

Serviceorienteret arkitektur og webservices anvendt i praksis

Peter Andersen, Arne Jørgensen og Lotte Simonsen

4. november 2005

MÅ OFFENTLIGGØRES PÅ BIBLIOTEKET

Indhold

1. Forord	7
2. Introduktion	7
2.1. Læsevejledning	8
3. Problemformulering	8
4. Introduktion til case: Brødrene Hartmann	10
4.1. Motivation	10
4.2. Virksomhedsbeskrivelse	11
4.3. Overordnet problembeskrivelse	11
4.4. Økonomiske fordele ved systemet	12
5. Metodisk fremgangsmåde	13
6. Foranalyse	15
6.1. Metodevalg	15
6.2. Betingelser omkring lokaliteten	16
6.3. Tidsplan og risici	17
7. Case: Om krav og kravindsamling	18
7.1. Kravsindsamling	18
7.2. Krav	20
7.3. Funktionelle krav	21
7.4. Ikke-funktionelle krav	21
7.5. Afgrænsninger	22
8. Case: Brugsmønsterrealisering	23
8.1. Aktørbeskrivelser	23
8.1.1. Lagerforvalter	23
8.1.2. Transportør	23
8.2. Brugsmønsterbeskrivelser	24
8.3. Kollaborationer	25
8.4. Klassediagram	26
8.5. Databasediagram	27
9. Serviceorienteret arkitektur og webservices	28
9.1. Serviceorienteret arkitektur	28
9.2. Oplæring i SOA	29
9.3. Webservices	30
9.4. Webservices i praksis	31
9.4.1. Case: Webservices som en in-house løsning	33

10. Udviklingsmæssige overvejelser	33
10.1. Tidsbesparende i drift og vedligehold	34
10.1.1. Case: Tidsbesparende i drift og vedligehold	36
10.2. Offentlig værktøjskasse	36
10.2.1. Case: Offentlig værktøjskasse	36
10.3. Portabilitet og interoperabilitet	37
10.3.1. Case: Portabilitet og interoperabilitet	37
10.4. Hastighed	38
10.4.1. Case: Hastighed	39
10.5. Økonomiske fordele ved at tilgå databaser via webservices	39
10.5.1. Case: Økonomiske fordele ved webservices	40
11. Arkitektur	40
11.1. Den overordnede arkitektur	41
11.2. Arkitektur i SOA	42
11.3. Case: Arkitektur	43
12. Sikring af databasebegreber i webservices	44
12.1. ACID	45
12.2. Case: Transaktionssikring	47
13. Sikkerhedsaspekter	47
13.1. Autentifikation	48
13.1.1. Case: Autentifikation	49
13.2. Autorisation	50
13.2.1. Case: Autorisation	50
13.3. Konfidentialitet	51
13.4. Dataintegritet	52
13.4.1. Case: Konfidentialitet og dataintegritet	52
13.5. Uafviselighed	52
13.5.1. Case: Uafviselighed	53
13.6. Sikkerhed i adgang til databasen	53
13.6.1. Case: Sikkerhed i adgang til databasen	54
13.7. Programteknisk sikkerhed	54
13.7.1. Case: Programteknisk sikkerhed	55
13.8. Anden sikkerhed	55
13.8.1. Case: Anden sikkerhed	56
14. Case: Programmeringsproblematikker	56
14.1. Post back-problematik	57
14.2. Brugervenlighed	58
14.3. Caching	61

15. Case: Test	61
15.1. Testforløb	63
15.2. Testresultater	64
15.2.1. Brugervenlighed	65
15.2.2. Ny funktionalitet	66
16. Case: Produktlevering	66
16.1. Produktet i drift	67
17. Oplæring i anvendelse af det nye system	67
17.1. Modstand mod forandring	68
18. Markedet i fremtiden	69
18.1. Case: Brødrene Hartmann i fremtiden	70
19. Perspektivering	71
19.1. Gruppen	71
19.2. Brødrene Hartmann kontra Niels Brock	71
19.3. Produktet	72
19.4. Testen	72
20. Konklusion	72
A. Tidsplan	75
B. Kollaborationsdiagrammer	76
C. Opgaver fra testen	78
C.1. Opgaver i WinClient (intern klientsoftware)	78
C.2. Opgaver til WebClient (ekstern klientsoftware)	79
D. WSDL-beskrivelse af webservicen	80
Litteratur	95

1. Forord

I forbindelse med dette projekt er der en række personer, vi gerne vil takke for deres samarbejde. Vi vil gerne takke Brødrene Hartmann for at have muliggjort, at vi kunne afprøve teori i praksis, og for at afsætte den fornødne tid til os.

Her tænker vi specielt på Ole Nygaard Andersen og hans imødekommenhed for at finde det rette projekt til os, hans afsmittende engagement og hans store sans for forplejning. Jesper Steenholdt for hans engagement og ideer til det nye system. Alle fra it-afdelingen, som har vist stor interesse for projektet. Brian for hans fantastiske rundtur på fabrikken i Tønder, og ikke mindst Tove Lindquist, som gjorde projektet muligt ved at formidle kontakten.

Vi vil også gerne takke vores vejleder Pia Petersen for god feedback og for gode kommentarer til rapporten.

Der skal også lyde en stor tak til Christine Bille for at have knoklet sig gennem vores rapport og givet god feedback. Det har været en stor hjælp at få andre til at læse opgaven og se på den med andre øjne.

2. Introduktion

I „Undervisnings- og eksamensplan for datamatikeruddannelsen ved Niels Brock“ [9] står der følgende om, hvad hovedopgaven skal indeholde:

Formålet med hovedopgaven er at den studerende som afslutning af uddannelsen får mulighed for at bearbejde en kompleks problemstilling i relation til en konkret opgave inden for det datamatiske område i samarbejde med en virksomhed.

Mål: at den studerende både skriftligt og mundtligt dokumenterer evnen til systematisk og analytisk at gennemføre en problemløsning samt at forbinde teori og praksis i et bredt sammenhængende fagligt perspektiv.

Til den praktiske erfaring har vi til hovedopgaven fået et samarbejde i stand omkring et transportbooking-system med virksomheden Brødrene Hartmann, som har hovedkontor i Lyngby og er storleverandør af emballage.

Vi er en sammentømret gruppe, som har god erfaring med at arbejde sammen fra tidligere semestre. Vi har alle forskellige baggrunde og forskellige interesser indenfor it-verdenen, og det betyder, at vi kan komme med forskellig input til diskussionerne i projektforløbet samtidig med, at vi supplerer hinandens stærke og svage sider godt.

2.1. Læsevejledning

Vi starter med en beskrivelse af det problemområde, vi vil beskæftige os med. Dette munder ud i vores problemformulering, som vi vil diskutere gennem rapporten og forsøge at besvare i konklusionen.

Efter problemformuleringen vil vi komme med en introduktion til Brødrene Hartmann, som er den virksomhed, vi skal udvikle et system for. Det er dette system, der skal ligge til grund for vores casestudy.

Dernæst vil vi i afsnittet „Metodiske fremgangsmåde“ beskrive de metoder og midler, vi vil bruge, både i forbindelse med udviklingen af produktet, men også hvilken litteratur vi vil anvende, når vi skal diskutere den problemstilling, vi står overfor.

De efterfølgende afsnit består af diskussioner af de emner, der hænger sammen med problemformuleringen. I en række afsnit sætter vi fokus på vores case, og diskuterer problemstillingerne set i forhold til udviklingen af projektet for Brødrene Hartmann.

Efter „problemløsningsafsnittene“ vil vi opsummere forløbet og i konklusionen give et bud på, om det var muligt for os at svare på det spørgsmål, vi stillede i problemformuleringen.

Vi forudsætter, at læseren af rapporten har kendskab til, hvad der svarer til mindst et fjerdesemesters-niveau på datamatikeruddannelsen.

Undervejs i rapporten snakker vi om en administrativ del og en offentlig del. Det henviser til de to delsystemer, vores produkt består af. En nærmere forklaring af dette kan findes i afsnit 7.2 på side 20.

Referencer af typen [7] refererer til litteraturfortegnelsen på side 95.

På side 75 starter vores bilag. Bilag A er vores tidsplan. I bilag B findes de kollaborationsdiagrammer, vi har lavet i forbindelse med udviklingen af vores system. Bilag C indeholder de testopgaver, vi brugte i forbindelse med testen hos Brødrene Hartmann. Til sidst har vi bilag D, der indeholder det WSDL-dokument, der beskriver vores webservices.

Vedlagt rapporten findes en cd-rom med kildekoden til transportbooking-systemet. På cd-rommen findes en fil ved navn README.TXT, der indeholder en beskrivelse af cd-rommens indhold.

3. Problemformulering

For gruppen er det vigtigt, at vi til hovedopgaven studerer et emne, som er fremadrettet i forhold til, hvad vi hver især gerne vil arbejde med i fremtiden, og i forhold til hvad der er nyt og fremme på markedet.

Desuden er det vigtigt, at der er plads til, at vi som gruppe arbejder med emner, som den enkelte er personligt interesseret i, og gerne vil have mere erfaring med. Emner som vi hver især har ønsket at arbejde med, er sikkerhed, databaser og brugervenlige grænseflader.

Brødrene Hartmann har mange projekter i gang samtidig. Ud fra vores ønsker, er vi kommet frem til, at et helt nyt transportbooking-system, som virksomheden på forhånd havde gjort sig nogle tanker om, ville være det rigtige projekt for os, da det indeholder alle de ønskede elementer. Den overordnede problemstilling hos Brødrene Hartmann er beskrevet i afsnit 4.3 på side 11.

For også at drage et fremadrettet emne ind over projektet, som vi alle har interesse i, har vi med godkendelse fra Brødrene Hartmann, valgt at implementere serviceorienteret arkitektur via webservices i projektet. Dette er for os interessant, fordi emnet spås en stor fremtid, og hvis dette bliver en realitet, vil vi sandsynligvis alle komme til at arbejde med emnet på et eller andet tidspunkt i vore it-karrierer. For Brødrene Hartmann som virksomhed ser de også mulige fremtidsperspektiver i serviceorienteret arkitektur, både som in-house løsninger og til udveksling af data med nogle af deres faste kunder, se eventuelt afsnit 18.1 på side 70.

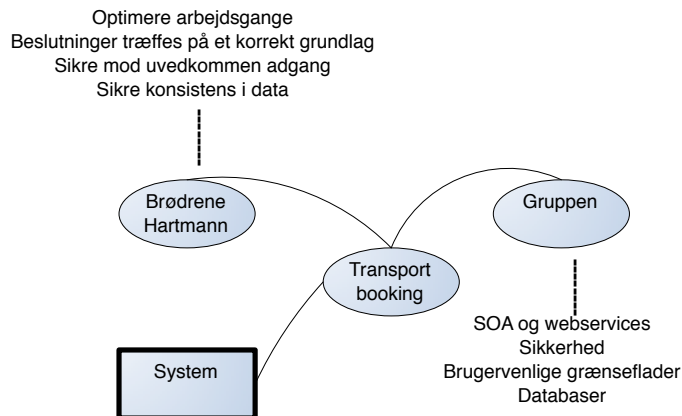
Vi er to i gruppen der, på fjerde semester har snuset lidt til den tekniske del af webservices i valgfaget Webtechnology. Den mere praktiske del med implementeringen af webservices i et virkeligt projekt, vil vi dog alle gerne afprøve. Vi vil også gerne studere emnet serviceorienteret arkitektur, da webservices kan være et middel til at opnå dette.

Vi synes, at det vil være interessant, at studere hvordan en strategi som fokuserer på løst-koblede forretningsforbindelser, kan implementeres i et brugervenligt interface, samtidig med at sikkerheden overholdes, og hvordan dette passer ind i samspillet med databaser.

Dette leder os frem til at vi i hovedopgaven vil besvare følgende spørgsmål:

„Hvordan kan vi udvikle et it-baseret system til Brødrene Hartmann, der kan forestå deres opgave vedrørende reservering af timeslots. Et system der har brugervenlige grænseflader, og gør brug af serviceorienteret arkitektur og webservices som adgang til databaser, og som kan fungere sikkerhedsmæssigt forsvarligt, så Brødrene Hartmann kan sikre sig mod uvedkommen adgang og konsistens i data, og som samtidig kan muliggøre optimering af arbejdsgange, og sikre at beslutninger træffes på et korrekt grundlag?“

Figur 1 på næste side forestiller en illustration af problemet, der skal udtrykke hvad Brødrene Hartmann og vi får ud af projektet.



Figur 1: Illustration af problemformuleringen.

4. Introduktion til case: Brødrene Hartmann

Vi fik kontakt til Brødrene Hartmann gennem en bekendt, som arbejder for dem, og hun henviste os til it-chefen Ole Nygaard Andersen. Vi fik et møde med ham, og præsenterede ham for, hvad vi gerne ville arbejde med og omfanget af hovedopgaven. Han tilbød os det projekt, vi har beskrevet i denne rapport. Vi har fra begyndelsen fået et godt forhold til virksomheden, og samarbejdet har fungeret rigtig godt.

4.1. Motivation

Brødrene Hartmanns motivation for at indgå i et projekt med os har været, at de har haft et behov for at få lavet nogle overvejelser omkring et system, men de har ikke haft tiden til at gå i dybden med problematikken. Det er sådan et projekt, der ellers kan få lov til at ligge længe på hylden, fordi der ikke er tid eller overskud til at gå i gang med det.

Deres samarbejde med os betyder, at de uden at have investeret alt for meget tid, har kunne få lavet en HI-FI-prototype af det system, de gerne vil have udviklet. Det betyder, at de ved projektets afslutning står med en analyse af problemstillingen. De står også med et produkt, de kan udvikle videre på, eller bruge som skabelon, hvis de vil udvikle systemet på en anden platform, eller i et andet sprog.

Alternativerne for Brødrene Hartmann ville enten være at udvikle systemet internt i it-afdelingen, hyre et konsulentbureau til opgaven, eller fortsætte uden systemet.

Det mest sandsynlige er at projektet ville blive udskudt, indtil der var tid i it-afdelingen til at udvikle systemet. Et valg der i praksis kan svare til at fortsætte uden.

4.2. Virksomhedsbeskrivelse

Brødrene Hartmann er en international produktionsvirksomhed, der blev grundlagt i 1917. Virksomheden producerede på daværende tidspunkt papirposer, indtil brødrene ved et besøg i USA i 1936, blev inspireret til at skifte produktionen over til produktion af æggebakker i formstøbt papir. I dag består deres produktion hovedsageligt af at lave æggebakker, men de laver også hospitalsudstyr og pakkematerialer til frugt og industrielle produkter. De laver og sælger også delene til produktionslinjerne, inklusive reservedele og selve støbeformene.

Brødrene Hartmanns hovedkontor ligger i Lyngby og de har salgskontorer i tyve lande, mens der i Danmark er en fabrik i Tønder og i udlandet fabrikker i Argentina, Ungarn, Brasilien, Tyskland, Malaysia, Kroatien, Israel, USA og Canada.

Virksomheden har omkring 2600 ansatte på verdensplan og en global markedsandel på 10 milliarder danske kroner.

4.3. Overordnet problembeskrivelse

I dette afsnit vil vi kort beskrive den problemstilling, som Brødrene Hartmann ønsker at løse ved hjælp af et nyt it-system. Problemet håndteres i dag ikke af it, og beskrivelsen af problemstillingen vil derfor være helt frigjort fra krav og ønsker til det kommende system.

På fabrikken i Tønder afhenter mellem 50 og 75 lastvogne dagligt færdigproducerede varer, og bringer disse ud til kunderne. Til læsning af lastbilerne har Brødrene Hartmann et antal lagermedarbejdere/truckførere, se figur 2 på den følgende side.

For at sikre en fornuftig belastning fordelt ud over en arbejdsdag tilstræbes det, at lastvognene ankommer jævnt fordelt hen over en arbejdsdag, samt at Brødrene Hartmann ved, hvornår de ankommer, så de kan udnytte eventuelle mellemprioriteter til at klargøre lasten.

Brødrene Hartmann meddeler dagligt transportørerne, hvilke varer de har klar til levering og til hvem. Det er herefter transportørernes opgave at disponere varerne på et antal lastvogne. Disponeringen af varerne meddeles Brødrene Hartmann elektronisk.

I dag meddeler transportørerne herefter via e-mail eller fax, hvornår de ønsker at afhente en given disponering, og Brødrene Hartmann kan så melde tilbage, hvorvidt dette kan lade sig gøre eller ej.

Transportørerne har en egen dagsorden, der tilsiger, at de har brug for videst muligt fleksibilitet i planlægningen af deres arbejdsdage. Tilbagemeldingerne fra transportørerne er derfor i praksis behæftet med en del usikkerhed, ligesom de ofte forsøger at ændre deres planer helt indtil sidste øjeblik – både hvad angår afhentningstider, samt disponeringen af, hvilke varer der skal sendes med hvilke lastvogne.



Figur 2: Læsning af lastbil med æggebakker.

Det er målet for Brødrene Hartmann, at et it-system skal sætte nogle fastere rammer for transportørerne, og motivere dem til at respektere disse. Transportørerne skal lære at overholde de tider, de har reserveret til afhentning af gods. Gør de dette, vil ventetiden for andre transportørere, og dermed også dem selv, kunne minimeres eller helt fjernes.

Herigennem forudses det for Brødrene Hartmann, at de vil kunne opnå en mere effektiv arbejdsgang på lageret.

4.4. Økonomiske fordele ved systemet

Den effektive arbejdsgang kan give nogle økonomiske fordele for Brødrene Hartmann. Hvis transportørerne begynder at overholde deres reserverede tider, kan det medføre, at fabrikken i Tønder kan komme hurtigere gennem dagen ved at afkorte den tid det tager at læsse en lastbil. Det betyder at de ikke skal have de timeansatte truckførere gående lige så længe som før, og det kan medføre en besparelse i lønninger.

Det kan også medføre, at hvor fabrikken før kunne nå at tage imod mellem 50 og 75 lastbiler om dagen, så kan de i fremtiden måske nå op på 75 til 100 lastbiler om dagen, og det kan i sidste ende betyde en højere omsætning for Brødrene Hartmann.

De økonomiske fordele ved at anvende webservices i et system, som det vi udvikler, er beskrevet i afsnit 10.5 på side 39.

5. Metodisk fremgangsmåde

I dette afsnit vil vi faktisk beskrive, hvilken fremgangsmåde vi vil anvende for at komme frem til besvarelsen af problemformuleringen.

Dokumentationen af teorien vi vælger at beskæftige os med for at komme frem til besvarelsen, vil komme senere som afsnit, hvor vi først diskuterer teorien, efterfulgt af case studies. I disse case studies vil vi belyse de problemstillinger, der har været omkring anvendelsen af teorien i praksis, i forbindelse med udviklingen af Brødrene Hartmanns transportbooking-system.

Vi vil i dette projekt arbejde sammen med virksomheden Brødrene Hartmann, dels for at få praktisk erfaring med systemudvikling i samarbejde med en virksomhed, men også for at have et produkt, hvori vi kan teste den nye viden, vi tilegner os undervejs i litteraturstudiet.

For at kunne tilegne os ny viden, har vi gennem vores vejleder ladet os inspirere til at anvende en bog om serviceorienteret arkitektur, SOA, af den danske forfatter og underviser Henrik Hvid Jensen [7]. Hans bog giver os en rigtig god introduktion til, hvad SOA og webservices er og kan bruges til, fordi den går meget i dybden med rigtig mange af emnerne. Vi vil bruge denne bog som hovedlitteratur i studiet, til bund for diskussioner omkring SOA og webservices.

Udover denne bog har vi fundet en del artikler på nettet, som vi også vil inddrage i diskussionerne. Flere af disse artikler er specielt rettet mod nogle dele af SOA og webservices, og de er derfor gode at trække ind i nogle forskellige afsnit i rapporten.

Sammen med artiklerne og Hvid Jensen [7] har vi også været på biblioteket og lede efter gode bøger om SOA og webservices. Udbuddet har desværre ikke været så stort som vi havde håbet, med mindre man vælger deciderede programmeringsbøger, hvor der ofte blot beskrives, hvordan man konkret skal implementere webservices i det konkrete programmeringssprog.

Til dette systemudviklingsprojekt vil vi anvende UP som metode. Denne metode har vi erfaring med fra tidligere semestre, og mener, at den vil egne sig godt til udviklingen af projektet.

Da vi arbejder med metoden på studiebasis, kommer vi dog til at anvende en lidt modificeret udgave af UP. Vi har blandt andet en aftale med Brødrene Hartmann om, at produktet ikke skal implementeres i virksomheden, da de selv agter at gøre dette på basis af vores løsning. Transitionsfasen kommer derfor ikke til at fylde så meget i vores projekt. Med en tidsplan for hovedopgaven der afsætter 4-5 uger til udvikling af produktet, er det endvidere begrænset, hvor mange og hvor omfattende iterationerne kan blive.

Som understøttende litteratur til UP vil vi anvende Bennett [2], som er blevet brugt i undervisningen på tidligere semestre. Overvejelserne vi gjorde os, omkring beslutningen om at anvende UP, kan ses i afsnit 6 på modstående side.

For at kunne udvikle et system er vi nødt til at indsamle en række krav fra virksomheden. For at indsamle disse krav vil vi arbejde med forskellige fremgangsmåder. Vi vil, ud over at snakke med vores kontaktperson for virksomheden, også forsøge at etablere et virksomhedsbesøg. Dette kan give os chancen for at snakke med nogle af dem, som i sidste ende vil komme til at bruge systemet, og ikke kun folk, der ser problematikken udefra.

Vi vil også arbejde med prototyper, fordi de kan være gode til at afsløre forståelsesmæssige fejl, og de kan give os en idé om, om vi er på vej i den rigtige retning, i forhold til hvad systemet skal kunne.

Desuden vil vi gøre brug af en midtvejspræsentation, for at vise virksomheden hvor langt vi er nået, og også for at sikre os, at vi kan nå at lave ændringer, hvis der er noget, vi har misforstået.

Udover kontakt til virksomheden, vil vi også bruge meget tid på diskussioner i gruppen. Vi er gode til at snakke om tingene og gå dem igennem, så alle i gruppen føler, at vi er på rette vej.

Vi vil løbende hver især teste de dele af programkoden, vi udvikler, og til sidst i udviklingsforløbet vil vi gøre brug af brugertest. Brugertesten skal sikre, at vores system er brugervenligt, og at det opfylder kravene, så Brødrene Hartmann får hvad de forventer og systemet bliver en succes.

Da vi har fået adgang til Microsofts Visual Studio .NET via Niels Brock, vil vi bruge dette udviklingsværktøj til at udvikle projektet i. Dels fordi webservices er en integreret del af denne platform, og dels for at få mere erfaring med programmeringssprogene C# og ASP.NET.

Som understøttende litteratur til programmeringen vil vi primært anvende Microsoft Visual Studio .NET 2003 Documentation. Hvis dette ikke er tilstrækkeligt, vil vi forsøge at finde kilder på internettet.

Som database vil vi bruge MySQL 4.1, da Brødrene Hartmann har ønsket, at vi bruger denne, og fordi vi kan hente den gratis på internettet.

Derudover vil vi bruge, og dermed få yderligere erfaring med de værktøjer, vi allerede kender og har brugt på tidligere semestre. Dette drejer sig blandt andet om versionsstyringssystemet Subversion. Systemet gør det nemt at redigere i filer, som andre samtidig arbejder på, og det bevarer gamle versioner af filerne.

Til rapportskrivning vil vi bruge L^AT_EX. Fordelen ved dette er, i modsætning til traditionel tekstbehandling, at et dokument let kan deles ud i flere filer, samt at disse filer integrerer godt med Subversion. L^AT_EX holder desuden nemt referencer, henvisninger, litteraturliste og indholdsfortegnelse opdateret.

6. Foranalyse

6.1. Metodevalg

Da vi skulle beslutte os for, hvilken udviklingsmetode vi ville anvende til udviklingen af Brødrene Hartmanns system, diskuterede vi fordelene og ulemperne ved de forskellige metoder, som vi har stiftet bekendtskab med gennem datamatikeruddannelsen.

Valget faldt på UP, som dækker livscyklusen:

Analyse – Design – Konstruktion – Test – Implementering

Vi mente, at det var den metode, som ville egne sig bedst set ud fra, at vi skulle alle faserne igennem undtagen den praktiske erfaring med implementeringsfasen, men vi ville alligevel gøre os nogle overvejelser omkring denne fase.

Metoder vi kender, som har samme livscyklus som UP, er OORP og XP.

OORP har den fordel, at man i samarbejde med de fremtidige brugere, nemt og hurtigt kan komme frem til designet og kravene til systemet, gennem iterationerne ved anvendelsen af LO-FI- og HI-FI-prototyper.

Vi vurderede, at dette for så vidt kunne have været en udmærket metode at anvende, hvis vi havde kunne etablere en tæt kontakt til brugerne. Dette kunne imidlertid ikke lade sig gøre, da den primære bruger af det administrative system er Jesper Steenholdt, som til dagligt er placeret i Tønder, og brugerne til det offentlige system er transportørerne, som befinder sig spredt ud over hele landet. Vi så derfor ingen fordel ved at anvende OORP, da iterationerne i forløbet formentligt ikke ville kunne lade sig gøre, i et hurtigt tempo.

