Rapportering i et service orienteret arkitektur

Skrevet af

Yusuf Karabulut

Kongens Lyngby 2005

IMM-B.ENG.-2005-13

Forord

Denne hovedopgave udgør afslutningen af min Diplom IT uddannelse på Danmark Tekniske Universitet. Projektet er udført i samarbejde med BRFkredit over en tidsramme på 10 uger.

Min hovedopgave har dels et teoretisk, dels et mere praksisorienteret sigte. Den teoretiske del af opgaven beskæftiger sig med analyse og design af en applikation som kan generere dynamiske rapporter. I den forbindelse vil jeg undersøge en række rapporterings værktøjer og jeg vil give et indblik i den service orienteret arkitektur. I den praksisorienteret del af opgaven, vil jeg implementere og teste en dynamisk rapport.

Lyngby, Oktober 2005
Yusuf Karabulut

Indholdsfortegnelse

1 PROJEKTDEFINITION 5

1.1 INDLEDNING 5
1.2 PROBLEMFORMULERING 5
1.3 AFGRÆNSNING AF PROJEKTET 6

2 FREMGANGSMETODE 7

2.1 METODE2 7
2.1.1 FASER 8
2.2 TIDSPLAN 10

3 ANALYSE 12

3.1 KRAVSPECIFIKATION: 12
3.2 USE-CASES 13
3.2.1 VIS RAPPORT 13
3.2.2 GEM RAPPORT 13
3.3 ANALYSE AF RAPPORTERINGS VÆRKTØJER 14
3.3.1 CRYSTAL REPORTS 15
3.3.2 ASP.NET 17
3.3.3 KONKLUSION 19

4.0 DESIGN 20

4.1 CRYSTAL REPORTS 20
4.1.1 RAPPORT DOKUMENT 20
4.1.3 CRYSTAL REPORTS PROCESSERINGS MODEL. 22
4.1.4 BINDINGS MULIGHEDER 24
4.1.5 SIKKERHED 26
4.1.6 CRYSTAL REPORTS LØSNINGSMODEL 27
4.2 SOA 28
4.2.1 BEGREBSFORKLARING 28
4.2.2 HVAD ER SOA 29
4.2.3 BRFKREDIT´S SOA ARKITEKTUR 34
4.2.4 DISKUSSION 43

5 IMPLEMENTATION 45

5.1 OVERORDNET SYSTEM FLOW 45
5.2 FUNKTIONS BESKRIVELSE 46
5.3 IMPLEMENTERING AF HENT METODE 47
5.3.1 IMPLEMENTERINGS EKSEMPEL 47

5.3.2 SEKVENS DIAGRAMMER 48
5.6 ANVENDTE PATTERNS 50
5.6.1 SINGLETON 50
5.6.3 FACADE / REMOTE FACADE 51
5.6.4 PROXY PATTERN 51
5.7 IMPLEMENTERINGS PROBLEMER 52

6 TEST 53

6.1 UNIT TEST 53
6.2 USE CASE TEST 54
6.2.1 GEM RAPPORT 54
6.2.2 VIS RAPPORT 56

7 KONKLUSION 58

7.1 PERSPEKTIVERING 58

8 BILAG 59

8.1 RAPPORT OPLÆG FEJL! BOGMÆRKE ER IKKE DEFINERET.
8.1.1 KREDITINDSTILLING **FEJL! BOGMÆRKE ER IKKE DEFINERET.**
8.2 GENERERET RAPPORT FEJL! BOGMÆRKE ER IKKE DEFINERET.
8.4 KILDEKODE 59
8.4.1 BRF.INDSTILLING.FRONTEND.WEBFORM PROCESSAGENT.CS 59
8.4.2 BRF.ENTITY.INDSTILLING INDSTILLING.CS 71
8.4.3 BRF.INDSTILLING.BACKEND.DATAACCESS INDSTILLING.CS 79
8.4.4 BRF.INDSTILLING.BUSINESS INDSTILLING.CS 81

1 Projektdefinition

1.1 Indledning

Når en kunde ønsker at optage et lån skal BRFkredit tage stilling til om de vil godkende lånet. Til dette formål skal man indsamle informationer blandt andet om kunden, om de ejendomme han/hun ejer eller ønsker at købe, kundens økonomiske status etc. Alle disse informationer skal præsenteres i en række overskuelige og strukturerede rapporter. Det er min opgave at undersøge hvordan man kan løse denne opgave, og til dels at implementere den valgte løsning. Til dette formål vil jeg analysere en række rapporterings værktøjer og på baggrund af denne analyse vil jeg udvælge og implementere de ønskede rapporter. Uanset hvilket værktøj eller løsningsmodel jeg vælger at bruge skal min løsning indgå i et større arkitektur. Denne arkitektur hedder SOA og står for Service Orienteret Arkitektur. Jeg vil undersøge denne arkitektur nærmere og præsentere den for læseren. Derefter vil jeg inddrage denne teori til at implementere en række metoder som er nødvendige for rapporterne. Forståelsen af SOA arkitekturen er en forudsætning for at man kan forstå analyse og implementeringen af rapporterne.

1.2 Problemformulering

Den overordnede problemstilling med projektet er at undersøge hvordan man kan generere dynamiske rapporter i en Service orienteret arkitektur. I den forbindelse er det nødvendigt at undersøge principperne bag SOA arkitekturen, og implementere en løsning som ikke overtræder disse principper. Forståelsen af den service orienteret arkitektur er en forudsætning for at forstå analyse og implementeringen. I min analyse af rapporterings værktøjer vil jeg sætte ekstra fokus på Crystal Reports og ASP.NET. Jeg vil opstille en række krav og kriterier som jeg vil bruge til at sammenligne værktøjerne. Grunden til at jeg netop har valgt disse to værktøjer, er at man i BRFkredit har erfaring med ASP.net som rapporterings værktøj, men på grund af høje udviklingstider ønsker at afprøve et nyt værktøj. Dette værktøj er Crystal Reports og jeg vil undersøge om den er egnet til at genere større rapporter i en service orienteret arkitektur. I den sammenhæng vil jeg vil undersøge mulighederne i Crystal Reports, og præsentere de løsningsmodeller som værktøjet stiller til rådighed. Endeligt vil jeg vælge en passende løsning og bruge den til at generere rapporterne.Jeg vil forsøge at besvare følgende spørgsmål:

Hvilke alternativer findes der til Crystal reports?

Hvordan kan man bruge Crystal Reports, i en SOA arkitektur ?

Hvilke problemstillinger kan man møde når man anvender Crystal Reports?

Hvad er grundprincipperne i SOA?

Hvilke problemstillinger kan man møde i et SOA arkitektur?

Hvad er forskellen mellem SOA og OO?

1.3 Afgrænsning af projektet

Da tidsrammen på projektet er 10 uger, vil det ikke være muligt at implementere alle rapporterne. Jeg vil tage udgangspunkt i en af rapporterne og implementere denne som et skelet som jeg løbende vil udvide. Det skal være muligt at vise og gemme rapporten.

2 Fremgangsmetode

Dette afsnit beskriver den fremgangsmåde jeg vil benytte for at besvare min problemformulering. Jeg vil præsentere den valgte arbejdsmetode og valg af begrebsdefinitioner og endelig en tidsplan for projektet.

2.1 Metode2

I større projekter som denne er det en god ide at bruge på forhånd definerede og afprøvede processer. Den process eller udviklingsmetode jeg har valgt at anvende hedder Metode2.
Metode 2 er navnet på en udviklingsmetode som bruges i BRFkredit. Typisk anvendes Metode2 til udvikling af IT-Systemer men jeg har også brugt den til at fremstille rapporten/dokumentationen af projektet. Metoden er kraftigt inspireret af Unified Process.
Unified Process er en metode til udvikling, implementering og vedligehold af IT-systemer. De væsentligste principper i Unified Process er:

- Iterativ og inkrementel systemudvikling
- Høj involvering af Forretningen gennem hele projektforløbet
- Test gennem hele projektforløbet
- Objektorienteret Analyse og Design
- UML som notationsform

I Metode2 arbejder man i faser. I hver fase udfører man en række discipliner og disse resulterer i en række artefakter. En artefakt kan defineres som et produkt. En artefakt kan således være en Use-Case og Aktør liste, en Projektbeskrivelse eller en stump kode.
En disciplin består af en række processer, aktiviteter og artefakter. I dette projekt har jeg beskæftiget mig med følgende discipliner:

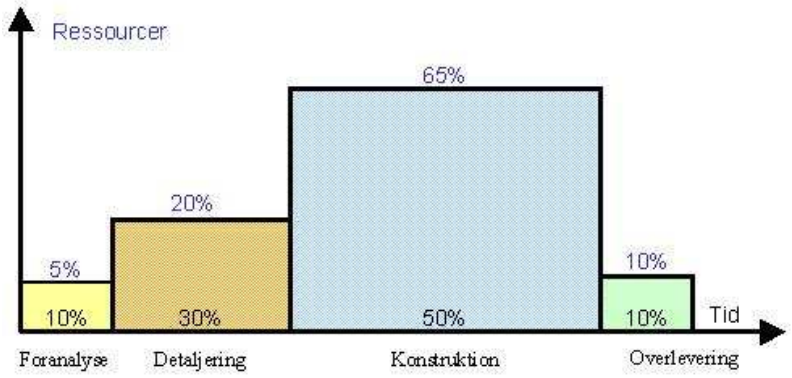
- Kravstyring
- Analyse og Design
- Implementering
- Dokumentering

- Test
- Projektstyring

2.1.1 Faser

En fase er en delsekvens i et projektforløb. Hver fase indeholder mange aktiviteter fra de forskellige discipliner, fasen genererer artefakter, og fasen afsluttes med en milepæl og et review. En fase består af en eller flere iterationer, i dette projekt har jeg besluttet at en iteration er på 10 mandedage eller 2 uger.

Der er forskel på tids- og ressourceforbrug i de enkelte faser. Et typisk mellemstort projekt vil have ressourcer og tid fordelt som vist her:



Figur 1 Typisk ressourceforbrug for et mellemstort projekt.

Da jeg skal skrive en del dokumentation er min Foranalyse og Detaljerings fase lidt længere end dem som er illustreret på figuren. Jeg har udeladt Overleverings fasen da den ikke har relevans for mit projekt. Jeg har beskæftiget mig med følgende Metode 2 faser:

Foranalyse:

Formålet med Foranalysen er at formulere og fastlægge systemets scope (mål og rammer) samt etablere en aftale (Projektbeskrivelse) med kravstilleren om hvad der skal udvikles. Da min opgave udgør en mindre del af en større opgave kan jeg nøjes med at aftale med projektlederen om en eventuel projektbeskrivelse. Udover de nævnte artefakter der skal laves en projektdefinitionen, som

indeholder indledning, problemformulering og afgrænsning. Denne fase skal være færdig efter første iteration.

Hovedaktiviteter i Foranalysen:

- Afdække og formulere projektets scope og systemets grænser.
- Overordnet at beskrive de fremtidige forretningsprocesser og systemets Use Cases.
- At udarbejde en overordnet projektplan for hele projektet samt en detaljeret plan for den første iteration i Detaljeringsfasen
- At udarbejde en Projektdefinition, herunder:
    - Indledning
    - Problemformulering
    - Afgrænsning
- At udarbejde og dokumentere en Analyse som ligger til grund får valget af løsningsmodellen.

Detaljering:

Det primære formål med Detaljeringsfasen er at få bygget en stabil arkitektur for den færdige løsning. Resultatet af denne fase er et "skeletagtigt" men kørende system, der demonstrerer den endelige applikations arkitektur. Med hensyn til dokumentationen skal analyse afsnittet skrives i denne fase. Denne fase skal være færdig efter 2 iterationer eller 4 uger.

Hovedaktiviteterne i Detaljeringsfasen er:

- At få en bedre forståelse for de indsamlede krav, System Use Cases
- At designe, implementere og verificere en arkitektur, der kan understøtte den færdige løsning

- At implementere de kritiske System Use Cases og hændelser, der blev identificeret i Foranalysefasen
- At sikre at miljøet omkring projektet, så som projekt databaser, udviklings- og test miljøer, værktøjer og lokaler der er behov for til Konstruktions fasen, er på plads
- At få defineret og beskrevet test-strategier og test planer
- At få dokumenteret Design delen af projektet.

Konstruktionsfasen:
Formålet med Konstruktionsfasen at udvikle den resterende del af løsningen, på baggrund af den arkitektur, der blev defineret og udviklet i Detaljeringsfasen. Det er også her at den resterende funktionalitet udvikles og iterativt indarbejdes i applikationen.
Det primære resultat af Konstruktionsfasen er en produktionsklar version af den samlede løsning, der opfylder de krav til funktionalitet, der er blevet indsamlet og godkendt under detaljeringsfasen. I denne fase skal Implementerings afsnittet af rapporten skrives. Denne fase varer 2 iterationer.
Hovedaktiviteter i Konstruktionsfasen:

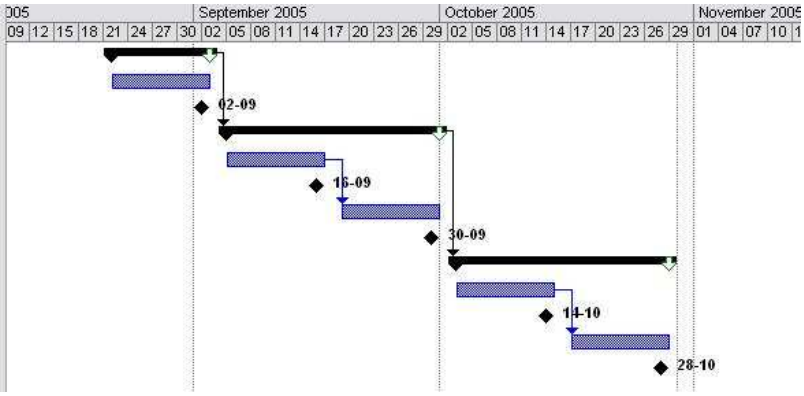
- Beskrivelse af de resterende krav, System Use Cases og Supplerende Specifikationer, der skal implementeres i de enkelte iterationer
- Færdiggøre Analyse og Design af den resterende funktionalitet
- Implementering af de resterende dele af løsningen og integrere disse i den samlede løsning
- Løbende test af de dele af løsningen der udvikles i de enkelte iterationer
- At få dokumenteret Implementerings delen af projektet.

I konstruktionsperioden er det implementeringen af den valgte løsningsmodel som har første prioritet. Det eneste der skal dokumenteres i denne fase de problemstillinger man vil komme til at møde når man implementerer.

2.2 Tidsplan

Dette afsnit er en af artefakterne fra Foranalyse fasen. Jeg har udarbejdet en Gantt tidsplan

	Task Name	Duration
1	- **Foranalyse**	**10 days**
2	Iteration 1	10 days
3	Rapport release (Projekt definition)	0 days
4	- **Detaljering**	**20 days?**
5	Iteration 1	10 days?
6	Rapport release (Analyse)	0 days
7	Iteration 2	10 days
8	Rapport release (Design)	0 days
9	- **Konstruktion**	**20 days**
10	Iteration 1	10 days
11	Rapport release (Implementation)	0 days
12	Iteration2	10 days
13	Rapport release (Test)	0 days



3 Analyse

I dette afsnit vil jeg analysere en række rapporterings værktøjer og vurdere hvilket af disse er bedst egnet til at generere de ønskede rapport typer. Først vil jeg analysere de forskellige rapport typer som brugerne ønsker, og identificere en række krav/kriterier til værktøjet. Derefter vil jeg kigge nærmere på de enkelte værktøjer. Sidst i afsnittet vil jeg udvælge en af værktøjerne og begrunde mit valg.

3.1 Kravspecifikation:

Brugerne har lavet en række oplæg til hvordan rapporterne kunne se ud. Efter aftale med projektet og brugerne har jeg valgt at holde mig til deres overordnede specifikationer, men samtidigt har jeg fosøgt at gøre designet mere indbydende.
Hvis man kigger nærmere på rapport oplægget, kan man se at der er mange afsnit som bliver genbrugt i flere af rapporterne. Det vil derfor være en god ide hvis man kan dele rapporterne op i mindre afsnit, som man kan genbruge. Jeg har udarbejdet en liste af de afsnit/del-rapporter som indgår i alle rapporter. Jeg prioriterer Kredit indstilling rapporten højst, da denne rapport er den vigtigste ifølge brugernes prioritering. Denne rapport stiller følgende krav til rapporterings værktøjet.

- Det skal være muligt at opdele rapporterne i en række genbrugelige delrapporter.
- Felterne i rapporterne skal være dynamiske. Hvis et felt er tomt, skal den ikke vises på printet og hvis værdien af feltet vokser skal feltet også vokse i størrelse.
- Det skal være muligt at hente billeder fra en database. Skal bruges i forbindelse med Koncerndiagrammet afsnittet.
- Det skal være muligt at gemme rapporterne i ESDH, som PDF dokumenter.
- Rapporterne skal kunne kaldes fra en .net applikation.
- Løsningsmodellen skal overholde principperne den service orienteret arkitektur.

