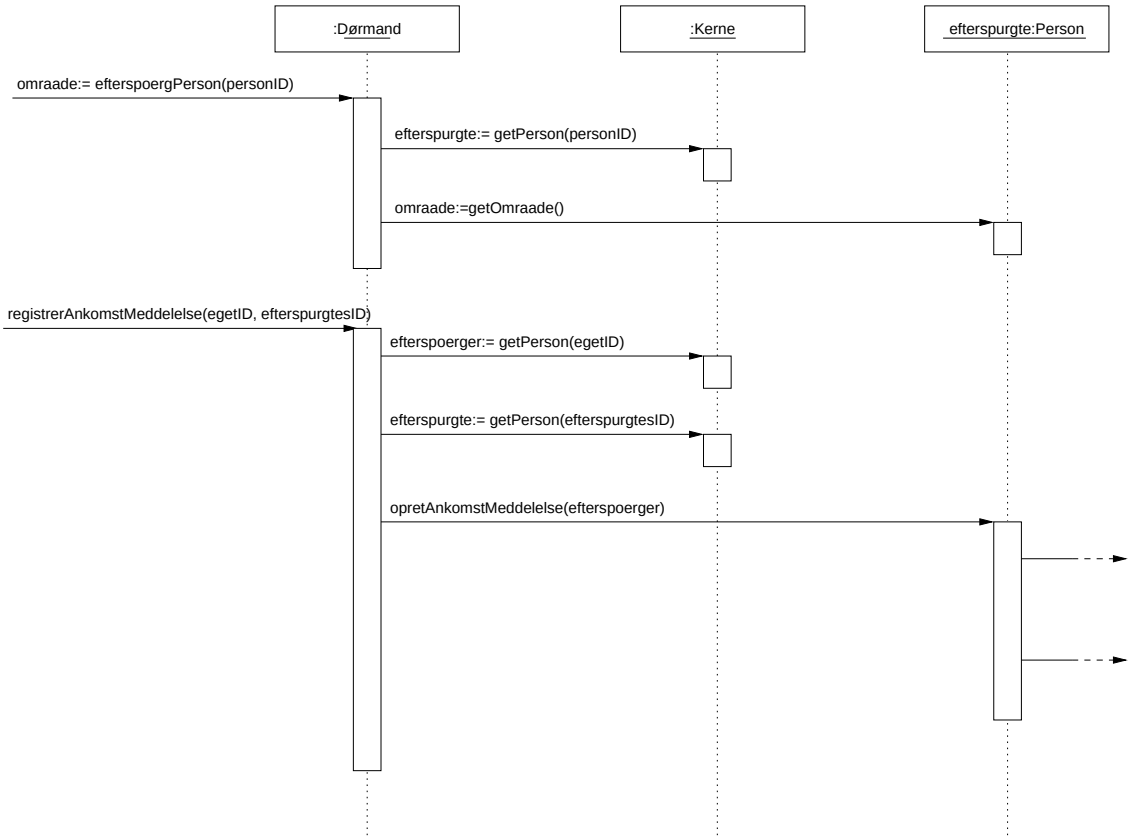
Bilag

A.1 Send besked

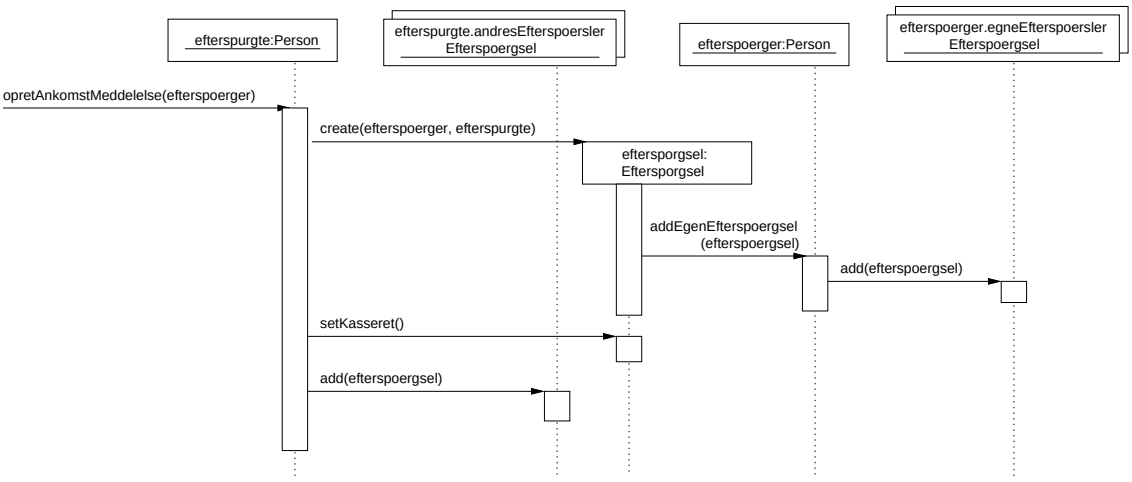


A.2 Anmod besked

Interaktionsdiagram over anmod besked.



