Kalender og aftalebooking til K. Translation

Omar Khalidan - [xx-xx-xx]
Morten Trolle Nielsen - [14-06-79]

Instruktor: Andreas Frisch

Projekt i systemudvikling 2014 (ProjDat2014)

21. april 2014

Indhold

1 Abstract

Dette er anden delrapport i faget Systemudviking. Vores udgangspunkt er første delrapport, som vi her udvikler og udbygger. Vi vil foresætte arbejdet med og præsentationen af den IT-løsning, som vi har udviklet til kunden; tolkeservicen K. Translation. Specielt vil vi fokusere på modelleringen af systemet. Vi vil præsentere anvendelsesområdet og problemområdet; både med tekst og UML-diagrammer. Det skal resultere i en bedre og umiddelbar forståelse af vores påtænkte it-løsning. Derefter vil vi også med afsæt i første delrapports kravspecificering videreudvikle denne og se på de funktionelle og nonfunktionelle krav. Vi vil tage modelleringen et skridt videre ved hjælp af use-cases, så vi kan understøtte vores modellering med konkrete eksempler på brug af systemet.
Vi vil afslutte denne anden delrapport med to korte referater og efterfølgende analyse og perspektivering af de to artikeler, der er et krav til rapporten.
Vi er selv førsteårs datalogistuderende ved Københavns Universitet, og det forudsættes, at læseren befinder sig på samme læringsniveau eller højere, idet der undervejs vil forekomme adskillige fagspecifikke termer.

2 Problemformulering

Tolkeservicen K. Translation er et lille, enmands tolkebureau med speciale i arabiske sprog. Indehaveren af bureauet er , og hun er som sagt både leder og eneste medarbejder i foretagendet. Hendes kunder er offentlige myndigheder som politiet og anklagemyndigheden og private aktører som advokatbureauer og forskellige virksomheder. Disse instanser kontakter K. Translation, når de har behov for en tolk til konkrete sager eller ved andet forefaldende tolkearbejde, og der laves en aftale. Det kan både være aftaler to eller tre uger ude i fremtiden, eller det kan være aftaler med kortere frist ved mere presserende sager.
K. Translation (KT) har i længere tid ønsket et it-system, som kan afhjælpe lidt af den daglige travle arbejdsbyrde. Derfor tog ejeren af KT kontakt til os, og anmodede om et møde omkring dette problem. Det første møde fandt sted fredag den 21. marts, og vi blev enige om at begynde et samarbejde, der skal resultere i et færdigt kalender- og bookingssytem til KT og KT's kunder. Kunden præsenterede os for en liste af de funktioner, som hun ønsker, at det nye system skal kunne levere. For det første ønskes der en hjemmeside, hvor KT's kunder kan logge på systemet med deres EAN-nummer. Da vi bl.a. snakker om offentlige myndigheder som politi og anklagemyndighed, skal login systemet fremstå professionelt og sikkert. Der skal være en tilhørende database indeholdende alle KT's kunder, og det skal være muligt løbende at tilføje kunder til databasen.
Når KT's kunde har logget på systemet, skal vedkommende præsenteres for en kalender opslået på dags dato med mulighed for visning af de næste to til tre uger. Det er KT's holdning og erfaring, at der ikke forekommer aftaler længere end tre uger ude i fremtiden; derfor denne begrænsning. KT's kunde skal i kalenderen kunne se alle de tidspunkter, hvorpå det er muligt at booke en aftale med K. Translation. Disse ledige tidspunkter kan evt. markeres med grønt. Allerede indgåede aftaler markeres med rødt for at tydeliggøre denne forskel. Når KT's kunde har besluttet sig for en dato og et tidspunkt, taster vedkommende dette ind i systemet. Derefter skal kunden oplyse typen på arbejdet; om det er mundligt eller skriftligt tolkearbejde, simultantolkning eller andet arbejde. Aftalen gemmes så i databasen og vises herefter i kalenderen. Det er vigtig for KT's ejer, at hendes samarbejdspartnere ikke skal bruge alt for lang tid på at booke en aftale, så enkelhed og intuitivitet skal prioriteres højt.

3 Projektaftale

Inden vi fortsætter med modelleringen og designet af systemet, vil vi beskrive præcist, hvilke krav og forventninger kunden K. Translation har til vores endelige løsning. Vi har delt den følgende kravspecifikation op i to dele: high-level funktionelle krav, som dækker samspillet mellem anvendelsesområdet og kalender- og bookingsystemet uden implementeringsspecifikke detaljer. Og nonfunktionelle krav, der dækker over ikke direkte funktionelle aspekter som bl.a. brugervenlighed og pålidelighed. Idet kravspecifikationen senere danner grundlag for afprøvningen af systemet, er det vigtigt, at den er "komplet, konsistent, entydig og korrekt."[1][p. 122]

3.1 Funktionelle krav

På det første kundemøde beskrev indehaveren af K. Translation, hvilke forventninger og ønsker hun havde til det færdige produkt: et integreret kalender og bookingsystem. Det dannede grundlaget for kravspecifikationen, som nu udgører rammerne indenfor hvilke, vi udvikler systemet. De funktionelle krav kan ses i den efterfølgende liste.

- K. Translation ønsker en it-løsning, der indeholder en kalender og et bookingsystem, som af bureauets kunder kan tilgås over intetnettet.
- På hjemmesiden skal KT's kunder præsenteres for en login menu, hvor de logger på systemet med virksomhedens eller institutionens EAN-nummer og et selvvalgt password.
- Efter succesfuldt login tilgår kunden kalenderen. Det skal være muligt for kunden at søge i kalenderen efter ledige tidspunkter udfra kriterier som dato og klokkeslet.
- KT's kunde skal vælge typen på arbejdet inden for et antal forudbestemte kategorier.
- Der skal sendes en bekræftende email tilbage til kunden, efter at vedkommende har booket en tid i kalenderen. Det blev af KT foreslået, at kunden modtager et vCard

- Det skal være muligt for K. Translation løbende at tilføje flere kunder eller samarbejdspartnere til databasen.
- KT's indehaver vil gerne kunne overføre sine private aftaler fra en ekstern kalender til vores kalendersystem. Hun forestiller sig en form for synkronisering, men vi har fra start gjort hende det klart, at dette måske ligger udover det mulige. Se evt. afsnittet om projektplan, hvor vi diskuterer prioriteringer.

Vi har i samarbejde med KT udviklet den ovenstående liste af funktionelle krav, men KT havde fra start et godt billede af, hvilke funktioner og egenskaber det færdige produkt skulle indeholde. Sammen identificerede vi først de aktører, der skal kunne tilgå systemet; altså i virkeligheden anvendelsesområdet. Derefter beskrev KT's indehaver en række typiske scenarier fra hendes daglige arbejde, hvor kunder til bureauet booker en tid hos tolkeservicen. Udfra disse scenarier kunne vi opstille flere af de funktionelle krav, og dette arbejde kulminerer iøvrigt senere i rapporten i afsnittet omhandlende use-cases. Vi synes, at de funktionelle krav på en god og dækkende måde beskriver de ønsker, som K. Translation har til det færdige program. Det skulle være muligt at teste, om vores løsning lever op til alle ovenstående krav, når systemet senere skal valideres på baggrund af kravspecifikationen.

3.2 Non-funktionelle krav

Indehaveren af K. Translation kom også med nogle forventninger og ønsker, som ikke umiddelbart hører til systemets funktionelle egenskaber. Det er bl.a. forventninger om brugervenlighed, pålidelighed og andre svært verificerbare egenskaber. Disse ønsker har vi samlet her i afsnittet med nonfunktionelle krav sammen med mere implementeringsspecifikke detaljer. Den efterfølgende liste indeholder de nonfunktionelle krav til systemet.