3.2 Use-cases

3.2.1 Vis Rapport

USE CASE 1	Vis rapport	
Formål	At generere og vise de ønskede rapport	
Betingelser	Den pågældende indstilling findes i databasen	
Succesfuld afslutning	Rapporten bliver genereret og vist på brugerens skærm	
Trigger	Der trykkes på ”Vis rapport” knappen.	
BESKRIVELSE	**Step**	
	1	Brugeren trykker på vis rapport
	2	Brugeren vælger den rapport type han hun ønsker at vise
	3	Rapporten bliver genereret og vist i browseren.
UDVIDELSE	**Step**	
	2.a	Brugeren vælger den rapport han/hun vil se

3.2.2 Gem rapport

USE CASE 1	Gem rapport	
Formål	At generere og gemme rapporten i ESDH	
Betingelser	Den pågældende indstilling findes i databasens	
Succesfuld afslutning	Rapporten bliver genereret og gemt i ESDH som pdf.	
Trigger	Hvornår skal den gemmes?	
BESKRIVELSE	**Step**	
	1	Brugeren trykker på gem rapport

3.3 Analyse af rapporterings værktøjer

I dette afsnit vil jeg analysere en række rapporteringsværktøjer og jeg vil opstille en række kriterier, og vurdere de enkelte værktøjer med disse. Kriterierne er udarbejdet på baggrund af en kort analyse af rapporterne og det udviklings miljø som de skal udvikles i.

Jeg vil vurdere rapporterne ud fra følgende kriterier.

Udviklingstid:

Grunden til at man overhovedet bruger et rapporterings værktøj frem for selv at udvikle rapporterne er at man gerne skulle spare tid. I sidste ende er det tids kriteriet som er den afgørende faktor når jeg skal vurdere værktøjet.

SOA grænseflade:

Jeg vil undersøge hvordan værktøjerne binder op i mod SOA arkitekturen. Den optimale grænseflade mellem rapporterne og SOA arkitekturen, er objekt bindinger. Grunden til dette er at besked strukturen i BRF´s SOA model er entitets klasser, derfor vil det være optimalt hvis rapporteringsværktøjet kan binde direkte på objekter og collections af objekter.

Export muligheder:

Det skal være muligt at exportere/gemme rapporterne. Grunden til dette er at brugerne ønsker at gemme rapporterne i en database. BRFkredit bruger en database som hedder ESDH (Elektronisk Sag og Dokument Håndtering) som bruges til at gemme diverse dokumenter. Rapporterne skal gemmes i PDF format. Grunden til dette er at denne filtype er både udbredt og ikke redigerbar. Hvis man eksempelvis gemte rapporterne som RTF filer, ville brugerne kunne redigere i rapporterne uden at gemme disse ændringer i databasen. Endvidere ville det betyde at man mistede historikken i rapporterne, det ville ikke være muligt se hvem der har redigeret rapporten.

Fleksibilitet:

Værktøjer skal gøre udviklerens arbejde nemmere uden at begrænse udvikleren valgmuligheder. Det skal være muligt at redigere den bagvedliggende muligvis auto genererede kode, og på den måde have 100% kontrol over hvordan data hentes, sorteres og behandles.

3.3.1 Crystal Reports

Crystal Reports er en rapportgenerator udviklet af Seagate Software. Crystal Reports er et af de førende værktøjer til behandling og analyse af data. Jeg har kigget nærmere på den version af Crystal Reports som følger med Visual Studio 2003. Hvis man er interesseret kan man også investere i Crystal Reports standalone designer som har flere funktioner og er nemmere at arbejde med.

En rapport i Crystal Reports består af flere sektioner. Nogle af de mest anvendte er Page header og Details sectionen. Detail sektionen bruges hvis man vil vise en eller flere rækker fra databasen.

Hvis man vil rapportere fra en tabel som indeholder en liste af ejendomme, og man ønsker at vise alle ejendommene, skal man blot flytte felterne fra ejendom tabellen til detail sektion. Crystal Reports vil iterere gennem tabellen og lave en ny række i rapporten for hver række i tabellen. Rapporten kan derfor vokse alt efter hvor mange rækker der findes i tabellen. Hvis man skal hente data fra flere tabeller og man har flere en til mange relationer kan man indsætte subrapporter i sin main rapport. En subreport har de samme sektioner som en normal rapport dokument. Den eneste begrænsning er at en subreport ikke kan indeholde en anden subreport.

Crystal Reports binder op imod datasets, hvilket ikke er helt ideelt til dette projekt. Dette indebærer at man skal konvertere fra objekter til datasets. Heldigvis findes der en række funktioner i .net som kan hjælpe mig med at opnå dette.

En anden cool feature med Crystal Reports, er at man ud fra hver enkelt felt og sektion kan programmere hvordan feltet skal vises. For at gøre dette skal man skrive en ”Formula”, dette kan man gøre i Crystal Reports eget kode sprog eller i VB. Denne funktion er meget god hvis man vil lave valideringer eller beregninger i rapporterne. Et af brugernes krav er, at hvis et felt ikke er udfyldt skal det udgå af printet. Her vil det være oplagt at skrive en simpel formula som kontrollerer om feltet er udfyldt. Hvis man ikke vil bruge formula funktionen, men hellere vil kode i C# kan man også gøre dette. Dog skal man huske at det er mere tidskrævende, idet man selv skal gennemsøge rapporten og finde det felt man ønsker at redigere/validere.

Crystal Reports er det værktøj som har mange eksporterings muligheder. Man kan eksportere til over 20 slags formater, incl PDF, RTF og EXCEL.

Når man har designet en rapport og ønsker at vise den skal man indsætte et crystalReportViewer komponent i sin win eller webform. Denne viewer indeholder en række knapper som brugeren kan benytte til at udskrive, gemme og navigere rundt på rapporten. Man kan vælge at gruppere data, så de fremstår som afsnit og underafsnit. Man kan f.eks lave en liste af ejendomme og når brugeren trykker på en af rækkerne, vil rækken ekspandere og man vil se yderligere information om den pågældende ejendom. Denne feature kan bruges til at gøre rapporten mere overskuelig og give et bedre overblik. Jeg har valgt ikke at implementere dette i mit projekt, da de genererede rapporter altid skal udskrives.

Udviklingstid:	På grund af de mange funktionaliteter som værktøjet stiller til rådighed er udviklings tiden kort. Hvis man skal lave større komplekse rapporter kan man blive nød til at implementerer funktionalitet som ikke findes i Crystal Reports.
SOA grænseflade:	Crystal Reports binder op imod datasets, hvilket ikke er optimalt da vi i dette projekt anvender objekt bindinger. En eventuel Crystal Reports løsning indebærer at man implementerer en række konverterings komponenter som kan konvertere objekter til datasets.
Export muligheder:	Crystal Reports kan eksportere til over 20 formatter, bl.a. Word, PDF, Excel og HTML. Crystal Reports er det bedste værktøj hvad angår eksporterings muligheder.
Fleksibilitet:	Crystal Reports er meget nemt at anvende og lære. Man kan stort set lave en rapport kun ved at bruge ”drag and drop” funktionen. Dette betyder også at værktøjet tager en række beslutninger som udvikleren ikke har kontrol over.

Konklusion

Crystal Reports er meget effektiv hvis man skal lave simple rapporter. Hvis man f.eks skal lave en rapport som henter data fra en række tabeller som har en parent – child relation er Crystal Reports uden tvivl et godt valg. Hvis man derimod skal lave store rapporter som henter data fra flere tabeller som dem jeg skal lave, har Crystal Reports sine begrænsninger. Man kan programmere sig uden om nogle af disse problemer, men det er tidskrævende.

3.3.2 ASP.net

ASP.NET er som bekendt ikke et rapporterings værktøj, men derimod et programmerings sprog. Grunden til at jeg har ASP.NET med i min analyse er at man i BRFkredit altid har brugt ASP.NET til at generere dynamiske rapporter. Der er en række fordele og ulemper ved at bruge ASP.NET som rapporterings værktøj. En af de største ulemper er, at det kan være tidskrævende at programmere. Simple features såsom sidetal bliver til store problemer, som man skal kode sig ud af.

Hvis man skal bruge ASP.NET til at designe rapporterne kan man bruge Visual Studios drag and drop funktioner til at oprette og redigere felterne og tabellerne i rapporten. Visual Studio autogenererer den bagvedliggende kode. Udvikleren kan dog redigere koden hvis det skulle være nødvendigt.
Når man skal udfylde felterne i rapporten, skal man selv mappe fra datakilden, som i dette projekt er entitets objekter, til felterne i rapporten. Dette skal man gøre for alle felterne i rapporten. Med simple objekter er dette ikke et problem. Man laver blot et felt i rapporten som man udfylder runtime. Hvis man derimod har lister i rapporten (f.eks. en liste af ejendomme), og man ikke kender antallet af elementer skal rapporten kunne vokse alt efter hvor mange elementer der findes i listen. I de rapporter som jeg skal designe er der lister flere steder, f.eks. kan en indstilling indeholde nul eller flere ejendomme. I dette tilfælde vil man have en tabel og der vil være en række for hver ejendom i listen. Runtime betyder det at man skal gennemløbe listen og for hvert element oprette en ny række i tabellen og derefter indsætte værdierne fra elementet ind i rækken. Dette skal man gøre for alle objekterne og det kan være tidskrævende. Dog skal man huske at det til tider kan være en fordel at man selv skal kode alt, fordi det indebærer at udvikleren har fuld kontrol over hvordan rapporten bliver genereret. Når man anvender Crystal Reports eller et af de andre værktøjer er man tit begrænset til de funktioner og features som værktøjet stiller til rådighed. Og derfor har man begrænset kontrol over hvordan rapporterne bliver genereret. I den forstand er ASP.NET meget fleksibel men også mere tidskrævende.

Udviklingstid:	En ASP.NET løsning er tidskrævende at designe og implementere, men til gengæld har man fuld kontrol over hvordan rapporten bliver genereret. Man kan sortere og konvertere objekterne som man vil og man kan udføre validering af objektet felter.
SOA grænseflade:	ASP kan bindes op imod alle datatyper. Man skal blot selv mappe og eventuelt konvertere felterne i datakilden. I dette projekt er datakilden entitets objekter og da ASP også er objekt orienteret behøver man ikke at konvertere objektet.
Export muligheder:	I ASP har man ingen eksport muligheder. Man kan generere en rapport men ikke gemme den. Man kan komme uden om problemet ved at anvende software som kan konvertere fra html til pdf dokumenter. Men som udgangspunkt stiller ASP ingen eksporterings funktioner tilrådighed.
Fleksibilitet:	Som tidligere nævnt er ASP meget fleksibel idet udvikleren selv skal kode sig ud af problemerne. Dette er selvfølgelig på bekostning af tid.

Konklusion

ASP er helt klart værd at overveje når man skal generere dynamiske rapporter. Det at man kan binde op imod objekter gør det meget nemt at arbejde med. Der skal ikke laves nogen former for konvertering. En anden fordel ved at anvende ASP er at man som udvikler har fuld kontrol over hvordan data præsenteres og vises på skærmen. En af de største ulemper med ASP er at det kan være tidskrævende at kode. En anden ulempe er at man som udgangspunkt ikke har nogen eksporterings/gem muligheder.

3.3.3 Konklusion

ASP.net er i stand til at lave objekt bindinger og derfor er det nemmere og hurtigere at implementere en ASP løsning hvad angår SOA kompatibilitet. Det er forholdsvis nemt at lave data bindingen til entitets objekterne. På den anden side stiller ASP.net ingen rapporterings funktionaliteter til rådighed, man skal kode en masse ASP.net hvilket er meget tidskrævende.
I modsætning til ASP.net bruger Crystal Reports dataset bindinger. Dette betyder der skal laves en form for konvertering af entitets objekter til datasets. Men til gengæld er det nemmere og hurtigere at designe og implementere rapporterne med Crystal Reports. Grunden til dette er at værktøjet stiller en række funktionaliteter til rådighed såsom sortering, konvertering, sidetal etc. Alle disse funktionaliteter skal man selv implementere med en ASP.NET løsning.

En af brugernes krav var at det skal være muligt at gemme en rapport til ESDH databasen i PDF format, Crystal Reports opfylder denne krav, det gør ASP.NET ikke. Man skal investere i et software komponent som kan eksportere ASP.NET sider til PDF sider.

I den nedenstående tabel sammenligner jeg værktøjerne:

	Crystal Reports	ASP.NET
Udviklingstid	Kort udviklingstid.	Lang udviklingstid.
Fleksibilitet	Begrænset til de funktioner som Crystal Reports stiller til rådighed.	Ingen begrænsninger.
Eksporterings muligheder	Kan eksportere til flere formater heriblandt PDF.	Ingen.
SOA kompatibilitet	Dårlig, da den anvender datasets	ASP.NET har den bedste grænseflade til SOA.

Selvom Crystal Reports ikke har en særlig god grænseflade mod SOA, stiller den en række funktioner til rådighed som kompenserer for dette.
Summa summarum vurderer jeg at en Crystal Reports løsning med en konverterings komponent er den mest passende løsning.

4.0 Design

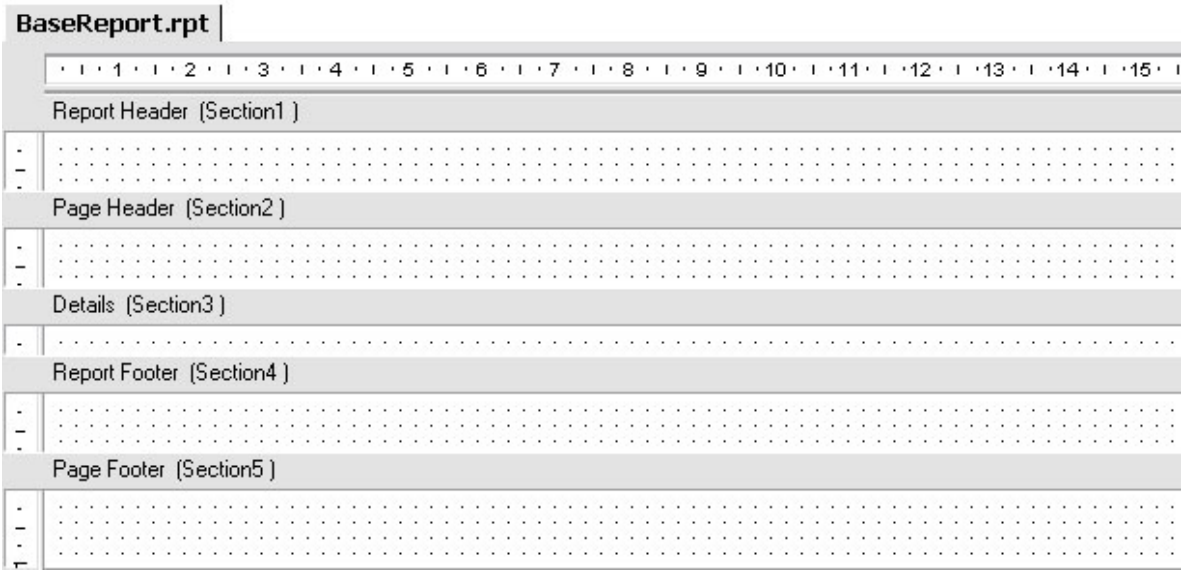
I dette afsnit vil jeg undersøge hvordan jeg kan designe og implementere rapporterne med Crystal Reports (CR). Da jeg både skal lave rapporterne og hente nødvendige data fra data laget, er det nødvendigt at undersøge både Crystal Reports og SOA arkitekturen nærmere.

4.1 Crystal Reports

Jeg vil kigge nærmere på de funktionaliteter som Crystal Reports stiller til rådighed. Jeg vil præsentere en række løsningsmodeller og endeligt vil jeg vælge den løsningsmodel som jeg synes passer bedst til den stillede opgave.

4.1.1 Rapport dokument

Et Crystal Reports dokument består af flere sektioner. De vigtigs sektioner er header, footer og details.



Header sektion vil være synlig øverst på alle sider, men footeren vil være synlig på bunden af siden. Den mest interessente sektion er detail sektioner. I modsætning til de andre sektioner, kan denne sektion udskrive flere rækker. Hvis man binder op imod en liste som indeholder 10 rækker, og man indsætter felter fra tabellen til detail sektionen, vil Crystal Reports iterere igennem tabellen og udskrive værdierne fra listen.

Hvis man henter data fra flere tabeller, skal man i Crystal Reports angive relationen mellem tabellerne. Når man har lavet en relation kan man vælge at lave den som inner eller outer join som man kender det fra SQL.