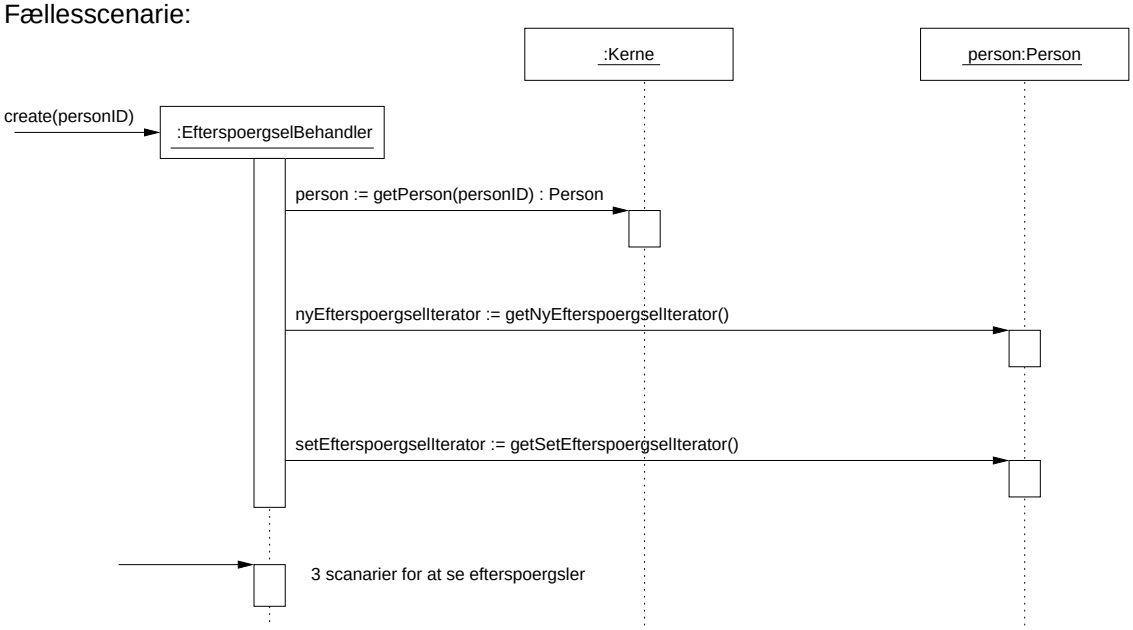
Interaktionsdiagram over opretAnkomstMeddelelse på Person-klassen.