- Kalender og bookingsystemet skal placeres på en hjemmeside.

- Hjemmesiden skal indeholde K. Translations kontaktinformationer. Der er ikke stillet særlige krav til designet udover, at det skal være enkelt og intuitivt, så personer uden store it-forudsætninger kan navigere på siden og booke en tid.
- Alle brugere skal kunne tilgå systemet med en almindelig web browser som Internet Explorer, Firefox eller Google Chrome.
- KT's kunder skal hurtigt og uden mulighed for misforståelse kunne skelne ledige tidspunkter i kalenderen fra optagede. Derfor skal ledige tidspunkter markeres med grøn skrift eller farve, og allerede indgåede aftaler markeres med rødt.[1]
- K. Translation ønsker den letteste og billigste implementering og hosting af systemet. Hun vil have et system, der ikke kræver løbende vedligeholdelse, men som bare "kører og passer sig selv".
- Da KT's kundegrundlag bl.a. udgøres af offentlige myndigheder og ministerier, må der ikke være tvivl om systemet integritet og pålidelighed.
- Der ønskes præcis og let forståelig dokumentation af systemet. Især ønsker indehaveren dækkende, enkle manualer i tilfælde af, at hun selv må stå for den løbende vedligeholdelse af systemet og i tilfælde af svigt og andre fejl.
- Kalendersystemet og backend databasen må ikke tabe data eller aftaler.

Som det fremgår, vil flere af de nonfunktionelle krav være svært verificerbare. Det er meget svært at fastslå med nogen som helst sikkerhed, hvornår en bruger opfatter en hjemmeside som enkel, intuitiv og let at overskue. Man kan stille parametre op, fastsætte gennemgående regler for designet og bruge afprøvede metoder, men der kan alligevel ikke gives nogen garantier. Derfor vil vi også i en senere rapport bruge tænke-højt-forsøg for at få en ide om, hvor godt eller dårligt vores design er i forhold til disse egenskaber. Andre af de nonfunktionelle krav udspringer af manglen på it-kompetancer i bureauet og af fraværet af en systemadministrator. Dette betyder højest sandsynligt, at vi placerer det færdige system hos

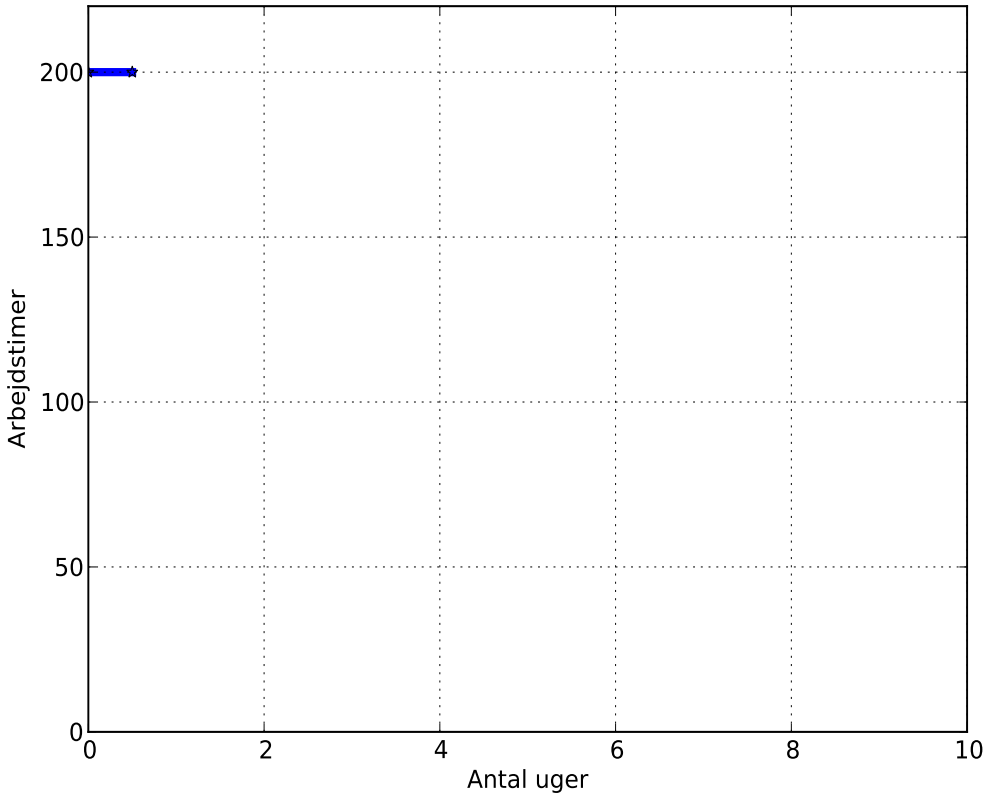
[1] Der kan godt argumenteres for, at dette punkt er en funktionel egenskab ved systemet, men vi har placeret det under de nonfunktionelle krav, da bl.a. omhandler brugervenlighed og implementeringen.

en udbyder af serverplads, da KT i så fald kommer til at stå for et minimum af systemvedligehold. Det vil også være den billigste løsning, da anskaffelsen af en server, strøm og evt. køling til en sådan vil resultere i en betragtelig udgift.
Da belastningen af systemet ikke forventes at blive særlig høj, kar KT. ingen performancemæssige krav til vores løsning. Der er ingen tids kritiske brugerfunktioner, og risikoen for samtidig brug af kalenderen må forventes at være minimal. Man kunne godt forestille sig to kunder, der vil booke den samme tid i kalenderen med race condition og deadlock til følge, men backend databasens transactionindstillinger burde kunne håndtere dette tilfredsstillende. Pålidelighed er på den anden side et vigtigt og naturligt issue, da K. Translation ikke kan acceptere, at aftaler tabes eller forsvinder.

4 Projektplan

Adskillige forelæsninger i Systemudviklingskurset har handlet om agil projektledelse og systemudvikling. Vi har fundet en sådan iterativ og inkremental tilgang til projektet spændende, og kunne derfor godt tænke os, at systemudvikle inden for rammerne af de agile principper, som vi bl.a. har stiftet bekendsskab med i artiklen "Jeff Sutherland's Scrum Handbook"[2]. Det vil dog ikke være muligt, at gennemføre projektet i komplet overensstemmelse med Scrum og alle de agile regler. For det første vil det betyde, at vores kunde skal afse betydelig mere tid til projektet, end hun umiddelbart har planlagt, hvis hun løbende skal opdate "Product Backlog" og deltage i prioriteringsmøder ved hver sprints begyndelse. Derudover vil det ikke være realistisk, at vi selv holder daglige Scrum møder, og at vi kan levere al den dokumentation et virkeligt Scrum forløb forudsætter som f.eks. de daglige overslag over vores egne fremskridt i forhold til de påtagede opgaver i den aktuelle sprint. Derfor vil vi slække på nogle af reglerne, og vi håber at kunne gøre det uden at bevæge os alt for langt væk fra den virkelige agile Scrum systemudvikling.
I stedet for de daglige estimeringer over projektets fremadskriden, har vi valgt at nøjes med et overordnet burndown diagram for hele projektet. Se figur 1. (Et brundown diagram for en enkelt sprint vil være magen til, men værdierne på førsteaksen vil være dage i stedet for uger, og værdierne på andenaksen vil være mindre).