Vi vurderede fra starten, at XP ikke ville være en god metode for os at anvende til dette projekt. Alene det at metoden foreskriver, at det er en nødvendighed for metodens succes, at man skal have en kunderepræsentant siddende til daglig sammen med udviklergruppen. Dette mente vi nok ville være urealistisk i vores situation. Heller ikke selvom vi havde fået en fast arbejdsstation hos Brødrene Hartmann, ville de forståeligt nok, ikke kunne afsætte den fornødne tid til dette.

XP egner sig også bedre til systemer, hvor kravene ofte skifter, hvorfor mindre dele af systemet testes og implementeres løbende. Vi vurderede at kravene til transportbooking-systemet formentlig ikke ville ændre sig i en sådan hast, at vi ville få fordel af denne fremgangsmåde. Desuden var kravet fra Brødrene Hartmann, at få leveret en funktionel prototype og ikke færdige „delsystemer.“

Når systemet er sat i drift hos kunden vil der helt sikkert være noget videreudvikling og vedligehold af systemet. Her vil UP være en god metode, fordi vedligehold følger den samme livscyklus, som beskrevet ovenfor. Det betyder at man kan bruge de samme arbejdsmetoder når man skal vedligeholde et system, som når man skal

udvikle det. Det kan være en fordel, da man så ikke skal skifte metode efter at systemet er udviklet.

De mange diagrammer kan have en stor fordel for kunden i forbindelse med senere udvikling og vedligehold. Udover klassediagrammet, som vi nævner ovenfor, er kollaborationsdiagrammerne også gode, fordi de er med til at beskrive arbejdsgangene i systemet. Disse diagrammer giver et godt overblik over, hvad der hænger sammen, så man kan se, hvilke konsekvenser eventuelle ændringer kan medføre.

Desuden synes vi, UP er god som metode, fordi den griber tingene an på en meget struktureret måde, og fordi den arbejder med iterationer. Iterationerne gør, at vi med god samvittighed kan løbe hurtigt henover nogle af faserne, fordi vi ved, at vi vil komme tilbage til dem igen på et senere tidspunkt.

Vi mener en af styrkerne ved UP, som for eksempel XP slet ikke benytter sig af, er de mange diagrammer, som udvikles med hovedvægten i elaborationsfasen. Vores erfaring som udviklere er, at de er med til at holde fokus på, at kravene gennem alle iterationerne i projektet holdes konsistente. Specielt under konstruktionsfasen, er klassediagrammet for eksempel en god støtte til forståelse af, hvordan de forskellige klasser skal implementeres med forbindelse til hinanden, og hvordan de forskellige medlemsfunktioner skal anvendes.

Vi har på tidligere semestre arbejdet med UP, og har god erfaring med denne metode. Dette har selvfølgelig også haft betydning for valget af UP.

Da vi stadig er i en læreproces, og hovedopgaven for vores vedkommende ikke som det primære mål, har haft til opgave at give os erfaring med en udviklingsmetode, har vi ikke undersøgt markedet for, om der måske ville have været, en udviklingsmetode som egnede sig bedre til Brødrene Hartmanns projekt end UP.

Da UP fokuserer på, hvem det handler om, hvad der sker, hvornår det sker og hvordan det skal ske, alt sammen drevet af kravene og risiciene, mener vi, at vi traf det rigtige valg i forhold til, at anvende UP som udviklingsmetode til transportbooking-systemet.

6.2. Betingelser omkring lokaliteten

Niels Brock anbefaler, at man til dagligt sidder og arbejder hos den virksomhed, man skal udvikle projektet sammen med, for dels at sikre at projektet bliver en succes, og dels for at de studerende skal have denne erfaring med i bagagen, til jobbet der venter ude i fremtiden.

Dette gjorde vi også en indsats for at arrangere. Brødrene Hartmann ville også gerne have imødekommet dette, men det lod sig desværre ikke gøre. Muligheden for at vi kunne sidde i et separat lokale, var ikke tilstede, og it-afdelingen hos Brødrene Hartmann er et åbent kontorlokale, hvor der ikke er mulighed for at presse flere arbejdspladser ind. De har i dette kontorlandskab et afskærmet mødelokale, hvor

vi fik tilbud om, engang imellem når det kunne lade sig gøre, at komme ud og arbejde.

Problemet ved denne ordning var, syntes vi, at vi på denne måde ikke ville få en fast arbejdsplads, hvor vi kunne efterlade vores ting. Vi ville formentlig heller ikke kunne opnå den samme dynamik ved vores gruppemøder, som vi plejer, da disse ofte går noget højlydt for sig. Vi ville måske komme til at forstyrre det faste personale, hvilket vi ikke ønskede.

Brødrene Hartmann var forstående for disse problematikker og for vores situation omkring besværlighederne, der ligger i at arbejde på Niels Brock. Da computerne deles imellem alle de studerende, kan de være geninstalleret næste morgen, når man møder og skal fortsætte sit arbejde, hvorfor vi ville risikere at skulle bruge en del tid på at geninstallere programmer og flytte filer.

Derfor udstyrede Brødrene Hartmann os hver især med en bærbar computer til låns, hvor vi kunne installere alt den software, vi havde brug for. Computerne leveres tilbage, når vi skal ud og aflevere det endelige produkt.

Løsningen blev således, at vi stort set hver dag, har siddet i et grupperum på Niels Brock, hvor de bærbare computere har været os en rigtig stor hjælp.

6.3. Tidsplan og risici

Efter første møde med Brødrene Hartmann fastlagde vi en tidsplan for hovedopgaveforløbet, se bilag A på side 75.

Tidsplanen afsætter cirka halvdelen af forløbet til udvikling af systemet til Brødrene Hartmann og cirka halvdelen af forløbet til dokumentation og rapportskrivning. Efter sammenhængende perioder havde vi afsat milepæle hvor periodens samlede opgaver skulle være færdiggjort og evalueret.

Vi har skønnet at der ville være et antal risikofaktorer der kunne forskyde tidsplanen. De væsentligste risici har været sygdom, større kravændringer og et gruppemedlems familieførøgelse.

Risiciene er imødegået ved at indlægge en smule luft i dagene op til milepælene, herunder inddragelse af weekenderne om nødvendigt.

Risikoen ved ændrede krav er forsøgt imødegået ved at indgå i en dialog med virksomheden, om hvorvidt kravene faktisk ønskedes implementeret og hvilke øvrige krav, der om nødvendigt måtte vige der for.

7. Case: Om krav og kravindsamling

7.1. Kravsindsamling

Når man skal samle krav ind til et it-systems funktionalitet, er der mange forskellige måder at gøre dette. Vi har primært benyttet os af den ekspertviden, som Ole Nygaard Andersen og Jesper Steenholdt hos Brødrene Hartmann besidder. Vi har desuden været på virksomhedsbesøg på fabrikken i Tønder. Vi har blandt andet også udviklet prototyper og testet systemet på virkelige brugere. Vi har i alt holdt fem møder med Brødrene Hartmann, inklusive virksomhedsbesøg og test der var heldagsarrangementer.

Allerede ved det første, indledende møde vi havde med Brødrene Hartmann, dannede vi os et billede over grundskellet til, hvad systemet skulle kunne. Vi havde ikke snakket om et projekt med Brødrene Hartmann før, kun at vi skulle mødes og se, om vi kunne finde noget, der havde fælles interesse for os alle, og om vi ønskede at samarbejde med hinanden.

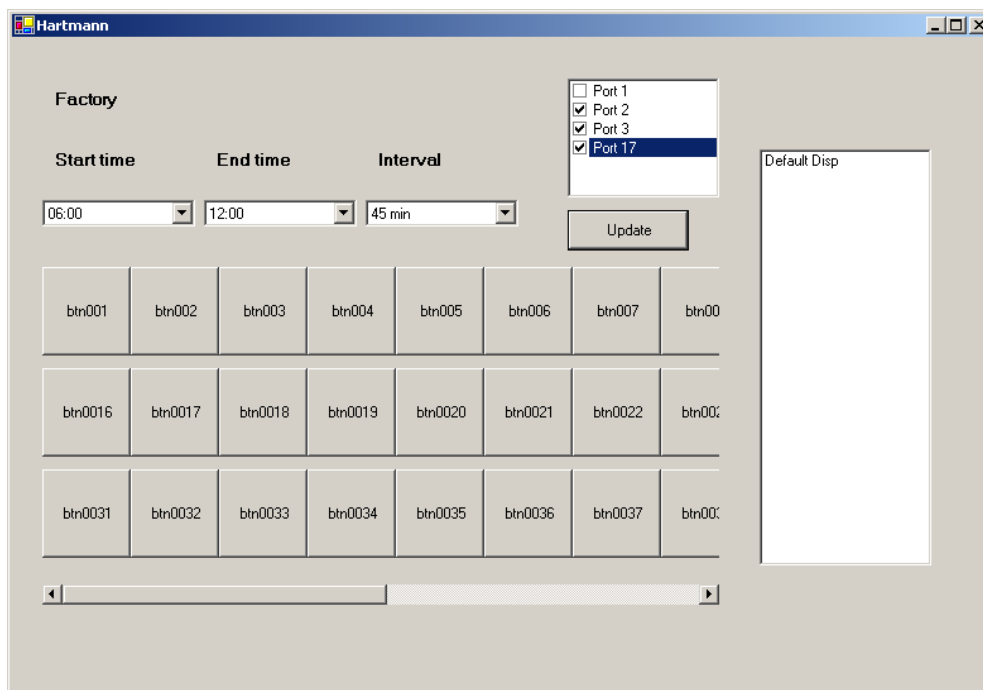
Mødet udviklede sig dog til, at Ole fremlagde transportbooking-systemet for os, som han mente ville passe til vores krav og ønsker. I løbet af denne samtale fik vi diskuteret mange af de specielle detaljer, som findes hos Brødrene Hartmann. Eksempler herfra er:

- Hvad er en disponering og hvordan identificeres den. Et disponerings-id omfatter den samling af gods som en transportør kan afhente ad gangen. Sammen med disponerings-id'et leveres oplysninger om, hvilken transportør der henter varerne, og hvilken dato og fabrik varerne skal hentes på.
- Vi fik også diskuteret, hvordan flowet foregår fra registreringen af, at varer skal afhentes, over hvordan man reserverer en timeslot, til hvordan selve afhentningen foregår.
- Vi diskuterede blandt andet også løsningsmodeller på, hvordan man grafisk kunne bygge morgendagens skema op.

Grundstenene til hvordan det endelige system kom til at se ud, blev dannet ved dette første møde.

Det der for gruppens vedkommende var vigtigt efter dette møde, var at finde ud af om vi havde en nogenlunde fælles opfattelse af, hvad systemet skulle kunne, inden vi sagde ja til projektet. Derefter kunne vi se, om vi hver især fandt projektet udfordrende i forhold til, hvad vi ville, og hvad vi i tidligere projekter havde beskæftiget os med. Da der var enighed om dette, gik vi i gang med en grundigere analyse af kravene ved anvendelse af UP og UML.

Næste store skridt var virksomhedsbesøget i Tønder, hvor vi for første gang skulle møde Jesper Steenholdt, som er ham der til daglig skal anvende den administrative del af systemet. Vi havde lavet en funktionel HI-FI-prototype, hvor vi allerede



Figur 3: Første prototype til det administrative system, som blev fremvist på mødet i Tønder.

hentede og gemte nogle data i databasen, mens andre data var skrevet statisk ind i programkoden, se figur 3.

Vi havde besluttet os for, at vi på mødet ville have Jesper til at beskrive med sine egne ord, hvad hans arbejdsrutiner til dagligt er, og hvordan han forestiller sig at systemet, vi udvikler til ham, kommer til at fungere, inden vi viste ham vores prototype. På denne måde mente vi, at vi bedre ville kunne se, om vi havde forstået arbejdsrutinerne korrekt. Og Jesper ville dermed heller ikke være farvet af oplevelsen med prototypen, både hvad angår positiv, såvel som negativ kritik.

Mødet viste sig at blive et temmeligt vigtigt møde. Der var nogle enkelte ting som gjorde, at vi skulle ændre en del i det bagvedliggende system. For eksempel skulle man kunne oprette et skema ud fra en skabelon. Dette havde vi slet ikke haft med i vores tidligere tankegang.

En anden vigtig ting som afsløredes ved mødet var, at Ole og Jesper ikke var helt enige om, hvor mange disponeringer man skulle kunne reservere i et enkelt timeslot.

Jesper oplever at Brødrene Hartmann, i den virkelige verden ofte må være fleksible over for, at et timeslot kan indeholde mere end en disponering, af den simple grund at transportørerne ikke udarbejder deres disponeringer korrekt.

Ole derimod oplever den virkelige verden som de procedurer Brødrene Hartmann og transportørerne har beskrevet og aftalt. Altså at der til ét timeslot kun hører én disponering.

Der blev taget en beslutning på mødet om, at Brødrene Hartmann i fremtiden må opdrage transportørerne, så de opretter disponeringer efter de fastlagte aftaler, og at det er ud fra den tankegang, vi skulle udvikle systemet. Til problematikken knyttedes endvidere at afregningen mellem Brødrene Hartmann og transportørerne afhænger af disponeringerne, og en forskellig opfattelse af disse kan derfor give anledning til en ukorrekt fakturering.

Udover mødet med Jesper i Tønder, fik vi også en halvanden times rundvisning på fabrikken. Dette gjorde, at vi fik et mere realistisk billede af, hvordan arbejdsrutinerne på fabrikken er.

Gennem udviklingsforløbet af programmet har vi løbende itereret over kravene, og løbende justeret disse hvis det har været nødvendigt. Dette har vi gerne gjort efter møder med Brødrene Hartmann, eller når der er opstået uklarheder, specielt under elaborations- og konstruktionsfasen.

Den sidste iteration af kravindsamling til systemet gennemgik vi ved evalueringen af bruger- og programtesten, som vi foretog hos Brødrene Hartmann.

Testen viste, at der blot var ønsker om yderligere funktionalitet, samt grænsefladeforbedringer fra testpersonernes side.

Disse nye krav og ønskerne aftalte vi med Brødrene Hartmann, at de selv må implementere på et senere tidspunkt. Hvordan selve testen forløb kan ses i afsnit 15 på side 61.

7.2. Krav

Opgaven fra Brødrene Hartmann går ud på, at vi skal lave et transportbooking-system til deres produktionsfabrik i Tønder.

Systemet skal bestå af to dele: en administrativ del og en „offentlig“ del.

I det administrative system skal det være muligt for lagerforvalteren at lave et skema over, hvornår der er åbent for at de transportører, der har et samarbejde med Brødrene Hartmann, kan komme og afhente varer, der er klar til transport.

Transportørerne skal så i det „offentlige“ system kunne se det oprettede skema via en webbrowser, og selv kunne reservere de tider, der er til rådighed. At systemet er „offentligt“ skal blot forstås således, at det kan tilgås af de transportører, der har et samarbejde med Brødrene Hartmann.

Allerede efter det første møde hos Brødrene Hartmann med Ole Nygaard Andersen, var vi kommet frem til grundstrukturen i systemet. Det viste sig ved det næste

møde, at der var meget få rettelser til vores forståelse af systemet. Det var mere en diskussion om forståelse af dataenes betydning, og en snak med Ole, om hvilke dele vi allerede havde besluttet os at ville implementere, og afgrænsninger som Brødrene Hartmann selv skulle implementere på et senere tidspunkt.

Nedenfor kan ses de funktionelle og de ikke-funktionelle krav til systemet, som vi ved iterationerne nåede frem til.

7.3. Funktionelle krav

Det skal være muligt for lagerforvalteren at oprette et skema, ud fra nogle specifikationer, han selv har valgt. Disse specifikationer er åbnings- og lukketid, samt timeslotlængden og antallet af lagerpersonale der er til rådighed. Skemaet skal kunne oprettes ud fra en skabelon, således at lagerforvalteren kan vælge en foruddefineret skabelon, som passer bedst til det skema, han ønsker at oprette. Lagerforvalteren skal også have mulighed for at gemme et skema som en skabelon.

Når skemaet er oprettet skal det være muligt for lagerforvalteren at låse de timeslots, der ikke skal stilles til rådighed for transportørerne. Desuden skal han have mulighed for at ændre, slette eller flytte reserveringer af timeslots. Når skemaet er oprettet, skal det være muligt for lagerforvalteren at ændre på indstillingerne for skemaet. Lagerforvalteren skal kunne få tidligere oprettede skemaer vist.

Transportørerne skal kunne logge på via en webbrowser. Transportørerne skal have mulighed for at reservere timeslots med deres egne disponeringer på skemaet. Transportørerne skal også have mulighed for at ændre, slette eller flytte de reserveringer, de selv har registreret. Det skal også være muligt for transportørerne at hente tidligere udfyldte skemaer til aflæsning.

Uden dog at have tilknyttet en brugergrænseflade til dette, skal det være muligt at tilføje, fjerne og redigere fabrikker og brugere i systemet.

En nærmere beskrivelse af aktører og yderligere realiseringer af krav kan ses i afsnit 8 på side 23.

7.4. Ikke-funktionelle krav

Systemet skal køre op imod en MySQL database. Lagerforvalteren og transportørerne skal have nogle brugervenlige grænseflader. Med brugervenlige forstås blandt andet, at menuer skal have sigende navne. Det betyder også at eventuelle genvejstaster skal have den samme effekt, som de har i systemerne, der bliver brugt til hverdag.

Systemet skal udvikles med baggrund i tankegangen omkring serviceorienteret arkitektur, og der skal bruges webservices i implementeringen af systemet. Det skal opbygges, så det er nemt at vedligeholde.

Systemet skal være sikkert at bruge. Det betyder, at der kun skal være adgang til systemet for en lukket gruppe af brugere, og at systemet skal sikre konsistens i data. Systemet skal medføre en optimering af arbejdsbyrderne på fabrikkerne, og være med til at opbygge en arbejdsgang der gør, at beslutninger kan træffes på et korrekt grundlag.

Der skal opsamles oplysninger fra systemet, så det er muligt, på et senere tidspunkt at lave statistikker. Der skal implementeres en form for automatisk opdatering af brugergrænsefladerne, så brugerne hele tiden har et forholdsvis opdateret skema at arbejde på. Der er også et ønske om, at brugergrænsefladen kan udbygges med forskellige sprog.

7.5. Afgrænsninger

Vores erfaring med at udvikle større systemer siger, at det altid er nødvendigt fra starten af, at foretage nogle afgrænsninger. Disse afgrænsninger fastsættes, for at kunne træffe beslutning om et fornuftigt sluttidspunkt, så man sikrer, at systemet ikke fortsætter i det uendelige.

I Brødrene Hartmann-projektet var en af de ting, der blev afgrænset fra starten, et grafisk administrationssystem til brugere og fabrikker. Da vi allerede skulle implementere administrationsdelen til reservering af timeslots, mente vi, at opgaven ville være meget lig denne, i forhold til at gemme og hente data. Derfor traf vi en beslutning om, at vi hellere ville bruge tiden på at implementere noget, som gav os helt nye udfordringer.

En anden ting, der blev afgrænset, var automatisk opdatering af brugergrænsefladerne. Denne beslutning blev truffet i samarbejde med Ole Nygaard Andersen. Da reserveringerne af timeslots kan foretages fra morgendagens skema frigives, frem til morgendagen starter, vil reserveringer i systemet ikke være så koncentrerede indenfor et kort tidsrum, som først antaget. Beslutningen blev, at brugeren af systemet selv må opdatere skærbilledet ved hjælp af for eksempel F5.

Muligheden for at samle information til statistik i en log er også afgrænset væk. Brødrene Hartmann har endnu ikke lagt sig fast på, hvilke oplysninger det er, de gerne vil gemme til dette formål. Vi mener ikke at dette ville have voldt os de store problemer, da vi har prøvet noget lignende på tredje semester, hvor vi opsamlede information fra en webserver til en logfil.

Sproghåndteringen blev i samarbejde med Ole og Jesper afgrænset væk undervejs i udviklingsforløbet. Det gjorde det, da der ved mødet i Tønder kom nye og vigtigere krav til systemet, i form af, at et skema skulle oprettes ud fra en skabelon. Da vi allerede så småt var begyndt at forberede implementeringen af sproghåndteringen i systemet, ligger der derfor et par tomme tabeller i databasen til senere implementering af dette.

8. Case: Brugsmønsterrealisering

I dette afsnit gennemgår vi i hovedtræk realiseringen af de identificerede brugsmønstre. Vi lægger ud med en beskrivelse af de involverede aktører, og giver derpå en beskrivelse af brugsmønstrene.

Herefter følges realiseringen via kollaborationsdiagrammer til klassediagrammet.

8.1. Aktørbeskrivelser

I systemet til Brødrene Hartmann er der to typer aktører, lagerforvalteren og transportørerne. Afsnittene nedenfor beskriver disse aktørers arbejdsrutiner.

8.1.1. Lagerforvalter

Lagerforvalteren forvalter lageret.

Hver dag klokken 10 sender lagerforvalteren en prognose til transportørerne, om hvad han forventer, der vil være af transporter den følgende dag. Klokken 13 udsender han den endelige ordre til transportørerne, om hvad de skal hente på fabrikken næste dag.

Når transportørerne har meldt deres disponeringer tilbage til Brødrene Hartmann, er det lagerforvalterens opgave at oprette et skema til morgendagen og inddele dagen i en række timeslots, hvor transportørerne kan afhente godset, de har knyttet til disponeringen.

Det er også lagerforvalterens opgave, at administrere egne reserveringer på dagens skema, eller hvis transportørerne, af en eller anden årsag, ikke selv kan oprette eller redigere deres reserveringer.

8.1.2. Transportør

Brødrene Hartmann har en række faste transportører tilknyttet hver fabrik. For eksempel har fabrikken i Tønder ca. 5.

Det er disse transportører, der står for at transportere de, af Brødrene Hartmann, fremstillede varer fra fabrikkerne ud til kunderne.

Når transportøren har modtaget næste dags ordre fra Brødrene Hartmann, laver han en disponering over hvilke og hvor mange vogne, han vil fordele godset i. Denne disponering afleveres til Brødrene Hartmann klokken 15.

Fra klokken 16 kan transportøren logge sig ind i systemet og reservere timeslots via sit disponerings-id til afhentning af gods. Hvis der er behov derefter, skal transportøren

selv ændre eller slette sine egne reserveringer, inden morgendagen starter – når Brødrene Hartmanns fabrik åbner.

Transportøren kan også logge sig ind i systemet og se tidligere skemaer med egne reserveringer.

8.2. Brugsmønsterbeskrivelser

I det følgende beskrives de syv brugsmønstre vi identificerede under kravsindsamlingen.

Book timeslot Transportøren eller lagerforvalteren skal kunne markere at i en bestemt timeslot på en dag, afhentes der et bestemt disponerings-id. Herefter kan timeslottet ikke disponeres til anden side.

Transportørerne kan kun lægge deres egne disponerings-id'er ind i timeslots. Lagerforvalteren kan lægge alle disponerings-id'er der er tilknyttet hans fabrik ind i timeslots, samt et specielt „default“-disponerings-id der indikerer at timeslotten ikke kan anvendes til almindelig reservering.

Show schedule Både transportører og lagerforvalteren skal kunne få vist et skema for en given dag.

Først vælges hvilken fabrik der ønskes et skema for. Dette skal ikke nødvendigvis ske interaktivt.

For den pågældende dag og fabrik findes skemaets grundoplysninger som består af starttid, sluttid, hvor meget personale der er til rådighed, timeslotlængde og dagens oprettede timeslots inklusive eventuelle disponeringer af disse.

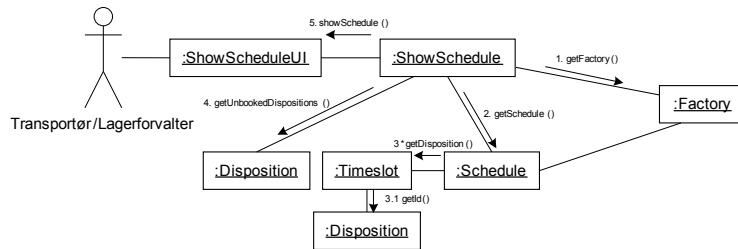
Move disposition Transportørerne og lagerforvalteren skal kunne flytte en disponering fra en timeslot på et givet skema til et andet timeslot på skemaet.

For transportørerne vil der være en begrænsning på hvilket tidsrum ændringer kan foretages i. Ændringerne kan ikke foretages så snart morgendagen er startet.

Submit schedule For dage hvor der endnu ikke eksisterer et skema, skal lagerforvalteren kunne oprette et skema.

Et skema består af en dato, en fabrik, en timeslotlængde, en starttid, en sluttid og hvor meget personale der er til rådighed på fabrikken. Herudfra genereres skemaets timeslots.

Et skema oprettes ud fra en skabelon. Dette betyder, at skemaet måske ikke behøver at blive tilrettet til dagen bortset fra, at skemaet får en dato som sættes automatisk.



Figur 4: Kollaborationsdiagram for ShowSchedule.

Eventuelle ændringer og andre ønsker må foretages via brugsmønstreret, som passer til den valgte handling.

Submit schedule template Lagerforvalteren kan gemme et skema som en skabelon.

Skabelonen skal gives et unikt navn, og kan valgfrit gives en beskrivelse.

Disponeringer i et skema gemmes ikke i skabelonen – undtaget herfor er de timeslots som lagerforvalteren har „blokeret“.

Update schedule For et givet skema skal lagerforvalteren kunne ændre i skemaets grundoplysninger, det vil sige timeslotlængde, starttid, sluttid og hvor meget personale der er til rådighed på fabrikken.