Man kan indsætte flere objekter til rapport dokumentet. Nogle af de mest anvendte Crystal Reports objekter er:

- Database felter
- Formula felter. En formula er det samme som en funktion i normal programmering. Når man skriver en formula skal man altid sætte en værdi for det reserverede ord ”Formula”. Dette svarer til en ”return” i normal programmering. Man kan bruge formula felter til at beregne felter som man kan dele med andre de andre subrapporter.
- Opsummerings felter
- Grafer
- Subrapporter. En subrapport er det samme som en normal rapport med den undtagelse at den ikke kan indeholde andre subrapporter. Jeg har implementeret de forskellige afsnit som subrapporter, på den måde kan jeg genbruge subrapporterne i flere rapporter.

4.1.2 Rapport visning

Når man skal præsentere en Crystal Reports rapport til brugeren skal man tilføje en Crystal Report viewer til sit projekt. En viewer indeholder en toolbar som tillader brugeren at navigere sig igennem rapporten og udføre forskellige handlinger på rapporten såsom gem, udskriv, søg osv..
Hvis man ikke ønsker at bruge Crystal Reports egen viewer kan man eksportere rapporten til pdf format og sende data direkte til browseren (on-the-fly). Dette betyder at rapporten vil blive åbnet og vist med Adobe Acrobat Reader som stiller langt flere funktioner tilrådighed for brugeren. Med denne løsning er man også fri for at kode ASP.

Export muligheder

Crystal Reports kan blandt andet eksportere til følgende formater.:

- Adobe Acrobat (.pdf)
- Crystal Reports for Visual Studio .NET (.rpt)
- HTML 3.2 and 4.0 (.html)

- Microsoft Excel (.xls)
- Microsoft Rich Text (.rtf)
- Microsoft Word (.doc)

4.1.3 Crystal Reports processerings model.

Når Crystal Reports skal genere en rapport kan lave op til fem gennemløb alt efter rapportens kompleksitet. Nedenstående figur illusterer denne processerings model.

Figur 2 Crystal Reports processerings model. Billede er hentet fra MSDN

Pre-Pass 1

Når Crystal Reports skal generere en rapport, vil den som det første evaluere konstante formularer. En konstant formula indeholder værdier som er statiske igennem rapporten. En konstant formular er ikke afhængig af data fra databasen, eksempel på en konstant formular kunne være en simpel beregning af to konstante værdier. En konstant formula bliver kun evalueret en gang. Denne process eller gennemløb hedder ”BeforeReadingRecords”. Denne specielle navn kan man bruge i en formula, for at tvinge Crystal Reports til at udføre den pågældende formula allerede inden den har hentet data fra databasen.

Pass 1

Efter "BeforeReadingRecords" processen, vil Crystal Reports hente data fra databasen. Mens data hentes vil Crystal Reports udføre følgende handlinger:

- Evaluere formularer som indeholder database felter.
- Filtrering af data. Hvis man i Crystal Reports har skrevet en select statement hvor man filterer data vil det blive eksekveret i dette gennemløb.
- Sortering, gruppering og summering af felter. Crystal Reports vil sortere data, opdele data i grupper og beregne eventuelle summerings felter i disse grupper.
- Generere krydstabeller.
- Gemme data i hukommelsen og i temp filer. Crystal Reports vil ikke oprette forbindelse til databasen igen, istedet vil den bruge de gemte værdier.

Pre-Pass 2

Under pre-pass 2 vil Crystal Reports sortere grupperne som blev oprettet under pass 1. Man kan sortere grupperne i top eller bottom N grupper som man kender det fra sql.

Pass 2

I pass 2 vil Crystal Reports generere siderne i rapporten. Siderne bliver genereret ”on demand”, dette betyder at en side bliver først genereret når brugereren anmoder om den. Når siden bliver genereret vil Crystal Reports udføre følgende:

- Filtrereng af grupper.
- Løbende summeringer.
- Evaluere formularer som er markeret med “WhilePrintingRecords”
- Charts and maps
- Genere eventuelle subrapporter.

Pass 3

I den tredje og sidste gennemløb vil Crystal Reports beregene sidetal. Denne gennemløb bliver kun udført i rapporter som viser total antal sider i hele rapporten.

4.1.4 Bindings muligheder

Når man anvender Crystal Report kan man vælge mellem flere bindings modeller og bindings datatyper. I dette afsnit vil jeg beskrive de muligheder som værktøjet stiller til rådighed. Der findes to overordnede modeller man kan vælge imellem det er Pull og Push modellen. I dette afsnit er det nødvendigt at tæt på den praktiske implementation af begge modeller for at afgøre, hvilken af dem er den er

4.1.4.1 Push

I push metoden skal udvikleren selv hente og indsætte data i rapporten. Push metoden kan benyttes ved at man får Crystal Reports til at binde op imod en lokal database fil (Dataset). Denne skema fil indeholder ingen data, dens eneste formål er at præsentere felterne og tabellerne i databasen. Nedenstående figur illusterer push modellen.

Man kan implementere en push model ved at man i design time opretter en xml fil som Crystal Reports binder op imod. Runtime skal hente data og konvertere data til xml format, derefter skal man sende data til rapporten. Det er vigtigt at data runtime og data i xml fil har samme struktur. Når man har oprettet en xml fil og man bruger denne som datakilde i Crystal Reports vil man design time se en liste af tabellerne og de felter som tabellerne indeholer. Nu kan man bruge drag & drop funktionen til at indsætte felterne i rapporten. For at fylde data i felterne skal man hente og konvertere data til dataset format. Derefter skal man sætte rapportens datasource til det dataset man har lige har hentet. Nu vil rapporten bruge datasettet i hukommelsen, fremfor den lokale dataset fil.

Det er derfor vigtigt at det lokale dataset fil og det dataset man henter er identiske, hvad angår felter og feltnavne. Hvis man design time tilføjer et felt i rapporten som ikke findes i det dataset som vi forsøger at bruge runtime vil applikationen fejle. Man kan dog komme udenom problemet ved at man laver en funktion som opdaterer den lokale fil hver gang man henter data. På den måde vil man

sikre at data i filen er synkroniseret med data i hukommelsen. Denne metode kan dog kun bruges i udvikling, da applikationen ikke har skrive rettigheder på produktions maskinen.

Hvis man vil bruge dataset filer skal man sørge for at konverere datakilden til dataset format. I mit tilfælde er datakilden entitets objekter, og derfor skal jeg konvertere fra objekter til datasets. Der findes ingen funktioner i .net frameworket som kan dette og derfor skal man selv udviklet en komponent/funktion som kan dette.

Hvis man ikke ønsker at binde op imod dataset filer kan man vælge XML. Fordelen ved at bruge XML fremfor XSD er at man nemt kan konvertere entitets objekter til XML og derefter til XSD. Man skal dog være opmærksom på at den XML format som jeg anvender, ikke indeholder information om datatyper og derfor vil alle felterne i datasettet være af typen string.
Dette kan være problematisk hvis man skal lave beregninger i rapporten. En mulig løsning til dette er at konvertere felter til et heltal og derefter udføre beregningerne. hvis det drejer sig om mange felter kan det være tidskrævende.

4.1.4.2 Pull

Hvis man vælger at anvende pull modellen, vil Crystal Reports binde direkte op imod tabellerne i databasen. Se nedenstående figur.

Denne metode er nemme at anvende da den ikke kræver at man skal kode noget. Det hele kan ordnes via de indbyggede funktioner i Crystal Reports. Applikationen udfører alt arbejdet for udvikleren, den åbner en forbindelse til databasen, læser data, indsætter data i rapporten og lukker forbindelsen.
Den største ulempe med denne fremgangsmetode er at frontenden vil have direkte forbindelse til databasen.

4.1.5 Sikkerhed

Hvis man har valgt at bruge push modellen, og dermed besluttet at udvikleren manuelt skal hente data fra databasen og at det er op til backend-delen af applikationen at oprette forbindelse til database og håndtere eventuelle sikkerheds problemet. Hvis man derimod anvender Pull-modellen skal man være opmærksom på nogle sikkerheds problemer.
Når man begynder at designe sine rapporter og man ønsker at hente data fra en database, bliver udvikleren bedt om at oprette en forbindelse til den database han/hun ønsker at bruge. I dette vindue kan man vælge at indtaste brugernavn og kode eller man kan vælge at bruge integreret sikkerhed. Hvis man vælger integreret sikkerhed vil Visual Studio bruge brugernes akkreditiver (”Credentials”) fra webserveren (I dette projekt har vi brugt IIS) til at forbinde til databasen. Hvis SQL serveren kører på samme maskine som IIS´en vil dette ikke skabe nogle problemer, men hvis man derimod anvender en fjern SQL server og man forsøger at køre rapporten runtime vil man få en login fejl. Crystal reports kan ikke overgive(impersonate) ens akkreditiver fra IIS´en til en fjern SQL serveren. Man skal selv angive eventuelle logon informationer såsom brugernavn og kode. Hvis man vælger at angive brugernavn og kode fremfor at vælge integreret sikkerhed, er det af sikkerhedsmæssige årsager kun brugernavnet der gemmes i rapporten. Derfor skal man igen selv angive logon informationen i koden. Til dette formål har Crystal Reports lavet et TableLogOnInfo objekt som indeholder tre properties. De er TableName, UserID og Password, man skal altid selv angive password propertien, mens de to andre felter bliver overført fra Crystal Reports design time. (Skal jeg slette denne afsnit ?)

4.1.6 Crystal Reports løsningsmodel

Da vi anvender en service orienteret arkitektur kan jeg ikke bruge pull modellen da dette vil gå udenom hele arkitekturen. Et af grundprincipperne i SOA er at Frontenden ikke har direkte adgang til databaser eller andre eksterne datakilder. Frontenden bør kun hente data via de udbudte services som backenden stiller til rådighed. Dette kan vi kun opnå ved push modellen.
Jeg vurderer at push metoden kombineret med XML binding er den mest passende løsning til den stillede opgave. Fordelen ved XML binding er at man nemt kan konvertere objekter til XML i .net Ulemper ved XML binding er at XML ikke indeholder information om datatyper. Alle datatyperne vil være string værdier, dette kan være problematisk da jeg skal foretage beregninger på nogle af felterne. Problemet kan dog løses ved at man manuelt konverterer de nødvendige felterne til heltal.

Alternativt kan man binde op imod XSD også kaldet XML Skema. I XSD filer ***skal*** man angive datatyperne på felterne. Hvis man skal anvende XSD skal man kunne konvertere Entity objekter til XSD klasser. Der findes ikke en indbygget funktion i .net som kan dette, derfor kræver det at man selv skal kode en komponent der kan løse opgaven. Jeg vurderer at den tid man skal bruge på at kode denne konvertering funktion ikke opvejer gevinsten ved at anvende XSD. Det er nemmere at konvertere de nødvendige felter til heltal felter.

Som tidligere nævnt har man flere muligheder for at vise rapporterne på skærmen, jeg vurderer at den bedste løsning er at generere rapporterne og præsentere rapporterne ”on the fly”. Dette betyder at når der kommer en forespørgsel skal rapporten genereres og vises på skærmen med det samme. Fremfor at vise en ASP side med en CrystalReportViewer komponent har jeg valgt at eksportere rapporten til PDF format og sende PDF data direkte til klientens browser. Dette vil medføre at rapporterne vil blive åbnet som normale PDF dokumenter med Adobe Acrobat Reader som er integreret med Internet Explorer. Både Acrobat og Internet Explorer er standard i BRF. Brugerne vil have langt flere muligheder hvad angår interaktion med rapporten via de indbyggede features i Acrobat Reader. En anden stor fordel er udviklings tiden. Med denne løsning skal man hverken kode html eller ASP.

4.2 SOA

I dette afsnit vil jeg kigge nærmere på SOA delen af projektet. Jeg vil beskrive de SOA principper som er standard i BRF.

4.2.1 Begrebsforklaring

4.2.1.1 Service

SOA står for Service Orienteret Arkitektur, og inden jeg går i dybden med emnet vil jeg kort forklare begrebet service. En service i SOA er i princippet det sammen som en service blandt mennesker. Dvs. det er en tjeneste eller opgave som udføres af nogen andre for os. Man kan have forskellige grunde til at benytte en service. 1) fordi en service udbyder er specialiseret i netop de services som den udbyder og dermed er den bedre end os selv. 2) fordi vi ikke selv kan eller gider, 3) fordi en service er billigere, 4) fordi vi ikke har nok ressourcer til at udføre opgaven selv. En service er en samling af relateret funktionaliteter som stilles til rådighed fra en *serviceudbyder* til en eller flere *serviceaftagere*. En service leverandør kan offentliggøre information om de udbudte services i et service register/katalog. Service aftagere kan slå en service op og hente information om den pågældende service de ønsker at bruge. Der har længe eksisteret en misforståelse om at web-services er det samme som eller synonymt med SOA. SOA er *ikke* en teknologi men en række karakteristika og ”best practice” principper om hvordan distribueret programmering bør laves. Det er derfor muligt at lave en applikation som aftager og udbyder web-services uden at overholde SOA principperne. Web-services er blot en af de mange kommunikations muligheder man kan vælge imellem når service aftagere og udbydere skal kommunikere. Det er meget almindeligt at services implementeres via webservices men det er ikke en forudsætning.

4.2.1.2 Løs kobling

Mange steder i rapporten bruger jeg udtrykket løs kobling, derfor er det nødvendigt at afklare hvad det vil sige at systemer er løst koblede. WC3 definerer løs kobling som (oversat fra engelsk): ”***Løs kobling går ud på at reducere de kunstige afhængigheder i videst mulig omfang”***

Man kan opdele afhængighed i ægte og kunstig afhængighed.

- Ægte afhængighed er de ydelser eller services som et system anvender fra andre systemer. Ægte afhængighed eksisterer altid og kan ikke reduceres.
- Kunstig afhængighed er det sæt af faktorer et system *er nødt til at efterkomme* for at kunne anvende ydelser eller services fra andre systemer. Af kunstige afhængigheder kan nævnes; 1) tegnsæt afhængighed, 2) API afhængighed, 3) platform afhængighed, 4) afhængighed af kendskab til servicens interne opbygning etc.

En konceptuel eksempel på løs kobling problemstillingen:

Hvis man rejser til udlandet og ønsker at medbringe elektroniske apparater er man i visse tilfælde nødsaget til at medbringe en strøm adapter. Den ægte afhængighed i dette tilfælde er strømbehovet og den kunstige afhængighed er at vores stik skal passe ned i den lokale kontakt, hvor de måske kører med en anden spænding og frekvens. I dette eksempel eliminerer vi den kunstige afhængighed vha. strøm adapteren. Man vil aldrig kunne fjerne alle de kunstige afhængigheder, men man kan reducere dem til et minimum og derved opnå løs kobling.

4.2.2 Hvad er SOA

SOA er en arkitektonisk fremgangsmåde til fremstilling af systemer som består af autonome services. Et af de vigtigste mål i SOA er at opnå løs kobling mellem software komponenterne. Dette gælder især mellem service aftagere og udbydere. En af SOA´s styrker er at man kan tage eksisterende it-systemer og få dem til at spille sammen, uden at skulle skifte platform.
Det er ikke muligt for mig at beskrive alle principperne og ideerne bag SOA. Jeg vil nøjes med at beskrive nogle grundprincipperne i SOA.

Princip 1: Design eksplicitte grænseflader

En SOA service kommunikerer ved at sende/modtage beskeder over veldefineret grænseflader. En grænseflade i denne sammenhæng henviser til grænsen mellem et service interface og dens interne implementation. En service grænseflade kan beskrives som en kontrakt som indeholder information om de udbudte services og om de datatyper som servicen forventer. Når man designer SOA applikationer skal man huske på at det er ydelses og tidsmæssigt dyrt at krydse denne grænse. Nogle af årsagerne til dette er at:

- Den fysiske placering af en service er ukendt.

- Sikkerheds/trust modeller kan ændre sig for hver gang man krydser grænsefladen.
- En ellers pålidelig service kan pludselig give nedsættelse i ydelse på grund af ny netværk konfiguration eller migrering til en anden fysisk placering.
- En service aftager ved normalt ikke hvordan servicen er implementeret. Service aftageren har begrænset kontrol over ydelsen af en given service.

Som det fremgår af listen er en service afhængig af netværket, herunder netværkets responstider, mulige netværksfejl og eventuelle distribueret systemfejl. En lokal implementation af en given funktion er ikke. Derfor bør man grundigt vurdere og vælge en given funktionalitet skal implementeres som en service eller som lokal komponent. Hvis man vælger at designe en komponent som en webservice bør man have følgende "best practice" principper i baghovedet.

- **Veldefineret grænseflader.** Alle SOA services skal stille en kontrakt til rådighed, som service aftageren skal overholde. Al kommunikation med services skal overholde denne kontrakt. Kontrakten definerer funktionerne som servicen udbyder samt de beskeder/datatyper som bruges i forbindelse med disse funktioner.