Vi har som udgangspunkt afsat mellem 8 og 10 uger til det agile projektforløb og bedømt den påkrævede arbejdsindsats til at være 200 timer, hvilket vil sige 100 timer til hver, da vi er to mand i gruppen. Dette overslag er dog yderst usikkert,



Figur 1: Burndown diagram over projektforløbet

men vi har aldrig prøvet at arbejde på denne måde før, så det bliver spændende at se, om vores estimationer bliver mere præcise undervejs. Vi har planlagt at udvikle i sprints af 14 dages varighed, men vi kan udvide dette til 3 uger, hvis et af punkterne i produkt backloggen forekommer mere omfangsrigt.
Indehaveren af K. Translation har ofte meget travlt og kan som sagt ikke deltage i hvert nyt sprintmøde. Derfor har vi besluttet at simulere disse møder ved, at vi selv skriver og opdater produkt backloggen og gør det med udgangspunkt i kravspecifikationen. Vi bliver herved in effect vores egen proxykunde ! Den indledende produkt backlog kan ses i tabel 1.

Punkt	Prioritet	Værdi	Indsats
Som bruger af KT's hjemmeside bliver man præsenteret for en kalender, hvori man kan booke en aftale med KT.	1	Høj	
Som KT's kunde møder man et login interface, når man navigerer til hjemmesiden.	2	Høj	15
KT skal have muligheden for løbende at tilføje kunder til databasen.	3	Høj	
Man skal som kunde modtage en bekræftende email efter at have lavet en aftale i kalenderen.	4	Middel	
Som kunde bliver man præsenteret for en lille menu, hvor man skal vælge typen på arbejdet.	5	Lav	
Som kunde skal man kunne søge i kalenderen ved hjælp af dato eller klokkeslet.	6	Middel	
Indehaveren af KT vil gerne kunne synkronisere sin eksterne kalender med hjemmesidens kalender og på den måde overføre sine private aftale.	7	Lav	
KT ønsker en hjemmeside med kontaktinformation, billeder og et enkelt design.	8	Middel	12

Tabel 1 Produkt Backlog

KT's ønsker står i prioriteret rækkefølge, og vi har yderligere tilføjet en kolonne, hvor KT kan estimere nytteværdien af punktet med Høj, Lav eller Middel. Sidste kolonne viser vores bedømmelse som udviklere af den påkrævede arbejdsindsats. Ofte vil punkterne i produkt backloggen være formuleret som små bruger historier eller endda som deciderede use cases. Dernæst har vi valgt ud, hvilke punkter vi koncentrerer os om i den første sprint. Dette fremgår af tabel 2, som er Sprint Backloggen. Punkterne bliver yderligere delt op i sprint opgaver, og hver udvikler påtager sig et antal opgaver og kommer igen med en bedømmelse af den påkrævede arbejdsindsats i timer. Sprint backloggen bliver dermed udgangspuktet for systemudviklingen i den efterfølgende sprint.

Backlog punkt	Sprint opgave	Frivillig	Indsats
Som KT's kunde møder man et login interface, når man navigerer til hjemmesiden.	Oprette en Django applikation	Omar	2
	Skriv login interfacet	Morten	6
	Test login interfacet	Omar	3
	Integrer interfacet med resten af hjemmesiden	Morten og Omar	4
KT ønsker en hjemmeside med kontaktinformation, billeder og et enkelt design.	Skrive basisskabelonen til Django	Morten	5
	Udvid basisskabelonen med "extend"-skabeloner	Morten	5
	Test hjemmesiden i flere forskellige browsere	Omar	2

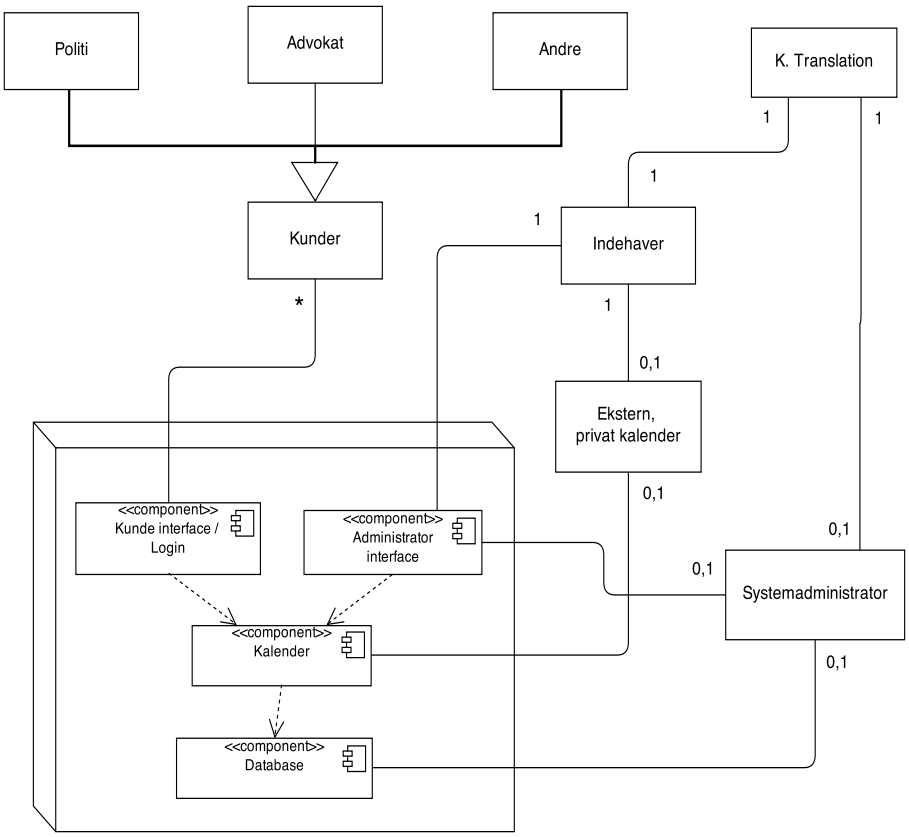
Tabel 2 Sprint Backlog

5 Anvendelsesområde

Vi vil i de næste afsnit kigge på to centrale dele af enhver modellering; nemlig anvendelses- og problemområdet. Vi begynder her med en analyse af anvendelsesområdet, og for overskuelighedens skyld præsenterer vi det først ved hjælp af et UML-diagram. Se figur 2.

Det skal dog præciseres, at det store kvadrat i nederste venstre hjørne i figur 2 ikke hører til anvendelsesområdet. Det er et high-level billede af problemområdet, som her er medtaget for at binde de to områder sammen, og som behandles indgående i næste afsnit.
Umiddelbart er anvendelsesområdet meget begrænset. Vi har fundet følgende klasser, der skal modelleres i anvendelsesområdet:

- K. Translation med indehaver (som samtidig er eneste medarbejder)
- Systemadministrator kos K. Translation
- K. Translations kunder som politi, advokater, etc.