Når skemaet ændres, mistes alle eksisterende reserveringer af timeslots.

Free timeslot Transportørerne og lagerforvalteren skal kunne fjerne en disponering fra en given timeslot.

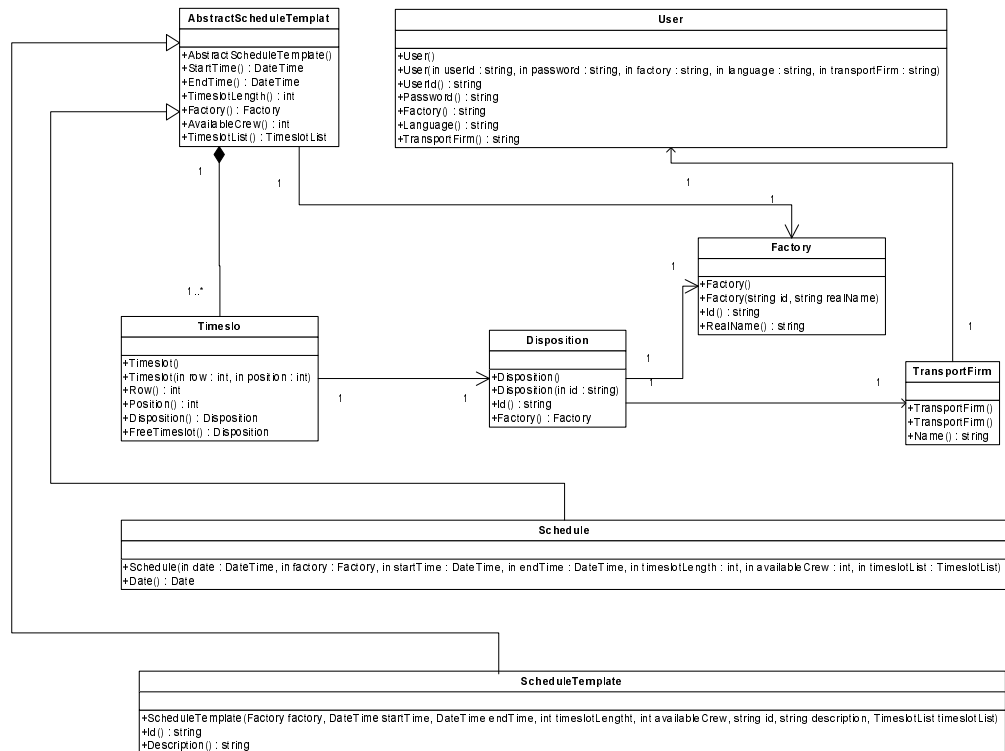
Transportørerne kan kun fjerne deres egne disponeringer.

8.3. Kollaborationer

Efter at have fundet frem til brugsmønstrene for systemet, skal disse gennem kollaborationer føres videre til kollaborationsdiagrammer. Vi har valgt ikke at vise kollaborationerne, da de ikke viser mere information, end kollaborationsdiagrammerne, men vi har været gennem dette trin i udviklingsforløbet.

Kollaborationsdiagrammerne er med til at vise arbejdsgangen for brugsmønsteret. Figur 4 viser et eksempel på et kollaborationsdiagram fra vores projekt. Det viser brugsmønsteret „ShowSchedule“, og tallene viser, i hvilken rækkefølge, tingene sker. Først finder vi ud af, hvilken fabrik der er tale om. Så hentes alle oplysningerne om skemaet, og alle de disponeringer, der stadig ikke er reserveret. Alle disse oplysninger sendes til brugergrænsefladen, hvor de præsenteres for brugeren.

De andre kollaborationsdiagrammer kan findes i bilag B på side 76.



Figur 5: Klassediagram.

8.4. Klassediagram

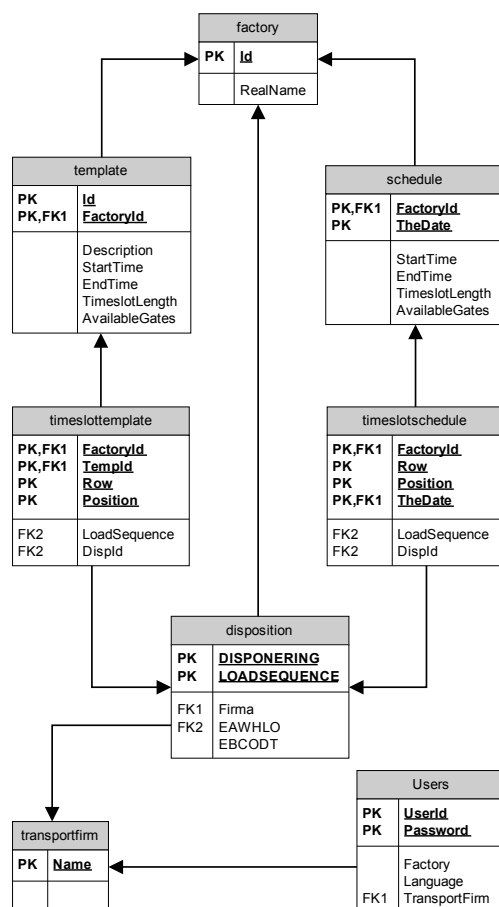
Klassediagrammet i figur 5 viser de klasser, vi fandt frem til gennem arbejdet med kollaborationsdiagrammerne. For overskuelighedens skyld har vi valgt kun at vise et udvalg af get- og set-funktionerne på diagrammet.

Forbindelsen mellem AbstractScheduleTemplate og Timeslot er en komposition, hvilket betyder at et objekt af klassen Timeslot ikke kan eksistere uden et objekt af klassen AbstractScheduleTemplate. Til et Timeslot-objekt kan der være et Disposition-objekt tilknyttet.

En instans af AbstractScheduleTemplate-klassen indeholder et Factory-objekt, samt et eller flere Timeslot-objekter.

Et Disposition-objekt har et Factory-objekt og et Transportfirm-objekt tilknyttet. Transportfirm-objektet er også tilknyttet et User-objekt.

AbstractScheduleTemplate er en abstract klasse, hvilket betyder, at det ikke er muligt at oprette en instans af klassen. Klasserne Schedule og ScheduleTemplate arver



Figur 6: Databasediagram.

egenskaber fra klassen AbstractScheduleTemplate.

8.5. Databasediagram

Databaselayoutet, der kan ses af figur 6, er skabt ud fra klassediagrammet på side 26, og har gennemgået en normalisering til tredje normalform.

„Disposition“-tabellen indeholder foruden de afbillede data endvidere en lang række øvrige data om disponeringerne. Disse data er dog ikke medtaget i figuren, da de ingen relevans har for det system, vi har udviklet, men derimod anvendes i andre af Brødrene Hartmanns it-systemer.

9. Serviceorienteret arkitektur og webservices

De følgende afsnit vil komme nærmere ind på teorien omkring SOA og webservices. Der vil være nogle generelle beskrivelser af emnerne, og vi vil kigge lidt nærmere på brugen af webservices.

9.1. Serviceorienteret arkitektur

Serviceorienteret arkitektur er et begreb, som de første par gange man hører om det, kan virke lidt diffust, da det umiddelbart ikke virker så håndgribeligt.

Tager man selve ordet *service* er det ifølge Hvid Jensen [7] defineret som følgende:

„Indkapslede, løst koblede, genbrugelige softwarekomponenter, som semantisk veldefineret beskriver dets funktionalitet og opførelse, og som kan tilgås via programmer.“

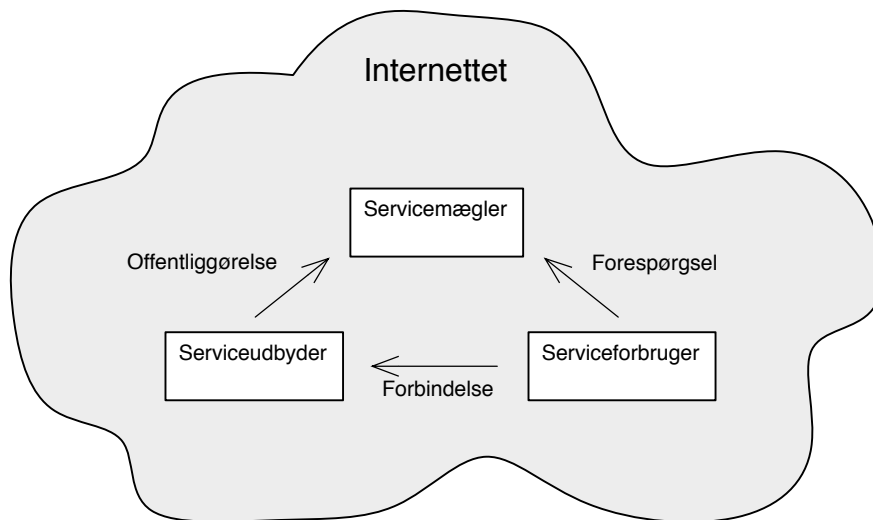
Kobler man dette sammen med ordene *orienteret arkitektur*, beskriver dette tilsammen en måde, hvorpå man kan designe softwarekomponenter.

Målet er, at man designer sine softwarekomponenter således, at de uanset hvilken enhed de skal anvendes på kan tilpasses med det samme, når behovene opstår. En enhed kunne en mobiltelefon eller en pc. Dette skal ske uden at det får indflydelse på applikationer, der i forvejen bruger softwarekomponenterne. De skal kunne forbruges og kommunikere med andre softwarekomponenter i andre netværk, ud fra det ønske, at blive en del af et større fleksibelt netværk, hvor data deles, uden at den enkelte virksomhed sælger ud af sine kernekompetencer.

Alligevel er serviceorienteret arkitektur mere end det. Det er også en arkitektur som tager udgangspunkt i virksomhedens behov. Forretningsprocesser skal kunne udarbejdes ud fra eksisterende komponenter, som kan sammensættes på forskellige måder til at understøtte ændringer i forretningsbehovene, både internt og eksternt. Dette skulle blandt andet være med til at give virksomheden en øget værdi ud af de it-ressourcer, de allerede har.

Serviceorienteret arkitektur skal også på det strategiske niveau, på en simpel og hurtig vis, give virksomheden en mere fleksibel måde at opnå kortsigtede forretningsinitiativer på.

Den serviceorienterede arkitektur kan illustreres ved, at der er services, der skal stilles til rådighed. Disse services skal så kunne findes af andre, for at de kan anvende dem. Se figur 7 på næste side for at finde sammenhængen mellem hvorledes udbyderen offentliggør servicen til en mægler, og hvorledes forbrugeren undersøger udbudet hos en mægler og derefter forbinder til den fundne service.



Figur 7: Serviceorienteret arkitektur udbydes, findes og bruges.

9.2. Oplæring i SOA

Når en virksomhed begynder at anvende serviceorienteret arkitektur, så er der flere afdelinger, der muligvis skal have efteruddannelse, eller i hvert fald noget oplæring i brugen af SOA [7].

Det it-orienteret personale skal dreje fokus over på at udvikle systemer med henblik på at kunne tilbyde udenforstående at dele services, og selv udvikle systemer sammensat af andres eller egne services. Services og applikationer skal kunne ændres uafhængigt af hinanden, og dette vil ske ved en mere trinvis implementering, efterhånden som behovene opstår. Levetiden for virksomhedens applikationer vil dermed øges, da man langsomt udskifter de dele som ikke længere er tidssvarende og ikke hele applikationer. Programmørerne skal vænne sig til at programmere langt mere hændelsesorienteret, og de vil i langt større grad til at skulle genbruge programkode samtidig med, at alt skal forblive løst-koblet.

Ved den øgede brug af webservices vil applikationer også efterhånden skulle dannes ud fra, at de på kørselstidspunktet kombineres af services fra forskellige placeringer, i stedet for, at de bliver tilknyttet moduler under selve udviklingen. En af de centrale ideer bag webservices er, at applikationer i højere grad skal kommunikere direkte med andre applikationer, i stedet for som oftest, at lade en bruger interagere med programmet. Det betyder, at programmørerne i højere grad end de førhen har gjort, skal til at kommunikere via fastlagte standarder. Programmering via disse bliver mere

fleksible, da programmørerne ved udviklingen eller anvendelsen af webservices ikke længere behøver at bekymre sig om, hvilket programmeringssprog, operativsystem eller database de bagvedliggende applikationer anvender.

It-orienteret personale skal også til at tænke it-sikkerhed på et helt andet niveau, end de førhen har været vant til. Gennem webservices vil man blive langt mere afhængig af, om andres sikkerhed lever op til det forventede og ønskede niveau. Man skal også til, at sikre sig, at andres services fungerer korrekt, for selv at kunne anvende dem i sine applikationer, uden at data mister sin semantiske korrekthed, og uden at man mister troværdigheden over for sine egne kunder.

Det er ikke nok kun at overveje sikkerheden, når man er forbruger af webservices. Det er også vigtigt at tænke sikkerhed, når man vil udbyde services. Det er for eksempel vigtigt at sikre, at det ikke er andre, end de kunder med de rette rettigheder, der får adgang til bestemte data, eller at virksomhedens services er sikret mod ondsindede angreb. Afsnit 13 på side 47 beskriver de sikkerhedsaspekter, der skal overvejes, ved brugen af SOA og webservices.

Det forretningsorienterede personale skal også til at se virksomheden med fokus på at være en del af et større forretningsnetværk. De skal kunne forudse, hvordan webservices vil kunne bruges til at gøre den nuværende forretning mere effektiv, og hvordan man via webservices kan skabe nye ydelser til forretningen, uden dog at sælge ud af sine kernekompetencer. Via den serviceorienterede arkitektur skal virksomheden betragte udarbejdelsen af forretningsprocesser som størrelser, der kan sammensættes på kryds og tværs for at understøtte nye eller ændrede forretningsbehov. Dette giver mulighed for en mere fleksibel strategi, hvor der skal sættes på hurtigere og mere kortsigtede forretningsmål.

Forretningsorienteret personale vil skulle kunne beskrive forretningen på en lettere måde end før, gennem aktiviteter som de kan sammensætte og adskille efter behov. Dette gør også, at de skal kunne gøre relationerne til andre virksomheder mere fleksible, og skal hurtigere kunne tilpasse forretningen efter øjeblikkelige ændringer.

9.3. Webservices

Ind under den serviceorienterede arkitektur kan webservices bruges som en specifik implementering. En webservice er i korte træk en bestemt type applikation, hvis formål er at levere service til andre applikationer ved at modtage data fra disse og sende data tilbage, defineret ved et standardiseret XML-format også kaldet WSDL-dokumentet (Web Service Description Language). Transaktionerne er dynamisk forbundet under kørslen og leverer derfor data efter princippet „Just in Time“ (JIT).

Webservices har ry for blandt andet, at ville kunne ændre radikalt i niveauet af interoperabiliteten, ved at minimere kravene til fælles forståelse. Serviceudbyderen og servicemodtageren skal kun sikre sig at de har samme forståelse af WSDL-dokumentet, for at kunne kommunikere med hinandens applikationer. Webservices siges også at

ændre måden hvorpå virksomheder kører deres forretningsmodeller, og baner en helt ny vej for softwareindustrien, se eventuelt afsnit 18 på side 69.

En virksomheds it-løsninger er ofte udviklet over mange år, og udviklet på den software og hardware, som var det nyeste og mest populære, da implementeringen fandt sted. Man koncentrerer sig om at finde den optimale løsning til netop det enkelte system, uden eller med kun sparsom hensyntagen til at data i fremtiden måske skulle kunne udveksles med andre systemer.

Med webservices er der derfor helt nye udfordringer i sigte indenfor integrationen af applikationer, på tværs af forskellige systemer, som oven i købet kan være ejet af forskellige virksomheder. Data vil på grund af WSDL-dokumentets udformning kunne udveksles og integreres uden hensyntagen til forskellige operativsystemer, objektmodeller og programmeringssprog.

Der findes også kritiske røster [8] angående brugen af webservices. Noget af kritikken går på, at webservices er for langsomme, mens andre siger, at de er for besværlige at arbejde med.

Vi mener ikke, at webservices er for langsomme at arbejde med, men at der dog kan være situationer, hvor de ikke er praktiske at anvende. Dette uddybes i afsnit 10.4 på side 38, der handler om hastighed.

Kritikken af, at webservices skulle være besværlige at arbejde med, mener vi ligger meget i, at folk ikke adskiller tingene korrekt. Hvor man før i tiden måske arbejdede med at levere totalløsninger, skal man nu til at se sig selv som dataleverandør. Den løse kobling gør, at man ikke længere har kontrol med hele systemet. Det er derfor vigtigt at aflære, at man ikke kan bare kan lave en ændring i en del af systemet, og så foretage eventuelle følgeændringer i resten af systemet.

Man kan argumentere for, at den løse kobling også findes i den almindelige lagdelte arkitektur. Det er selvfølgelig også svært at ændre noget op gennem flere lag i en lagdelt arkitektur, men det er stadig i mange tilfælde en totalløsning. Det betyder, at så længe man har adgang til hele løsningen, er det muligt at ændre dele i flere lag. Dette er ikke altid muligt når det kommer til SOA og webservices. Hvis der for eksempel er tale om en offentlig webservice, er det ikke muligt at vide, hvem der bruger den. Derfor kan man ikke uden videre ændre noget, der har betydning for interfacet af webservicen.

9.4. Webservices i praksis

Som beskrevet i afsnit 9.3 foregår kommunikationen mellem en webserviceudbyder og -forbruger i et XML-baseret dataformat. Transporten af disse data er principielt uafhængig af webservicen, men oftest vil det foregå via HTTP-protokollen.

```

POST /TransportBooking/default.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 562
SOAPAction: "https://localhost/TransportBooking/default.asmx/GetFactory"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://
www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <AuthHeader xmlns="https://localhost/TransportBooking/default.asmx">
      <Username>brugernavn</Username>
      <Password>hemlig</Password>
    </AuthHeader>
  </soap:Header>
  <soap:Body>
    <GetFactory xmlns="https://localhost/TransportBooking/default.asmx">
      <id_>pto</id_>
    </GetFactory>
  </soap:Body>
</soap:Envelope>

```

Kodeeksempel 1: Eksempel på en SOAP XML-forespørgsel inklusive HTTP-headere til webservicen `getFactory`.

```

<?xml version="1.0" encoding="utf-8" ?>
<Factory xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns="https://localhost/TransportBooking/default.asmx">
  <Id>pto</Id>
  <RealName>Toender</RealName>
</Factory>

```

Kodeeksempel 2: XML-svar på forespørgslen i kodeeksempel 1.

Webservices kan opfattes som et funktionskald over nettet. Der overføres altså et antal parametre fra forbrugeren til udbyderen, hvorefter denne foretager et antal handlinger og returnerer en datamængde som svar.

Af kodeeksempel 1 ses hvorledes et kald af webservicen `getFactory` med parametere-
ren „pto“ er serialiseret til XML og sendt via en HTTP POST-kommando. På en sådan
forespørgsel kan der returneres et svar som i kodeeksempel 2.

Forbrugere af en webservice kan orientere sig om hvilke parametre en webservice
forventer ved at konsultere dens beskrivelse i form af et WSDL-dokument. Som
eksempel kan ses WSDL-dokumentet fra Brødrene Hartmann-casen i bilag D på
side 80 – dette indeholder blandt andet beskrivelsen af `getFactory`-servicen.

Til en del programmeringssprog og -miljøer er udviklet biblioteker til håndtering af
websites, både som forbruger og udbyder. Det er således ikke nødvendigt selv at

konstruere og parse forespørgsler og svar. Visual Studio .NET er et eksempel på et miljø, hvor webservices er meget integreret.

9.4.1. Case: Webservices som en in-house løsning

Det var ikke et krav fra Brødrene Hartmann, at vi implementerede webservices og dermed serviceorienteret arkitektur i projektet. Det var derimod et ønske, vi selv havde, da vi gerne ville tilføre projektet noget funktionalitet, som vi mente ville have en værdi for os i form af, at vi ville komme til at skulle sætte os ind i nyt stof og dermed lære nyt.

Hvid Jensen anbefaler i sin bog [7], at man anvender serviceorienteret arkitektur på alle nye projekter, dog med en vis forsigtighed ved projekter hvor ydeevnen er særdeles vigtig. Han siger også, at man ikke skal serviceorientere alt på en gang, men vente til at man har fået tilstrækkelig erfaring og viden med at anvende webservices og SOA.

Brødrene Hartmann benytter sig ikke i forvejen af webservices og serviceorienteret arkitektur, og har på nuværende tidspunkt heller ingen planer om at benytte eller udbyde services offentligt. Derfor blev projektet en in-house webserviceløsning, så Brødrene Hartmann kan opnå erfaringer med teknologien og forsøge sig med at tage de første skridt til at omlægge virksomheden til at anvende en serviceorienteret arkitektur. Det kan de gøre, uden at det kommer til at have de helt store udviklingsmæssige og økonomiske risici for dem.

Beslutter Brødrene Hartmann sig for på et senere tidspunkt, at de vil gå over til at offentliggøre services og dermed omlægge virksomheden til en serviceorienteret arkitektur, er det vigtigt, at der fra ledelsens side bliver truffet en strategisk beslutning om dette. Hvis ikke de gør dette, kan de risikere at alle de mindre tiltag som hele tiden vil blive gjort i ønsket om at anvende SOA, ikke kører i den samme retning, hvilket kan have konsekvenser i form af manglende overblik.

10. Udviklingsmæssige overvejelser

Når man skal se på de overvejelser, der er under et udviklingsforløb, hvori der indgår webservices, kan man se situationen fra to forskellige vinkler, forbrugerens og udbyderens. Forbrugerens vinkel er det at anvende webservices i sin applikation, og udbyderens vinkel er selve udviklingen af webservices.

Ser vi først på overvejelserne fra forbrugerens side, giver en webservice let adgang til udveksling af data og kan derfor nemt integreres i en applikation. Programmøren behøver ikke at have kendskab til designet af en eventuel database, men skal kun sætte sig ind i hvilke metoder der er til rådighed i webservicen. Hvis webservicen man henter data fra, bruger det samme dataformat som den applikation dataene skal

bruges i, sparer programmøren tid på at skulle konvertere data. Det kan derfor være nyttigt at bruge tid på at sætte sig ind i hvilken standard webservice leverer data i, og derefter selv anvende samme standard.

Hvis man kan finde en webservice der dækker ens behov, vil det tidsmæssigt og økonomisk være en fordel at anvende en sådan fælles ressource, som andre i forvejen har lagt deres erfaringer og arbejde i, i stedet for at skulle udvikle alt fra bunden selv. Der er også mulighed for, at man kan anvende flere forskellige webservices i samme applikation, og man slipper defor selv for at skulle holde styr på at vedligeholde databaserne.

Fra udbyderens side er det nemt at vedligeholde en webservice. Webservicen kan tilføjes ny funktionalitet uden at det får konsekvenser for de applikationer som bruger servicen i forvejen, såfremt interfacet til webservicen ikke ændres. Udviklerne kan nøjes med at opdatere webservicen et lokalt sted, og skal derfor ikke sørge for at vedligeholde dataaccess-klasser og frigive disse hos de forskellige kunder. I et udviklingsværktøj som for eksempel Visual Studio .NET, som vi har brugt til udviklingen af Brødrene Hartmanns projekt, er det meget nemt at oprette en webservice. Basalt set udvikler man blot webservicen som en almindelig klasse og markerer denne som en webservice, samt markerer de metoder i klassen, man ønsker publiceret.

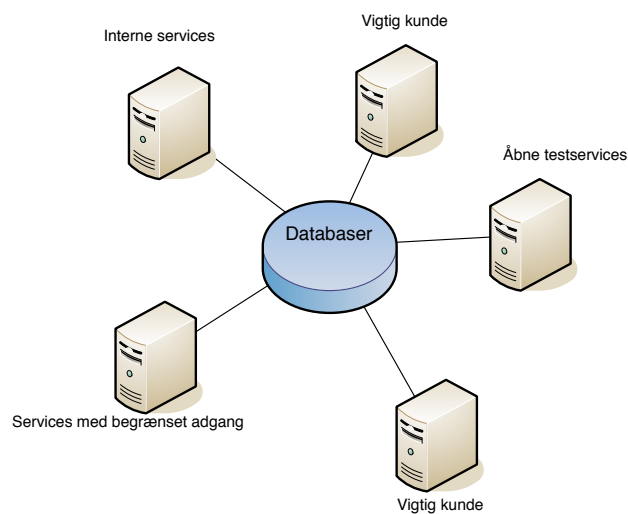
Det er udbyderens opgave at udforme webservicen i en logisk struktur og programmere de metoder, som skal stilles offentligt til rådighed. Visual Studio .NET sørger selv for at oprette mange af de grundlæggende ting, der skal til for at kunne kommunikere med webservicen. For eksempel autogenereres WSDL-dokumentet ud fra, hvordan man har opbygget sin webservice, og ud fra hvilke, annoteringer man har anvendt. Der bliver også automatisk dannet en brugergrænseflade til webservicen, hvor man kan teste inddata og uddata af servicens metoder.

10.1. Tidsbesparende i drift og vedligehold

Om driften og vedligeholdelsen af en webservice er tidsbesparende i forhold til ikke at anvende en webservice, er svært at måle direkte.