- **Det skal være nemt at aftage en service.** Når man designer en service, skal det være nemt for andre udviklere at aftage services. Service kontrakten skal designes med henblik på at servicen kan videre udvikles uden at bryde kontrakten med eksisterende service aftagere.

- **Undgå RPC (Remote Procedure Call) lignende grænseflader.** RPC anvendes typisk i en distribueret objekt arkitektur. RPC modellen er funktions centreret, dvs. grænsefladen udstiller en række funktioner som aftageren skal kalde for at udføre en given opgave. Dette betyder ofte at service aftageren skal udføre flere service kald, måske endda i en bestemt rækkefølge til at udføre en given opgave. Dette vil resultere i en stærk kobling mellem service udbydere og aftagere, og det vil også være svært at identificere eventuelle afhængigheder. Fremfor RPC lignende kald bør service aftageren kalde en service med en veldefineret besked, som dikterer den handling der skal udføres.

- **Design kun få grænseflader for hver service.** Hvis en service udstiller mange interfaces, vil den være sværere at aftage og vedligeholde. Derfor er det bedre at designe og udstille

færre interfaces til en service. Når man har færdig designet et interface, er det vigtigt at den forbliver statiks. Man skal så vidt muligt undgå at redigere i service interfaces og i stedet lave nye interfaces, hvis servicen skal udvides.

- **Implementerings detaljerne skal forblive internt i servicen.** Implementerings detaljerne for en service må ikke ”sive” ud over service grænsefladen. En service aftager bør ikke have implementerings detaljer om en service, hvis dette er tilfældet kan man risikere at service aftageren udvikler kode der er afhængig af den specifikke implementering af servicen. Hvis implementeringen af servicen ændres, vil det betyde at service aftageren også skal foretage ændringer, hvilket ikke er optimalt. Hvis man har designet en service som bruger flere komponenter/funktion til at løse en given opgave, bør disse ikke være tilgængelige i grænsefladen, da det siger noget om selve implementationen af den pågældende service. Servicen skal kun udstille den generelle komponent som anvender hjælpe metoderne. På den måde kan hjælpe funktionerne redigeres og udvides uden at det påvirker service aftagere.

Princip 2: En service bør være autonom

En service er en uafhængig selvstyrende entitet. En service kan igangsættes, versioneres og håndteres uafhængigt. Pointen med at designe autonome services er at vi ønsker at opnå løb kobling mellem service aftagere og udbydere.

En service kan f.eks. kaldes ved hjælp af en URI adresse. Dette betyder at den fysiske placering af serveren som hoster den pågældende service kan ændre sig. Dette vil ikke have nogen indflydelse på selve servicen, men kan derimod medføre uforventede konsekvenser for service aftageren. Det kan resultere i højere responstider og/eller ustabil service stabilitet. Derfor bør service aftagere og udbydere påtage et pessimistisk synspunkt om hvordan servicen kan aftages/udbydes. Fra en service aftagers synspunkt indebærer det at der skal implementeres funktionalitet til fejl dedektering og håndtering i forbindelse med service kald.

Det er ikke kun service aftagere der skal være pessimistisk, en serviceudbyder skal også være pessimistisk når de udbudte services skal aftages. En serviceaftager kan fejle, uden at give besked om det og en serviceudbyder bør ikke stole på serviceaftageren til at kalde servicen rigtigt. En

service aftager kan bevidst/ubevidst forsøge at misbruge servicen, ved at sende forkerte eller korrupte beskeder. Derfor bør service udvikleren tage højde for disse problemer, blandt andet ved at validere input beskeder.

Selvom man designer sin service til at være autonome entiteter, kan jeg med det samme fortælle at en service er ikke en ø. En SOA baseret applikation består næsten altid af en række services som er designet til at løse en specifik problemstilling. Hvis man forsøger at designe autonome services vil man hurtigt opdage at der ikke kan eksistere en service som kalder en anden service. Dette vil være imod den autonome tankegang, da servicen vil være afhængig af en anden service. I praksis vil man ikke udvikle 100% autonome services, men man kan tilstræbe på at isolere sine services så vidt muligt og på den måde opnå løs kobling.

- **Design autonome services så vidt det er muligt.** Services bør igangsættes og versioneres uafhængigt af systemerne som skal aftage servicen.
- **Service interfacet/kontrakten er endegyldig.** Kontrakten eller interfacet mellem system og service bør designes med henblik på at den ikke kan ændres når den først er designet. Denne fremgangsmåde vil tvinge udviklere til at indbygge fleksibilitet i deres kontrakt og/eller besked struktur.
- **Design robuste services.** Hvis man påtager et pessimistisk syn, til både service udbyderen og aftageren vil man opdage mulige svagheder. Fra en service aftagers synspunkt indebærer dette at, man skal tage højde for manglende/ustabil tilgængelighed og ydelse fra servicen. Fra en service udbyders synspunkt, indebærer dette at man skal designe sit system med henblik på at aftageren kan misbruge servicen.
- **En service ejer data.** Flere forskellige services må ikke tilgå de samme data. Hvis flere services deler den samme domæne område og man laver en ændring i domæne modellen vil det resultere i at man skal tilrette flere services. For at minimere denne afhængighed skal man kun lave en service pr domæne område.

Princip 3: En services udveksler xml beskeder fremfor klasser

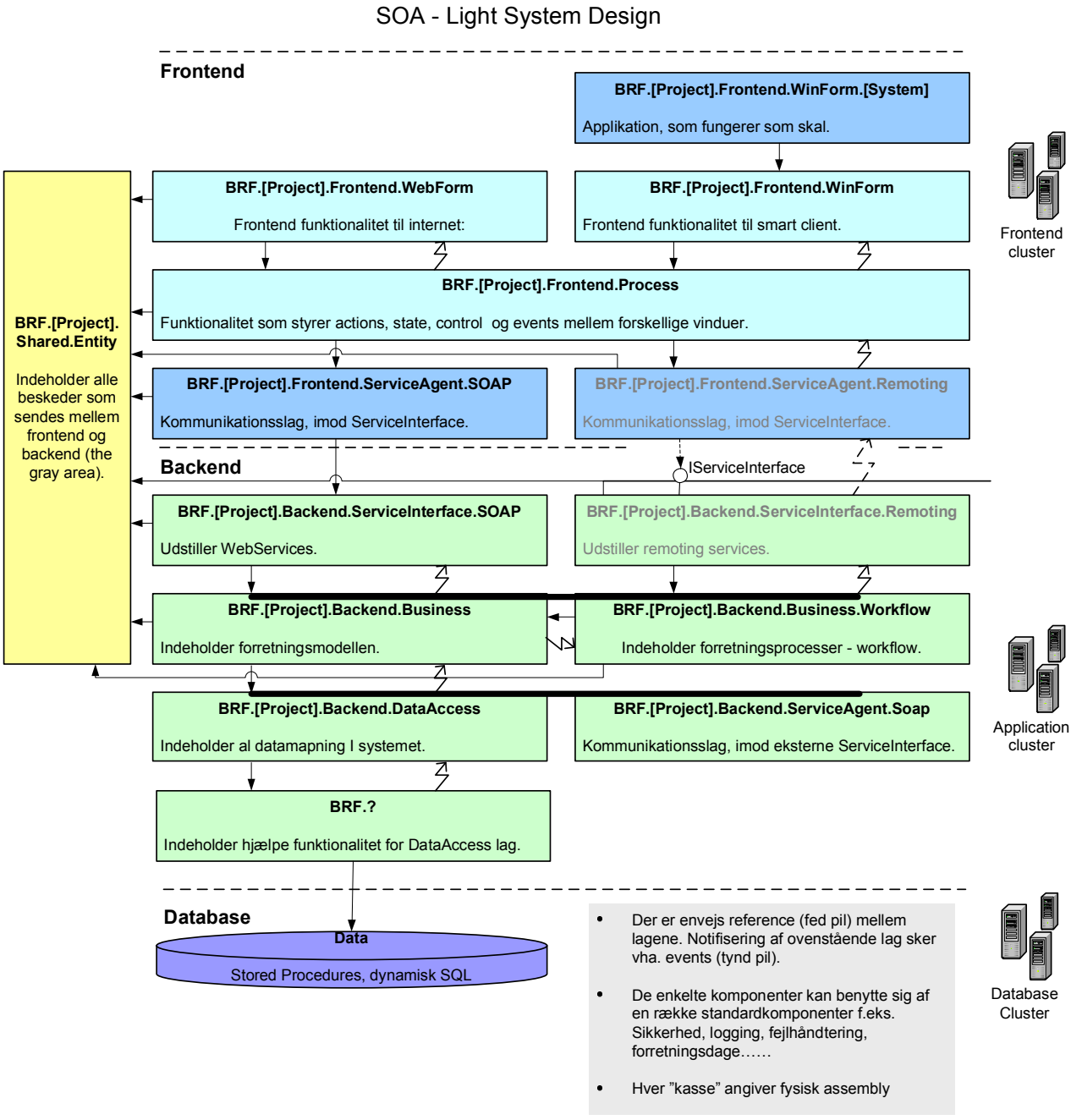
Som tidligere nævnt skal services kun udveksle på forhånd definerede beskeder som er defineret ud fra en service kontrakt. En af de mest anvendte service kontrakt typer er den xml baserede WSDL. For yderligere information om WSDL, se kapitel 4.2.3.2.4

Når man designer og implementerer en applikation anvender man klasser som repræsenter entiteter i et forretnings specifikt problem (Indstilling, Ejendom, Interessent). Disse klasser indeholder kun data. En klasse er både programmeringssprog og platform afhængig. Hvis services udvekslede klasser fremfor xml beskeder, ville to forskellige services, der er udviklet i forskellige sprog og platforme ikke kunne kommunikere med hinanden. For at kommunikere skal parterne have en fællesnævner som de begge forstår, i dette tilfælde er det xml. En SOA service udstiller datatyperne som .net frameworket konverterer xml baseret beskeder som er platform uafhængige. Det er i kraft af XML teknologien at SOA services kan opnå platform uafhængighed og interoperabilitet. XML beskederne som udveksles mellem service aftager og udbyder kan defineres i en WSDL fil. Denne service kontrakt indeholder information om de xml beskeder som servicen skal bruge. Både service aftageren og udbyderen skal bruge disse xml beskeder til at kommunikere. Det er derfor meget vigtigt, især for service aftageren at kontrakten er statisk og ikke ændrer sig. En service kontrakt skal tage højde for at beskederne kan ændre sig uden at det påvirker service aftagerne. Dette vil tvinge service udvikleren til at implementere muligheden for udvidelser i besked strukturen.

- Service kontrakten skal være statisk.
- Service kontrakten bør designes med henblik på at services kan udvides. Udvidelsen skal ikke have indflydelse på service aftagerne. Dette kan man opnå ved at versionere service interfacet.
- Grænsen mellem en privat og tilgængelig data skal styrkes. Servicens interne data bør skjules fra service aftageren mens de tilgængelige datatyper bør være ”immutable” (Kan ikke ændre sig)
- Hvis det er nødvendigt kan man versionere service kontrakten. Denne fremgangsmetode bevirker at man ikke bryder kontrakten med eksisterende service aftagere.

4.2.3 BRFkredit´s SOA arkitektur

4.2.3.1 Arkitektur diagram



4.2.3.2 Arkitektur beskrivelse

I dette afsnit vil jeg beskrive BRFkredit´s SOA arkitektur. Derudover vil jeg forklare nogle af de design overvejelser jeg har gjort mig. I den forrige afsnit har jeg beskrevet det konceptuelle design af SOA arkitekturen og i dette afsnit vil jeg lave en mere detaljeret design.

4.2.3.2.1 Brugergrænseflade

Størstedelen af alle applikationer skal have en brugergrænseflade. Man kan implementere brugergrænseflader med Windows forms, ASP.net eller ved hjælpe af andre teknologier. Formålet med brugergrænsefladen er at formatere og vise data til brugerne, og/eller modtage og validere input fra brugeren. I dette projekt har vi to brugergrænseflader, den ene er lavet i ASP.net/Crystal Reports og bruges udelukkende til at vise de forskellige rapporter. Den anden brugergrænseflade er lavet med WinForms og den bruges til at indtaste, validere og beregne. Det resulterende data gemmes i databasen som senere bliver hentet og genereret til en rapport.

En brugergrænseflade er opbygget af flere kontrol komponenter såsom knapper, tekstfelter og dropdown tabeller. Når brugeren aktiverer en komponent, rejses der et event(hændelse) som håndteres af kontrollens eventhandler. Eventhandleren skal derefter kalde en mere generel funktion som udfører den ønskede handling. Den generelle funktion skal implementeres med henblik på at den skal kunne kaldes fra flere kontroller, på den måde kan man genbruge denne funktion flere steder. Pointen er at adskille den udførende del af koden fra den komponent specifikke eventhandler.

Når man designer brugergrænsefladen bør man følge disse retnings linier:

Brugergrænseflade komponenterne bør ikke have kendskab til hinanden.
Man bør undgå at hårdkode relationer mellem de forskellige forms. I stedet bør process laget sørge for at åbne de rigtige forms og synkronisere data og events. Man bør osgå undgå at hårdkode relationer mellem Parent og child forms. F.eks. hvis man har en liste komponent (parent), som har til opgave at vise en oversigt af ejendomme og hvis man har en ”vis detaljer” komponent, bør man ikke implementere funktionalitet i ”vis detaljer” komponenten som direkte binder op imod ejendom liste komponenten. På den måde er det muligt at kalde ”vis detaljer” komponenten fra flere steder i applikationen.

Input fra bruger skal valideres

Det er en god ide at validere input fra brugerne i brugergrænseflade komponenterne. Validering af input bevirker at applikationen skal lave færre funktionskald til server siden med forkert input.

Brugergrænseflade komponenterne bør ikke styre flowet i applikationen.

Brugergrænseflade komponenterne bør designes med henblik på at gøre dem til isoleret komponenter som kun eksisterer i deres egen scope uden kendskab til de andre brugergrænseflade komponenter. Det er proces lagets opgave at holde styr på flowet i applikationen og kalde de rigtige komponenter på de rigtige tidspunkter.

Vis en brugervenlig fejlside i tilfælde af fejl.

Det er altid en god ide at implementere en generel fejl-side som bliver vist i tilfælde af fejl. Fejl-siden bør vise en brugervenlig fejl beskrivelse og eventuelt en mulig løsning på problemet. Hvis man ikke gør dette kan man risikere at brugeren får en almindelig .net fejl side som er meget teknisk og intetsigende for brugeren.

Process agent

Alle brugergrænseflade projekter uanset, hvilken teknologi man har valgt, bør have en process agent komponent. Som navnet antyder er process agent en slags agent som kommunikerer med process laget, på vegne af brugergrænseflade komponenterne. For at opnå dette bør process agenten have en reference til process laget. Process agenten udstiller de relevante metoder som proces laget stillet til rådighed. Derudover indeholder Processagenten en række funktioner som kun er relevante for de enkelte brugergrænseflader projekter. I Crystal Reports brugergrænsefladen har jeg brug for en række konvertering funktioner til at konvertere fra objekter til datasets. Denne type af funktioner har kun relevans for Crystal Reports projektet og derfor bør de ligge i process agent klassen i Crystal Reports projektet.

4.2.3.2.2 Proces lag

Proces laget skal indeholde og udstille en række proces komponenter som kan kaldes fra flere forskellige brugergrænseflader. Når man designer en proces komponent skal man tænke globalt og

derfor skal implementeringen af proces komponenter ikke være brugergrænseflade specifikt. Man skal forsøge at adskille brugergrænsefladen fra proces laget for at opnå løs kobling. Dette kan man opnå ved at bruge delagates (funktionspointere), events og eventhandlers. Et event er beskeder der sker i et system når der sker en bestemt hændelse. En bestemt hændelse kan for eksempel være når brugeren trykker på en knap eller når en metode har udført en bestemt handling. Den sidstnævnte event skal man selv rejse, mens den første bliver rejst af .net frameworket. Alle events skal have en eventhandler tilknyttet til dem. En eventhandler indeholder den kode der bliver eksekveret når eventet bliver rejst. Delegates er type sikre funktionspointere, der tillader at sende en reference til en metode og eksekvere denne metode, uden direkte at kalde metoden. Når man laver en delegate skal den have samme signatur som den metode den skal eksekvere (samme parametre og samme retur type).

Når en proces komponent har udført en bestemt handling vil den rejse et event. Med andre ord vil proces komponenten signalere at den er færdig med at udføre en bestemt handling. På brugergrænsefladen vil der være en eller flere komponenter som lytter/venter på denne event. Når event bliver rejst vil de pågældende brugergrænseflade komponenter fange eventet og reagere på den.