Figur 2: UML-diagram over anvendelsesområdet

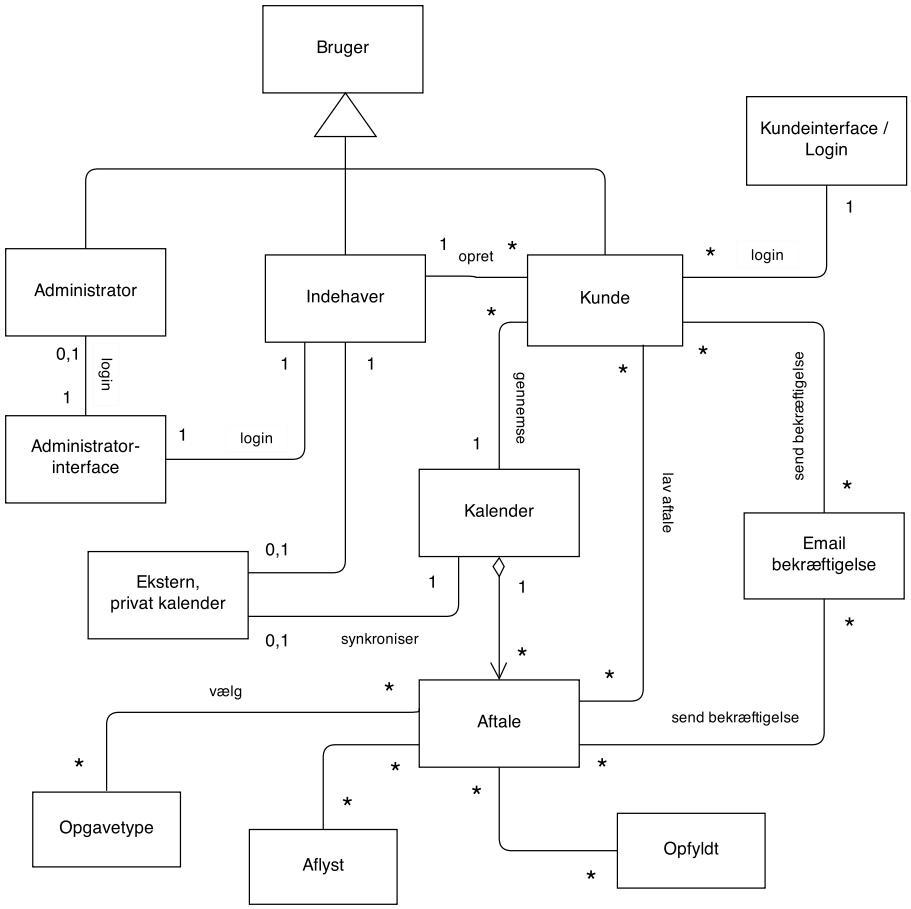
- Eksternt, privat kalendersystem

K. Translations indehaver, som samtidig er den eneste medarbejder, skal som den første modelleres i systemet. Hun skal bruge kalenderen og bookingssystemet i sit daglige arbejde, og hendes ønske er, at kunne kombinere hendes private kalender med vores it-løsning, så hun har alle sine aftaler samlet et sted. Derfor har vi modelleret hende som indehaver af K. Translation og bruger af kalenderen, men vi har samtidig udvidet modellen med en ekstern kalenderklasse mellem indehaveren og systemet, hvor hendes private aftaler ligger, og hvor den eksterne kalender kan synkronisere med vores bookings- samt kalendersystem og overføre de private aftaler. Det skal dog siges, at vi på nuværende tidspunkt er meget usikre på denne funktion, og at vi potentielt må fortælle K. Translation, at dette ligger uden for vores formåen, så vi er påpasselige med at stille hende for meget i udsigt.

Derfor har den også i UML-diagrammet fået multiplicitet 0, 1, da vi højest regner med 1 ekstern kalender, der skal integreres. Det kunne godt gå hen og blive en stor og kompliceret opgave, så derfor får den også i første omgang en lavere prioritet end andre mere tilgængelige ønsker. Se evt. afsnittet om projektplan.
Vi har i UML diagrammet givet indehaveren af KT multiplicitet 1. Man kunne sagtens forestille sig flere ejere af bureauet, men vi har taget udgangspunkt i den nuværende situation, og der er ikke umiddelbart udsigt til nogen ændringer. Selvom indehaveren ikke besidder it-kundsskaber ud over det almindelige, har vi i erkendelse af, at hun også er eneste medarbejder, i UML-diagrammet givet hende adgang til administrationsinterfacet, da hun sikkert vil komme i en situation, hvor hun selv er nødt til at tilgå systemet med alle rettigheder for at ændre eller oprette nye kunder.
Hvis vi skal foresætte modelleringen af anvendelsesområdet, så er der en Systemadministratorklasse associeret til K. Translation. Indehaveren af KT er ganske almindelig it-bruger, og her snakker vi om mail, netbank, facebook, etc, men er ikke it-kyndig udover dette. Derfor er vi nødt til at modellere en systemadministrator, der har overblik over systemet, og som kan yde support, hvis der opstår problemer. Hvis det færdige system ikke kommer til at ligge på en intern server hos KT, men ender med at blive hosted af en ekstern udbyder af serverplads, så vil nødvendigheden af denne klasse mindskes betragteligt, men der vil altid være et vist behov for central systemadministration hos KT i tilfælde af ændringer i kontaktinformation eller kalenderbrug med dertil hørende ændringer i kodebasen og efterfølgende nye upload til serveren. K. Translation må selv tage stillig til, hvor meget systemadministration der er nødvendig efter afleveringen og implementeringen af systememet, men vi vil naturligvis indgå i en dialog med hende omkring dette emne, når det bliver aktuelt.
Den sidste klasse, der skal modelleres i anvendelsesområdet, er KT's kunder og samarbejdspartnere. Denne klasse er omdrejningspunktet for hele systemet, da en af hovedpræmisserne for vores it-løsning er, at KT's kunder på en hurtig og overskuelig måde kan booke en aftale med tolkeservicen. Kundegrundlaget er offentlige myndigheder som politi, ministerier og hospitaler samt private aktører som advokater og andre samarbejdspartnere. Når de møder hjemmesiden, skal de kunne tilgå kalenderen via et loginsystem med deres EAN-nummer og derfra føres videre til selve kalenderen, hvor der kan bookes en aftale. Vi må forvente, at kundesegmentet har meget svingende it-kundsskaber, men at de i kraft af deres daglige arbejde er vant til at arbejde med diverse login- og kalendersystemer, der findes overalt i f.eks. den offentlige sektor.

6 Problemområde

Efter at vi har modelleret tilgangen til it-systemet gennem anvendelsesområdet, er turen kommet til systemets hjerte: problemområdet. Det er her, vi med hjælp af modelleringen senere skal implementere vores løsninger på KT's krav og ønsker. For at binde modelleringen sammen går alle klasserne fra anvendelsesområdet igen i problemområdet. Vi præsenterer igen for overskuelighedens skyld først et UML-diagram over området. Se figur 3. Dernæst kommer der en kort opsummerende liste med klasserne efterfulgt af et længere forklarende afsnit.



Figur 3: UML-diagram over problemområdet

Her følger en liste af de klasser vi har identificeret i problemområdet:

- Bruger: Kunde, administrator, indehaver. Går igen fra anvendelsesområdet
- To interfaces. Et til kunderne og et til administratoren
- Et kundekartotek, hvor KT's indehaver løbende kan tilføje kunder
- Kalenderen, som kunderne kan bruge, når de skal finde en ledig tid, og som KT's indehaver kan bruge i sit daglige, travle arbejde
- Aftale. Kunderne booker en aftale i kalenderen.
- En aftale kan både være aflyst og opfyldt, hvis den ikke længere er aktuel
- Da KT har både mundtlige, skriftlige og andre opgaver, skal kunden oplyse typen på arbejdet.
- Når kunden har booket en aftale, sendes der en bekræftende email tilbage til kunden med det aftalte tidpunkt
- KT's indehaver har et ønske om at kunne synkronisere hendes eksterne, private kalender med vores it-løsning