Vi mener at vedligeholdelse af en webservice, der tilgår en database, må være nemmere og hurtigere end hvis man havde en dataaccess-klasse, der er distribueret til alle klientapplikationer. Dataaccess-klassen kan ligge hos flere forskellige kunder og på mange lokaliteter, og man skal derfor holde styr på, om alle kunder er opdateret med den nyeste version. Webservicen er kun tilgængelig et sted. Dette betyder at der oftest kun skal vedligeholdes dette ene sted, hvorfor det er mere problemfrit at vedligeholde denne.

Webservicen behøver ikke at være lukket ned under vedligeholdelsen. Kunden vil allerhøjest opleve et par sekunder hvor webservicen ikke er tilgængelig når den nye version frigives. Ændrer man ikke på eksisterende funktionalitet, men tilføjer ny,



Figur 8: Gruppering af services.

vil det ikke give anledning til problemer for kunderne som anvender webservicen i deres applikationer.

Hvis en virksomhed udbyder mange webservices kan det være en fordel at gruppere dem efter forskellige kriterier. Disse kriterier kan være, hvor åbne eller lukkede servicene skal være. Det kan også være efter hvilke forretningsområder, de tilhører. Hvis man har mange services udbudt til en fast kunde, kan man med fordel lægge denne kundes services i en gruppe for sig, se figur 8.

Det betyder også, at hvis man har lavet nogle Service Level Agreements med nogle kunder, hvori der står, at de services virksomheden stiller til rådighed for dem, skal være oppe 99% af tiden, så kan gruppeinddelingen være med til hurtigt at identificere hvilke services, der skal holdes ekstra øje med.

Ved at gruppere virksomhedens webservices, kan man blandt andet også skabe en større stabilitet. Hvis man er ude for, at der en overgang er en meget stor belastning på sine services og de underliggende databaser, så kan man lukke ned for nogle af grupperne. Det kan for eksempel være de grupper af services, der ikke direkte er tilknyttet nogle bestemte kunder eller lignende. Det kan betyde, at man kan undgå databasenedbrud ved at lade sine webservices fungere som stødpude mellem kunder og virksomhedens data.

10.1.1. Case: Tidsbesparende i drift og vedligehold

Selvom Brødrene Hartmann i første omgang kun vil bruge webservices internt i virksomheden, kan det stadig godt betale sig for dem at tænke i grupper. De kan eksempelvis dele deres services op, så dem til transportørernes system ligger et sted, mens fremtidige services til for eksempel deres ERP-systemer kan ligge et andet sted. På den måde undgår de, at forstyrrelser på det ene system er til gene for de medarbejdere, der bruger de andre systemer.

Brødrene Hartmann kan lægge det meste af det nye system på deres maskiner i Lyngby, og det kan spare dem en masse tid, når systemet skal vedligeholdes. Ved at have database, webservices og webserveren liggende hos it-afdelingen i Lyngby kan de blandt andet spare den tid det tager at sende en medarbejder til Tønder, hvis der er sket et nedbrud i systemet. Da ændringer internt i services også kan foregå lokalt, sparer man igen en masse rejsetid.

Windows-applikationen er det eneste, der bliver nødt til at ligge udenfor Lyngby. Her kan Brødrene Hartmann igen spare tid på vedligehold, ved at lave eventuelle ændringer i Lyngby, og så sende DLL-filen og EXE-filen til Tønder. Så skal medarbejderne i Tønder bare overskrive den tidligere version af filen, og så skulle tingene køre videre.

10.2. Offentlig værktøjskasse

I virksomheder med mange data/databaser og mange dertilhørende webservices, kan det være fordelagtigt at opbygge en værktøjskasse bestående af disse services for at danne et samlet overblik over hvad der udbydes. Værktøjskassebegrebet kendes for eksempel også fra komponentudvikling.

En sådan værktøjskasse kan være med til at give et overblik over servicene på en struktureret måde og derigennem give en øget genavendelighed af etablerede services, i stedet for ny- eller genudvikling. Det kan øge fokus på de muligheder, der faktisk ligger i virksomheden for at anvende deres services adgang til data, og eventuelle nye kombinationer af dette.

Som værktøjskasse til webservices kan anvendes UDDI-teknologien [1, kap. 3]. UDDI-teknologien er egentlig designet til at annoncere og registrere udbudet af webservices virksomheder i mellem, men internt i specielt større virksomheder kan teknologien også anvendes for at give et samlet overblik over hvilke webservices, man internt har udviklet og udbyder.

10.2.1. Case: Offentlig værktøjskasse

Da dette projekt er det første der anvender SOA og webservices i Brødrene Hartmann og virksomheden ikke foreløbig står over for at implementere SOA og webservices

yderligere, ser vi i den nuværende situation ingen grund til, at der implementeres en offentlig værktøjskasse i form af en UDDI-server i virksomheden. Vi har derfor ikke analyseret teknologien nærmere.

Såfremt der på længere sigt omlægges til brug af SOA og webservices, tegner billedet sig dog anderledes. Med Brødrene Hartmanns vidtstrakte brug af it i ERP-systemer og til opsamling af produktionsdata, vil det være en klar fordel, såfremt data og services fra disse publiceres via webservices, at der til den tid skabes et samlet overblik over udbuddet ved hjælp af for eksempel en UDDI-service.

10.3. Portabilitet og interoperabilitet

Når man snakker systemudvikling, har det altid været en vigtig overvejelse, hvordan man kunne gøre sine programmer portable. Portabilitet betyder, at man uden alt for meget besvær kan flytte sine programmer fra én platform, for eksempel Windows, til en anden, som for eksempel Linux eller Mac.

Java har i lang tid været velanset når det kommer til portabilitet, fordi man kan oversætte et java-program til java-bytecode på én platform, og så køre programmet på en hvilken som helst platform, så længe der er en Java virtuel maskine (JVM) installeret.

Det er lidt den samme fremgangsmåde .NET virker på. Hvis man har installeret .NET-frameworket på sin maskine er det muligt at køre de programmer, der er udviklet på .NET. Det er dog ikke på alle platforme der understøttes af .NET-frameworket endnu. Der er blandt andet ved at blive udviklet et framework til Linux, som hedder MONO, der skal medføre at .NET-programmer også kan afvikles på Linux.

En ting, som er blevet lidt misforstået omkring webservices, er at de også skulle være portable. Man kan sige, at de sagtens kan udvikles, så de er portable, i de tilfælde, hvor udbyderen vælger at udskifte den platform, de kører på. Ideen bag webservices er derimod, at de skal have en høj interoperabilitet [3]. Interoperabilitet betyder, at beskeder sendt fra ét program på én platform kan modtages og forstås af et andet program, der kører på en anden platform. Her passer webservices godt ind, da det ikke er meningen, at disse services skal lægges ud på alle de maskiner, der skal bruge dem. Virksomhedens webservices skal derimod kun ligge ét sted, og så skal de andre maskiner tilgå dem dette ene sted. Grunden til at det kan lade sig gøre at sende disse beskeder på tværs af platforme og programmeringssprog er, at der er blevet udarbejdet en række standarder og specifikationer af World Wide Web Consortium.

10.3.1. Case: Portabilitet og interoperabilitet

Vi har udviklet Brødrene Hartmanns system til Windows på .NET-plattformen. De vil dog kunne portere systemet til Linux, hvis MONO er installeret, og hvis de

komponenter, der bliver brugt i vores system er blevet skrevet til frameworket i MONO. Vi forestiller os dog ikke at dette vil ske foreløbig, da Brødrene Hartmanns it-politikker foreskriver, at virksomheden kun anvender Windows på arbejdsstationer.

Fordi vi har implementeret webservices, vil det være muligt for Brødrene Hartmann, på et tidspunkt, at lade transportørerne selv stå for udviklingen af deres brugergrænseflader, uden at Brødrene Hartmann kender noget til de systemer, som transportørerne bruger.

Transportørerne kan godt arbejde med eksempelvis Linux på deres systemer, og så udvikle grænseflader i Java, der bruger de services, som Brødrene Hartmann stiller til rådighed. Det kan også være at der på et senere tidspunkt bliver stillet webservices til rådighed, som transportørerne kan bruge direkte i deres eksisterende systemer. Det kunne for eksempel være statistikoplysninger eller lignende.

10.4. Hastighed

Før man i en applikation udskifter den direkte tilgang til en database med en tilgang via en webservice, må man overveje, hvilken konsekvens det får for afviklingshastigheden af applikationen.

I webservices pakkes både forespørgsel og svar ind i XML og HTTP-protokollen, mens den direkte forespørgsel foregår i et mere kompakt format.

Den samlede datamængde der transmitteres ved to ens forespørgsler og svar via en webservice og en direkte adgang, må derfor være større via webservicen på grund af det overhead XML-strukturen leverer.

Der vil både være en hastighedsnedgang i forbindelse med, at forespørgsler og svar skal serialiseres samt pakkes ind og ud af webservicens XML-struktur, og der vil være en hastighedsnedgang ved at overføre en større mængde data via nettet.

Nettet vil være flaskehalsen i kommunikation, og det ekstra tidsforbrug der opstår ved serialisering samt ind- og udpakning, vil derfor ikke være udslagsgivende. På servere med meget høj belastning vil det stadig være netværkskapaciteten, der vil være flaskehalsen.

XML-overheadet udgør procentuelt mere på små datamængder når de er serialiseret, end det gør på større „klumper“ af data, som for eksempel indlejrede billeder.

I en konkret anvendelse må man derfor vurdere, hvilken type data man har brug for og med hvilken hyppighed. Et system der har brug for små datamængder, men til gengæld har brug for at opdatere disse i et hurtigt skiftende realtidssystem, kan flaskehalsen og overheadet udgøre et problem, mens de i andre typer systemer kan negligeres til fordel for de øvrige fordele, ved at anvende webservices som tilgang til databaser.

10.4.1. Case: Hastighed

I projektet, vi har udviklet for Brødre Hartmann, har vi målt datamængden til typisk mellem 3% og 8% af et webservice XML-svar, men den samlede datamængde udgør dog kun op til 27 kb, og opdateringsfrekvensen i klienterne er ikke så stor, at det nedsætter anvendeligheden af disse. Erfaring med udviklingen af systemet siger desuden, at brugen af webservices ikke udgør et problem for hastigheden i systemet.

10.5. Økonomiske fordele ved at tilgå databaser via webservices

En af ideerne med SOA og webservices er at tilvejebringe et værktøj, der kan skabe let og billig adgang til data.

Ved at tilgå databaser via webservices, bliver koblingen mellem de forskellige applikationer, løsere end ved traditionel anvendelse af applikationer med tilknyttet databaselag. Dette betyder blandt andet, at skal der ske ændringer i for eksempel en tabelstruktur, kan man måske nøjes med at udvide webservicen med en enkelt ny funktion, eller rette i et par enkelte eksisterende funktioner, ét sted. Dette er meget tidsbesparende i forhold til den traditionelle måde, hvor man skal ind og opdatere datalaget i flere forskellige applikationer på flere forskellige maskiner, som anvender databasen.

Udover dette skal programmørerne ikke bekymre sig så meget om systemets infrastruktur mere, men mere have fokus på at skrive en samarbejdende applikation og have fokus på at levere en bedre og optimeret service. Hvis programmørerne fra starten opbygger webservicen således, at der hele tiden nemt kan genbruges, rettes eller tilføjes ny funktionalitet, vil det i sidste ende kunne betale sig økonomisk.

Et eksempel kunne være en webservice, som returnerer et navn og en adresse via cpr-nummer. Webservicen kunne opbygges, så den havde en funktion, som kun returnerer navnet, en funktion som kun returnerer adressen, og en funktion som både returnerer navn og adresse via den pågældende persons cpr-nummer. Senere kunne der tilføjes en funktion der returnerer navnene på eksempelvis 100 personer ad gangen. På denne måde garanterer man også, at man ikke får en allerede eksisterende applikation, som benytter webservicen, til at fungere uhensigtsmæssig, når der tilføjes ny funktionalitet.

Det kunne tænkes, at mange virksomheder kunne få økonomiske fordele ved at udbyde allerede eksisterende applikationsmoduler som webservices. Inddeler man adgangen til sine webservices i forskellige grupper, kan man også bedre målrette en webservice efter den tiltænkte målgruppe. For kunden giver dette større tilfredshed, og det kan tiltrække flere kunder, når man kun behøver at investere i den del af webservicen, man specifikt har brug for.

For virksomheder der kunne have udbytte af at udveksle data med andre virksomheder, kan man også samarbejde om at udbyde en webservice, og dermed deles om de økonomiske udgifter og indtægter.

Økonomiske fordele ved at anvende en webservice er også, at det ikke har nogen betydning hvilket operativsystem eller programmeringssprog servicen skal integreres i. Det betyder, at man ikke skal til at sætte sig ind i en ny teknologi, men kan fortsætte med den som man plejer, hvorfor hastigheden hvormed en applikation designes og bygges sandsynligvis øges.

Det er vigtigt at sætte sig ind i interfacet til webservicen for at kunne anvende den korrekt, og anvender man de samme standarder der foreskrives i WSDL-dokumentet, vil det lette integreringen en del og samlet set vil det oftest bringe de totale omkostninger ned.

10.5.1. Case: Økonomiske fordele ved webservices

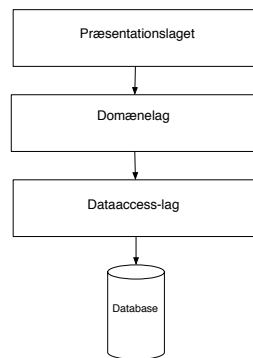
At vi har implementeret en webservice som tilgår databasen hos Brødrene Hartmann, har ikke de store økonomiske fordele for dem direkte, da systemet er en in-house løsning. At webservicen er nem at vedligeholde vil blive en økonomisk fordel for dem, se afsnit 10.1 på side 34.

På længere sigt kunne vi forestille os, at Brødrene Hartmann vil kunne få en økonomisk gevinst ved at tilgå deres eksisterende datamængder gennem webservices. De vil hurtigere og billigere kunne imødekomme nye behov, og vil nemmere og på en mere ensartet måde kunne tilgå data i deres ERP-system, og dermed også få et bedre overblik over den samlede datamængde. Se også afsnit 4.4 på side 12 der beskriver de økonomiske fordele ved indførelsen af det nye system.

11. Arkitektur

Arkitekturen i en applikation kan bygges op i lag. Det er meget almindeligt at dele op i lag, da det giver en lang række fordele, både når programmer skal skrives, og hvis der skal ændres noget, når programmet først er i brug. Arkitekturen består ofte af tre lag; præsentations-, domæne- og dataaccess-lag, se figur 9 på næste side. Domænelaget kaldes også nogle gange forretningslogiklaget. Teorien omkring lagdeling beskrives blandt andet i Bennett [2].

Princippet med at opdele et program i mindre dele gør også udviklingsprocessen mere overskuelig. Hvis udviklingsgruppen fra begyndelsen aftaler interfacet for de forskellige lag, både hvad angår argumenter, returtyper og betydning, så har programmøren for så vidt frie tøjler til lave hvad han vil, så længe han overholder lagets interface.



Figur 9: Standardarkitekturen i følge Bennett [2].

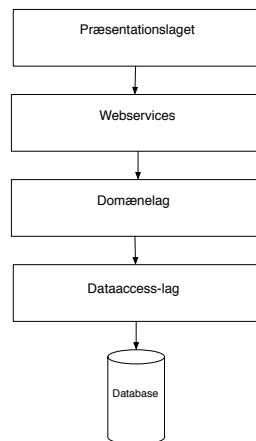
Lagdelingen gør det også muligt at udskifte et lag, uden at det har nogen betydning for de andre lag, så længe man ikke ændrer noget ved lagets interface.

11.1. Den overordnede arkitektur

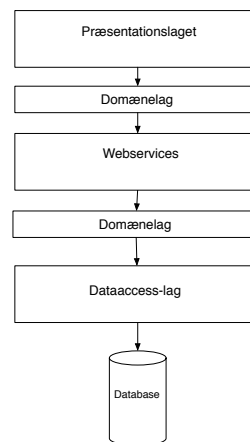
Vi mener, at der er flere steder i den lagdelte arkitektur, hvor webservices kan passes ind. Et sted hvor de kunne placeres ville være mellem præsentationslaget og domænelaget. Webservices ville her bruge de underliggende lag, i de services de stiller til rådighed. Der er dog en lille forskel i forhold til at have hele applikationen liggende, da det er nødvendigt at referere til webservicen via en internetadresse i stedet for en mappe på den lokale maskine.

Man ville også kunne argumentere for at domænelaget ligger på begge sider af webservicelaget. Det mener vi fordi domænelaget bliver stillet til rådighed gennem webservicen. Det betyder at man kan oprette objekter i præsentationslaget og i princippet arbejde med domænestrukturen i dette lag. Det betyder også at det er muligt at arbejde med nogle objekter, der ikke er opdateret, og det kan være med til at give brugeren et forkert billede af virkeligheden. Det er dog ikke muligt som udbyder at forhindre at dette sker, da det ikke er muligt at styre, hvordan virksomhedens webservices bliver brugt af forbrugerne. Figur 10 og 11 på den følgende side illustrerer hvordan disse lagdelinger kan se ud.

Det at webservices passer så godt ind i de allerede eksisterende arkitekturer, ser vi som en stor fordel, da det så ikke er nødvendigt at ændre sine udviklingsmetoder, for at kunne indkorporere webservices i sine applikationer. Webservices kan bare sættes ind oven på domænelaget, uden at det har nogen synderlig indflydelse på de andre lag.



Figur 10: Webservices placeret over domænelaget.



Figur 11: Domænelaget over og under webservicene.

Det giver mest mening at kigge på den overordnede arkitektur, når man udvikler applikationer med webservices til internt brug i virksomheden. Hvis man bruger webservices som andre virksomheder har udviklet, kan det give mere mening at kigge på den arkitektur, der er beskrevet i afsnit 11.2.

11.2. Arkitektur i SOA

Der er bred enighed om, at SOA overordnet set består af tre dele, applikations-, service- og komponentarkitekturen [11].

Applikationsarkitekturen er den del, der bruger de webservices, der er stillet til rådighed i opbygningen af større eller mindre programmer.

Servicearkitekturen er mellemeddet mellem applikationerne, der bruger services, på den ene side og implementationen på den anden.

Komponentarkitekturen er med til fortælle hvordan services er bygget op. Det betyder, at man kan se på webservices fra to sider, fra udbyderens side og fra forbrugerens side.

Udbyderen behøver for så vidt ikke vide, til hvad eller hvordan forbrugeren bruger de services, han har stillet til rådighed. Udbyderens opgave er at sørge for, at hans services virker og returnerer det de skal.

Arkitekturen af udbyderens applikation, i denne sammenhæng en eller flere webservices, ligner meget arkitekturen fra figur 9 på side 41, dog er webservices ikke et nyt lag, men ligger som erstatning for præsentationslaget.

Forbrugeren af webservices behøver omvendt ikke vide, hvordan disse services er implementeret, så længe de overholder de beskrivelser der findes for dem.

Arkitekturmæssigt kan vi se på det på to måder.

Den ene måde er hvis forbrugeren både bruger webservices og nogle lokale komponenter i sin applikation. Dette giver arkitekturen fra figur 9 på side 41, og så vil de klasser, der bruger webservices, ligge i domænelaget sammen med de klasser der bruger det lokale dataaccess-lag.

Den anden måde er at forbrugeren kun bruger webservices i sin applikation. Det giver, set fra hans side, kun et par lag, præsentationslaget og muligvis et domænelag med nogle kontrolklasser, der gør brug af webservices.

11.3. Case: Arkitektur

I forbindelse med udviklingen af programmet til Brødrene Hartmann har vi arbejdet med UP som metode, og vi har derfor valgt at tage udgangspunkt i den lagdeling, der står beskrevet i Bennett [2]. Grunden til at vi bruger den lagdeling er at vi skal udvikle et system fra bunden, og det er en god model at gå ud fra, da det er nemt at arbejde med de forskellige lag hver for sig.

Det, at vi har skullet udvikle et system fra bunden, har også betydet, at vi har fået mulighed for at arbejde med webservices set både fra udbyderens og forbrugers synsvinkel.

Fordi vi arbejdede med denne lagdelte arkitektur, var det nemt at dele systemet op i mindre dele, som vi kunne arbejde på hver for sig. Vi brugte en del tid på at aftale de interfaces, der skulle være mellem lagene. Derfor kan man sige at når vi udviklede præsentationslaget, så havde vi forbruger-kasketten på, mens de resterende lag blev udviklet med udbyder-kasketten på.

Opdelingen i server og klienter virkede rimelig ligefrem. Fordi vi arbejder med SOA synes vi, at det giver mening at sende hele objekter eller objektlister over nettet via webservices-kaldene. Derfor ligger alle klasserne fra klassediagrammet i domænemodellen på serversiden.

For at finde webservice-metoderne har vi kigget på de funktionskald, der gik fra kontrolklassen til entitetsklasserne i vores kollaborationsdiagrammer. De viser, hvad det er, man skal bruge i klienterne, for at kunne vise data på en sammenhængende måde i en brugergrænseflade.

I vores arkitektur har vi et dataaccess-lag liggende over vores database. Dette lag sørger for tilgangen til databasen, og for at lave en fleksibel tilgang, har vi lagt

forbindelsesoplysningerne til databasen i en fil for sig. På den måde kan vi ændre disse oplysninger, uden at vi skal genoversætte dette lag.

Over dette lag ligger vores domænelag som indeholder vores objektstruktur, og dette lag er bygget op med udgangspunkt i vores klassediagram.

Vi har lavet en webserviceklasse til alle vores webmetoder. Den gør brug af dataaccess-laget for at hente de oplysninger der er nødvendige, for at kunne oprette instanser af de klasser, der er beskrevet i domænelaget.

Disse tre lag beskriver samlet set den server, der danner grundlaget for vores applikation. Ved siden af denne server har vi to forskellige brugergrænseflader. Den ene grænseflade er en windowsapplikation, mens den anden er en webapplikation, der kan tilgås via en vilkårlig webbrowser.

Vi arbejder også med domænelaget i brugergrænsefladerne. Forskellen på den traditionelle måde at gøre det på, og denne måde er, at vi i dette tilfælde tilgår klasserne gennem webservicelaget. Selvom dette kan betyde, at man kan komme til at arbejde med gamle data, synes vi at det er en god måde at opretholde klassestrukturen på tværs af webservices.

En alternativ måde at bruge domænelaget i brugergrænsefladen er at opbevare en lokal kopi af klassestrukturen på den maskine, som grænsefladen ligger på. Så kan man lade webservicene returnere de oplysninger, der skal bruges til at oprette et objekt, i stedet for at returnere et objekt, som vi gør nu. Dette er dog ikke særlig smart fordi vi mister noget af det overblik, vi kan give ved at returnere objekter. Det er svært at se, hvad det er for et objekt, vi får tilbage, hvis det bare er en lang liste af basale typer.

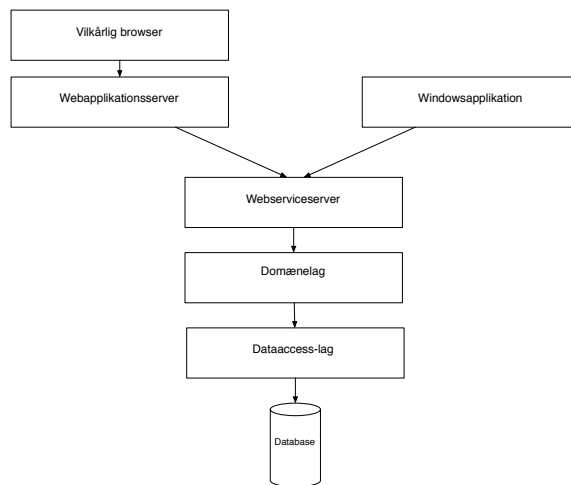
Da projektet blev aftalt leveret til Brødrene Hartmann som en prototype til brug for deres egen implementation – og siden hen til videreudvikling – får Brødrene Hartmann nytte af de samme fordele som udviklere, med den beskrevne arkitektur, som vi har haft som udviklere.

Opdelingen i en klient- og serverarkitektur vil endvidere være en fordel for Brødrene Hartmann da det administrative klientsoftware vil blive anvendt på fabrikken i Tønder, mens serverdelen og webklienten med stor sandsynlighed vil blive anvendt og drevet fra hovedkontoret i Lyngby.

Figur 12 på modstående side viser den tilstræbte arkitektur i projektet.

12. Sikring af databasebegreber i webservices

Med skiftet fra at tilgå en database direkte, til i stedet at se dataene via en webservice, vil forbrugeren af webservicen således ikke længere være tilknyttet den del af udviklingsprocessen, der har med databaser at gøre.



Figur 12: Arkitekturen i projektet.

Igennem mere end 40 år er der, i tilknytning til databaser, udviklet en omfattende teori om disse og hvorledes data sikres. Når databasen forsvinder ud af webserviceforbrugerens synsfelt kan man frygte, at denne teori forsvinder med.

Anskuer vi udbuddet af datatilgangen med SOA-briller, vil det være naturligt at opfatte én transaktion som én sammenhængende service. Derfor taler dette for, at transaktionen skal udbydes som ét samlet webservicekald. På den måde sikrer udbyderen af en webservice, at databasebegreberne overholdes, og forbrugeren ved, at transaktionerne sikres andetsteds.