Når brugeren skal udføre en opgave/handling skal man typisk lede dem igennem en proces, hvor der er flere stadier, det kan være information indsamling, beregning, og resultat visning. De enkelte brugergrænseflade komponenterne bør ikke holde styr på rækkefølgen af denne processen, i stedet bør de kalde funktioner i proces laget som derefter bestemmer hvilken side/stadie brugeren skal videresendes til.

Proces laget bør udstille følgende funktionalitet:

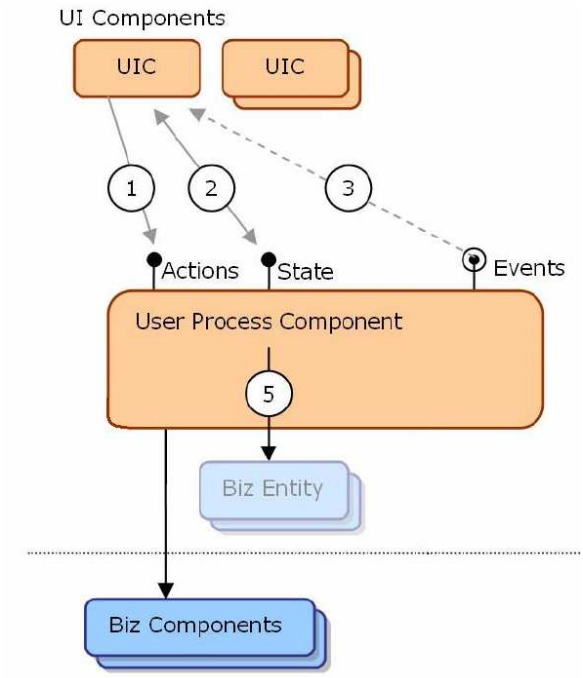
Udstille forretnings funktionaliteter (1)
Alle kald ned til backenden bør indkapsles i disse metoder. Fremfor at ruturnere forretnings objekterne direkte, vil disse funktioner opdatere de lokale kopier af forretnings objekterne i staten.

Opdate og udstille state objekter (2).

Proces laget bør udstille forretnings objekterne som simple get og set metoder. Alle tilgængelige objekter ligger i en state manageren, og bliver opdateret af de udførende funktioner. En state manager er frontendens cache af data.

Rejse events (3)

Når en proces komponent har udført en handling og har opdateret state manageren skal proces komponenten rejse en event, og på den måde give beked til brugergrænsefladen at den nu kan hente den opdaterede forretnings objekt fra staten. Normalt skal udvikleren selv rejse denne event, men med visse datatyper vil .net frameworket automatisk rejse en event når data har ændret sig. Dette gælder blandt andet for dataset.



Figur 3 Interface mellem proces og brugergrænseflade. Figuren er hentet fra Appliacation Architecture for .net

Kommunikationen mellem en brugergrænseflade og en proces komponente bør se således ud: Brugergrænseflade komponenten bør kalde proces laget for at udføre en bestemt handling på en forretnings objekt (1). Det kunne f.eks være at vise en bestemt indstillingen ud fra dens id. Proces laget vil kontrollere om det ønskede indstilling allerede findes i staten, hvis objektet ikke findes, vil proces laget kalde backenden for at hente denne indstilling. Når indstillingen er hentet vil proces komponenten tilføje den til staten og derefter rejse en bestemt event (3). Brugergrænsefladen komponenten vil fange event´et og hente indstillings objektet fra staten(2).

4.2.3.2.3 Service agent

Dette lag eller komponent har til funktion at kommunikere med udbudte services og udstille de relevante service funktioner. For at opnå dette kan man bruge facade pattern, som afskærmer og simplificere service aftagerens tilgang til servicen. Det er trods alt ikke vigtigt for service aftageren at vide hvordan man opretter forbindelsen til servicen eller at den bruger SOAP eller XML. For mere om denne pattern se afsnit 5.4.3.
Service aftageren og service udbyderen bruger forskellige datatyper, og det er service agent klassernes opgave at konvertere disse service datatyper til de tilsvarende datatyper i service aftager systemet. For at opnå dette kan man bruge proxy pattern. For mere information om denne pattern se afsnit 5.4.4.

4.2.3.2.4 ServiceInterface

Dette projekt har til formål at udstille de services som backenden stiller til rådighed.
Når man designer webmetoder er det vigtigt at metoderne er ”idempotent” dette betyder at hvis servicen modtager den samme besked flere gange vil resultatet/svaret være den samme.

Webservices
En webservice består af de følgende komponenter:

- ***En service proces.*** Dette er en software komponent som er i stand til at bearbejde en xml dokument, som den har modtaget fra en transport/applikations protokol. Hvordan komponenten er implementeret er irrelevant. Den eneste krav er at den skal kunne bearbejde en veldefineret XML dokument.
- ***En række dokumenter.*** Når en proces skal kommunikere med en webservice skal den sende parametre i form af xml dokumenter. Dokumenterne som en webservice proces kan bearbejde, dvs. modtage eller returnere er beskrevet i en separat XML skema dokument. Både service udbyderen og aftageren skal have adgang til denne dokument beskrivelse som indeholder information om de xml dokumenter som kan udveksles med den udbudte

webservice. Denne dokument er typisk beskrevet med Web Service Description Language (WSDL). Et WSDL dokument består af følgende sektioner:

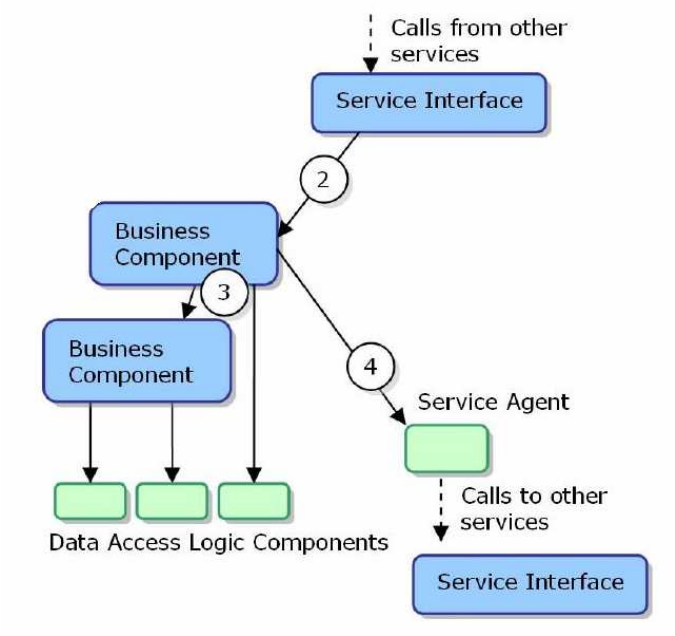
- *PortType.* Indeholder information om de funktioner som services stiller tilrådighed og de beskeder som funktionerne forventer. Portype sektionen kan sammenlignes med en klasse, i den forstand at en klasse kan indeholde flere metoder.
- *Message.* Denne section indeholder information om beskederne som indgår i service funktionerne. En besked kan bestå af flere dele. Delene kan sammenlignes med det der svarer til funktions parametre i traditionelt programmering.
- *Types:* Indeholder information om de datatyper som servicen bruger. For at servicen skal være platform uafhængig bruger vi XML til at definere syntaksen af data typerne.
- *Binding:* Indeholder information om kommunikations protokollen som bruges af webservicen, det kan eksempelvis være SAOP.

- ***En adresse.*** En webservice skal have en adresse, også kaldet port reference. Denne adresse skal bruges når man ønsker at bruge en webservice.
- ***Indkapsling***. Man skal bruge noget funktionalitet for at indkapsle beskederne og en række system informationer som skal bruges af kommunikations processerne. For at opnå dette kan man anvende SOAP protokollen. Denne protokol er baseret på xml og anvendes til at sende XML baserede beskeder. En SOAP besked består af to elementer: En soap header som indeholder diverse system informationer og en soap body som indeholder selve xml dokumentet som skal bearbejdes af servicen. SOAP protokollen er den mest anvendte protokol i forbindelse med webservices.

4.2.3.2.5 Business lag

Business eller forretnings laget er kernen i applikationen. Det er her alle forretnings/domæne funktionaliteterne er implementeret.

Forretnings komponenterne kan tilgås på følgende måder.:



Figur 4Forretnings komponent. Figuren er hentet fra Appliacation Architecture for .net

- Forretnings laget kan udstille funktionalitet som kan kaldes fra service interfacet (2)
- Forretnings komponenterne kan bruge data access komponenter til at hente, indsætte, og opdatere data i en database (3).
- En forretnings komponent kan via en service agent bruge eksterne services (4). Hvis man bruger eksterne services skal man implementere funktionalitet for at kompensere eventuelle fejl fra service.

4.2.3.2.6 Data Access lag

DataAccess laget er objekt representationen af database tabellerne. For at opnå dette har man i BRFkredit udviklet en standardkomponent som ved hjælp af Objekt Relation Mapping (ORM) kan tilgå en database typisk SQL og foretage skrive, læse og søge operationer. Denne komponent hedder DataMaster og bruger Stored Procedurer til at tilgå databasen. Tabellerne i databasen repræsenteres med dataaccess klasser, derfor er der en 1:1 forhold mellem database tabeller og dataaccess klasser. Til hver dataaccess klasse skal man lave en tilhørende læse, skrive og slet SP. Mapningen fra tabel værdier til objekt felter foregår via. attributter. Datamaster komponenten mapper rækkerne i tabellen til dataaccess objekter. Hvis man forventer at en SP vil returnere nul eller flere rækker skal man oprette en indexer klasse. Denne klasse fungerer som et liste objekt som indeholder flere dataaccess objekter. Man kan bruge denne indexer klasse til at iterere igennem listen og mappe dataaccess objekter til entitets objekter.

4.2.3.2.7 Entity lag

Dette lag indeholder alle forretnings objekterne. En forretnings objekt er en entitet i bestemt problemstilling. Entitets klasserne indeholder kun felter som er tilgængelige via get/set metoder, de bør ikke indeholde funktioner eller logik. Entitets klassernes eneste formål er at fungere som data bærende beskeder. Med andre ord er entity objekterne applikationens besked struktur. Disse klasser bliver serialiseret til XML dokumenter/beskeder ud fra et xml skema. XMLskemaet som indeholder information om klasserne kan findes i service kontrakten (wsdl) som udstilles af servicen.

De steder hvor det har været nødvendigt at anvende en liste af objekter har vi implementeret typestærke collections. En type stærk collection er en liste som kun kan indeholder den samme type af objekter. (Arrays er type stærke, arraylists er ikke). For at oprette en type stærk collection skal man nedarve fra collectionBase klassen. Denne klasse implementerer både ICollection og IList.

Fordi collectionBase klasse implementerer IList vil den nedarve en liste objekt, som kan fyldes ved at kalde add metoden. Som udgangspunkt er denne liste objekt ikke type stærk, men det kan den blive ved at lave en add metode, som kun accepterer en bestemt type af objekter. For at XMLSerializeren kan konvertere liste objektet til en xml format, er det nødvendigt at tilføje en indekser og en add metode.

4.2.4 Diskussion

Tidligere i rapporten har jeg beskrevet de principper som SOA arkitekturen foreslår at man overholder. Hvis man læser disse principper igennem vil man opdage at flere af dem er i modstrid med de mere traditionelle objekt orienterede principper. Jeg vil undersøge om SOA er en videre udvikling af den traditionelle OO tankegang eller om det er en videreudvikling af DO (distribueret objekter).

En af de områder hvor SOA og OO er vidt forskellige er deres forståelse og anvendelse af objekter. I traditionel OO er et objekt en instans af en bestemt klasse som både indeholder adfærd (funktioner) og data (properties og member variabler). Funktionerne i et objektet vil typisk operere på objektets egen data. I SOA arkitekturen bruger man også objekter, men i modsætning til traditionelle objekter har man adskilt objektets funktioner (adfærd) fra objektets klient specifikke data (state). Objekterne i SOA indeholder ingen logik, de består udelukkende af klient specifikt data. Disse objekter bliver konverteret til xml beskeder som udveksles mellem service udbyder og aftager. Set fra SOA arkitekturens side er objekter det samme som beskeder. Funktionerne som tidligere var i samme klasse som data, er implementeret som statiske funktioner i business laget.
I objekt orienteret applikationer opretter man instanser af forskellige klasser, og derefter kalder man funktioner i de enkelte objektet som operere på objektet egen data/tilstand. Eksempelvis myIndstilling.Save(), i dette tilfælde vil man gemme den pågældende instans af en klasse. I SOA arkitekturen har man adskilt objekternes data fra deres udførende funktioner. For at gemme et bestemt objekt i SOA arkitekturen skal man lave en gem funktion og som parameter kan man sende det objekt man ønsker at gemme. Eksempelvis GemIndstilling(indstilling). Dette betyder at man skal implementere og udstille gem metoder for alle de objekt typer man ønsker at gemme i databasen.

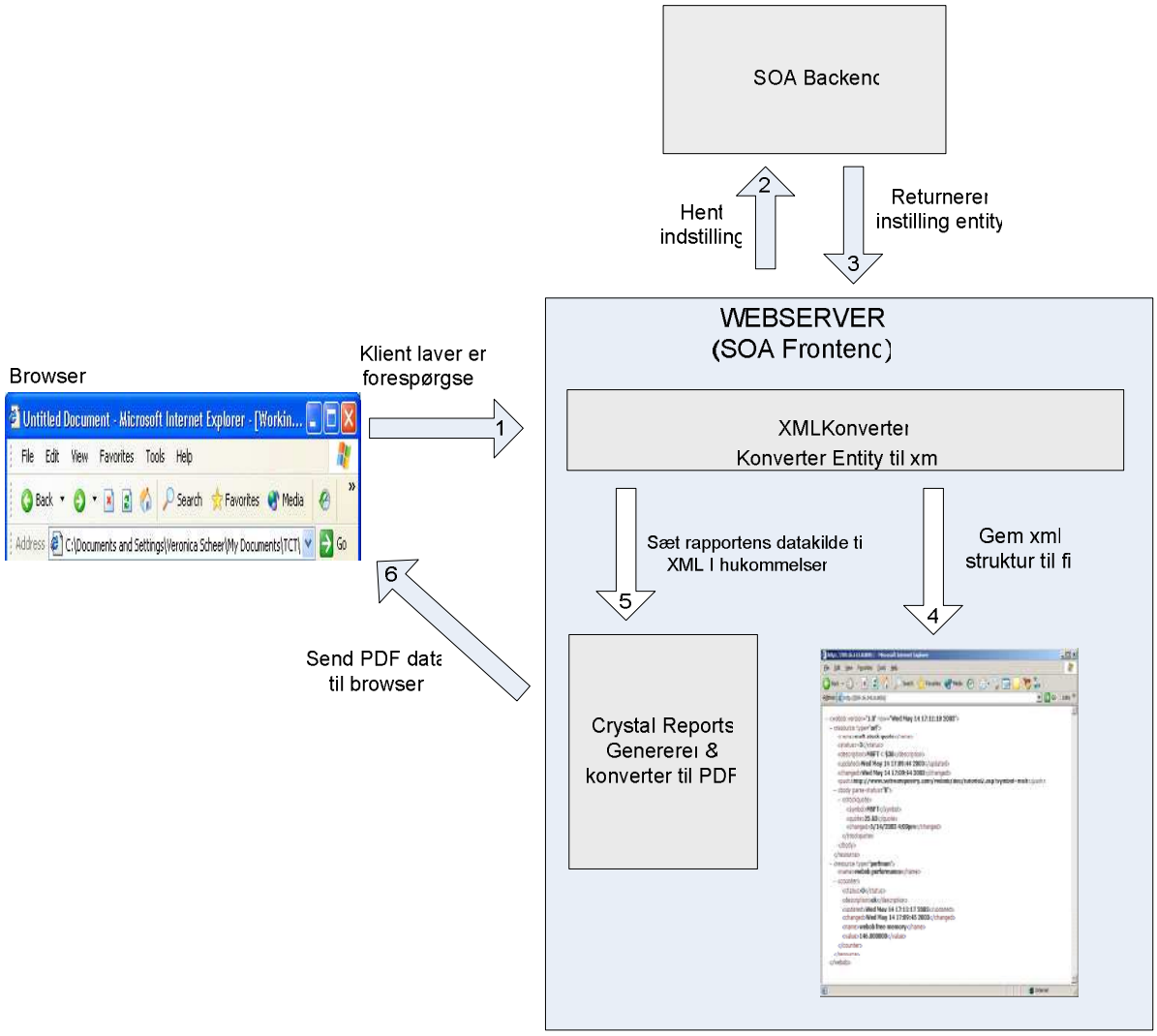
På grund af den grundlæggende forskellige opfattelse af et objekt, kan man påstå at SOA er ikke objekt orienteret. Men det betyder ikke at man ikke kan benytte OO teknikker i et SOA arkitektur. SOA arkitekturen siger intet om hvordan man skal implementere de enkelte forretnings processer. Man kan sagtens anvende traditionelle objekter når man skal designe og implementere forretnings processerne, man kan oprette objekter som både indeholder adfærd og state, og man kan bruge alle de OO teknikker man har lyst til, den eneste krav er at man ikke skal udstille disse objekter.