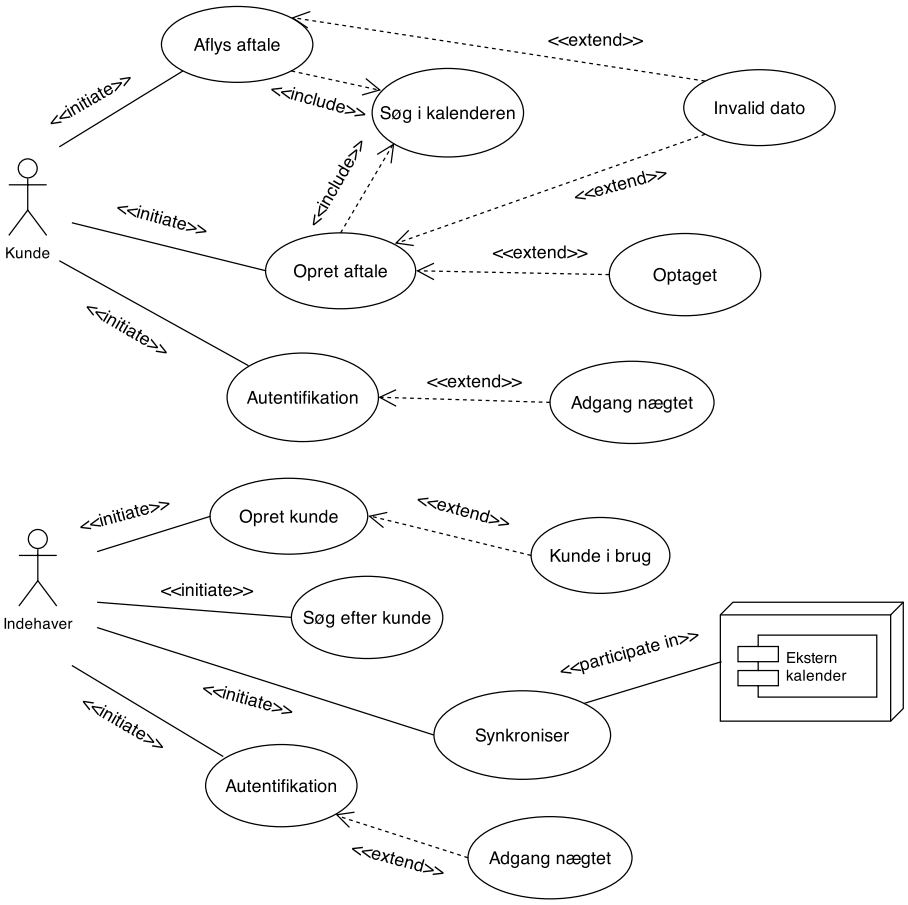
Vi begynder denne gennemgang af problemområdet med de to interfaces, hvor brugerne først møder systemet. Systemadministratoren får sit eget selvstændige interface, idet vedkommende skal kunne tilgå systemet med samtlige rettigheder. Som vi også nævnte under gennemgangen af anvendelsesområdet, så skal KT's indehaver alene i kraft af, at hun er eneste medarbejder, også have adgang til systemet gennem administratorinterfacet, så hun f.eks. kan tilføje nye kunder til kundekartoteket. KT's kunder vil også blive præsenteret for et login interface, når de navigerer til hjemmesiden. Efter login vil der være adgang til kalender og bookingsystemet.
Omdrejningspunktet i problemområdet må siges at være kalenderen, da hele systemets berettigelse hviler på denne. Vi skal have fundet en eller anden kalenderform, der passer til opgaven og KT's arbejdsgange. Indehaveren af K. Translation har som udgangspunkt fortalt, at der ikke forekommer aftaler længere end 2-3 uger ude i fremtiden, men det vil også være for meget for en almindelig computerskærm at skulle vise to eller tre hele kalenderuger med aftaler, så derfor er vi nødt til at finde et acceptabelt kompromis på, hvordan kalenderen skal præsenteres. Vi kunne begynde med at vise kunden dags dato og inkorporere søgefunktioner på dag, uge og klokkeslet, men det endelige design er på nuværende tidspunkt ikke

fastlagt.
Det er også et krav til kalenderen, at kunden let kan skelne mellem ledige tidspunkter og allerede bookede aftaler ved hjælp af et farvesystem. Her vil ledige tidspunkter være markeret med grøn skrift og aftaler med rød skrift.
Dette fører modelleringen af problemområdet videre til aftale klassen. Der vil være en naturlig sammenhæng mellem kalender og aftale, og derfor er de sammenkoblet ved aggregering. En kunde kan lave en eller flere aftaler på samme tid, og vedkommende vil efter hver indgået aftale automatisk modtage en bekræftende email med dato og tidspunkt. Dette ses til højre i UML-diagrammet, hvor emailen autogeneres og sendes tilbage til kunden. Vi har suppleret aftale klassen med to tilhørende klasser: aflyst og opfyldt, som er de to tilstande, en aftale kan være i. (En igangværende aftale har vi ikke fundet vigtig nok til at modellere).
Der skal i kalenderen være mulighed for at aflyse en allerede indgået aftale, og proceduren vil være den samme som ved oprettelsen. Kunden logger på, aflyser aftalen og modtager en autogeneret email som bekræftigelse. Hvorvidt kunden er interesseret i at beholde gamle og udløbne aftaler i kalenderen, står endnu ikke helt klart. De kunne bibeholdes som en form for kalenderhistorie eller logbog, hvis KT skal gennemgå gamle aftaler i forbindelse med fakturering eller anden opgørelse, men det ville være mest hensigtsmæssigt at sætte en grænse, så databasen ikke fyldes op med gamle, ligegyldige aftaler.
Det sidste, vi vil berøre i problemområdet, er det usikre punkt om samspillet mellem en ekstern kalender og vores eget kodede kalendersystem. KT's indehaver har sin egen private Google kalender, som hun gerne vil synkronisere med vores it-løsning for at overføre hendes private aftaler, så alt er samlet i en kalender. Det ville være en smuk løsning at kunne implementere noget sådant, men vi er nødt til seriøst at overveje kompleksiteten af dene opgave. Måske må vi stille K. Translation i udsigt, at private aftaler skal indtastes manuelt i kalenderen, men det må vi tage stilling til, når kernefunktionerne er implementerede.
Hermed slutter vi gennemgangen af anvendelses- og problemområdet. Den ovenstående modellering vil ligge til grund for det videre design og endelige

7 Use-cases

I dette afsnit vil vi beskrive og analysere de vigtigste funktionelle egenskaber ved systemet. Udgangspunktet er kravindsamlingen, som her udvikles til en high-level use case model, hvori vi efterfølgende identificerer de tre vigtigste use cases. Vi har samlet alle use cases og præsenterer dem i det medfølgende højniveau-

diagram. Se figur 4.



Figur 4: Højniveau-diagram bestående af samtlige identificerede use cases

De enkelte use cases er udviklede fra de funktionelle krav, og konteksten er problemområdet, der vækkes til live i historierne. Aktørerne i højniveau-diagrammet er alle hentede fra anvendelsesområdet. Indehaveren af KT er i dette scenario også systemadministrator, idet hun selv vil skulle udfylde denne rolle på længere sigt, og derfor er denne aktør ikke medtaget som en selvstændig klasse. Vi mener heller ikke, at det vil tilføje modellen mere værdi også at modellere systemadministratoren, da vedkommendes use cases i så fald vil være et subsæt af indehaverens. Hvis