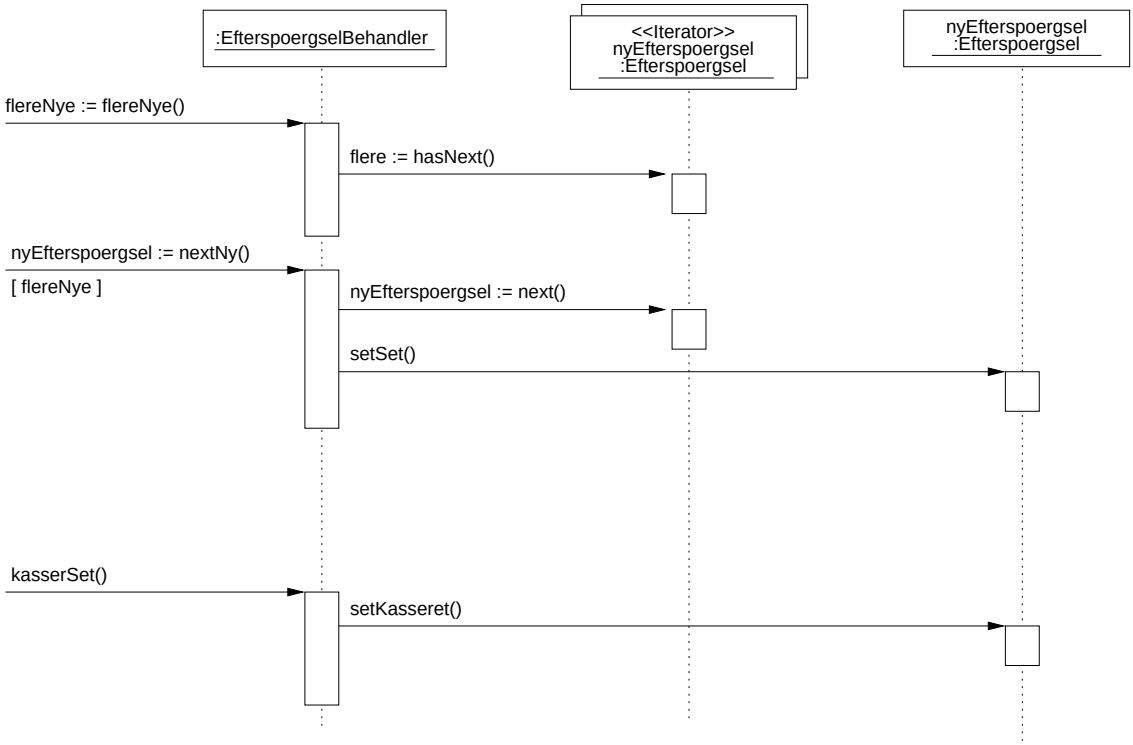
A.3 Er jeg efterspurgt

Interaktionsdiagram for usecases er jeg efterspurgt.

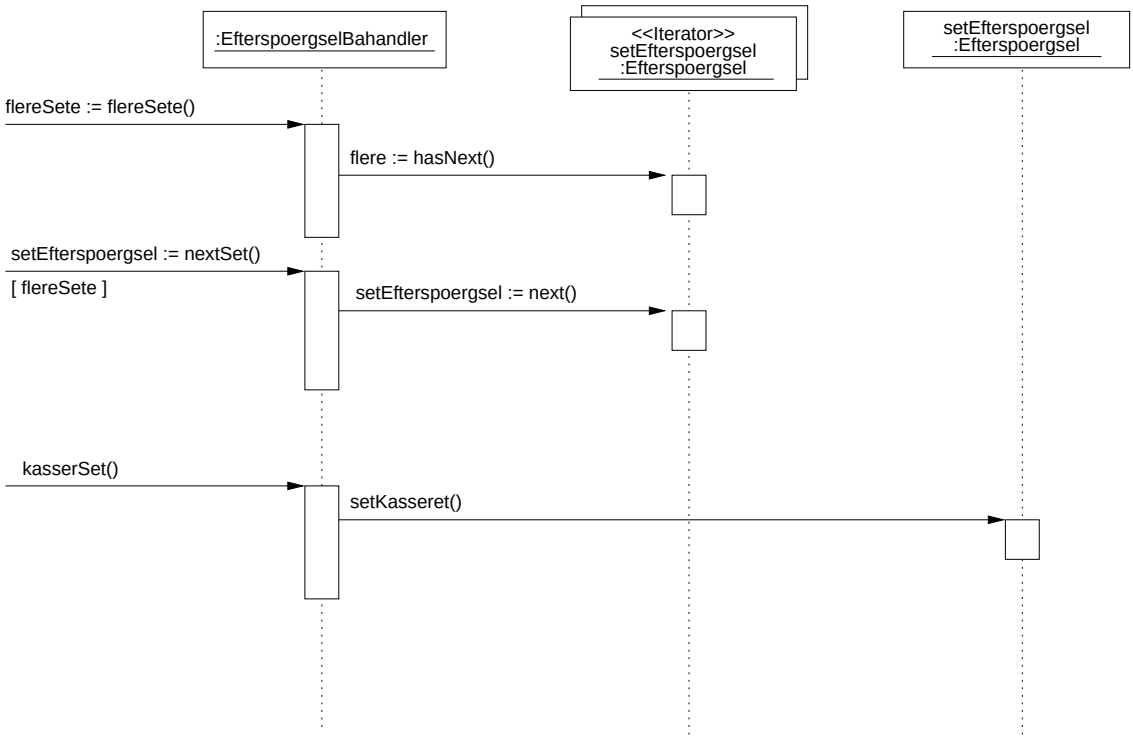
Fællesscenarie:



Hovedscenarie (gennemse nye):



Alternativt scenarie (gennemse sete):



Alternativt scenarie (slet alle sete):