Når man designer DO systemer (distribueret objekter) bør objekterne udstille et interface, og de bør kun indeholde grovkornede funktioner. Dette minder meget om de principper man skal overholde når man skal designe SOA services. En service skal også udstille et interface og den bør også kun udstille grovkornede funktioner. Der er mange ligheder mellem SOA og DO, men hvis man kigger nærmere på teknikkerne. Den største forskel mellem DO objekter og SOA services er at man i SOA udveksler XML beskeder, mens man i DO udveksler objekt referencer. XML beskeder er både sprog og platform uafhængig, det er objekt referencer ikke.

5 Implementation

5.1 Overordnet system flow

Nedenstående figur illustrerer konceptuelt flow i applikationen.



5.2 Funktions beskrivelse

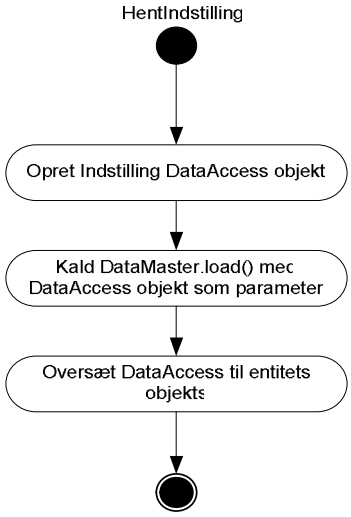
I dette afsnit vil jeg beskrive nogle af de vigtigste funktioner i applikationen.

GemRapport()	Denne funktion bruges når en bruger skal gemme en rapport. Funktionen kalder ned til serviceAgent laget i backenden, og sender indstilling id´et og rapport data som parameter. Rapport data hentes med HentRapport() funktionen.
GetRapportData()	Denne metode bruger Crystal Reports PDF eksport mulighed til at hente rapportens data som byte[].
MakeReport()	Denne metode generer rapporten, alt efter rapport type. I url´en skal man angive indstilling id og rapport type. Jeg har implementeret dette med en case sætning, som bruger rapporttypen til bestemme hvordan rapporten skal sættes sammen. Funktionen returnerer en RapportDokument.
ShowSubReport()	Denne metode bruges til at ”tilføje” eller ”fjerne” en subreport fra rapporten. Subrapporten bliver ikke fjernet fra rapporten siden, men den bliver undertrykt så den ikke vises. Til at starte med er alle rapporterne lukkede. Det er MakeReport() funktionen der bestemmer hvilke subrapporter der skal vises.
KonvertObjectToXML()	Denne metode konverterer et objekt til et xml dokument. For at opnå dette har jeg brugt XMLserializeren i .net frameworket. Denne funktion kalder UpdateDataSource().
BindReportToObject()	Denne metode bruger KonvertObjectToXML
UpdateDataSource()	Denne metode bruges udelukkende i udvikling. Jeg bruger den til at synkronisere Crystal Reports datakilde med de faktiske objekter jeg modtager fra backenden. Metoden skriver til en fil på maskinen og i produktions miljø har applikationen ikke rettigheden til at skrive til serveren.
ConvertObjectToDataset()	Denne funktion modtager et objekt og et dataset. Funktion bruger reflection til at læse objektets felter, og tilføjer en ny datatabel som hedder det samme som klasse navnet på objektet. Kolonnerne i

	datatabellen svarer til felterne i objektet. På nuværende tidspunkt er det kun primitive datatyper som indgår i datatabellen. Man kunne nemt udvide funktionen til også at håndtere objekt referencer og collections. Funktionen kan oprette en ny datatabel for hver objekt reference. I tilfælde af collections kunne man spørge om objektet nedarver fra collectionbase, hvis dette er tilfældet kan man iterere gennem listen, konvertere element objektet til en datarow klasse og tilføje den til et dataTable objekt.

5.3 Implementering af hent metode

Jeg har implementeret en række hent metoder som henter data fra databasen. Alle hent metoderne følger samme skabelon. Først oprettet man et dataaccess objekt, derefter henter man data fra databasen ved at kalde load metoden i DataMaster komponenten og endeligt konverterer man dataaccess objektet til et entitets objekt.



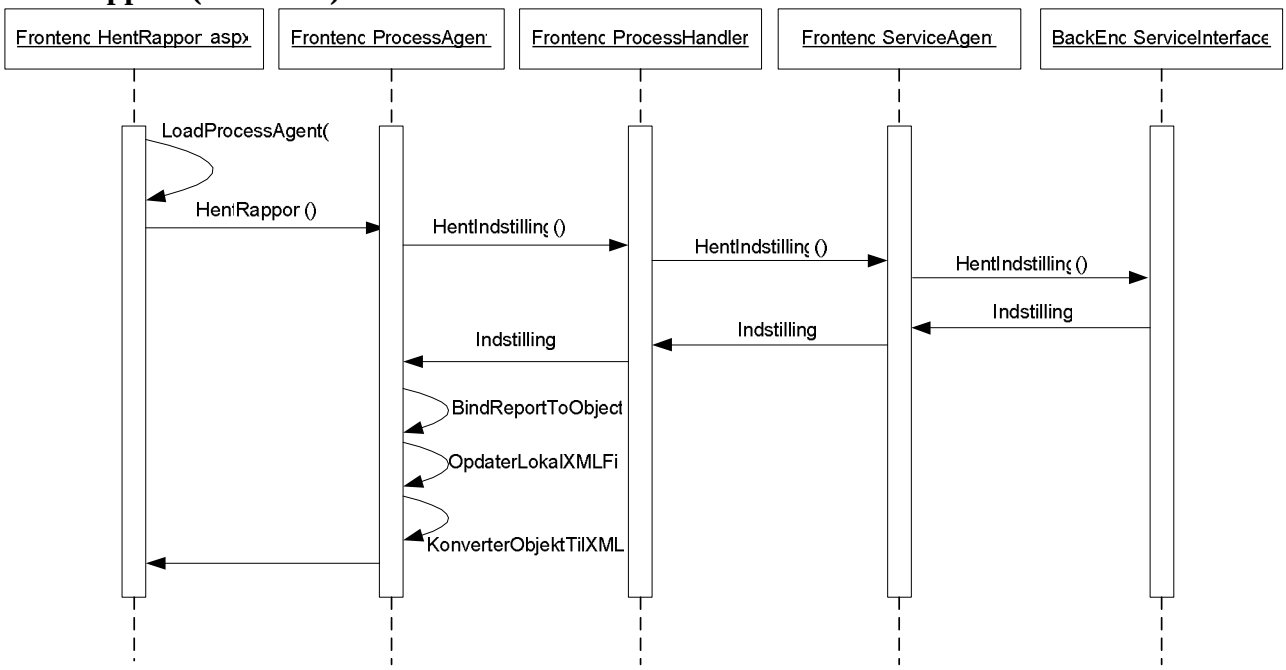
5.3.1 Implementerings eksempel

Jeg opretter en dataAccess klasse og sender indstillingsid´et som parameter, den skal bruges i stored proceduren. DataMaster.Load() kaldet vil fylde dataaccess objektet med data fra databasen. Da funktionen skal returnere et entitets objekt skal jeg konvertere dataaccesobjektet. Inden jeg returnere indstilling objektet skal jeg hente dens underobjekter. Nedenstående kode eksempel er taget fra Backend.Business.Indstilling klassen og den viser hvordan jeg har implementeret hentIndstilling metoden.

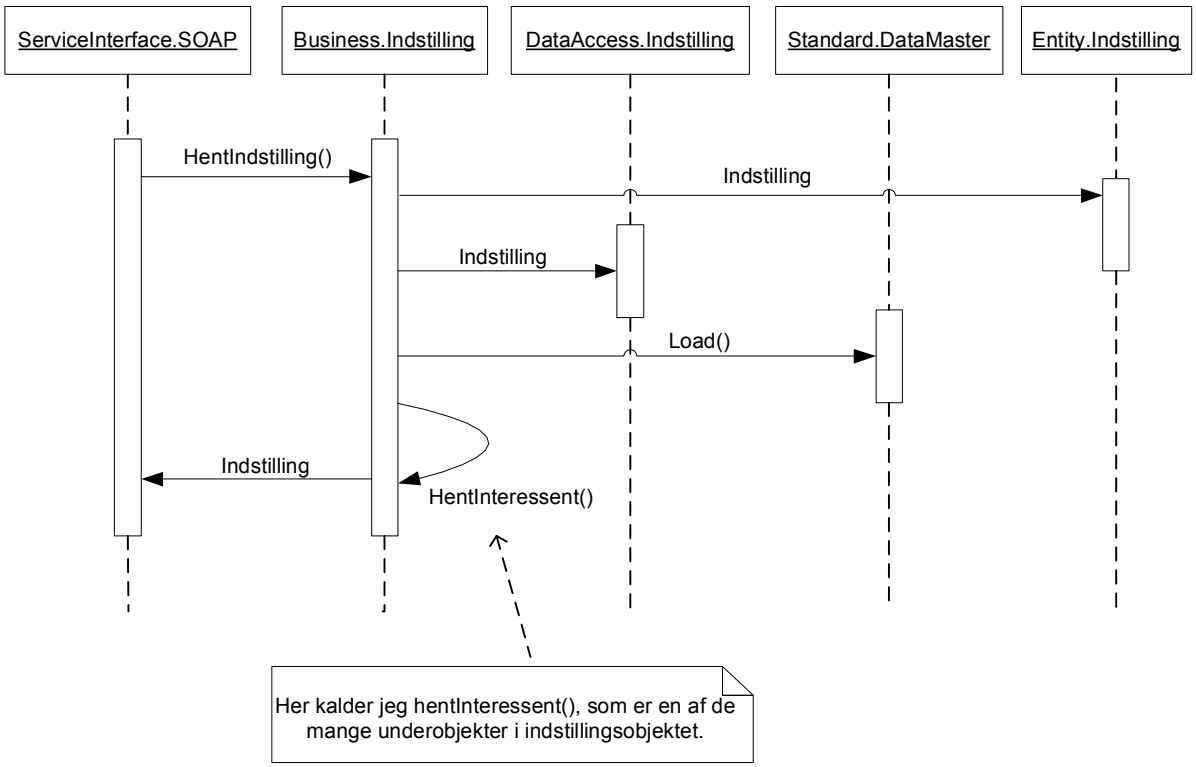
```
public static Entity.Indstilling.Indstilling HentIndstilling(int indstillingId)
{
    DataAccess.Indstilling dataAccessIndstilling = new DataAccess.Indstilling(indstillingId);
    DataMaster.Load(dataAccessIndstilling);
    Entity.Indstilling.Indstilling entityIndstilling = (Entity.Indstilling.Indstilling)
        DataMapper.DataMapToEntity(dataAccessIndstilling, typeof(Entity.Indstilling.Indstilling));

    entityIndstilling.Interessenter = Interessent.HentInteressenter(indstillingId);
    entityIndstilling.Koncern = Koncern.HentKoncern(entityIndstilling.KoncernId);
    entityIndstilling.IndstillingType = Type.HentType(GruppeTypeEnum.IndstillingType, entityIndstilling.IndstillingTypeId);
    entityIndstilling.BevillingCollection = HentBevilling(indstillingId);
    entityIndstilling.PrissaetningArray = HentPrissaetning(indstillingId);
    entityIndstilling.Ejendomme=Ejendom.HentEjendomme(indstillingId);
    return entityIndstilling;
}
```

5.3.2 Sekvens diagrammer

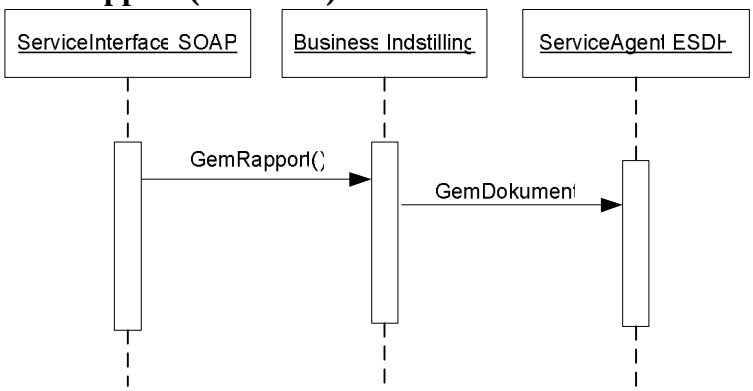
Hent rapport (Frontend):



Hent rapport (BackEnd):



Gem rapport (BackEnd)



5.6 Anvendte patterns

I dette afsnit vil jeg kort beskrive de patterns jeg har brugt i applikationen.

5.6.1 Singleton

Flere steder i applikationen har jeg brugt singleton pattern, blandt andet i proces laget. Proces laget skal have en reference til backend delen af indstillings applikationen, denne referenece/forbindelse skal konfigureres ved at hente information fra en databasen, hvilket er tidskrævende. Ved at bruge singleton pattern sikrer vi at der altid kun findes en service agent objekt i hukommelsen og derfor behøver kun at konfigurere forbindelsen én gang.

Generel problemstilling: Kun at tillade netop én instans af en given klasse.
Løsning: Man kan implementere en singleton ved hjælp af:

- En privat statisk variabel som indeholder en reference til det ønskede singleton objekt.
- En statisk konstruktør som opretter en instans af det ønskede objekt. En statisk konstruktør i C# eksekveres kun når den pågældende klasse bliver instantieret eller når en statisk member bliver refereret.
- En statisk property som udstiller singletonobjektet

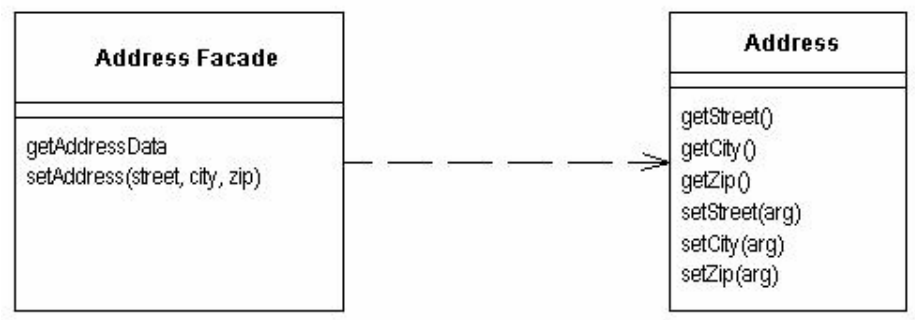
```
using System;
using BRF.Standard.Xception;

namespace BRF.Indstilling.Frontend.Process
{
    internal abstract class ServiceAgentSingleton
    {
        private static IServiceInterface myServiceAgent;

        static ServiceAgentSingleton()
        {
            myServiceAgent = new Frontend.ServiceAgent.Soap.ServiceAgent();
        }

        internal static IServiceInterface ServiceAgent
        {
            get
            {
                return myServiceAgent;
            }
        }
    }
}
```

Figur 5 Implementering af singleton pattern

Denne klasse er markeret som abstrakt, dette betyder at den ikke kan instantieres og derfor kan der heller ikke forekomme flere instanser af denne klasse. Klassen og property feltet er markeret som internal, dette betyder blot at denne klasse kun kan tilgås af andre klasser i samme assembly.

5.6.3 Facade / Remote facade

Grundprincippet i facade og remote facade pattern er at afskærme og simplificere klientens tilgang til en række objekter. Man kan implementere et facade pattern ved at indføre et facade objekt mellem klienten og det objekt system som klienten forsøger at tilgå. Facade objektet fungerer som et interface, som kun udstiller de relevante funktioner. Hvis der findes en fysisk grænseflade mellem klienten og objektsystemet som klienten forsøger at tilgå, bør man bruge Remote pattern. Grundprincippet bag denne facade er at man skal designe grovkornede interfaces, dette betyder at man skal designe få funktioner, som returnerer større datamængder. Hvis man designer mange finkornede funktioner, skal man udføre flere kald til backenden hvilket vil resultere i højere netværkstrafik. Jeg bruger dette pattern til at opfylde SOA princip 1 i afsnit 4.2.2.
Denne pattern er brugt i service interface og serviceagent laget. I service interfacet udstiller jeg en metode som henter en indstilling fra databasen. Indstillings objekt indeholder flere underobjekter som hver har deres egen hent metode. Men fremfor at udstille disse hent metoder i underobjekterne udstiller jeg en generel hent metode som returnerer et stort indstillings objekt. Nedenstående figur illusterer dette.