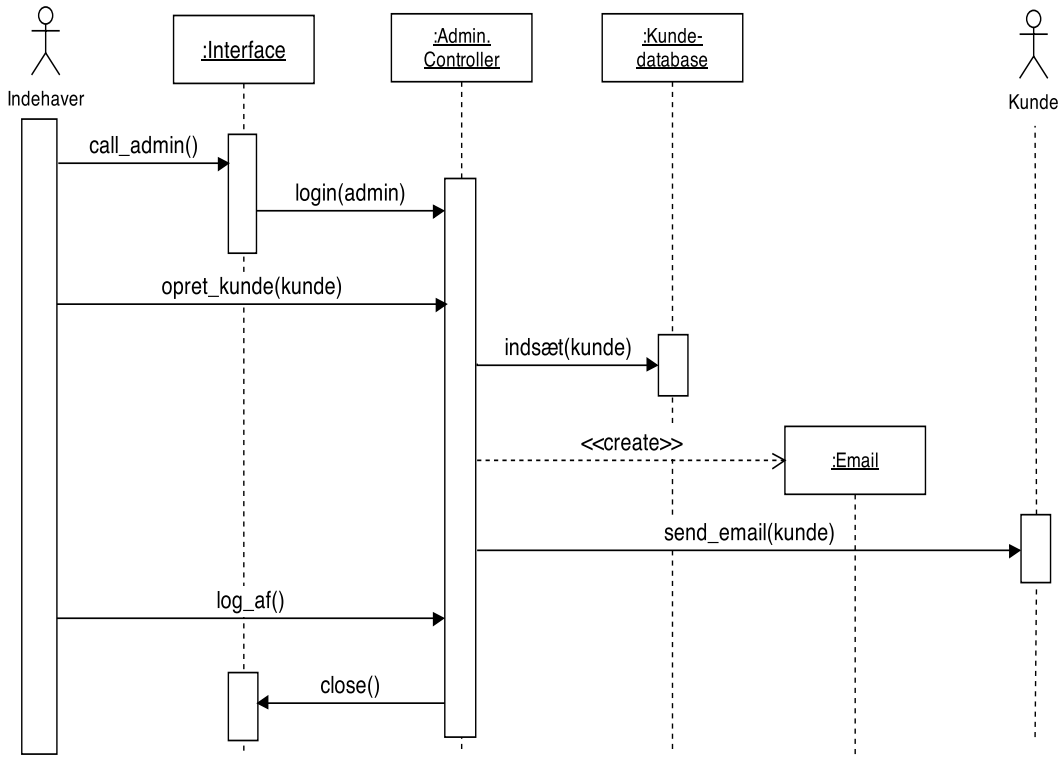
to use cases indeholder det samme begrænsede handlingsforløb, har vi modelleret dette med et "include" forhold. F.eks. kan en potientel kunde søge i kalenderen, både når vedkommende opretter en aftale og når vedkommende aflyser en aftale. Dette er med til at nedsætte kompleksiteteten i use case modellen og fjerne redundans. Det samme gør sig gældende med "extend" forholdene, der modellerer undtagelser og fejltilstande som f.eks. systemets reaktion på ulovlige datoer eller mislykket login.
Vi har valgt de tre vigtigste use cases ud og præsenterer dem her med sekvensdiagrammer. Første use case "Opret kunde" kan ses i tabel 3.

Use case navn	Opret kunde
Deltagere	• Initieret af KT's indehaver • Kommunikerer med kunde
Handlingsforløb	• Indehaveren af KT logger på systemet som administrator. • Administratorinterfacet viser menuen. • Indehaveren navigerer til menupunktet, hvor man kan tilføje nye kunder. • Interfacet præsenterer en formular til indehaveren • KT's indehaver udfylder samtlige formularfelter med kundens stamdata og anden kontaktinformation. Derefter sender hun formularen. • Administratorinterfacet tilføjer kunden til kundekartoteket og autogenererer en email, der sendes til den aktuelle kunde med besked om, at vedkommende nu kan logge på systemet for at booke en tid. • KT's indehaver logger af systemet. • Interfacet lukkes.
Indgangs betingelse	• KT's indehaver er logget på administratorinterfacet. • Pågældende kunde er ikke oprettet i kundekartoteket.
Exit betingelse	• Pågældende kunde er nu tilføjet til kundekartoteket. • Der ligger en bekræftende email i den pågældende kundes indbakke. • KT's indehaver er logget af systemet.

Tabel 3 Opret kunde use case.

Denne use case beskriver, hvordan K. Translations indehaver opretter en ny kunde i kundedatabasen. I vores gennemgang vil vi fokusere på at identificere "entity", "boundary" og "control" objekterne i use casen, og dette kan ses som en forløber for vores valg af "Model-View-Controller" arkitekturen og Django frameworket i afsnittet omhandlende softwarearkitektur, eftersom "Model" kan kortlægges til "entity", "view" til "boundary" og "controller" til "control". I det efterfølgende vil vi dog bruge de danske betegnelser, som er henholdsvis entitet, grænseflade og kontrol.
De to aktører i use casen er KT's indehaver og en kunde. Indehaveren igangsætter use casen og er den aktive part, mens kunden kun medvirker passivt. Grænsefladeobjekterne i use casen er administratorinterfacet, hvor indehaveren logger på systemet, og formularen, hvor hun indtaster kundens stamdata. Den bekræftende email er også en del af grænsefladen, selvom den ikke går til indehaveren men til kunden. Kundekartoteket er den ene entitet i use casen og kunde den anden. Her tænker vi ikke på kunden som aktør, men som en entitet, der oprettes undervejs udfra de indtastede stamdata. Det er umiddelbart lettere at identificere kontrollen i det medfølgende sekvensdiagram, så vi henviser til figur 5. Her fremgår det, at systemet har en administrator kontrol, der er ansvarlig for at oprette entiteten: Kunde og grænsefladen: Email samt Aftaleformular. Dette er måske snarere en for tidlig truffet design beslutning, som ikke fremgår af use casen, men krontrol objekter skabes ofte af grænseflader, der tilgås af aktøren for at igangsætte handlingsforløbet.

Den anden use case, vi har valgt, hedder "Opret aftale". Se tabel 4. Her logger en kunde på systemet for at booke en aftale med tolkeservicen, og vedkommende skal både oplyse dato, klokkeslet og opgavetype. Igen er der to aktører: kunden og KT's ejer. Men rollerne er byttede om i forhold til den første use case. Nu er kunden den aktive part, og KT's ejer er den passive, idet hun kun deltager i use casen for at modtage en email.
Grænsefladeobjekterne er KT's hjemmeside, som use casen indledes fra, og formularerne, der udfyldes af kunden undervejs. Den bekræftende email er ligeledes en del af grænsefladen. Kalenderen og aftalen er entiteter, og som det fremgår af det medfølgende sekvensdiagram, så udgøres kontrolobjektet af en "kalender controller". Se figur 6.