Hvis udbuddet af webservicen sker i en verden, hvor der ikke nødvendigvis er tæt samspil mellem udbyder og forbruger af servicen, er det umuligt at sikre, at forbrugerne af servicen laver et afsluttende webservicekald, der commit'er transaktionen til databasen. Dette kan medføre unødvendig låsning af databasen og nedsat tilgængelighed af webservicen.

De ovenstående forhold taget i betragtning kan vi anbefale altid at have en „én til én“-sammenhæng mellem transaktioner og webservices. Det vil sige, at har man en handling, der skal udføres som en transaktion, skal hele denne handling håndteres af ét webservicekald.

12.1. ACID

Et af de bedst kendte begreber fra databaseverdenen er ACID-begrebet, der skal sikre at en transaktion altid udføres fuldstændigt og konsistent.

ACID-begrebet består af [5, kap. 17.3]:

Atomicity En transaktion er at opfatte som en udelelig enhed. Enten udføres hele transaktionen eller også udføres den slet ikke.

Consistency preservation En transaktion skal, hvis den udføres, bringe databasen fra en sammenhængende tilstand til en ny sammenhængende tilstand.

Isolation Udførelsen af en transaktion sker isoleret fra øvrige transaktioner så disse ikke kan påvirke udfaldet af transaktionen.

Durability De ændringer der påføres en database skal være blivende og må ikke kunne gå tabt.

Webservices kommunikerer de facto altid via HTTP-protokollen, og da HTTP-protokollen er tilstandsløs mellem de enkelte kald, er det som udgangspunkt ikke muligt at sikre en transaktion over mere end ét HTTP-/webservice-kald.

I webverdenen findes en række teknikker til at omgå HTTP-protokollens tilstandsløshed, og disse kan også anvendes i sammenhæng med webservices [6, kap. 5].

Det kan for eksempel opnås ved at etablere en såkaldt sessionsvariabel, der identificerer den igangværende transaktion. Værdien af sessionsvariablen medsendes så af klienten ved hvert kald af webservices.

Ved hjælp af sessionsvariablen vil det således være muligt at knytte flere webservicekald til den samme session, og dermed til samme transaktion. Vi kan dermed teoretisk set sikre transaktionen. I praksis kan dette dog være svært, da de standardbiblioteker til databasehåndtering, der følger med for eksempel .NET, kræver at transaktionerne knyttes til en åben databaseforbindelse, og denne er svær at holde åben på tværs af to webservicekald.

Selvom forbrugeren af en webservice ikke længere sidder med oplevelsen af at arbejde med en database, er det vigtigt at transaktionsbegrebet ikke går tabt. Der er en risiko for at forbrugeren med tiden, og når forretningslogikken ændrer sig, vil gøre brug af eksisterende webservices på en ny måde. Risikoen kan opstå, hvis en ændring i forretningslogikken gør at det, der tidligere kunne udføres som to uafhængige webservicekald, og dermed transaktioner, nu må opfattes som værende én transaktion. Forbrugeren må være opmærksom på, at der er tale om én transaktion, og at sådanne kræver særlige hensyn fra webservicens side.

I stedet for at forbrugeren blot sammensætter en ny applikation af flere webservices, for at kunne udføre sin transaktion, må han henvende sig hos udbyderen og anmode om, at der stilles en ny service til rådighed, der håndterer hele transaktionen.

```

DataAccess db = null;

try {
    db.transaction = db.connection.BeginTransaction();

    // her udføres diverse SQL-sætninger

    db.transaction = db.connection.Commit();
}
catch {
    db.transaction.Rollback();
    db.Dispose();
    SoapException se = new SoapException(...);
    throw se;
}
finally
{
    db.Dispose();
}

```

Kodeeksempel 3: Opbygning af programkode med transaktionsstyring.

12.2. Case: Transaktionssikring

For at opretholde transaktionsbegrebet internt i webservicene bruger vi .NET-klassen `System.Data.Odbc.OdbcTransaction`.

Klassen stiller blandt andet metoderne `BeginTransaction`, `Commit` og `Rollback` til rådighed.

Vi har valgt en løsning, hvor alle databasesætninger udføres inde i en `try`-blok, inklusive `BeginTransaction` og `Commit`. Se kodeeksempel 3.

Støder vi under udførelsen på forhold, der kræver et `Rollback`, kaster vi en `exception`, og udfører `rollback`'et i den til `try`-blokken hørende `catch`-blok.

Det har vi valgt som generel tilgangsvinkel, da det også sikrer os mod ikke-forventede fejl, der måtte smide en `exception` og for at samle oprydningdelen, i forbindelse med et `rollback`, ét sted for hver webservicemetode.

13. Sikkerhedsaspekter

Næsten lige så længe som der har været computere tilgængelige på internettet, har der været folk, som uberettiget har forsøgt at tilgå data på disse maskiner. Derfor er sikkerhed en meget vigtig brik i forbindelse med SOA og udviklingen af webservices.

Når man ønsker at bruge soA-tankegangen til at stille services til rådighed, er det meget vigtigt at huske, på at der kan komme forandringer i sikkerheden. Hvis en virksomhed tilbyder data gennem services, som andre virksomheder kan bruge i applikationer eller lignende, så udvider den samtidig sin sikkerhedsbarriere til også at indeholde de virksomheder, der anvender dens services. Det kan sagtens være, at der er lavet nogle gode Service Level Agreements mellem to virksomheder, og udbyderen har måske også opbygget en god sikkerhedsstruktur i sine services. Alligevel kan et hul i forbrugers sikkerhed betyde, at udbyderens data ikke længere er sikre.

Når vi taler om sikkerhed i en kommunikation mellem to parter, som for eksempel mellem en webserviceforbruger og en webserviceudbyder, eller mellem en webapplikation og slutbrugeren, er der fem punkter vi må sikres os (Hvid Jensen [7, kap. 9.1]):

Autentifikation Hermed menes at modtageren af en ordre, webservicekald og lignende skal have en sikker identitet af afsenderen.

Autorisation Verifikation af at afsenderen er berettiget til at foretage en given ordre, webservicekald eller lignende.

Konfidentialitet Kun de involverede parter må kunne få adgang til ordren, webservicekaldet eller lignende.

Dataintegritet Sikkerhed for at ordren er uændret under transport fra afsender til modtager.

Uaføiselighed Ingen af parterne må kunne benægte deres deltagelse i ordreafgivelsen efterfølgende.

Der er mange overvejelser omkring hvor „åbne“ webservices skal være, altså hvor mange der skal have adgang til den funktionalitet, de tilbyder. Dette kommer meget an på, hvilken slags webservices, der er tale om. Hvis der er nogle enkelte virksomheder, der gennem aftaler har fået adgang til noget data, skal der nogle stramme sikkerhedsopsætninger til, så det kun er medarbejdere fra disse virksomheder, der kan anvende de tilgængelige webservices.

På den anden side kan man vælge at tilbyde nogle webservices til alle, der har lyst til at bruge dem, så her kan man have en meget åben forbindelse til disse.

Det er dog altid nødvendigt at sikre sig den programtekniske sikkerhed, se afsnit 13.7 på side 54, så webservicekald ikke opnår andre rettigheder, end de oprindeligt var tiltænkt, upåagtet at autentifikation og autorisation er succesfulde.

13.1. Autentifikation

Autentifikation kan blandt andet opnås gennem kryptering med anvendelse af nøgler. Det kan for eksempel være asymmetrisk kryptering, hvor der ved hjælp af offentlige og private nøgler gives sikkerhed for, hvem den anden part i en kommunikation er.

```

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Header>
    <AuthHeader>
      <Username>brugernavn</Username>
      <Password>kodeord</Password>
    </AuthHeader>
  </soap:Header>
  <soap:Body>
    <WebServiceMethodName xmlns="Webservice Namespace" />
  </soap:Body>
</soap:Envelope>

```

Kodeeksempel 4: SOAP-besked med brugernavn og kodeord i AuthHeader.

Mere simpelt kan det foretages ved hjælp af brugernavn og kodeord. Her bør det dog sikres, at konfidentialitetskravet er opfyldt så brugernavn og kodeord ikke tilfalder udenforstående under transporten.

En simpel metode at anvende brugernavn og kodeord i webservices er at indlejre disse i SOAP-beskedernes header, se kodeeksempel 4. Metoden er ikke standardiseret, men er bredt anvendt, se Pollard [10], og understøttes blandt andet i .NET-rammearket.

Når brugernavn og kodeord skal valideres, vil det være en fordel at benytte eksisterende valideringssystemer, såfremt de findes og bruges. Det kan være sig en Active Directory- eller LDAP-server, der i forvejen bruges til adgangskontrol. Derved undgås dobbeltadministration af brugere.

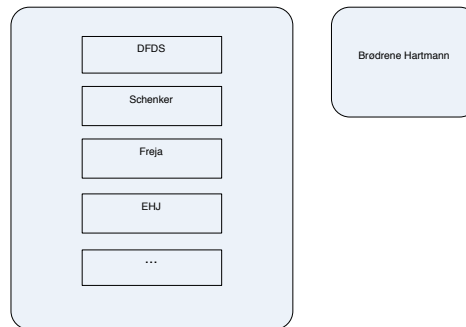
Findes der ingen eksisterende adgangskontrolsystemer, må man introducere et til lejligheden. Igen kan det være en overvejelse værd, om man skal tage skridtet fuldt ud og iværksætte et decideret adgangskontrolsystem som Active Directory, eller om man kan nøjes med validering af brugernavn og kodeord på baggrund af opslag i et databasesystem eller en fil på disken.

13.1.1. Case: Autentifikation

I Brødrene Hartmann var der ingen eksisterende adgangskontrolløsning for transportørere i det, de som ekstern leverandør ikke har adgang til it-systemerne hos Brødrene Hartmann.

Det var altså nødvendigt at indføre et adgangskontrolsystem for transportørerne. Valget faldt her på at administrere dette i en databasetabel for ikke at introducere ekstra kompleksitet i projektet, og da transportørerne ikke står overfor at skulle have adgang til andre systemer.

Vi har valgt at overføre brugernavn og kodeord ved hjælp af en „AuthHeader“-løsning som beskrevet i foregående afsnit.



Figur 13: Inndeling af brugergrupper i flere niveauer.

13.2. Authorisation

En nødvendig forudsætning for at kunne autorisere brugen af en webservice er at identifikationen er på plads. Med andre ord må man først sikre sig autentiteten af den bruger, der ønsker at anvende servicen, og dernæst kan man tage stilling til, om brugeren er autoriseret til at få handlingen udført.

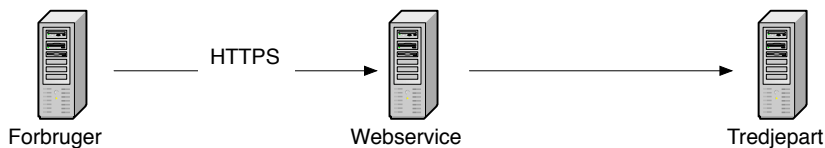
Selve autorisationsspørgsmålet er afhængigt af den problemstilling, der ønskes løst. I visse tilfælde giver det mening af inddele brugere i grupper og autorisere på baggrund af gruppetilhørsforholdet. I andre problemstillinger er det identiteten af brugeren selv, der er afgørende.

Til mere avancerede autentifikations- og autorisationsmuligheder er der udarbejdet en række udvidelser til webservicestandarderne, blandt andet Security Assertions Markup Language (SAML). SAML er designet til at beskrive autentifikations- og autorisationsinformation.

Det åbner mulighed for, at selve autentifikations- og autorisationsprocesserne foregår hos en tredjepart, der således via SAML, leverer oplysninger om en bruger er autoriseret til en bestemt service eller ej.

13.2.1. Case: Authorisation

I Brødrene Hartmann-projektet sker autorisationen på baggrund af grupper. Alle brugerere er tilknyttet en virksomhed, og man kan således tale om én gruppe der indeholder alle brugere hos en given transportør, men også om gruppen af transportører i det hele taget, i modsætning til de brugere der er ansat hos Brødrene Hartmann, se figur 13.



Figur 14: Ved kryptering på transportlaget opnås der kun kryptering fra forbruger til udbyder, men ikke direkte videre til en eventuelt tredjepart.

Transportørere har som helhed adgang til udvalgte oplysninger, mens Brødrene Hartmann har adgang til samtlige oplysninger. Endvidere har en given transportør adgang til data om egne disponeringer, men ikke til konkurrenternes.

13.3. Konfidentialitet

For at beskytte udvekslingen af oplysninger så uvedkommende parter ikke kan følge med, anvendes der traditionelt kryptering af dataene.

De fleste webserviceløsninger der implementeres stiller webservicen til rådighed via HTTP, og det er derfor fristende at overlade krypteringen til transportlaget, og således stille webservicen til rådighed ved at anvende SSL over HTTP.

Fordelene er, at metoden er velkendt og anvendes yderst hyppigt i andre sammenhænge end webservices.

I en række tilfælde vil det dog ikke være tilstrækkeligt at anvende HTTPS. I princippet er webservices uafhængige af transportlaget og såfremt andre transportprotokoller ønskes anvendt, må disse ligeledes kunne anvende kryptering på transportniveauet. Et eksempel kunne være s/MIME ved e-mail.

Et andet problem ved at overlade krypteringen til transportniveauet, som for eksempel HTTPS, er at krypteringen herved begrænses til kun at kunne foretages mellem de to parter i forbindelsen, og kun under selve transporten. Det er således ikke muligt at sikre en ubrudt krypteret forbindelse fra forbrugeren af webservicen og til en involveret tredjepart, se figur 14. Kryptering under videretransporten må derfor håndteres særskilt.

Såfremt en webserviceimplementation anvender andre webservices, eller webservicen kræver at oplysninger videresendes til en tredjepart, er det således ikke muligt, at sikre kryptering hele vejen fra forbrugeren og til tredjemanden.

World Wide Web Consortium har udviklet en specifikation [12] til kryptering af XML-dokumenter generelt. Denne specifikation kan også anvendes til webservices.

Ved at indlejre krypteringen i XML-dokumentet gøres krypteringen uafhængig af transportlaget, og krypteringen kan derved også opretholdes, når webservicekaldet videresendes til en tredjepart.

XML-krypteringen kan foretages på hele dokumentet eller blot elementvis i XML-dokumentet. Med elementvis kryptering kan forskellige dele af et webservicekald krypteres uafhængigt af hverandre. Dette kan for eksempel anvendes til at kryptere indholdet af enkelte elementer, så disse ikke kan læses af alle parter, der involveres i håndteringen. Et typisk eksempel vil være at kryptere kreditkortoplysninger, så disse kun kan læses af den tredjepart, der måtte stå for betalingen, og ikke den part der håndterer selve ordreafgivelsen.

13.4. Dataintegritet

At opretholde dataintegriteten kan sikres ved at anvende en digital signatur på XML-dokumentet. Er der ændret i dataene efter afsendelsen fra forbrugeren, vil signaturen således ikke længere være gyldig, og transaktionen kan derfor afvises.

Ved anvendelse af kryptering anvendes typisk også digitale signaturer, og anvender man derfor kryptering, vil dataintegriteten også være sikret. De samme praktiske forhold som gør sig gældende i forbindelse med konfidentialitet, se afsnit 13.3, gør sig derfor også gældende i forbindelse med dataintegritet.

13.4.1. Case: Konfidentialitet og dataintegritet

I projektet for Brødrene Hartmann er det ikke nødvendigt at kunne foretage elementvis kryptering, og der indgår ingen tredjeparter i udøvelsen af webservicene. Desuden gives der ikke adgang til andre transportformer end HTTP.

Vi har derfor valgt at basere konfidentialiteten på transportlaget, og webservicen er derfor kun tilgængelig som HTTPS. Valget er truffet af to grunde. Dels fordi kryptering på transportniveauet er tilstrækkeligt i forhold til problemstillingen. Dels fordi det således var muligt at trække på de erfaringer om konfigurerings af SSL i Microsofts Internet Information Services, som vi under alle omstændigheder ville opnå, ved at den webbaserede klient skulle fungere over HTTPS. Derved sikredes også dataintegriteten.

13.5. Uafviselighed

Uafviseligheden af en webservice sikres også ved hjælp af digitale signaturer. Har en deltager i en webservice underskrevet XML-dokumentet med sin digitale signatur, er dette et tegn på, at vedkommende har været involveret, da andre ikke har adgang til den private nøgle.

Af hensyn til en eventuel senere bevisførelse, vil det være en fordel, at benytte digitale signaturer der integreres i XML-dokumentet, som for eksempel ved hjælp af XML-kryptering, i stedet for at basere sig på digitale signaturer på transportprotokollen. Herved kan hele XML-beskeden gemmes for eftertiden, inklusive den digitale signatur.

Anvender man HTTP over SSL er det ikke simpelt at gemme XML-dokumentet inklusive signaturer, da XML-dokumentet typisk først når frem til applikationen, efter at webserversoftwarens har håndteret transporten. Teoretisk er det naturligvis muligt at gemme forløbet af transporten inklusive de digitale signaturer, men i praksis er dette ikke muligt uden at gå udenom de gængse webserverimplementationer på markedet, for eksempel Microsofts Internet Information Services eller Apache webserveren.

Anvendes e-mail som transportform med s/MIME som digital signatur er det straks nemmere, at sikre uafviseligheden, i det man blot skal gemme e-mailen som bevis.

13.5.1. Case: Uafviselighed

Som beskrevet i afsnit 13.4.1 om konfidentialitet i Brødrene Hartmann-projektet har vi baseret os på HTTP over SSL, og vi er derfor ikke i stand til at sikre uafviseligheden i projektet.

Det har været vores vurdering, at problematikken ikke vejer synderligt tungt i den samlede problemstilling. Webservicene løser et planlægningsproblem på „dag til dag“-basis, og eventuelle uoverstemmelser mellem transportørere og Brødrene Hartmann opfanges og løses derfor løbende.

Løsningen udbydes endvidere til en lukket kreds af samarbejdspartnere, ligesom værdien af en enkelt reservering af en timeslot dårligt kan opgøres, men ikke kan antages at være særlig høj.

Uafviselighed har da heller ikke været udtrykt som et krav fra virksomheden.

13.6. Sikkerhed i adgang til databasen

Med hensyn til sikkerhed på databaseniveau, kan brugersikkerhed løses på forskellige måder. Den ene måde er, at man opretter brugerne i selve databasesystemet, så det brugernavn og kodeord man sender med webservicen, bliver brugt til at skabe forbindelse til databasen. En anden mulighed er at man bruger ét brugernavn og kodeord til at skabe forbindelsen, og supplerer dette med en kontrol af at det medsendte brugernavn og kodeord findes i en tabel i databasen.

Ovenstående beskriver hvilke ting man skal overveje, når man udbyder webservices til virksomheder og privatpersoner udadtil, men hvad skal man så overveje, hvis man udbyder webservices internt i virksomheden. Det er ikke nødvendigvis alle i

virksomheden, der har behov for eller lov til at se de data eller bruge den funktionalitet, som de webservices man har, stiller til rådighed. Derfor er det lige så vigtigt at virksomhedens webservices internt, som det er eksternt, er sikret. Her kan man bruge de samme sikkerhedsforanstaltninger, som er beskrevet i de ovenstående afsnit.

13.6.1. Case: Sikkerhed i adgang til databasen

Hos Brødrene Hartmann har vi sikret adgangen til databasen ved hjælp af ét enkelt brugernavn i databasesystemet, suppleret med kontrol af det medsendte brugernavn og kodeord i en brugertabel.

13.7. Programteknisk sikkerhed

Når man udvikler programmer, der kræver at brugeren indtaster oplysninger, er det meget vigtigt, at man sikrer sig imod ondsindet brug af programmet. Eksempler på dette kunne være buffer overflow- eller SQL-injection-angreb.

Buffer overflow betyder, at man sender en meget længere streng end programmet forventer, og på den måde kan man skrive til områder af programmets hukommelse, som man ikke burde have adgang til. Dette kan enten resultere i at programmet går ned, eller at den ondsindede bruger får udført nogle ting, som er meget uhensigtsmæssige.

Et SQL-injection-angreb sker, når en ondsindet bruger i stedet for at indtaste forventede data, indtaster dele af en SQL-kommando, der får den oprindelige SQL-kommando til at udføre handlinger, som det oprindeligt ikke var tiltænkt.

Et eksempel på en SQL-injection kunne være, at den ondsinde person gætter, at brugervalidering sker på baggrund af, om den følgende SQL-sætning returnerer rækker eller ej:

```
SELECT * FROM Users WHERE userid = 'brugernavn' AND password = 'hemlig';
```

Hvis den ondsindede bruger ikke er i besiddelse af et gyldigt brugernavn og kodeord, kunne han forsøge sig med brugernavnet z' OR 'x'='x, samt det tilsvarende kodeord z' OR 'x'='x. Dette vil, hvis der ikke tages særlige forbehold, resultere i en SQL-sætning, der returnerer alle rækker i tabellen, nemlig:

```
SELECT * FROM Users WHERE userid = 'z' OR 'x'='x' AND password = 'z' OR 'x'='x';
```

For at beskytte sig mod ovenstående angreb er det således nødvendigt, når der indkorporeres data man ikke har fuld kontrol over i SQL-sætninger, at kontrollere om disse indeholder en apostrof. Gør de dette må man foretage en passende handling – enten afvise at udføre sætningen eller udskifte tegnet med et uskadeligt.

```
private string SqlSafe(string str)
{
    str = str.Replace("'", "'");
    return str;
}
```

Kodeeksempel 5: Funktion til at erstatte „farlig“ karakter.

Et andet eksempel på SQL-injections kunne være, at man som argument indsætter et semikolon, der afslutter SQL-strengen for tidligt. Herefter er det muligt for eksempel at skrive en SQL-streng der sletter hele databasen – **DROP** database "databasenavn".

Hvis man kender lidt til tabellernes opbygning, kan man med SQL-injections forsøge at ændre oplysninger i databasen. Her er kodeord igen et populært valg. Det er fordi, man ved at ændre kodeord kan få adgang til områder, man ikke har rettighed til.

13.7.1. Case: Programteknisk sikkerhed

I Brødrene Hartmann-projektet viste nogle enkelte tests hurtigt, at vi skulle tage højde for SQL-injections. Vores applikation var sårbar i de tilfælde, som vi har beskrevet i afsnit 13.7. Vi er dog forholdsvis sikret imod de angreb, hvor den ondsindede bruger forsøger at få sendt yderligere oplysninger, end de tilladte, retur. Det er vi, fordi vi opretter objekter med de oplysninger, vi henter fra databasen, og sender objektet retur. Det betyder, at eventuelle ekstra oplysninger ikke vil blive sendt retur til brugeren.

For at sikre os imod de andre angreb, har vi valgt at lave en funktion, der erstatter den karakter, der kan medføre, at vi bliver sårbare. Kodeeksempel 5 viser den funktion, der erstatter ' med " i de strenge, brugeren sender med som argumenter, til webservice-kaldene.

Da vi har valgt at anvende programmeringssproget C#, har vi ikke behovet at bekymre os om buffer overflows, da C#'s string-klasse ikke er sårbar over for dette.

13.8. Anden sikkerhed

På sikkerhedsfronten er der mange overvejelser at gøre, og emnet har til tider karakter af en aldrig sluttende kamp mod „de onde.“

Et generelt råd er, altid at tage højde for den værst tænkelige situation, og altid kontrollere alle faktorer, der indgår. I webservices der publiceres, vil det være nødvendigt at sikre sig, at inddata ser ud som forventet og ikke stole på, at specifikationerne i WSDL-dokumentet er overholdt. For en debat af dette, set specielt i forhold til databaser og brugerinddata, se afsnit 13.7 på forrige side.

En anden angrebstype er et såkaldt „genafspilningsangreb“. Angrebet går ud på at aflytte og gentage tidligere, gyldige operationer. I en webservicesammenhæng kunne det være at gentage et kald af en webservicemetode med de samme parametre og derved for eksempel gentage en én gang afgivet bestilling.

Genafspilningsangreb kan afværges ved at indføre tidsstempling i beskederne, så man ikke gentager ordrer med et allerede set tidsstempel.

Da webservices typisk publiceres på webservere via HTTP, er det nødvendigt at holde disse servere konstant opdateret med de nyeste sikkerhedsrettelser. Historisk set har webservere og -applikationer været målet for mange angreb.

Ud over at sikre at servicen ikke misbruges, vil det også være nødvendigt at sikre sig, at servicen kan opretholdes. De seneste år har vi set en lang række „Denial of Service“-angreb, hvor servere udsættes for så stor en belastning, at servicene reelt ikke er tilgængelige for de retmæssige brugere.

13.8.1. Case: Anden sikkerhed

I projektet vi har udviklet for Brødrene Hartmann har vi ikke indført tidsstempling. Vores valg af kryptering på HTTP-niveau gør at kommunikationen mellem forbruger og udbyder ikke kan aflyttes og derfor heller ikke genafspilles.

Det er ikke muligt at sikre sig totalt mod et Denial of Service-angreb, men vi har konfigureret Microsofts Internet Information Services til kun at kunne håndtere et begrænset antal simultane forespørgsler. I tilfælde af et angreb, skulle disse derved blive afvist hurtigere, end hvis hver forespørgsel skulle gennemføres som et webservicekald.

14. Case: Programmeringsproblematikker

I dette case-afsnit præsenterer vi et par overvejelser, der har været relevante at foretage under programmeringen af systemet til Brødrene Hartmann. Problematikkerne stikker ikke teoretisk dybt, men har været relevante at gøre ud fra et teknisk synspunkt.