5.6.4 Proxy pattern

Denne pattern kan bruges til at generere lokale klasser som repræsenterer objekter i et remote system. I et service orienteret arkitektur betyder det at man opretter proxy klasser i service aftageren. .NET frameworket genererer disse klasser ud fra WSDL beskrivelsen. Når en service aftager modtager et SOAP svar fra en service udbyder, vil .net frameworket deserialisere xml

beskeden og oprette de nødvendige proxy objekter. Disse proxy objekter skal konverteres til entitets objekter inden de returneres.

5.7 Implementerings problemer

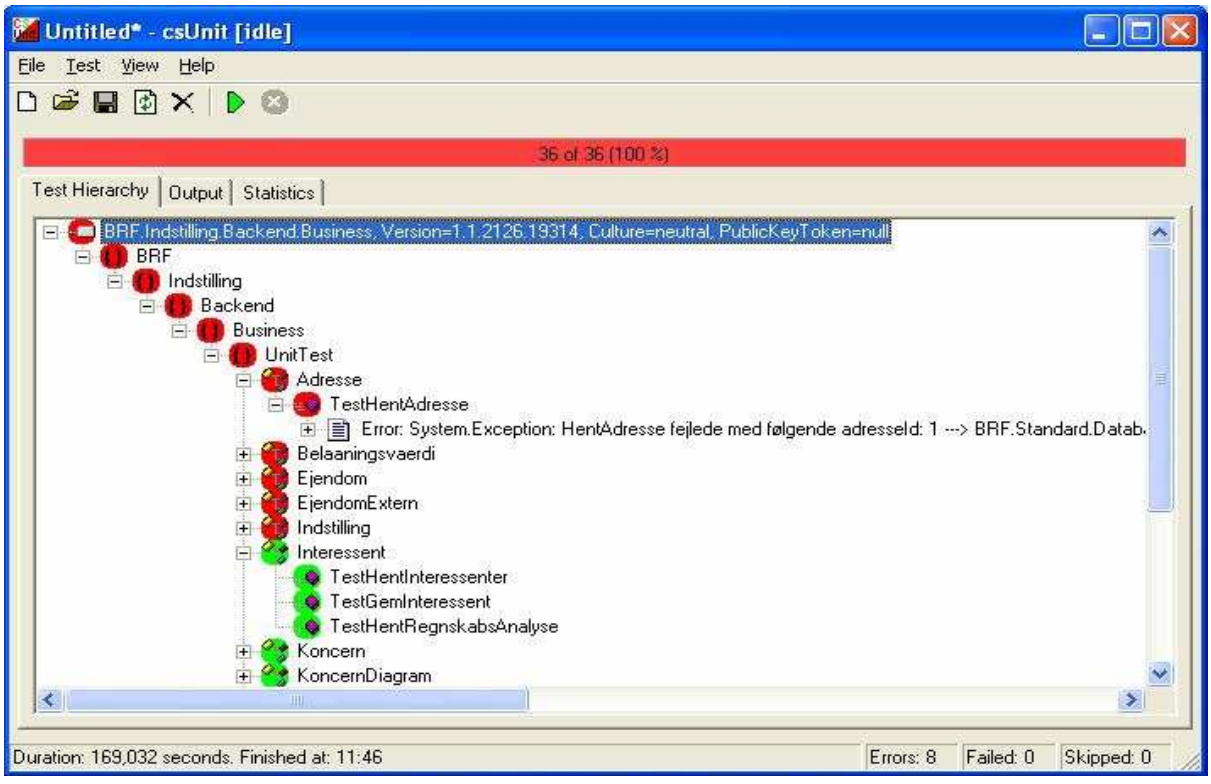
Når man bruger XML bindinger mister man information om datatyperne i Crystal Reports. Dette betyder at alle kolonnerne i det resulterende dataset er af typen string. Det har vist sig at det er meget besværligt at konvertere typerne i Crystal Report. Grunden til at jeg gerne vil have typer på felterne er at Crystal Reports har en række nyttige funktioner såsom konvertering og sortering af data. Når man skal sortere en tabel skal man angive et felt, og fortælle om den skal sorteres stigende eller faldende. Dette kræver selvfølgelig at feltet som skal sorteres skal være numeriske, hvilket ikke er tilfældet med XML bindinger. Hvad angår konvertering har det vist sig at alle dato typerne og flere decimal værdier har forkert format. I databasen er dato felter oprettet som DateTime felter, og når man konverterer et dato felt til XML vil feltet bliver konverteret som long date format, som indeholder både data og tid. Hvis feltet havde været af typen DateTime ville man i Crystal Reports vælge formatet af dato, og kun vise dato uden tid.

6 Test

6.1 Unit test

Ideen med unit testing er at kontrollere enkelte komponenter eller metoder. Jeg har løbende lavet unit testing. Fordelen med denne type test er at jeg kan teste flere funktionerHent indstilling metoden indeholder mange objekter som hver især indeholder en hent metode, med unit test kan man nemt og hurtigt kontrollere om alle metoderne fungerer.

For at bruge csUnit test skal man oprette en række test klasser og test funktioner. Test klasser skal markeres med [csUnit.TestFixture()]og test funktioner skal markeres med [csUnit.Test()] attributtet. En test funktion er en simpel funktion som kalder den funktion man ønsker at teste. For at vurdere om testen lykkedes eller om den fejlede skal man opstille en række betingelser. De mest anvendte er csUnit.Assert.NotNull(værdi) og csUnit.Assert.Greater(værdi, 0). Den først nævnte kontrollerer om den angivne parameter er forskellig fra null mens den sidste kontrollerer om det første parameter er større end det andet. Nedenstående skærmbillede illustrerer test resultatet af unit testen, disse fejl rettes løbende.



Figur 6 Skærmbilelde af csUnit.

6.2 Use case test

6.2.1 Gem rapport

TESTPLAN Gem rapport				
Testtype: Unit test ☑ Use-case test ☑ Installations test	**Dato:** 25/10 2005	**Udført af:** Yusuf Karabulut	**Test-plan ID.:** §01	**Test-plan Version:** 1.0
Test-formål: Denne test går ud på at teste gem rapport funktionen som er beskrevet i use casen ”Gem rapport” under afsnit 3.2.2				
Overordnet testbeskrivelse: Indstillings applikationen startes og brugeren opretter en indstilling.				
Test-forudsætninger: Ingen.				
Specifikke test-cases: o Indstilling applikationen startes o Brugeren opretter en ny rapport og udfylder eventuelle obligatoriske felter. o Brugeren trykker på gem rapport. o Åbn ESDH brugergrænsefladen fra browseren o Søg og åbn den oprettede rapport.				
Tilladte afvigelser / fejl: Ingen				
Korrigerende handlinger ved fundne fejl: Kilde koden kigges igennem, fejl findes, og testes igen				

TESTRAPPORT

Gem rapport

Testtype:	**Dato:**	**Udført af:**	**Anvendt test-plan:**	**Test-rapport ID:**
Unit test ☑	25/10 2005	Yusuf Karabulut	§01	§01
Use-case test ☑			**Version:**	**Version:**
Installations test			1.0	1.0

Test-resultater:

Nedenstående screenshot viser ESDH´s søge resultat. Som det fremgår bliver rapporten gemt i ESDH.

Fejl / afvigelser:

Ingen

Test-konklusion:

Som det fremgår af figur x bliver dokumentet oprettet i ESDH. Hermed kan det konkluderes at gem rapport virker efter hensigten.

6.2.2 Vis rapport

TESTPLAN

Vis rapport

Testtype:		**Dato:**	**Udført af:**	**Test-plan ID.:**	**Test-plan Version:**
Unit test	☑	25/10 2005	Yusuf Karabulut	§02	1.0
Use-case test	☑				
Installations test					

Test-formål:

Denne test går ud på at teste vis rapport funktionen som er beskrevet i use casen vis rapport under afnit 3.3.3

Overordnet testbeskrivelse:

Åbn et browser vindue og indtast adressen hvor applikationen er hostet. I adressen skal man sende to parametre, den ene er rapport typen og den anden er rapports id.

Test-forudsætninger:

At den indstilling man forsøger findes i databasen.

Specifikke test-cases:

- Åbn en IE browser
- http://localhost/BRFIndstillingFrontendWebForm/HentRapport.aspx?RapportId=71&RapportType=NyKreditIndstilling
- Kontroller at data i den generede rapport stemmer med data i dabasen.

Tilladte afvigelser / fejl:

Ingen

Korrigerende handlinger ved fundne fejl:

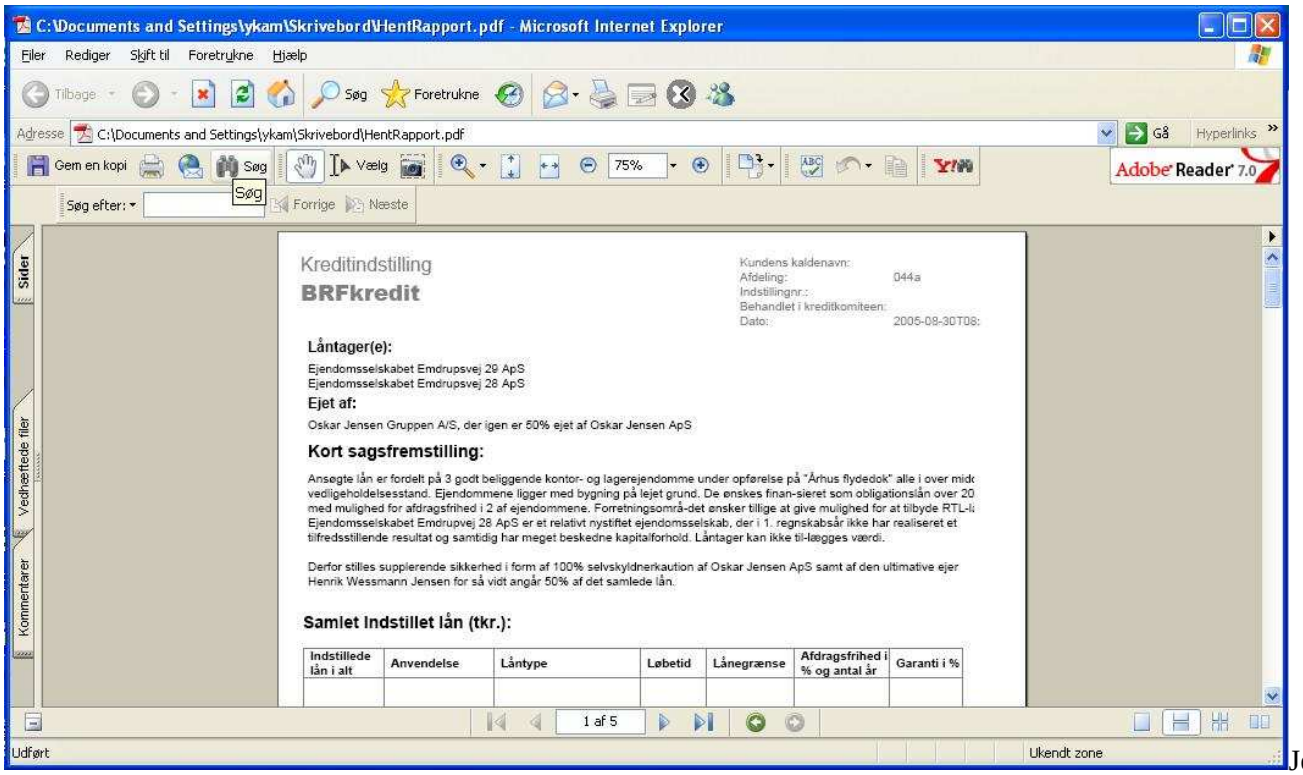
Kilde koden kigges igennem, fejl findes, og testes igen

TESTRAPPORT

Vis rapport

Testtype:		**Dato:**	**Udført af:**	**Anvendt test-plan:**	**Test-rapport ID:**
Unit test	☑	25/10 2005	Yusuf Karabulut	§02	§02
Use-case test	☑			**Version:**	**Version:**
Installations test				1.0	1.0

Test-resultater:



Jeg har vedlagt den genererede rapport under bilag i afsnit 8.2.

Fejl / afvigelser:

Ingen

Test-konklusion:

Som det fremgår af ovenstående skærmbillede bliver rapports data hentet og en pdf rapport bliver genereret og vist på browseren.

7 Konklusion

Som beskrevet i problemformuleringen harGenerere dynamiske rapporter I den forbindelse har jeg lavet en analyse. Meget tidligt i analyse fasen gik det op for mig at det eneste rigtige alternativ til Crystal Reports var ASP.NET. Jeg har sammenlignet ASP.NET med Crystal Reports ud fra en række kriterier, såsom fleksibilitet, SOA kompatabilitet og udviklingstid. På baggrund af denne analyse har jeg valgt at implementere rapporterne med Crystal Reports. Men da jeg ikke har erfaring med Crystal Reports har det været nødvendigt at undersøge værktøjet og de løsningsmodeller som den stiller til rådighed. Når man anvender Crystal Reports kan man vælge mellem flere forskellige modeller, se afsnit xx. Jeg har valgt den løsningsmodel som passer bedst til BFRkredits arkitektur. I den forbindelse har jeg undersøgt principperne bag den service orienteret arkitektur. Endeligt har jeg implementeret en af rapporterne med den valgte løsningsmodel. Det skal dog siges at jeg har haft en del problemer med imlementeringen af rapporten. Det har vist sig at den løsning jeg har valgt ikke er særlig hensigtsmæssig, fordi den begrænser mange af Crystal Reports nyttige funktioner såsom sortering og konverterering af data. Jeg har beskrevet problemet i detaljer under afsnit 5.5.

Det er lykkedes mig at designe og implementere en applikation som kan generere dynamiske rapporter i en service orienteret arkitektur. Under bilag har jeg vedlagt både en genereret rapport og brugernes oplæg, som det fremgår har jeg valgt at holde mig tæt på brugernes oplæg.
Hvis jeg havde valgt XML skema bindinger fremfor normal XML ville jeg have undgået mange af implementations problemer som jeg har stødt på. Trods disse problemer er det lykkedes mig at besvare spørgsmålene i problem formuleringen og implementere en applikation som kan generere dynamiske rapporter i et service orienteret arkitektur.

7.1 Perspektivering

Hvis jeg havde mere tid og flere ressourcer ville lave om på den nuværende implementation af rapporterne, jeg ville implementere XML skema bindinger. En anden god feature ville være at implementere kommentar og text felter som Rich Text Format. Dette vil tillade brugerne at kopierer og indsætte dele af word dokumenter i rapporter. RTF format kan indeholde tabeller og punktopstillinger og text formattering, dette vil give rapporterne et mere professionelt udseende.

8 Bilag

8.3 Referencer

Bøger:
Crystal Reports .NET Programming af Brian Bischof
Application Architecture for .NET: Designing Applications and Services

Crystal Reports artikler:
http://www.dotnetjunkies.com/Article/790775A0-C493-46D8-ABE0-40CA588D33D3.dcik
http://www.codeproject.com/aspnet/crystal_report.asp
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/crystlmn/html/crconreportprocessingmodel.asp

SOA artikler:
http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html
http://msdn.microsoft.com/architecture/soa/default.aspx?pull=/library/en-us/dnmaj/html/aj1soa.asp
http://informationweek.webservicespipeline.com/57704023
http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnbda/html/soadesign.asp
http://www.xml.com/pub/a/2002/02/20/endpoints.html
http://hr.svalevej.dk/95artikler/SOA_forklaret.asp

8.4 Kildekode

Jeg har vedlagt nogle af de vigtigste filer. Det skal dog bemærkes at i Business.indstilling har lavet hent og unitest delen, de andre funktioner i denne klasse er lavet af andre personer i projektet. Jeg har ikke alle lagene med.