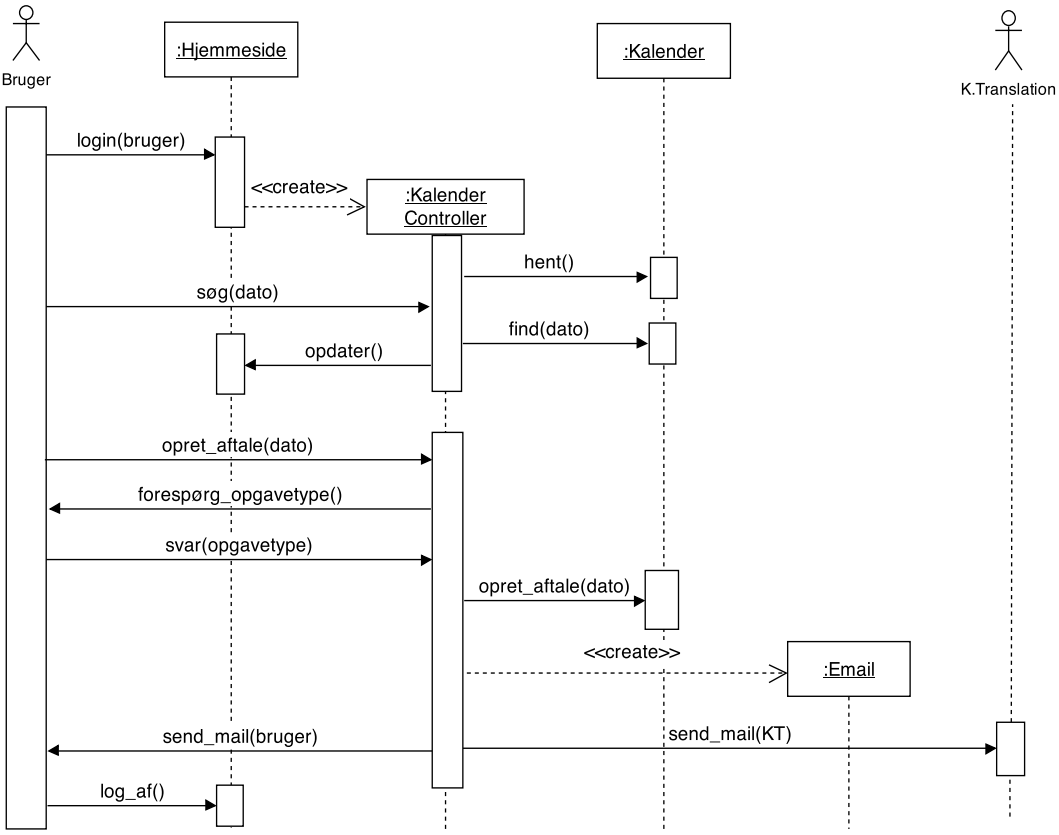


Figur 5: Sekvensdiagram for Opret kunde use case

Use case navn	Opret aftale
Deltagere	• Initieret af KT's kunde • Kommunikerer med Indehaveren af KT
Handlingsforløb	• KT's kunde navigerer til KT's hjemmeside, hvor kunden logger ind med det personlige password. • Kalender kontrollen henter kalenderen og viser den til kunden. • Kunden vælger at oprette en aftale. • Kalender kontrollen henter en formular, som kunden skal udfylde. • Kunden indtaster datoen og klokkeslettet for den ønskede aftale og bekræfter. • Krontrollen validerer dato samt klokkeslet og præsenterer kunden for en menu med forskellige aftaletyper. • Kunden vælger den ønskede aftaletype og bekræfter. • Krontrollen opretter aftalen i kalenderen og sender automatisk en email til både kunden og indehaveren af KT. • Kunden logger af kalenderen.
Indgangs betingelse	• KT's kunde er logget på kalenderen. • Kalenderen indeholder ingen aftaler på det af kunden ønskede tidspunkt.
Exit betingelse	• Der er oprettet en aftale i kalenderen på det ønskede tidspunkt. • Der ligger en bekræftende email i den pågældende kun-

Tabel 4 "Opret aftale" use case.

Igen består første kolonne af den initierende aktør, anden kolonne er grænsefladen, som aktøren bruger til at igangsætte use casen, og tredie kolonne indeholder kontrol objektet, som undervejs når at oprette to grænsefladeobjekter mere samt endnu en entitet.



Figur 6: Sekvensdiagram for "Opret aftale" use case

Efter at have vist to almindelige use cases, har vi til sidst valgt, at fokusere på de extraordinære forhold, der gør sig gældende ved "extend" use cases. Derfor viser vi to små use cases: "Optaget" og "Ulovlig dato", men vi gennemgår kun selve undtagelserne, eftersom konteksten for disse use cases er "Opret aftale" use casen fra tabel 1 og figur 6. Handlingsforløbene før og efter undtagelserne træder i kraft vil være præcise gentagelser af "Opret aftale" use casen; man skal altså

forestille sig denne use case på et givent tidspunkt i forløbet blive udvidet med en af de følgende to use cases. Se tabel 5 og 6.

Use case navn	Optaget
Deltagere	• Kommunikerer med KT's kunde.
Handlingsforløb	• Kunden oplyser typen på opgaven og bekræfter. • Kalender kontrollen prøver at oprette en aftale i kalenderen på det givne tidspunkt, men bliver nægtet adgang, fordi der allerede ligger en aftale på dette tidspunkt. Kontrollen beder kunden om at finde en ny tid. • Kunden indtaster et nyt tidspunkt i formularen og bekræfter. • Formularen validerer tidspunktet.
Indgangs betingelse	• Denne use case udvider "Opret aftale" use casen. Den initieres, når kontrollen prøver at indsætte en aftale i kalenderen på et tidspunkt, hvor der i forvejen eksisterer en aftale.
Exit betingelse	...

Tabel 5 "Extend" use case: "Optaget".

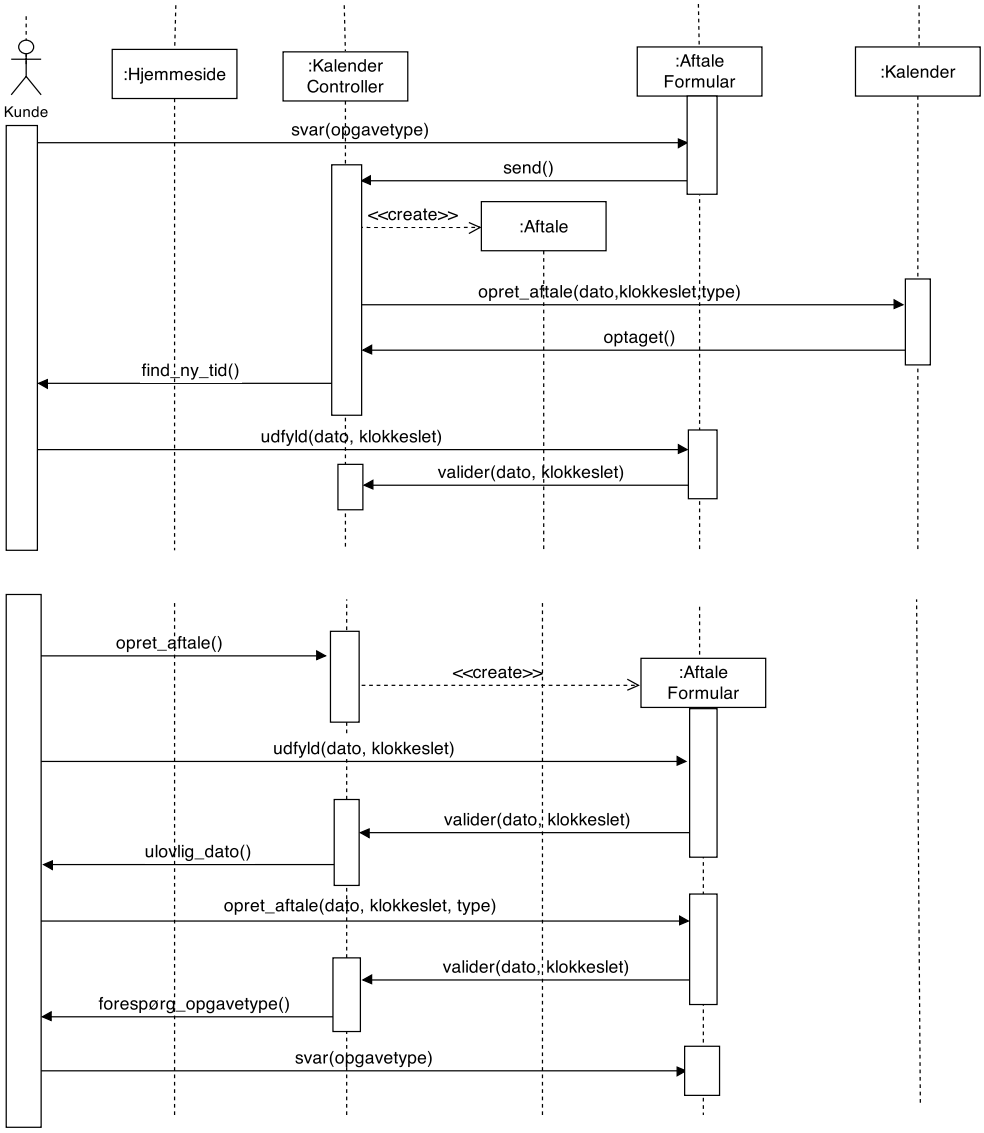
Use case navn	Ulovlig dato
Deltagere	• Kommunikerer med KT's kunde.
Handlingsforløb	• Kunden oplyser kontrollen om, at vedkommende vil oprette en aftale. • Kontrollen opretter en formular til aftalen. • Kunden indtaster dato og klokkeslet for aftalen og bekræfter. • Formularen prøver at validere tidspunktet, men den fejler. Kontrollen beder kunden indtaste et nyt tidspunkt. • Kunden indtaster dato og klokkeslet for den nye aftale og bekræfter. • Formularen prøver at validere tidspunktet og godkender. Kontrollen spørger kunden om opgavetypen. • Kunden oplyser opgavetypen og bekræfter.
Indgangs betingelse	• Denne use case udvider "Opret aftale" samt "Aflys aftale" use cases. Den initieres, når formularen ikke kan validere tidspunktet for kundens aftale, fordi kunden har indtastet et ikke eksisterende tidspunkt.
Exit betingelse	...