Vi har før arbejdet med Windows-grænsefladen, så den har ikke skabt de store problemer i udviklingsprocessen. Web-grænsefladen skabte lidt flere problemer. Det er dels fordi vi ikke har brugt webudvikling så meget tidligere, og dels fordi vi ikke har arbejdet med dynamiske oprettede komponenter før. Vi havde desuden lidt problemer med, at få den ønskede brugerinteraktion, se afsnit 14.1, der handler om post back-problematikken.

14.1. Post back-problematik

I forbindelse med udviklingen af webapplikationsklienten har vi haft nogle overvejelser angående brugergrænsefladen.

Et af klientapplikationens hovedformål er at kunne give transportørerne et altid opdateret overblik over, hvilke timeslots der er ledige, og hvilke der er optaget, herunder hvilke timeslots transportørerne selv har reserveret.

En overvejelse gik på at lade grænsefladen i webapplikationen genindlæse sig selv med faste intervaller, for eksempel hvert 20. sekund. Dette er muligt ved at indføre et *meta-tag* i HTML-koden:

```
<meta http-equiv="refresh" content="20">
```

Praktisk erfaring med dette siger dog, at genindlæsningen kan virke forstyrrende. For det første tager genindlæsningen typisk et par sekunder, i hvilke siden ikke vil være tilgængelig, og for det andet vil genindlæsningen ske, selvom brugeren er i færd med at anvende siden og dele af arbejdet kan derfor gå tabt.

Vi har derfor valgt, at genindlæsninger af skemaer i webapplikationen kun skal ske på brugerens eksplicite ønske. Det har været et mål fra vores side, at ønsket om genindlæsning sker på den mest intuitive måde for brugeren – nemlig at brugeren beder browseren om dette ved at trykke på „genindlæs“-knappen eller genvejen dertil, typisk at taste F5.

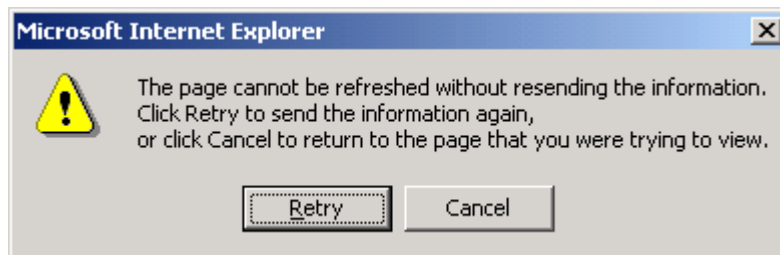
At få dette til at fungere gnidningsfrit har dog krævet et par tekniske finurligheder grundet webapplikationers og ASP.NET's natur, idet de bygger på HTTP-protokollen.

Handlinger i en webapplikation foregår ved at brugerens valg omsættes til HTTP-kommandoen POST. For eksempel klik på knapper eller i vores tilfælde timeslots. Sammen med POST-kommandoen sendes tilstanden af blandt andet radioknapper og lister i brugergrænsefladen til den webserver, der driver webapplikationen. På baggrund af disse tilstande udføres de ønskede handlinger på webserveren, hvorefter der returneres en, eventuelt opdateret, side til brugeren.

ASP.NET holder styr på tilstandene i både brugergrænsefladen, og programmet i øvrigt, på tværs af disse POST-kommandoer, og det er derfor ikke umiddelbart nødvendigt for os som udviklere at håndtere dette specielt.

I forbindelse med ønsket om gnidningsfri genindlæsning af siden opstår der dog det problem, at såfremt brugeren trykker på „genindlæs“-knappen eller F5, forsøger browseren at gentage den HTTP-kommando der frembragte siden. Dette er ofte en POST-kommando baseret på grænsefladetilstande, der var gældende før det aktuelle skærbillede. Typisk vil browsere endvidere advare brugeren om, at han er ved at gensende data, der muligvis kan være forældede, se figur 15 på den følgende side.

For at omgå dette, og derved give brugeren en så gnidningsfri oplevelse som muligt, har vi i forbindelse med håndteringen af alle POST-kommandoer valgt, at efter den



Figur 15: Forsøg på at genindlæse en webside der er resultatet af en HTTP POST-kommando.

ønskede handling er udført, så afslutter vi med at instruere browseren i at gå til en ny adresse, der dog er den samme som man kommer fra. Dette bevirker at browseren som resultat af en POST-kommando, bliver instrueret i at gå til en ny adresse, og dette gør den ved at udføre HTTP-kommandoen GET. GET-kommandoen medsender ikke yderligere data, og den resulterende side har derfor ingen problemer med genindlæsning.

I ASP.NET holdes der derimod ikke styr på tilstande på tværs af GET-kald, da disse potentielt set kan gå til en vilkårlig side på internettet. Vi er derfor nødt til selv at tage hånd om dette.

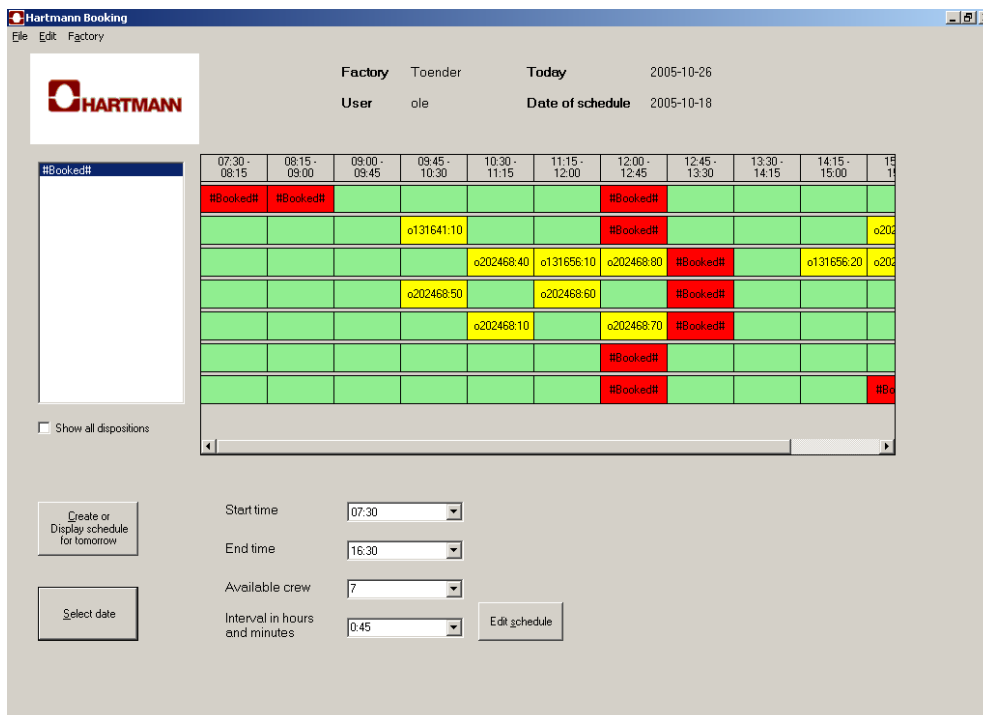
Der findes flere måder at håndtere dette på. Da vores applikation kun har brug for at kende tilstanden af enkelte brugergrænsefladeelementer, på tværs af de fremtvungne GET-kommandoer, har vi valgt at gemme disse tilstande i browser-cookies, før vi fremtvinger GET-kommandoen, og så efterfølgende genindlæse tilstanden fra cookies'ene i forbindelse med sidegenereringen ved GET-kommandoen.

En anden løsningsmodel kunne være at gemme tilstanden på serveren og lade denne identificere af en *session*-værdi. Dette kan være en fordel såfremt tilstanden der skal gemmes er mere kompleks, end den har været i vores tilfælde.

14.2. Brugervenlighed

Der er mange måder, hvorpå man kan sikre brugervenlighed i et it-system. Dette afsnit beskriver nogle af de tiltag, vi har gjort omkring designet af og kommunikationen på brugergrænsefladerne, for at opnå dette. Hvordan vi har sikret brugervenligheden i form af, om systemet lever op til kravene, og om brugerne kan finde ud af at anvende systemet, er beskrevet i afsnit 7 om kravindsamling, og i afsnit 15 om test.

Fælles for Windows-klienten og web-klienten er, at vi i skemaet har brugt farverne rød, gul og grøn til at markere tilstanden af timeslottet. Farverne er valgt ud fra ønsket om, at brugerne skal kunne genkende budskabet, som de allerede kender i forvejen fra lyssignalerne i trafikken.



Figur 16: Windows-klienten til transportbooking-systemet.

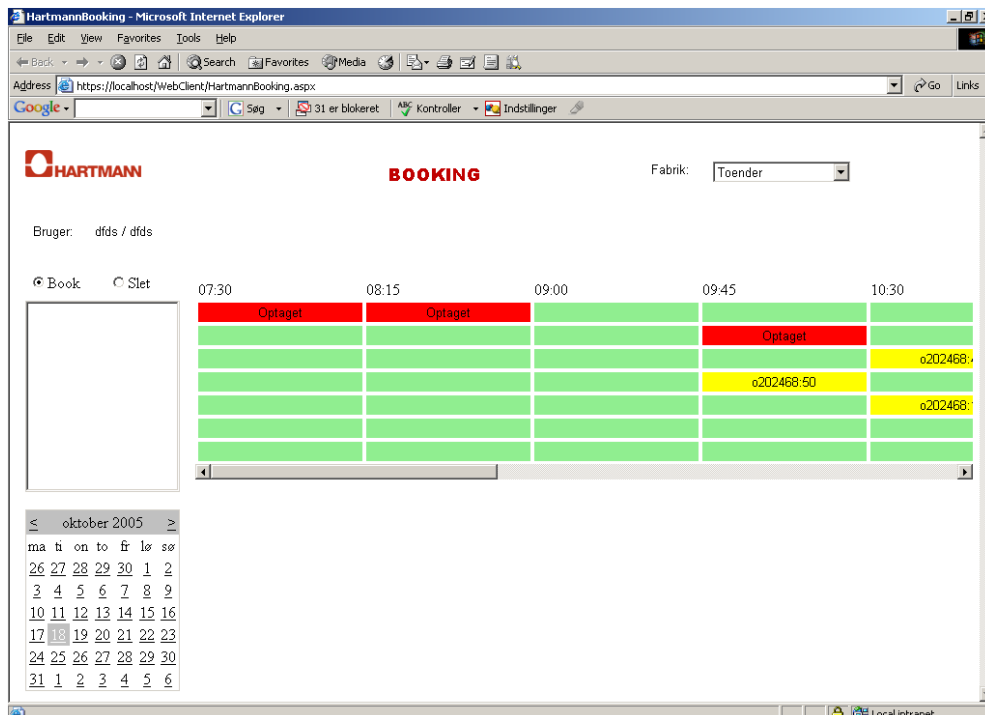
I transportbooking-systemet betyder rød, at timeslottet er reserveret. Grøn betyder at timeslottet er ledigt og derfor fri til reservering. Gul viser de timeslots, der er reserveret af brugeren selv.

Figur 16 og figur 17 på den følgende side viser vores to grænseflader, som de kom til at se ud i den endelige version af programmet.

Vi har forsøgt at lægge os så tæt som muligt op af de designs, som mange efterhånden betragter som standarder, for at gøre det lettere at overskue for brugeren. For eksempel er disponerings-id'erne placeret i venstre side af skærbilledet, således at brugeren opfatter disse som en slags menu de kan vælge fra.

Også i Windows-klienten har vi brugt menuer, som de fleste brugere, der er vant til at arbejde på Windows-platforme, vil kunne genkende. Disse, og i øvrigt også knapperne på denne brugergrænseflade, er understøttet af genvejstaster.

Udover at man kan opdatere skærmen med tryk på F5, som vi har taget med, fordi det er den tast, de fleste systemer bruger som opdateringstast, har vi også anvendt ALT+BOGSTAV, som genvejstast. For eksempel kan man komme til menupunktet „File“



Figur 17: Web-klienten til transportbooking-systemet.

i Windows-klienten ved at trykke ALT+F. Disse genvejstaster giver brugeren mulighed for hurtigt, at udføre handlinger ved brug af tastaturet.

En anden ting, som også kan ses på Windows-klienten, er måden hvorpå vi har grupperet ting, som hænger logisk sammen. Ved denne gruppering får brugeren en visuel opfattelse af, hvilke ting der hører sammen.

Over skemaet ses for eksempel en gruppe af data, som kun er til information. Under skemaet ses gruppen af felter, som skal bruges til at redigere selve opbygningen af skemaet med. Ved at indtastningsfelterne er grupperet på denne måde, betyder det også, at den afstand musen skal rykkes bliver minimal, og er dermed med til at øge hastigheden, hvormed opgaver kan udføres. Bruges tastaturet har vi også sørget for at tabuleringen foregår i en logisk sammenhæng.

En lidt mere skjult faktor, der også er med til at øge brugervenligheden, er forståelig information omkring de handlinger, der udføres. Når brugeren reserverer eller sletter reserveringer af timeslots, vises dette øjeblikkeligt på skemaet, så brugeren kan se, at handlingen er udført.

Opstår der fejl, har vi forsøgt at skrive forståelige beskeder til brugeren. Vi sørger for

at fortælle hvorfor fejlen er opstået, så de har mulighed for, at forstå hvad de skal gøre for at udrede dette. Det er dog ikke alle fejl brugeren selv vil kunne udrede. For eksempel systemfejl vil der skulle en fra Brødrene Hartmanns it-afdelingen til at udrede. Vi har derfor været grundige med at skrive forståelige fejlbeskeder med oplysninger om, hvor i systemet og i hvilken funktion, fejlen opstod, så den hurtigt kan udbedres.

Designet af webklienten har vi, udover de allerede diskutererede emner, ikke forskønnet med hverken baggrundsfarve eller kontraster til at øge brugervenligheden med. Brødrene Hartmann skal selv passe denne ind i deres allerede eksisterende system, hvorfor de også selv vil tilpasse et stylesheet til dette. Webklienten kan ses på figur 17 på forrige side.

14.3. Caching

Webservices i .NET-frameworket giver mulighed for at angive, at et givet webservicekald kan caches i en fastsat periode. Det vil sige, at svaret på webservicekaldet kan i cacheperioden gemmes, og et efterfølgende kald kan besvares direkte, uden at udføre selve metoden.

Ved arbejde med databaser må man vurdere fra metode til metode, om det giver mening at cache resultatet. En metode der opdaterer databasen bør ikke caches, og en metode der læser værdier, der hyppigt opdateres i databasen, bør heller ikke caches. Metoder der aflæser værdier, der derimod sjældent opdateres, kan caches i kortere eller længere perioder.

En fordel ved caching kan være at nedbringe belastningen på selve databasen, samt at svartiden generelt forkortes, idet metoden ikke skal udføres ved hvert kald af webservicen.

I projektet for Brødrene Hartmann har vi vurderet de enkelte webservicemetoder og valgt enten at cache disse i et døgn, eller helt at undlade at cache dem. Det er således kun data, der sjældent varierer, der caches. For eksempel caches hvilke fabrikker der er tilknyttet transportbooking-systemet i et døgn.

15. Case: Test

Da vi skulle planlægge, hvordan vi ville teste systemet, var det vigtigt først at gøre os det klart, hvad det var, vi ville teste. Vi kom frem til, at vi ville lave en samlet bruger- og funktionalitetstest.

Brugertesten skal vise om brugerne forstår logikken i systemet. Brugeren skal kunne servicere sig selv ved at kunne finde oplysninger, eller kunne udføre opgaver uden

at møde forhindringer undervejs. Det er også vigtigt, at brugeren forstår at anvende hele systemet og ikke kun en lille del, for at få det rigtige udbytte ud af det.

Funktionalitetstesten skal vise om de handlinger, brugeren udfører i systemet, giver de forventede resultater. Opstår der uventede situationer i form af systemfejl, er det vigtigt, at den rette fejlbesked vises til brugeren, så brugeren enten selv kan forstå problemet, eller kan henvende sig til it-afdelingen med en forklarende fejlmeddelelse. I Brødrene Hartmanns system kunne eksempler på dette være, hvis der ingen forbindelse er til databasen, eller hvis brugeren ikke kan identificeres af systemet.

Hos Brødrene Hartmann er det en meget lille del af personalet, som kommer til at anvende den administrative del af systemet. Alligevel er det vigtigt, at disse personer får en positiv oplevelse med at anvende systemet. Alternativt kunne det tænkes, at de udviser modstand mod at anvende systemet fremover, se afsnit 17.1 på side 68 om modstand mod forandring.

Den offentlige del af systemet er den, alle transportørerne skal anvende. Det er ligeså vigtigt, at denne del fungerer optimalt, så arbejdsgangene bliver optimeret, og for at Brødrene Hartmann kan bevare sit gode forhold som kunde hos transportørerne, og til deres egne kunder, som skal modtage de producerede varer, på det aftalte leveringstidspunkt.

Vi har lavet en testplan, hvor vi har sørget for, at der bliver stillet opgaver, så brugeren kommer rundt i hele systemet. Disse opgaver har vi udformet med henblik på, at de ikke må være ledende i deres formuleringer, og med henblik på at de passer til de virkelige arbejdsgange og situationer, man kan komme ud for, når man skal registrere og finde data i systemet. Testopgaverne kan ses i bilag C på side 78.

Vi vil til selve testen sørge for at brugerne får en kort introduktion til selve programmet uden dog at informere dem om, hvordan man anvender det og navigerer rundt. Brugerne vil få udleveret én opgave ad gangen på et stykke papir. På denne måde får vi brugerne til kun at fokusere på den udleverede opgave, og kan derfor bedre styre, hvordan brugerne oplever at løse denne opgave som en isoleret handling. Brugerne skal aflevere papiret tilbage når de selv mener, at de har løst opgaven, og får derefter udleveret en ny opgave. Det er her vigtigt at vi ikke afbryder, hvis vi kan se, at den enkelte bruger ikke har løst opgaven korrekt. Er opgaven ikke løst korrekt, må vi ved en senere evaluering finde ud af, om det var opgaven, som var formuleret misvisende, eller om vores program er ulogisk bygget op.

Undervejs skal brugerne tænke højt, når de udfører testen. Vi kan så bedre danne os et billede af, om brugerne synes at noget er ulogisk, eller se hvad der i deres øjne fungerer rigtig godt.

Vi vil afslutte testen med en diskussion med brugerne om, hvordan de oplevede brugervenligheden og funktionaliteten i programmet.

Under selve testen vil vi hver især have forskellige roller. Én kommer til at styre opgaveudvekslingen med brugeren, og registrere om brugeren løser opgaven korrekt, og om det var inden for et acceptabelt tidsinterval. En anden skal koncentrere sig

om at registrere, hvordan brugeren navigerer rundt i systemet, og den sidste skal koncentrere sig om at registrere brugerens kommentarer undervejs.

Vi har selv testet programdelene løbende under udviklingen, og har også testet systemet som en helhed, blandt andet ved hver især et par gange, at gennemføre alle testopgaverne som vi har forberedt. Alligevel kan vi ikke være sikre på at alt fungerer som det skal. Vi har et klart billede af hvordan arbejdsgangene i systemet skal foregå, og vi ved på forhånd hvordan systemet bør reagere på de forskellige handlinger. Dette ved den almindelige bruger ikke, og vil måske derfor opfører sig helt anderledes, end vi forventer.

Ved at sammenfatte observationerne ved testen, mener vi at kunne se, om der er noget i systemet, der skal laves om, inden den endelige aflevering.

15.1. Testforløb

Vi havde aftalt en testdag hos Brødrene Hartmann, hvor de ville stille testpersoner til rådighed for os. Testen foregik i den ene ende af et åbent kontor, og det betød at der til tider var meget uro. Udover larm og småsnak i baggrunden var der også uro omkring testpersonen, da andre medarbejdere ville se, hvad der skete.

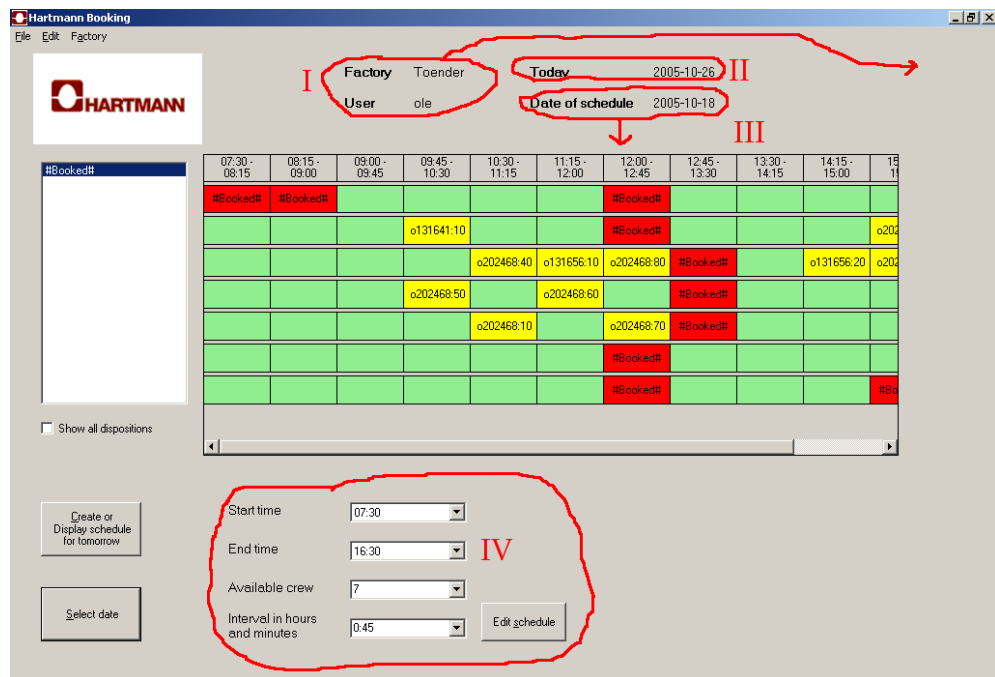
Det er første gang, at vi har haft „rigtige“ testpersoner at arbejde med i et udviklingsprojekt og vores manglende erfaring skinnede lidt igennem i både opgaver og opgaveformuleringer. Vi har selv afprøvet opgaverne inden testdagen, men vi kender jo selv vores system og ved, hvad vi skal gøre for at løse opgaverne.

Vi havde fire testpersoner til denne test og det var kun én af dem, der kommer til at arbejde med systemet til daglig. To af personerne sidder til daglig i Lyngby og kender ikke indgående til problemstillingerne, mens den sidste var vores kontaktperson Ole Nygaard Andersen, som godt kender til problemstillingerne og som har kendskab til systemet.

Der var lidt problemer med opgave 1 i winklienten. Det var en lidt svær opgave at starte med, da den forudsatte, at testpersonen kendte indgående til problemstillingen i vores projekt.

Det at nogle af testpersonerne ikke kendte til problemstillingerne gjorde også, at enkelte af deres kommentarer og forestillinger ikke stemte overens med de krav, der var sat til systemet og den hverdag, det skal fungere i. Det har derfor været nødvendigt at vurdere disse ønsker i dette lys. De havde dog ideer og ønsker til forbedringer af brugergrænsefladerne, som skal med i overvejelserne af, hvad der kan komme med i fremtidige versioner af systemet.

Der var også lidt problemer med nogle formuleringer i opgaverne. For eksempel opgave 3 i winklienten, hvor man skulle reservere timeslots. Her var det en smule forvirrende at der stod „til“ et klokkeslæt, når vores ide med opgaven var, at det skulle være „fra“ det klokkeslæt.



Figur 18: Ændringer i administrationssystemet.

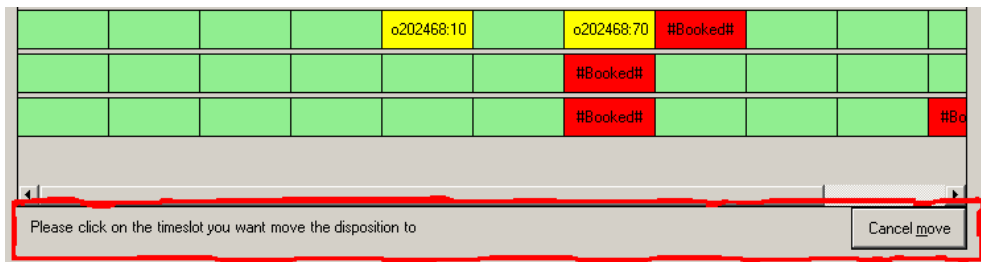
Vi havde også haft den tanke under testplanlægningen, at nogle testpersoner skulle starte med den ene klient, mens andre skulle starte med den anden. Dette var for at teste hvordan de brugte webklienten, når de ikke kendte til funktionaliteten i winklienten, og omvendt. Desværre glemte vi det helt, da vi var godt igang med forløbet. Vi ville gerne have haft det med, men vi mener ikke, at det har nogen stor betydning for testresultatet.

15.2. Testresultater

Testen forløb godt og viste, at der ikke var noget, vi havde misforstået med hensyn til systemet. Vi har opfyldt kravene, og systemet fungerer, som det skal.

Der kom en række gode kommentarer og ønsker fra testpersonerne. Det er alt sammen noget, man med fordel kan inkludere i fremtidige versioner af systemet. Disse nye krav vil Brøderene Hartmann selv implementere.

Vi vælger at dele kommentarene op i to kategorier. Den ene kategori er brugervenligheden, mens den anden kategori er forslagene til ny funktionalitet.



Figur 19: Ændringer ved flytning af disponeringer.

15.2.1. Brugervenlighed

En af de ting som flere nævnte som noget, der kunne ændres for at øge brugervenligheden, var at flytte de oplysninger, der står i toppen af administrationssystemet. Der opstod også lidt forvirring omkring, hvordan man ændrede et skema, herunder hvornår ændringen træder i kraft. Man skal indtaste de nye oplysninger og trykke på knappen „Edit schedule“, men testpersonerne forventede at se ændringerne med det samme.

Figur 18 på modstående side viser de ændringer, man kunne foretage, for at løse nogle af disse misforståelser. Der står nogle romertal (I-IV), der viser fire mulige ændringer der beskrives i det følgende.

Den første ting kunne være at flytte fabrikkens navn og brugernavnet (I) ud til højre, så de ikke tager så meget fokus.

Der var flere der blev forstyrret af dags dato (II), så den kunne man fjerne helt.

Da nogle af testpersonerne ikke opdagede sammenhængen mellem skemaet og datoen for skemaet (III), kunne man med fordel flytte denne dato tættere på skemaet.

Så kunne man flytte „Edit schedule“-funktionaliteten (IV) op i menuen, eller ud i en form for sig, der kom frem ved tryk på en knap.

Der var også nogle problemer, når der skulle ombyttes disponeringer. Figur 19 viser de oplysninger der kommer frem, når man vælger at flytte en disponering.

Disse oplysninger blev overset af flere testpersoner, og man ville derfor med fordel kunne gøre denne tekst mere synlig, så man ville lægge bedre mærke til den.

Derudover kunne man også opdele den knap der hedder „Create or display schedule for tomorrow“, da det var lidt forstyrrende at have de to funktionaliteter samlet. Der blev flere gange udtrykt ønske om at få yderligere oplysninger fra databasen om disponeringerne vist, så det var også en ting, der kunne være med til at øge brugervenligheden.

15.2.2. Ny funktionalitet

Udover ændringerne til brugervenligheden, var der også en række ting, som testpersonerne gerne ville have, som vi mener ville gå ind under kategorien ny funktionalitet. Det er ting, som ville være gode at have med, men som ikke er en del af de krav, der var blevet opstillet.

Der kom et ønske om forskellige versioner af #Booked#-disponeringen, der er Brødrene Hartmanns „default“-disponering, som kan bruges til reservere timeslots til internt brug. Lige nu er det ikke muligt at se om et timeslot er spærret fordi der ikke er nogen på arbejde, eller fordi det for eksempel bruges til at fylde en trailer, der først skal hentes dagen efter. Derfor kunne det være smart at have en #Intet_mandskab#-disponering og en #Bruges_internt#-disponering, så det var til at kende forskel på dem.

Så var der nogle af testpersonerne, der gerne ville have mulighed for at skrive kommentarer til disponeringerne. Den kunne for eksempel være til de interne disponeringer, så man kunne læse, hvad det var, de skal bruges til. For eksempel om en disponering skulle bruges til at læsse en trailer som nævnt ovenfor, eller om det er det tidsrum, hvor nogle af medarbejderne skulle til frokost.

Noget som nogle af testpersonerne også ønskede, var muligheden for at flytte en disponering til en anden dag, for eksempel til dagen efter. Det kan være smart hvis transportørerne har lavet en disponering til én dag, men pludselig står i en situation, hvor de bliver nødt til at flytte den til dagen efter. Dette kræver dog lidt mere arbejde end nogle af de andre ændringer, da disse disponeringer også bliver brugt i andre af Brødrene Hartmanns it-systemer og ændringen derfor har mere vidtgående konsekvenser.

Endelig havde brugerne et ønske om at der også kunne oprettes et skema uden brug af en skabelon.

16. Case: Produktlevering

Brødrene Hartmann havde som udgangspunkt den idé, at transportbooking-systemet, som vi har udviklet til dem, kun skulle være en slags prototype for dem. Det var så meningen at de selv, på et senere tidspunkt, ville implementere det i deres eksisterende system, efter at de havde tilrettet og videreudviklet det.

Det betød derfor, at vi fra projektets start ikke havde en aftale om, at vi skulle sætte systemet i drift i virksomheden, ej heller udforme hverken en brugermanual eller en systemmanual. På baggrund af dette har vi gjort os ekstra umage i forsøget på, at lave en velstruktureret programkode.

Det endelige produkt, som vi skulle levere til Brødrene Hartmann, skulle være den funktionelle HI-FI-prototype. Denne prototype skulle bestå af en Windows-klient til det administrative system og en webklient til det offentlige system, samt en webserviceserver og en databaseserver. Udover dette ville de modtage dokumentation i form af et eksemplar af vores rapport.

Efter at Brødrene Hartmann har set det produkt vi har udviklet til dem, har de ændret mening, og de vil nu gerne have implementeret systemet som det er, for at sætte det i drift med det samme. De vil så justere og tilføje ny funktionalitet løbende, når behovet opstår.

Derfor har vi lavet en aftale med dem, om at vi sætter en dag af, hvor vi tager til Lyngby og sætter systemet i drift sammen med it-afdelingen. Vi skal derudover også demonstrere programkoden og systemdesignet for dem, så de selv uden større problemer, kan arbejde videre derfra.

Der er en del rejseaktivitet hos Brødrene Hartmann, som skal koordineres, men vi håber på, at implementeringen af transportbooking-systemet vil kunne lade sig gøre, umiddelbart kort tid efter vores aflevering af hovedopgaven, så vi når at gøre os denne erfaring inden eksamen.

16.1. Produktet i drift

Vi mener ikke, at der skal så mange ændringer til for at få sat systemet i drift hos Brødrene Hartmann. Databasen skal sættes op på deres egne servere, og der skal ændres nogle oplysninger i den fil, som blandt andet indeholder adressen, brugernavnet og kodeordet til databasen.

Der skal oprettes et certifikat, som skal bruges til SSL-forbindelsen. Adressen på webservicen i de to klienter skal ændres fra at være localhost til at være den internet-adresse, hvor webservicen bliver placeret.

17. Oplæring i anvendelse af det nye system

Overvejelserne i dette afsnit bygger på organisations- og systemudviklingsteori fra tidligere semestre, Hvid Jensen [7].

Når man laver et it-system, hvad enten det er et helt nyt system, eller det er et eksisterende system, som skal omstruktureres eller opgraderes, undgår man ikke, at der vil være en indkøringsperiode, inden brugerne er helt fortrolige med det nye system. Ændringer i et eksisterende system kan have lige så store indkøringsproblemer som et nyt, da det kan være lige så svært at aflægge sig gamle vaner, som til at skulle lære helt nye at kende.

Systemet, vi har udviklet til Brødrene Hartmann, vil have forskellige grupper af brugere, som skal lære det at kende. Set fra brugersiden er der brugerne hos Brødrene Hartmann som skal anvende det administrative system som redskab til at udføre deres arbejdsfunktioner. Der er transportørerne som skal anvende den offentlige brugergrænseflade til at reservere timeslots. Set fra serversiden er der gruppen af programmører som skal vedligeholde systemet. En sidste gruppe er de personer som skal ajourføre forretningsgangene i systemet.

Aftalen med Brødrene Hartmann er, at vi ikke skal oplære deres brugere i at anvende det afleverede system. Alligevel vil det være sådan, at når vi har udført testen, vil de personer som testede systemet, have fået et kendskab til hvordan systemet virker på brugergrænsefladerne. Disse personer vil, i indkøringsperioden efter at systemet er installeret og integreret som et dagligt arbejdsredskab, kunne virke som sparringspartnere eller få roller, som en slags eksperter for andre, der ikke var en del af projektet inden frigivelsen.

Hvis det viser sig, at systemet bliver et pilotprojekt til at indføre serviceorienteret arkitektur hos Brødrene Hartmann, skal både det it-orienterede og det forretningsorienterede personale til at tænke i helt nye baner. Se afsnit 9.2 på side 29 for en beskrivelse af, hvad personalet skal overveje, hvis serviceorienteret arkitektur indføres.

17.1. Modstand mod forandring

Når systemet bliver sat i drift hos Brødrene Hartmann, kan dette udmønte sig i en modstand mod forandringerne. Vi vil i dette afsnit diskutere nogle af årsagerne til dette. Afsnittet bygger primært på Christiansen [4, kap. 3.2].

Allerede før systemet er sat i drift hos Brødrene Hartmann, har virksomheden underrettet os om, at der kan forventes en vis modstand mod, og utilfrelshed med, systemet fra transportørernes side.

Grunden til modstanden, ligger i at nogle transportører ikke følger de retningslinjer, der er beskrevet i deres kontrakter med Brødrene Hartmann.

I dag har de rimeligt frie tøjler til at „bøje“ spillereglerne. For eksempel lægger de flere disponeringer ind under sammen sending og dermed fratager de Brødrene Hartmanns overblik over hvilke varer, der skal afhentes samlet.

Med det nye system vil transportørerne derimod være tvunget til at foretage reelle disponeringer af transporterne, og transportørernes egen planlægning fastlåses dermed helt op til et døgn tidligere end i dag.

Der kan altså forventes modstand fra transportørerne, da deres mulighed for at planlægge deres arbejdsgange forringes med systemet – omend de ikke forringes ud over, hvad de allerede i dag har forpligtet sig til kontraktligt.

Ønsker Brødrene Hartmann at afbøde denne modstand mod forandring, kan de i følge Bennett [2, kap. 19.8] overveje, at indføre systemet i faser, i stedet for at indføre det fra dag til dag. Det kan for eksempel ske ved at sætte systemet i drift i en form, der gør det muligt for transportørerne fortsat at bøjere reglerne, og først efter en tilvænningsperiode at håndhæve reglerne. Derved afkobles sammenhængen mellem indførelsen af systemet og håndhævelsen af de kontraktlige aftaler.

Brødrene Hartmann forventer ingen nævneværdig modstand mod systemet internt i virksomheden. Der vil dog altid være nogle risici man skal være opmærksom på. For medarbejderne betyder systemet for eksempel, at mange af de oplysninger, der nu findes på papir, vil blive indtastet og være tilgængelige i systemet.

Det betyder at medarbejderne skal vænne sig til de nye arbejdsrutiner, og det kan medføre, at nogle medarbejdere vil synes, at de gamle arbejdsrutiner var bedre og nemmere end de nye. Det kunne tænkes at de følte en forringelse af deres ansættelsesforhold, og dermed også deres arbejdsindsats. Det er en meget almindelig følelse blandt medarbejdere, når der indføres nye rutiner. Dette er fordi, mange ofte har fået opbygget sig en lang række arbejdsmetoder, som virker godt for dem, og det nye system betyder, at de både skal lære en masse nye ting, men de skal måske samtidig „aflære“ flere af deres gamle rutiner.

For Brødrene Hartmann ville dette kunne påvirke deres virksomhed i negativ retning. En eventuel intern modstand mod systemet kan forebygges ved at give en god indføring og oplæring i det nye system.

18. Markedet i fremtiden

Webservices bliver på nuværende tidspunkt spået en stor fremtid, og bliver betragtet som måden, vi fremover vil udveksle og dele data, business to business, med hinanden, ved kommunikation over internettet.

Webservices har potentialet til evolutionært at ændre den forretningsmodel, som de fleste virksomheder anvender i dag, og ændre den måde hvorpå de bruger software til at løse forretningsproblemer på. Virksomheder kan i deres egne applikationer benytte sig af data fra andre virksomheder. Dette betyder også, at virksomheder, der anvender SOA og webservices bliver mere dynamiske, og på tværs af verdensdele kan udveksle data og måske, i stedet for kun at konkurrere med hinanden, i stigende grad vil samarbejde.

Man kan forestille sig, at mange af de eksisterende applikationer som findes i dag, vil blive udbudt som webservicemoduler. Hvor man end har sin interesse, kan man så købe sig adgang til data via et webservicemodul, som er tilpasset ens behov, i stedet for at skulle købe en hel softwarepakke, hvor man måske kun har brug for et lille hjørne af applikationen.

Med SOA og webservices i bagagen har traditionelle it-afdelinger også mulighed for ændre sig fra rene supportorganisationer til ægte strategiske forretningsaktiver ved at publicere data, indkapsle disse og udbyde dem som webservices og eventuelt sælge adgangen til disse.

Webservices binder sig ikke til et bestemt operativsystem eller programmeringssprog, hvorfor det er med til mangfoldiggøre udbuddet og udviklingen af for eksempel gratis open source-systemer. Til gengæld kan det tænkes, at der i fremtiden vil være større grad af brugerbetaling, for at kunne anvende data, som måske ikke tidligere var udbudt som en webservice men som en almindelig webapplikation.

I dag er store virksomheder så småt begyndt at udbyde webservices. Blandt andet er Google begyndt at tilbyde muligheden for, at man via webservices kan implementere søgninger i sine applikationer. Dette er dog stadig på teststadiet. Også Amazon er begyndt at udbyde webservices, man kan bruge i sine egne programmer. Igen gives der blandt andet mulighed for at implementere søgefunktioner fra Amazon i ens egne applikationer. Amazon har færdigudviklet flere webservices, og de har indført brugerbetaling til flere af disse services.

Det er disse trends, der kan blive en del af fremtiden, hvis webservices for alvor bryder igennem.

18.1. Case: Brødrene Hartmann i fremtiden

Til et møde hos Brødrene Hartmann havde vi en snak med Ole Nygaard Andersen om, hvordan nutiden hos dem ser ud, og hvordan deres fremtid kunne tænkes at se ud, i forhold til serviceorienteret arkitektur og webservices.

På nuværende tidspunkt benytter de sig ikke af serviceorienteret arkitektur eller webservices, og har heller ikke gjort sig forsøg omkring dette, kun tanker.

I fremtiden kunne de i første omgang godt tænke sig, at forsøge sig lidt med in-house løsninger, og måske på et senere tidspunkt, når tiden er moden til det, gå over til også at anvende det i samarbejde med deres kunder.

Men Brødrene Hartmann vil ikke være med på bølgen som nogen af de første. Det er ikke deres kernekompetencer, da de ikke er en it-virksomhed. De er en produktionsvirksomhed som ikke kan tåle, at deres produktion risikerer at gå ned på grund af eksperimenter med it. Det ville være alt for risikabelt og dyrt for dem. Som Ole formulerede det:

„Leading edge is bleeding edge“

Nogle af deres kunder, har dog ytret ønske om i fremtiden at kunne samkøre systemer via serviceorienteret arkitektur. Ønskerne gik for eksempel på elektronisk fakturering.

Omkring transportbooking-systemet, kunne man forestille sig at transportørerne i fremtiden kunne gives adgang til systemet via offentlige webservices. På den måde kunne de således integrere data fra webservicen i deres egne interne systemer til for eksempel statistik og planlægning af ture.

19. Perspektivering

I dette afsnit vil vi komme ind på, hvordan projektet i sin helhed er forløbet. Vi vil blandt andet komme ind på samarbejdet i gruppen og med Brødrene Hartmann. Vi vil også skrive lidt om, hvad vi kunne have gjort anderledes, og hvor vores løsning er blevet som vi gerne ville have den.

19.1. Gruppen

Samarbejdet i gruppen er det meste af tiden gået godt. Vi har været gode til at snakke tingene igennem, når der er opstået tvivlsspørgsmål i forbindelse med projektet. Der har været tidspunkter undervejs i projektet, hvor der har været lidt småproblemer. Her har vores tidligere samarbejde og professionelle indstilling gjort, at vi har kunnet løse problemerne med det samme, og vi har snakket om problemerne og løst dem inden vi gik hjem, eller ved følgende møde.

19.2. Brødrene Hartmann kontra Niels Brock

Arbejdet med Brødrene Hartmann har fungeret rigtig godt, og det har været en god erfaring at få med. Vi har dog overvejet, om det har været et problem, at vi ikke har siddet ude hos dem og arbejdet, i forhold til at sidde på skolen. Vi mener at der er fordele og ulemper ved begge dele.

Det kunne have været en fordel at sidde hos Brødrene Hartmann i udviklingsprocessen, da det havde givet os mulighed for, hurtigere at få afklaret eventuelle problemer. Vi ville have haft mulighed for at spørge Ole „henover frokostbordet“ og få svar med det samme, i stedet for at skulle sende en email, og så vente et par dage på svaret.

På skolen har vi haft mulighed for at spørge lærerne, hvis der har været programmeringsproblemer. Vi har dog mest haft små problemer med forståelsen af problemstillingerne hos Brødrene Hartmann, og her har lærerne ikke haft de store muligheder, for at kunne hjælpe.

Ved hoveddelen af rapportskrivningen mener vi, at det ikke ville være så godt at sidde i virksomheden. Man kunne lidt frygte, at Ole eller andre hele tiden ville kigge ind og høre, hvordan det gik, og at det ville være et forstyrrende element. Derfor synes vi, at det har været godt at sidde på skolen, samt hjemme, og skrive rapport.

Det har været spændende at arbejde tæt sammen med en virksomhed, der har givet sig tid til at arbejde med os. Vi synes, at vi er blevet behandlet som medspillere på lige fod med andre medarbejdere i virksomheden, og der har været en stor interesse for vores arbejde. Vi ville gerne have opnået erfaringen ved at sidde i virksomheden at arbejde, men desværre var omstændighederne ikke til det.

19.3. Produktet

Vi er overordnet set godt tilfredse med det endelige produkt, men set i bakspejlet, er der ting vi kunne have gjort anderledes. Vi har fået lavet en god opdeling i lag, specielt med hensyn til brugergrænsefladen, domænelaget og webservicelaget. Vi kunne godt have lavet en bedre lagdeling på udbydersiden, da denne er meget tæt koblet til dataaccess-laget.

Hvis vi kigger på SOA-tankegangen, er vores produkt lidt for tæt koblet sammen. Vi mener, at det er fordi, vi har skulle skullet udvikle hele løsningen, både på forbruger- og udbydersiden, og så er det svært ikke at koble tingene sammen. Desuden har det spillet ind, at vi har skullet udvikle en in-house løsning til Brødrene Hartmann.

19.4. Testen

Det er første gang, at vi har arbejdet med test, hvor vi har arbejdet med testpersoner fra en virksomhed. Det har også betydet at vi har fået en masse erfaring med dette emne. Vi har fået erfaring med at skrive testopgaver, og har i den forbindelse lært, at man skal være meget præcis med sine formuleringer. Vi har også lært vigtigheden af et roligt miljø omkring testpersonen.

20. Konklusion

I problemformuleringen stillede vi følgende spørgsmål, som skulle danne grundlag for studiet og dermed gennemførelsen af hovedopgaven:

„Hvordan kan vi udvikle et it-baseret system til Brødrene Hartmann, der kan forestå deres opgave vedrørende reservering af timeslots. Et system der har brugervenlige grænseflader, og gør brug af serviceorienteret arkitektur og webservices som adgang til databaser, og som kan fungere sikkerhedsmæssigt forsvarligt, så Brødrene Hartmann kan sikre sig mod uvedkommen adgang og konsistens i data, og som samtidig kan muliggøre optimering af arbejdsgange, og sikre at beslutninger træffes på et korrekt grundlag?“

Vi har et velfungerende, kørende system, der løser problemstillingen for Brødrene Hartmann omkring reservering af timeslots.

Vi har gennem rapporten dokumenteret de teoretiske områder, omkring de problemstillinger vi har beskæftiget os med i forbindelse med transportbooking-systemet, og vi har derefter i case-afsnittene dokumenteret hvordan, vi praktisk har håndteret og implementeret dette.

Vi har sørget for gennem design, kravindsamling og test at gøre grænsefladerne brugervenlige.

Vi har implementeret serviceorienteret arkitektur via webservices. Gennem disse webservices tilgår vi databasen på en sikkerhedsmæssig forsvarlig måde. Vi sikrer at transaktioner enten bliver gennemført helt eller slet ikke, og vi sikrer, at de rigtige relationer, og dermed konsistens, mellem data opretholdes, ved at have normaliseret databasen til tredje normalform.

Via autentifikation og autorisation som kontrolleres og registreres ved login, sikrer vi mod uvedkommende adgang til såvel system som data.

Brødrene Hartmann er overbevidst om, at systemet vil optimere arbejdsgangene på lageret i Tønder. De vil ikke have så meget spildtid, som de førhen har haft, specielt på grund af transportørerne der nu bliver nødt til, at overholde deres aftaler og dermed også tidsplanen. Dette vil gøre at lagerpersonalet vil kunne træffe de rigtige beslutninger i forhold til hvilke varer der skal stilles frem til afhentning, ud fra de oprettede disponeringer i skemaet.

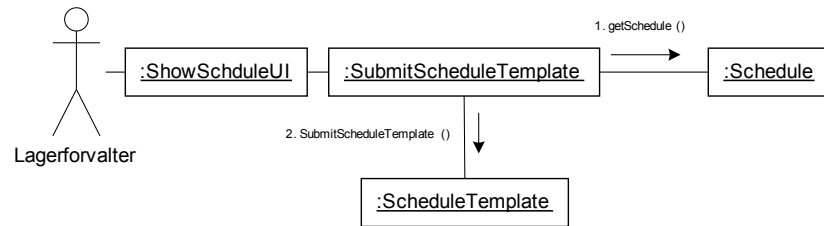
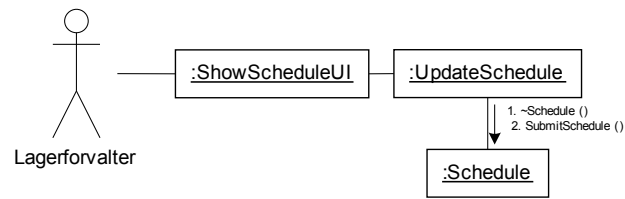
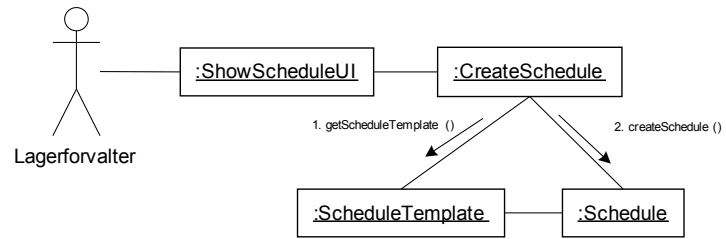
Sammenfattende mener vi, at vi har besvaret spørgsmålet tilfredsstillende og løst den stillede opgave. Vi ser det som et positivt tegn at Brødrene Hartmann har godkendt systemet og vil tage det i anvendelse, som det er.

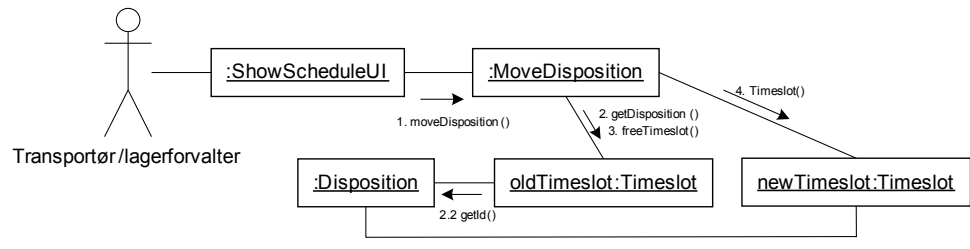
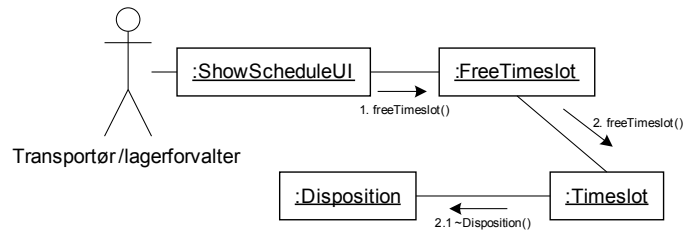
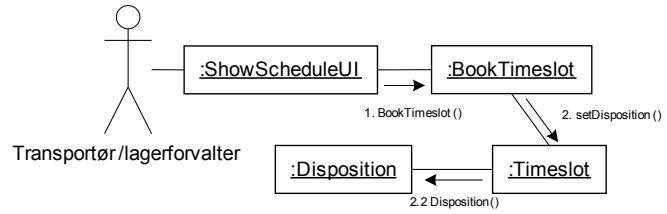
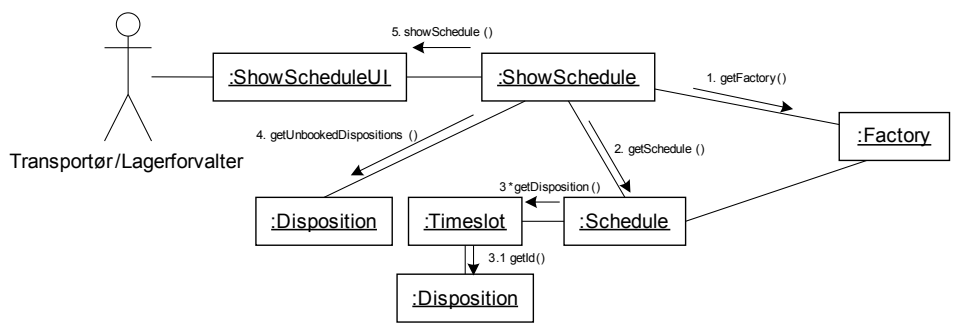
A. Tidsplan

<i>Uge 34</i>	
23	Møde hos Brødrene Hartmann kl. 9.00
26	Møde hos Brødrene Hartmann kl. 13.00
<i>Uge 35</i>	
29 + 30 + 31	Valg, afklaring, research og disponering til hovedopgaven.
1 + 2	Analyse – kollaborationsdiagrammer, klassediagram mm.
<i>Uge 36</i>	
5	Analyse – kollaborationsdiagrammer, klassediagram m.m.
6	MySQL, svn, Visual Studio .NET installation osv.
7	Tønder
8	Normalisering og oprettelse af tabeller i databasen
9	Implementation af win-client, forbindelse til server, flyt, slet, blokér timeslot mm. Impl. af sprog
<i>Uge 37</i>	
12	Implementation af win-client + forbindelse til server + flyt, slet, blokér timeslot mm. Implementering af server. Implementering af web-client + forbindelse til server + vis skema, gem timeslot.
13	Implementering af server. Implementering af web-client + forbindelse til server + vis skema, gem timeslot. Sikkerhed – login i begge klienter + ssl
14	Sikkerhed — login i begge klienter + ssl
15	Status og opsamling
16	Møde hos Brødrene Hartmann kl. 10.00–14.00. Fremlæggelse af foreløbige produkt.
18	<i>Milepæl</i>
<i>Uge 38</i>	
19 – 23	Stikord til alle punkter i hovedopgaven
<i>Uge 39</i>	
26 – 30	Tekstskrivning til alle punkter i hovedopgaven evt. afprøve teorien i produktet.
2	<i>Milepæl</i>
<i>Uge 40</i>	
3 – 6	Tekstskrivning til alle punkter i hovedopgaven evt. afprøve teorien i produktet.
6-7	Sikkerhed og diverse kodning.
<i>Uge 41</i>	
10	Test — testplan
11 – 12	Test — udførelse af test + evaluering af test.
13 - 14	Tekstskrivning.
16	<i>Milepæl</i>
<i>Uge 42</i>	
17 – 21	Tekstskrivning til hovedopgaven.
<i>Uge 43</i>	
24 – 28	Finpudsning af teksten til hovedopgaven.
<i>Uge 44</i>	
31	<i>Milepæl</i>
1 – 3	Korrekturlæsning, smårettelser og trykning af hovedopgave.
4	Aflevering af hovedopgaven senest kl. 12.00

Tabel 1: Tidsplan.