Tabel 6 "Extend" use case: "Ulovlig dato".

Grænseflade-, kontrol- og entitetobjekterne i de to "extend" use cases vil være magen til objekterne i de oprindelige use cases, der bliver udvidet, som det også fremgår af de medfølgende sekvensdiagrammer. Se figur 7 for begge sekvensdiagrammer.

8 Softwarearkitektur

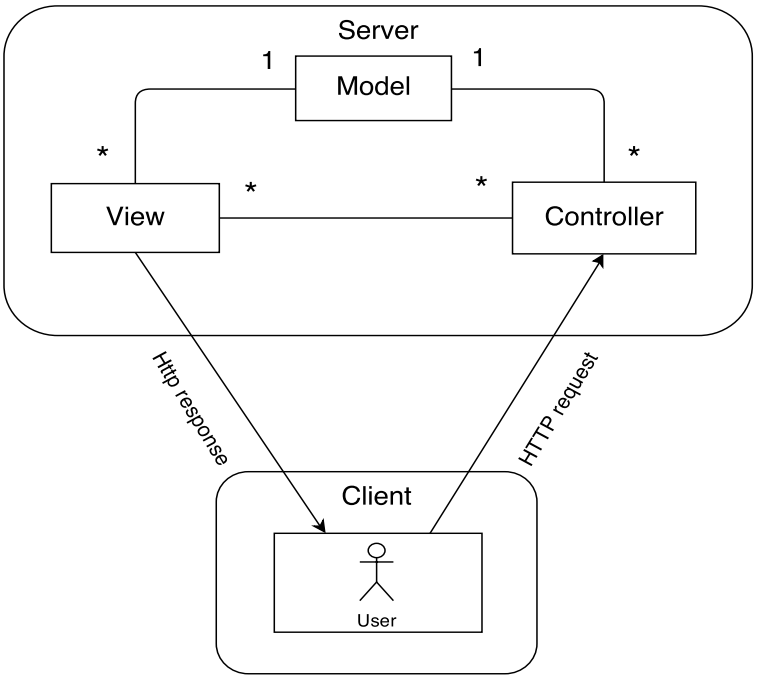
Det kan være en stor udfordring, at finde den rigtige softwarearkitektur i et systemudviklingsprojekt. Vi har selv overvejet flere forskellige arkitekturer og vil i dette afsnit argumentere for vores valg og fravalg. Først overvejede vi en multilagdelt arkitektur i form af 3-tier arkiekturen. Her kunne vi i datalaget gemme de oprettede brugere og bookede aftaler i en database. Forretningslogikken kunne placeres i mellemlaget og kodes med PHP. Brugerinterfacet ville ligge i præsentationslaget og tilgås gennem brugerens browser. Vi fravalgte dog denne løsning af



Figur 7: Sekvensdiagram for “Optaget” samt “Ulovlig dato”.

flere grunde. Vi fandt det ikke streng nødvendigt med tre lag, da det tredie lag ikke ville betyde nævneværdige forbedringer i forhold til en almindelig client-server model. Derudover kunne vi også uforvarent komme til at introducere flere sikkerhedsbrister, hvis vi havde kodet mellemlaget i PHP, da dette kræver, at man er ekstrem opmærksom på validering af brugerinput. Én maliciøs SQL-injection kan

slette en hel database og alle aftaler, hvilket vil være katastrofalt for KT. Da vores kalendersystem ikke kræver andet for at fungere end en bruger med en browser, der kan logge på KT's server, besluttede vi derfor, at holde arkitekturen så overskuelig som mulig ved at bruge client-server modellen.
Dernæst valgte vi det Model-View-Controller (MVC) baserede web framework Django til at understøtte client-server arkitekturen. Se figur 8.



Figur 8: Model-View-Controller arkitektur superimposeret på client-server

Django kommer med et enkelt og praktisk administratorinterface, der kan blive særdeles nyttig for os under systemudviklingen og for indehaveren af KT på længere sigt. Desuden får vi mulighed for at benytte Django's mange indbyggede applikationer. Vi kommer til at bruge moduler, der understøtter implementering af html formularer, og som også er i stand til at validere brugerinput i disse, og moduler for brugeroprettelse og autentifikation. Det betyder, at der er færre muligheder for os, til at introducere fejl og sikkerhedsmangler til systemet. En af Djangos helt store styrker, som udspringer af MVC arkitekturen, er den løse kobling og strenge adskillelse mellem de forskellige dele af modellen. Det betyder, at vi let kan ændre, slette eller tilføje i eksisterende dele uden at bekymre os om

afhængigheder mellem delene.
Fordi views i Django terminologi består af templates, og controllers består af viewfunktioner, er det måske mere rigtigt at kalde Django for en Model-View-Templates (MTV) arkitektur, men vi bibeholder den normale konvention og skriver MVC. Vores model kommer til at bestå af de aftaler, som KT's kunder booker ind i kalenderen. D.v.s at domænerne i den bagvedliggende database, som modellen kortlægges til og fra, udgøres af datoen og klokkeslettet for aftalen plus anden tilhørende information. Django's templates står som sagt for præsentationen, og vi påtænker at bruge en basisskabelon til hjemmesiden, der kan udvides med tilpassede templates, når logikken kræver det. Selve forretningslogikken, der er systemet hjerte, bliver implementeret i Django's viewfunktioner. (Controller i MVC). Det er bindeleddet mellem modellen og præsentationen, og det er her kernefunktionaliteten kommer til at sidde. Eftersom Django i virkeligheden er en samling Python biblioteker, vil vi kode systemet i programmeringssproget Python. Det er også et godt valg, fordi det ene medlem i vores tomandsgruppe har erfaring med Python, og det andet medlem ikke har. Python kan læres relativt hurtigt, så det ene medlem får muligheden for at lære det undervejs i projektet, mens det andet medlem kan lære det fra sig evt. gennem pair-programming.

9 No silver bullit

10 den anden artikel

Litteratur

[1] Bernd Bruegge and Allen H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns and Java*. Pearson Education Limited, Edinburgh, Third Edition, 2014.

[2] Jeff Sutherland, *Jeff Sutherland's Scrum Handbook* Scrum Training Institute, Massachusetts, Årstal: ?.