B. Kollaborationsdiagrammer





Figur 20: Projektets kollaborationsdiagrammer

C. Opgaver fra testen

C.1. Opgaver i WinClient (intern klientsoftware)

Opgave 1

Du skal klargøre et skema til i morgen.

Skemaet skal have strukturen:

- tidsinterval: 7:00 - 15:15 4 mand
- tidsinterval: 9:15 - 16:45 4 mand
- Inddelt i intervaller af 3 kvarter
- Reservér to timeslots til Hartmann kl.13.00

Opgave 2

Du skal gemme det skema som du lige har oprettet for i morgen som en skabelon til fremtidig brug.

Opgave 3

Schenker ringer. Deres system er gået ned og du skal derfor reservere deres timeslots for dem i skemaet for i morgen.

- Reservér disponering 131656:10 til kl. 9.15
- Reservér disponering 131656:20 til kl. 13.45
- Reservér disponering 131641:10 til kl. 16.00

Opgave 4

Du skal frigive en af de timeslots (efter eget valg) som Hartmann har reserveret til intern brug i morgendages skema.

Opgave 5

DFDS ringer. De kan ikke afhente o202468:50 klokken 9:45 i dag som lovet.

Flyt disponeringen til en ledig timeslot om eftermiddagen.

Opgave 6

Schenker ringer igen. Deres system er stadig nede. Du skal nu fjerne reserveringen af deres disponering i morgendagens skema kl. 13:45.

Opgave 7

Log ud af systemet.

C.2. Opgaver til WebClient (ekstern klientsoftware)

Opgave 1

Login som en transportør fra DFDS (brugernavn: dfds, kodeord: dfds).

Opgave 2

Reservér de disponeringer for i morgen du har til rådighed.

Opgave 3

Dine indtastninger skal ændres:

En disponering skal flyttes til et vilkårligt ledigt sted.

Opgave 4

Dine indtastninger skal ændres:

To disponeringer skal skifte plads med hinanden.

Opgave 5

En af dine chauffører ringer og spørger hvad tid han skal hente disponering o202468:20 hos Brødrene Hartmann i dag.

D. WSDL-beskrivelse af webservicen

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/" xmlns:soap="http://
schemas.xmlsoap.org/wsdl/soap/" xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:tns="https://
localhost/TransportBooking/default.asmx" xmlns:tm="http://microsoft.com/wsdl/mime/
textMatching/" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace=
"https://localhost/TransportBooking/default.asmx" xmlns:wsdl="http://schemas.
xmlsoap.org/wsdl/">
<wsdl:types>
  <s:schema elementFormDefault="qualified" targetNamespace="https://localhost/
  TransportBooking/default.asmx">
    <s:element name="GetLanguage">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="language_" type="s:string" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="GetLanguageResponse">
      <s:complexType>
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="1" name="GetLanguageResult" type="
            tns:ArrayOfAnyType" />
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:complexType name="ArrayOfAnyType">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="unbounded" name="anyType" nillable="true" />
      </s:sequence>
    </s:complexType>
    <s:element name="AuthHeader" type="tns:AuthHeader" />
    <s:complexType name="AuthHeader">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Username" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Password" type="s:string" />
      </s:sequence>
    </s:complexType>
    <s:element name="GetUser">
      <s:complexType />
    </s:element>
    <s:complexType name="User">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="UserId" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Password" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Factory" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="Language" type="s:string" />
        <s:element minOccurs="0" maxOccurs="1" name="TransportFirm" type="s:string" />
      </s:sequence>
    </s:complexType>
    <s:element name="GetUserResponse">
      <s:complexType>
        <s:sequence>
```

```

    <s:element minOccurs="0" maxOccurs="1" name="GetUserResult" type="tns:User" /
    >
  </s:sequence>
</s:complexType>
</s:element>
<s:element name="GetFactory">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="id_" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="Factory">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Id" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="RealName" type="s:string" />
  </s:sequence>
</s:complexType>
<s:element name="GetFactoryResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetFactoryResult" type="
        tns:Factory" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetSchedule">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="date_" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="AbstractScheduleTemplate" abstract="true">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="StartTime" type="s:dateTime" />
    <s:element minOccurs="1" maxOccurs="1" name="EndTime" type="s:dateTime" />
    <s:element minOccurs="1" maxOccurs="1" name="TimeslotLength" type="s:int" />
    <s:element minOccurs="0" maxOccurs="1" name="Factory" type="tns:Factory" />
    <s:element minOccurs="0" maxOccurs="1" name="TimeslotList" type="
      tns:ArrayOfTimeslot" />
    <s:element minOccurs="1" maxOccurs="1" name="AvailableCrew" type="s:int" />
  </s:sequence>
</s:complexType>
<s:complexType name="ArrayOfTimeslot">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="unbounded" name="Timeslot" nillable="true"
      type="tns:Timeslot" />
  </s:sequence>
</s:complexType>
<s:complexType name="Timeslot">
  <s:sequence>
    <s:element minOccurs="1" maxOccurs="1" name="Row" type="s:int" />
    <s:element minOccurs="1" maxOccurs="1" name="Position" type="s:int" />
  </s:sequence>
</s:complexType>

```

```

    <s:element minOccurs="0" maxOccurs="1" name="Disposition" type="tns:Disposition" />
  </s:sequence>
</s:complexType>
<s:complexType name="Disposition">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Id" type="s:string" />
    <s:element minOccurs="0" maxOccurs="1" name="Factory" type="tns:Factory" />
    <s:element minOccurs="0" maxOccurs="1" name="TransportFirm" type="tns:TransportFirm" />
  </s:sequence>
</s:complexType>
<s:complexType name="TransportFirm">
  <s:sequence>
    <s:element minOccurs="0" maxOccurs="1" name="Name" type="s:string" />
  </s:sequence>
</s:complexType>
<s:complexType name="Schedule">
  <s:complexContent mixed="false">
    <s:extension base="tns:AbstractScheduleTemplate">
      <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="Date" type="s:dateTime" />
      </s:sequence>
    </s:extension>
  </s:complexContent>
</s:complexType>
<s:element name="GetScheduleResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetScheduleResult" type="tns:Schedule" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetScheduleTemplate">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="id_" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="GetScheduleTemplateResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetScheduleTemplateResult" type="tns:ScheduleTemplate" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:complexType name="ScheduleTemplate">
  <s:complexContent mixed="false">
    <s:extension base="tns:AbstractScheduleTemplate">
      <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="Id" type="s:string" />
      </s:sequence>
    </s:extension>
  </s:complexContent>
</s:complexType>

```

```

        <s:element minOccurs="0" maxOccurs="1" name="Description" type="s:string" />
    </s:sequence>
</s:extension>
</s:complexContent>
</s:complexType>
<s:element name="GetTemplateList">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetTemplateListResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetTemplateListResult" type="
                tns:ArrayOfAnyType" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetFactoryList">
    <s:complexType />
</s:element>
<s:element name="GetFactoryListResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetFactoryListResult" type="
                tns:ArrayOfAnyType" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetDispositionList">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="date_" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetDispositionListResponse">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetDispositionListResult" type="
                tns:ArrayOfAnyType" />
        </s:sequence>
    </s:complexType>
</s:element>
<s:element name="GetUnbookedDispositions">
    <s:complexType>
        <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
            <s:element minOccurs="0" maxOccurs="1" name="date_" type="s:string" />
        </s:sequence>
    </s:complexType>
</s:element>

```

```

<s:element name="GetUnbookedDispositionsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="GetUnbookedDispositionsResult"
        type="tns:ArrayOfAnyType" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SubmitSchedule">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="schedule_" type="tns:Schedule" /
        >
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SubmitScheduleResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="SubmitScheduleResult" type="
        s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SubmitScheduleTemplate">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="schedulescheduletemplate_" type="
        tns:ScheduleTemplate" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SubmitScheduleTemplateResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="SubmitScheduleTemplateResult"
        type="s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="UpdateSchedule">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="schedule_" type="tns:Schedule" /
        >
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="UpdateScheduleResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="UpdateScheduleResult" type="
        s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>

```

```

<s:element name="FreeTimeslot">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="row_" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="pos_" type="s:int" />
      <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="date_" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="FreeTimeslotResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="FreeTimeslotResult" type="
        s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="BookTimeslot">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="row_" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="pos_" type="s:int" />
      <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="date_" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="dispId_" type="s:string" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="BookTimeslotResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="BookTimeslotResult" type="
        s:boolean" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SwapBookings">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="0" maxOccurs="1" name="date_" type="s:string" />
      <s:element minOccurs="0" maxOccurs="1" name="factoryId_" type="s:string" />
      <s:element minOccurs="1" maxOccurs="1" name="row1_" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="pos1_" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="row2_" type="s:int" />
      <s:element minOccurs="1" maxOccurs="1" name="pos2_" type="s:int" />
    </s:sequence>
  </s:complexType>
</s:element>
<s:element name="SwapBookingsResponse">
  <s:complexType>
    <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="SwapBookingsResult" type="
        s:boolean" />
    </s:sequence>
  </s:complexType>

```

```

    </s:element>
  </s:schema>
</wsdl:types>
<wsdl:message name="GetLanguageSoapIn">
  <wsdl:part name="parameters" element="tns:GetLanguage" />
</wsdl:message>
<wsdl:message name="GetLanguageSoapOut">
  <wsdl:part name="parameters" element="tns:GetLanguageResponse" />
</wsdl:message>
<wsdl:message name="GetLanguageAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="GetUserSoapIn">
  <wsdl:part name="parameters" element="tns:GetUser" />
</wsdl:message>
<wsdl:message name="GetUserSoapOut">
  <wsdl:part name="parameters" element="tns:GetUserResponse" />
</wsdl:message>
<wsdl:message name="GetUserAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="GetFactorySoapIn">
  <wsdl:part name="parameters" element="tns:GetFactory" />
</wsdl:message>
<wsdl:message name="GetFactorySoapOut">
  <wsdl:part name="parameters" element="tns:GetFactoryResponse" />
</wsdl:message>
<wsdl:message name="GetFactoryAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="GetScheduleSoapIn">
  <wsdl:part name="parameters" element="tns:GetSchedule" />
</wsdl:message>
<wsdl:message name="GetScheduleSoapOut">
  <wsdl:part name="parameters" element="tns:GetScheduleResponse" />
</wsdl:message>
<wsdl:message name="GetScheduleAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="GetScheduleTemplateSoapIn">
  <wsdl:part name="parameters" element="tns:GetScheduleTemplate" />
</wsdl:message>
<wsdl:message name="GetScheduleTemplateSoapOut">
  <wsdl:part name="parameters" element="tns:GetScheduleTemplateResponse" />
</wsdl:message>
<wsdl:message name="GetScheduleTemplateAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="GetTemplateListSoapIn">
  <wsdl:part name="parameters" element="tns:GetTemplateList" />
</wsdl:message>
<wsdl:message name="GetTemplateListSoapOut">
  <wsdl:part name="parameters" element="tns:GetTemplateListResponse" />
</wsdl:message>
<wsdl:message name="GetTemplateListAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>

```

```

</wsdl:message>
<wsdl:message name="GetFactoryListSoapIn">
  <wsdl:part name="parameters" element="tns:GetFactoryList" />
</wsdl:message>
<wsdl:message name="GetFactoryListSoapOut">
  <wsdl:part name="parameters" element="tns:GetFactoryListResponse" />
</wsdl:message>
<wsdl:message name="GetFactoryListAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="GetDispositionListSoapIn">
  <wsdl:part name="parameters" element="tns:GetDispositionList" />
</wsdl:message>
<wsdl:message name="GetDispositionListSoapOut">
  <wsdl:part name="parameters" element="tns:GetDispositionListResponse" />
</wsdl:message>
<wsdl:message name="GetDispositionListAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="GetUnbookedDispositionsSoapIn">
  <wsdl:part name="parameters" element="tns:GetUnbookedDispositions" />
</wsdl:message>
<wsdl:message name="GetUnbookedDispositionsSoapOut">
  <wsdl:part name="parameters" element="tns:GetUnbookedDispositionsResponse" />
</wsdl:message>
<wsdl:message name="GetUnbookedDispositionsAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="SubmitScheduleSoapIn">
  <wsdl:part name="parameters" element="tns:SubmitSchedule" />
</wsdl:message>
<wsdl:message name="SubmitScheduleSoapOut">
  <wsdl:part name="parameters" element="tns:SubmitScheduleResponse" />
</wsdl:message>
<wsdl:message name="SubmitScheduleAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="SubmitScheduleTemplateSoapIn">
  <wsdl:part name="parameters" element="tns:SubmitScheduleTemplate" />
</wsdl:message>
<wsdl:message name="SubmitScheduleTemplateSoapOut">
  <wsdl:part name="parameters" element="tns:SubmitScheduleTemplateResponse" />
</wsdl:message>
<wsdl:message name="SubmitScheduleTemplateAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="UpdateScheduleSoapIn">
  <wsdl:part name="parameters" element="tns:UpdateSchedule" />
</wsdl:message>
<wsdl:message name="UpdateScheduleSoapOut">
  <wsdl:part name="parameters" element="tns:UpdateScheduleResponse" />
</wsdl:message>
<wsdl:message name="UpdateScheduleAuthHeader">
  <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
</wsdl:message>
<wsdl:message name="FreeTimeslotSoapIn">

```

```

    <wsdl:part name="parameters" element="tns:FreeTimeslot" />
  </wsdl:message>
  <wsdl:message name="FreeTimeslotSoapOut">
    <wsdl:part name="parameters" element="tns:FreeTimeslotResponse" />
  </wsdl:message>
  <wsdl:message name="FreeTimeslotAuthHeader">
    <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
  </wsdl:message>
  <wsdl:message name="BookTimeslotSoapIn">
    <wsdl:part name="parameters" element="tns:BookTimeslot" />
  </wsdl:message>
  <wsdl:message name="BookTimeslotSoapOut">
    <wsdl:part name="parameters" element="tns:BookTimeslotResponse" />
  </wsdl:message>
  <wsdl:message name="BookTimeslotAuthHeader">
    <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
  </wsdl:message>
  <wsdl:message name="SwapBookingsSoapIn">
    <wsdl:part name="parameters" element="tns:SwapBookings" />
  </wsdl:message>
  <wsdl:message name="SwapBookingsSoapOut">
    <wsdl:part name="parameters" element="tns:SwapBookingsResponse" />
  </wsdl:message>
  <wsdl:message name="SwapBookingsAuthHeader">
    <wsdl:part name="AuthHeader" element="tns:AuthHeader" />
  </wsdl:message>
  <wsdl:portType name="TransportBookingSoap">
    <wsdl:operation name="GetLanguage">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Provides a list of all
        translated items. Identified by language_ (ex. Dansih, English...)</documentation>
      <wsdl:input message="tns:GetLanguageSoapIn" />
      <wsdl:output message="tns:GetLanguageSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetUser">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">This method provides
        information about the user.</documentation>
      <wsdl:input message="tns:GetUserSoapIn" />
      <wsdl:output message="tns:GetUserSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetFactory">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">This method provides
        information about the factory identified by id_</documentation>
      <wsdl:input message="tns:GetFactorySoapIn" />
      <wsdl:output message="tns:GetFactorySoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetSchedule">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">This method returns a
        transport booking schedule for a given factory and a given date.</documentation>
      <wsdl:input message="tns:GetScheduleSoapIn" />
      <wsdl:output message="tns:GetScheduleSoapOut" />
    </wsdl:operation>
    <wsdl:operation name="GetScheduleTemplate">
      <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">This method returns a
        template for a transport booking schedule for a given factory. The template is identified by
        id_ and factoryId_</documentation>
      <wsdl:input message="tns:GetScheduleTemplateSoapIn" />
    </wsdl:operation>
  </wsdl:portType>

```

```

    <wsdl:output message="tns:GetScheduleTemplateSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetTemplateList">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Provides a list of transport
      booking schedule templates for a given factory.</documentation>
    <wsdl:input message="tns:GetTemplateListSoapIn" />
    <wsdl:output message="tns:GetTemplateListSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetFactoryList">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Provides a list of transport
      booking schedule templates for a given factory.</documentation>
    <wsdl:input message="tns:GetFactoryListSoapIn" />
    <wsdl:output message="tns:GetFactoryListSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetDispositionList">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Provides a list of all
      dispositions.</documentation>
    <wsdl:input message="tns:GetDispositionListSoapIn" />
    <wsdl:output message="tns:GetDispositionListSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="GetUnbookedDispositions">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Provides a list of all
      unbooked dispositions for a given factory and date.</documentation>
    <wsdl:input message="tns:GetUnbookedDispositionsSoapIn" />
    <wsdl:output message="tns:GetUnbookedDispositionsSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="SubmitSchedule">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Submit a new transport
      booking schedule to transport booking system.</documentation>
    <wsdl:input message="tns:SubmitScheduleSoapIn" />
    <wsdl:output message="tns:SubmitScheduleSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="SubmitScheduleTemplate">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Submit a new transport
      booking schedule to transport booking system.</documentation>
    <wsdl:input message="tns:SubmitScheduleTemplateSoapIn" />
    <wsdl:output message="tns:SubmitScheduleTemplateSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="UpdateSchedule">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Update a transport booking
      schedule to the transport booking system.</documentation>
    <wsdl:input message="tns:UpdateScheduleSoapIn" />
    <wsdl:output message="tns:UpdateScheduleSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="FreeTimeslot">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Remove the booking from a
      timeslot of a given factory, date, 'row' and 'position'.</documentation>
    <wsdl:input message="tns:FreeTimeslotSoapIn" />
    <wsdl:output message="tns:FreeTimeslotSoapOut" />
  </wsdl:operation>
  <wsdl:operation name="BookTimeslot">
    <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Book a timeslot at a given
      factory, date, 'row' and 'position' with a disposition with dispId.</documentation>
    <wsdl:input message="tns:BookTimeslotSoapIn" />
    <wsdl:output message="tns:BookTimeslotSoapOut" />
  </wsdl:operation>

```

```

<wsdl:operation name="SwapBookings">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">Swap bookings between two
    timeslots at a given factory at a given date.</documentation>
  <wsdl:input message="tns:SwapBookingsSoapIn" />
  <wsdl:output message="tns:SwapBookingsSoapOut" />
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="TransportBookingSoap" type="tns:TransportBookingSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="GetLanguage">
    <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
      GetLanguage" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
      <soap:header message="tns:GetLanguageAuthHeader" part="AuthHeader" use="literal" /
        >
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetUser">
    <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/GetUser
      " style="document" />
    <wsdl:input>
      <soap:body use="literal" />
      <soap:header message="tns:GetUserAuthHeader" part="AuthHeader" use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetFactory">
    <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
      GetFactory" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
      <soap:header message="tns:GetFactoryAuthHeader" part="AuthHeader" use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetSchedule">
    <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
      GetSchedule" style="document" />
    <wsdl:input>
      <soap:body use="literal" />
      <soap:header message="tns:GetScheduleAuthHeader" part="AuthHeader" use="literal" /
        >
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>

```

```

<wsdl:operation name="GetScheduleTemplate">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    GetScheduleTemplate" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:GetScheduleTemplateAuthHeader" part="AuthHeader" use="
      literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetTemplateList">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    GetTemplateList" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:GetTemplateListAuthHeader" part="AuthHeader" use="
      literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetFactoryList">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    GetFactoryList" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:GetFactoryListAuthHeader" part="AuthHeader" use="
      literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetDispositionList">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    GetDispositionList" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:GetDispositionListAuthHeader" part="AuthHeader" use="
      literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="GetUnbookedDispositions">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    GetUnbookedDispositions" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:GetUnbookedDispositionsAuthHeader" part="AuthHeader"
      use="literal" />
  </wsdl:input>

```

```

<wsdl:output>
  <soap:body use="literal" />
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="SubmitSchedule">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    SubmitSchedule" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:SubmitScheduleAuthHeader" part="AuthHeader" use="
      literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="SubmitScheduleTemplate">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    SubmitScheduleTemplate" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:SubmitScheduleTemplateAuthHeader" part="AuthHeader" use
      ="literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="UpdateSchedule">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    UpdateSchedule" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:UpdateScheduleAuthHeader" part="AuthHeader" use="
      literal" />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="FreeTimeslot">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    FreeTimeslot" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:FreeTimeslotAuthHeader" part="AuthHeader" use="literal"
      />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="BookTimeslot">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    BookTimeslot" style="document" />
  <wsdl:input>

```

```

    <soap:body use="literal" />
    <soap:header message="tns:BookTimeslotAuthHeader" part="AuthHeader" use="literal"
      />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="SwapBookings">
  <soap:operation soapAction="https://localhost/TransportBooking/default.asmx/
    SwapBookings" style="document" />
  <wsdl:input>
    <soap:body use="literal" />
    <soap:header message="tns:SwapBookingsAuthHeader" part="AuthHeader" use="literal"
      />
  </wsdl:input>
  <wsdl:output>
    <soap:body use="literal" />
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="TransportBooking">
  <documentation xmlns="http://schemas.xmlsoap.org/wsdl/">This is the BrÃ_drene Hartmann
    A/S TransportBooking webservice.</documentation>
  <wsdl:port name="TransportBookingSoap" binding="tns:TransportBookingSoap">
    <soap:address location="https://localhost/TransportBooking/default.asmx" />
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```


Litteratur

- [1] Dietrich Ayala et al. *Professional Open Source Web Services*. Wrox Press Ltd., 2002. ISBN 1-861007-46-9.
- [2] Simon Bennett, Steve McRobb og Ray Farmer. *Object-Oriented Systems Analysis And Design Using UML*. McGraw-Hill, anden udgave, 2002. ISBN 0-07-709864-1.
- [3] Jason Bloomberg. *The portability pitfall*, 2004. URL http://searchwebservicestechtarget.com/originalContent/0,289142,sid26_gci960871,00.html.
- [4] Poul Erik Christiansen, Henrik Kjær, Hans Jørgen Skrivers og Erik Staunstrup. *Organisation*. Trojka, tredje udgave, 2003. ISBN 87-90701-49-6.
- [5] Ramez Elmasri og Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, fjerde udgave, 2003. ISBN 0-321-20448-4.
- [6] Alex Ferrara og Matthew MacDonald. *Programming .NET Web Services*. O'Reilly, første udgave, 2002. ISBN 0-596-00250-5.
- [7] Henrik Hvid Jensen. *Service Orienteret Arkitektur – integration som konkurrenceparameter*. Litera, første udgave, 2004. ISBN 87-91242-34-7.
- [8] Dan Mygind. SOA ned på jorden. *Computerworld*, nr. 35: s. 26, 30. september 2005.
- [9] Niels Brock. *Undervisnings- og eksamensplan for datamatikeruddannelsen ved Niels Brock*, 1999.
- [10] Daniel Pollard. *Authentication for Web Services (using SOAP headers)*, 2003. URL <http://www.codeproject.com/cs/webservices/authforwebservices.asp>.
- [11] David Sprott og Lawrence Wilkes. *Understanding Service-Oriented Architecture*, 2004. URL <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmaj/html/aj1soa.asp>.
- [12] World Wide Web Consortium. *XML Encryption*, 2001. URL <http://www.w3.org/Encryption/2001/>.