8.4.1 BRF.Indstilling.FrontEnd.Webform ProcessAgent.cs

```
using System;
using System.IO;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Data;
using System.Configuration;
using BRF.Standard.Xception;
using BRF.Entity.Indstilling;
using BRF.Indstilling.Frontend.WebForm;
using CrystalDecisions.CrystalReports.Engine;
using System.Collections;
using System.Reflection;
using System.Text;
using System.Xml;
using System.Xml.Serialization;

namespace BRF.Indstilling.Frontend.WebForm
{
    /// <summary>
```

```
    /// ProcessAgent klassen sørger for at styre processen.
    /// Herunder at kalde de rigtige SubRapporter til vores rapportside.
    /// </summary>
    public class ProcessAgent
    {
     #region Declarations

     private static Frontend.Process.ProcessHandler myProcessHandler;
     private int myRapportId;
     private string myRapportType;

     #endregion

     #region Constructors

     /// <summary>
     /// Konstruktøren for klassen som sørger for at der startes en ProcessHandler op.
     /// </summary>
     public ProcessAgent()
     {
      myProcessHandler = new Process.ProcessHandler(Process.BusinessContextEnum.Standalone);
     }

     #endregion

     #region Public methods

     /// <summary>
     /// Denne metode sørger for at gemme rapporten.
     /// </summary>
     public void GemRapport(int indstillingId)
     {
      //TODO: Her skal udvides sådan at indstillingen bliver sendt med til backend.
      ProcessHandler.GemIndstilling(indstillingId);
     }

     /// <summary>
     /// Denne metode sørger for at hente rapportdata ud af et rapportdokument
     /// og returnere disse data som et array af bytes.
     /// </summary>
     /// <param name="reportDocument"></param>
     /// <returns>byte[] Rapportdata</returns>
     public byte[] GetReportData(ReportDocument reportDocument)
     {
      byte[] dataBytes;

      try
      {
       //Her laves en ny data Stream ud af rapportdokumentet.
       System.IO.Stream rptStream =
reportDocument.ExportToStream(CrystalDecisions.Shared.ExportFormatType.PortableDocFormat);
       reportDocument.Close();
       //Her laves en ny binær stream reader med data fra rapportdokumentet.
       System.IO.BinaryReader br = new System.IO.BinaryReader(rptStream);
       dataBytes = new byte[rptStream.Length];
       //Her laves et array af data bytes som vi udfylder med data fra rapportdokumentet
       for(int i=0; i<rptStream.Length; ++i)
                dataBytes[i] = br.ReadByte();
```

```
        }
        catch(Exception exception)
        {
         throw new Exception("Kunne ikke konvertere Rapport dokument til array af Bytes.", exception);
        }

        return dataBytes;
       }

       /// <summary>
       /// Denne metode henter en rapport ud fra et i URL'en angivet RapportId og RapportType.
       /// Og returnerer et ReportDocument.
       /// </summary>
       /// <returns>ReportDocument</returns>
       public ReportDocument HentRapport()
       {
        try
        {
         //Her hentes variable fra URL'en.
         myRapportId = int.Parse(HttpContext.Current.Request.QueryString["RapportId"]);
         myRapportType = HttpContext.Current.Request.QueryString["RapportType"];

         //Her instantieres en ny Crystal Report : Fremtidig Indstilling
         CrystalReport.Reports.FremtidigIndstilling fremtidigIndstilling = new
BRF.Indstilling.Frontend.WebForm.CrystalReport.Reports.FremtidigIndstilling();

         //Her hentes indstillingen med det pågældende Id fra Frontend.Process laget.
         Entity.Indstilling.Indstilling indstilling = ProcessHandler.HentIndstillingTest(myRapportId);

         //Baseret på hvilken rapporttype brugeren har valgt at se, vælges herunder de aktuelle SubReports for den type.
         switch(myRapportType)
         {
                  case "NyKreditIndstilling":
                                    ShowSubLaanTager(fremtidigIndstilling, indstilling);
                                    ShowSubEjer(fremtidigIndstilling, indstilling);
                                    ShowSubKortSagsFremstilling(fremtidigIndstilling, indstilling);
                                    ShowSubIndstilledeLaan(fremtidigIndstilling, indstilling);
                                    ShowSubSamletEngagement(fremtidigIndstilling, indstilling);
                                    ShowSubEngagementUdvikling(fremtidigIndstilling, indstilling);
                                    ShowSubIndstillingKommentar(fremtidigIndstilling, indstilling);
                                    ShowSubEjendomme(fremtidigIndstilling, indstilling);
                                    ShowSubLejerKommentar(fremtidigIndstilling, indstilling);
                                    ShowSubSupplerendeSikkerhed(fremtidigIndstilling, indstilling);
                                    ShowSubPrissaetning(fremtidigIndstilling, indstilling);
                                    ShowSubKonklusion(fremtidigIndstilling, indstilling);
                                    ShowSubBevillingKreditKontor(fremtidigIndstilling, indstilling);
                                    ShowSubBevillingKreditKomite(fremtidigIndstilling, indstilling);
                                    ShowSubKoncernDiagram(fremtidigIndstilling, indstilling);
                                    ShowSubkautionist(fremtidigIndstilling, indstilling);
                                    ShowSubEjendomSkemaNyInds(fremtidigIndstilling, indstilling);
                                    break;
                  default:
                                    return null;
         }

         //Bind data til den fremtidige indstilling.
         BindReportToObject(fremtidigIndstilling, indstilling);
```

```
        return fremtidigIndstilling;
       }
       catch(System.FormatException formatexception)
       {
        throw new Exception("Det angivne RapportId er ikke korrekt. RapportId´et skal være et heltal.",
formatexception);
       }
       catch(Exception exception)
       {
        throw new Exception("HentRapport fejlede med RapportId: "+myRapportId , exception);
       }
      }

      #endregion

      #region Private methods

      /// <summary>
      /// Denne metode skifter farven på alle tabeller med den angivne endelse.
      /// </summary>
      /// <param name="objectEndingWith"></param>
      /// <param name="reportObjects"></param>
      private void GetObjectsEndingWith(string objectEndingWith, ReportObjects reportObjects)
      {
       foreach (ReportObject reportObject in reportObjects)
        if(reportObject.Name.EndsWith(objectEndingWith))
                reportObject.Border.BorderColor = System.Drawing.Color.Gray;
      }

      /// <summary>
      /// Denne metode finder subRapporten udfra Rapport dokumentet og sætter EnableSuppress værdien derefter.
      /// </summary>
      /// <param name="subReportName"></param>
      /// <param name="report"></param>
      /// <param name="supress"></param>
      private void ShowSubReport(string subReportName, ReportDocument report, bool supress)
      {
       try
       {
        SubreportObject subreportObject = (SubreportObject)report.ReportDefinition.ReportObjects[subReportName];
        if(supress)
                subreportObject.ObjectFormat.EnableSuppress = true;
        else
                subreportObject.ObjectFormat.EnableSuppress = false;
       }
       catch(Exception exception)
       {
        throw new Exception("Kunne ikke finde subRapport "+subReportName, exception);
       }
      }

      /// <summary>
      /// Binder en rapport til et object. Objektet konverteres først til XML og derefter
      /// opretter vi et dataset som vi bruger til at binde med rapporten.
      /// </summary>
      /// <param name="targetReport"></param>
      /// <param name="sourceObject"></param>
      private static void BindReportToObject(ReportDocument targetReport, object sourceObject)
```

```
    {
     try
     {
      DataSet dataSet = new DataSet();
      dataSet.ReadXml(new StringReader(KonverterObjektTilXML(sourceObject, true)));
      targetReport.SetDataSource(dataSet);
     }
     catch(Exception exception)
     {
      throw new Exception("Kunne ikke binde "+targetReport.Name+" rapporten med "+sourceObject+" objektet",
exception);
     }
    }

    /// <summary>
    /// Denne metode konverterer et objekt til xml-streng vha XMLSerializer
    /// </summary>
    /// <param name="sourceObject"></param>
    /// <param name="updateFile">Flag som indikerer om vi skal opdatere lokal fil</param>
    /// <returns></returns>
    private static string KonverterObjektTilXML(object sourceObject, bool updateFile)
    {
     try
     {
      XmlSerializer xmls = new XmlSerializer(sourceObject.GetType());
      StringBuilder sb = new StringBuilder();
      StringWriter sw = new StringWriter(sb);
      XmlTextWriter xmltw = new XmlTextWriter(sw);
      //Formaterer XML strengen.
      xmltw.Indentation = 4;
      xmltw.Formatting = Formatting.Indented;
      //Serialiserer objektet.
      xmls.Serialize(xmltw, sourceObject);
      if(updateFile)
                  //Opdaterer filen i lokationen fundet i SettingsDatabasen.
                  OpdaterLokalXMLFil(sb.ToString(),ConfigurationSettings.AppSettings["XMLFileLocation"]);

      return sb.ToString();
     }
     catch(Exception exception)
     {
      throw new Exception("KonverterObjektTilXML fejlede med følgende objekt :"+sourceObject.GetType().Name,
exception);
     }
    }

    /// <summary>
    /// Denne metode skriver en xml string til en fil.
    /// Bruges når vi skal opdatere vores xml fil som vi har bundet op imod i design time.
    /// Når vi bruger drag & drop til at få felter ind i vores rapport, bruger vi xml filen som datasource.
    /// I runtime giver vi Crystal Reports den samme xml-dokument, blot med data i.
    /// </summary>
    /// <param name="XMLString"></param>
    /// <param name="XMLFilePath"></param>
    private static void OpdaterLokalXMLFil(string XMLString, string XMLFilePath)
    {
     try
     {
```

```
        //Her laver vi en ny fil som vi streamer vores data ned i.
        FileStream file = new FileStream(XMLFilePath, FileMode.Create, FileAccess.Write);
        StreamWriter sw = new StreamWriter(file, Encoding.UTF8);
        sw.Write(XMLString);
        sw.Close();
       }
       catch(Exception exception)
       {
        throw new Exception("OpdaterLokalXMLFil fejlede. Forsøgte at skrive til:\n\r "+XMLFilePath+"\n\rXML
String:\n\r "+XMLString, exception);
       }
      }
      /// <summary>
      /// Denne metode konverterer et objekt til et dataset. Objektet må ikke indeholde referencer til andre objekter.
      /// Kun kendte datatyper.
      /// </summary>
      /// <param name="sourceObject">Objekt som skal konverteres til dataset</param>
      /// <returns></returns>
      private static DataSet ConvertObjectToDataSet(object sourceObject, DataSet dataSet)
      {
       //Her findes typen på objektet.
       System.Type type = sourceObject.GetType();
       ArrayList values = new ArrayList();
       //For alle properties på vores objekts type tilføjer vi værdien i vores values arraylist.
       foreach(PropertyInfo prop in type.GetProperties(BindingFlags.Public|BindingFlags.Instance))
       {
        if(!(prop.Name.Equals("IsNew")||prop.Name.Equals("IsDirty")||prop.Name.Equals("IsDeleted")))
        {
                    object result = type.InvokeMember(prop.Name, BindingFlags.Public | BindingFlags.Instance |
BindingFlags.GetProperty, null, sourceObject,null);
                    values.Add(result);
        }
       }
       //Her indsættes rækken fra vores values arralist i tabellen.
       dataSet.Tables[0].Rows.Add((object[])values.ToArray((typeof(object))));

       return dataSet;
      }

      #endregion

      #region Properties

      public static Process.ProcessHandler ProcessHandler
      {
       get
       {
        return myProcessHandler;
       }
      }

      #endregion

      #region Crystal Reports

       #region SubLaanTager

       /// <summary>
```

```
/// Denne metode undersøger om der er interessenter på indstillingen, og viser derefter Sub Rapporten.
/// </summary>
/// <param name="reportDocument"></param>
/// <param name="indstilling"></param>
private void ShowSubLaanTager(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
{
 try
 {
		if(indstilling.Interessenter!=null && indstilling.Interessenter.Count > 0)
		{
				ShowSubReport("SubLaanTager", reportDocument, false);
		}
 }
 catch{}
}

#endregion

#region SubEjer

/// <summary>
/// Denne metode undersøger om det er en interessenttype vi vil vise, og viser derefter Sub Rapporten.
/// </summary>
/// <param name="reportDocument"></param>
/// <param name="indstilling"></param>
private void ShowSubEjer(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
{
 try
 {
		if(indstilling.Interessenter != null && indstilling.Interessenter.Count > 0)
		{
				foreach(Entity.Indstilling.Interessent interessent in indstilling.Interessenter)
				{
						if(interessent.InteressentTypeId == 16)
						{
								ShowSubReport("SubEjer", reportDocument, false);
						}
				}
		}
 }
 catch{}
}

#endregion

#region SubKortSagsFremstilling

/// <summary>
/// Denne metode undersøger om der findes en KortSagsFremstilling på indstillingen, og viser derefter Sub
Rapporten.
/// </summary>
/// <param name="reportDocument"></param>
/// <param name="indstilling"></param>
private void ShowSubKortSagsFremstilling(ReportDocument reportDocument, Entity.Indstilling.Indstilling
indstilling)
{
 try
 {
```

```
                if(indstilling.KortSagsFremstilling != null && !indstilling.KortSagsFremstilling.Equals(""))
                {
                        ShowSubReport("SubKortSagsFremstilling", reportDocument, false);
                }
        }
        catch{}
        }

        #endregion

        #region SubIndstilledeLaan

        /// <summary>
        /// Denne metode undersøger om der findes nogle ejendomme på indstillingen, og viser derefter Sub Rapporten.
        /// </summary>
        /// <param name="reportDocument"></param>
        /// <param name="indstilling"></param>
        private void ShowSubIndstilledeLaan(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
        {
        try
        {
                if(indstilling.Ejendomme!=null && indstilling.Ejendomme.Count > 0)
                {
                        foreach(Ejendom ejendom in indstilling.Ejendomme)
                        {
                                if(ejendom.LaanOversigt.LaanCollection.Count != 0)
                                {
                                        ShowSubReport("SubIndstilledeLaan",
reportDocument, false);
                                }
                        }
                }
        }
        catch{}
        }

        #endregion


        #region SubEngagementUdvikling

        /// <summary>
        /// Denne metode undersøger om der er noget i EngagementUdvikling for indstillingen, og viser derefter Sub
Rapporten.
        /// </summary>
        /// <param name="reportDocument"></param>
        /// <param name="indstilling"></param>
        private void ShowSubEngagementUdvikling(CrystalReport.Reports.FremtidigIndstilling reportDocument,
Entity.Indstilling.Indstilling indstilling)
        {
        try
        {
                if(indstilling.Koncern.EngagementUdviklingCollection != null &&
indstilling.Koncern.EngagementUdviklingCollection.Count > 0)
                {
                        ReportDocument subEngagementUdvikling = new ReportDocument();
                        //Her oprettes et nyt Engagements objekt på baggrund af rapport dokumentet.
```

```
                                SubreportObject ObjectEngagementUdvikling =
(SubreportObject)reportDocument.ReportDefinition.ReportObjects["SubEngagementUdvikling"];
                                //Her åbnes vores sub rapport.
                                subEngagementUdvikling =
reportDocument.OpenSubreport(ObjectEngagementUdvikling.SubreportName);
                                //Her instantierer vi vores engagementudviklingsobjekter på baggrund af sub rapporten.
                                ReportObjects engagementUdviklingObjekter =
subEngagementUdvikling.ReportDefinition.ReportObjects;

                                //Her gennemløbes vores engagementudviklingsobjekter, hvor dem som ikke
indeholder data bliver udelukket med farveskift.
                                for(int i = 1; i < indstilling.Koncern.EngagementUdviklingCollection.Count+1; i++)
                                {
                                        foreach (ReportObject reportObject in engagementUdviklingObjekter)
                                        {
                                                if(reportObject.Name.EndsWith("No"+i))
                                                {
                                                        reportObject.Border.BorderColor =
System.Drawing.Color.Gray;
                                                }
                                        }
                                }
                                ShowSubReport("SubEngagementUdvikling", reportDocument, false);
                        }
                }
                catch{}
        }

        #endregion

        #region SubIndstillingKommentar

        /// <summary>
        /// Denne metode undersøger om der er en kommentar på indstillingen, og viser derefter Sub Rapporten.
        /// </summary>
        /// <param name="reportDocument"></param>
        /// <param name="indstilling"></param>
        private void ShowSubIndstillingKommentar(ReportDocument reportDocument, Entity.Indstilling.Indstilling
indstilling)
        {
                try
                {
                        if(indstilling.IndstillingKommentar != null && !indstilling.IndstillingKommentar.Equals(""))
                        {
                                ShowSubReport("SubIndstillingKommentar", reportDocument, false);
                        }
                }
                catch{}
        }

        #endregion

        #region SubEjendomme

        /// <summary>
        /// Denne metode undersøger om der er ejendomme på indstillingen, og viser derefter Sub Rapporten.
        /// </summary>
        /// <param name="reportDocument"></param>
```

```
        /// <param name="indstilling"></param>
        private void ShowSubEjendomme(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
        {
         try
         {
                        if(indstilling.Ejendomme!=null && indstilling.Ejendomme.Count > 0)
                        {
                                ShowSubReport("SubEjendomme", reportDocument, false);
                        }
         }
         catch{}
        }

        #endregion

        #region SubLejerKommentar

        /// <summary>
        /// Denne metode undersøger om der er lejerkommentarer på indstillingen, og viser derefter Sub Rapporten.
        /// </summary>
        /// <param name="reportDocument"></param>
        /// <param name="indstilling"></param>
        private void ShowSubLejerKommentar(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
        {
         try
         {
                        if(indstilling.LejerKommentar != null && !indstilling.LejerKommentar.Equals(""))
                        {
                                ShowSubReport("SubLejerKommentar", reportDocument, false);
                        }
         }
         catch{}
        }

        #endregion

        #region SubSupplerendeSikkerhed

        /// <summary>
        /// Denne metode undersøger om der er supplerende sikkerhed på indstillingen, og viser derefter Sub Rapporten.
        /// </summary>
        /// <param name="reportDocument"></param>
        /// <param name="indstilling"></param>
        private void ShowSubSupplerendeSikkerhed(ReportDocument reportDocument, Entity.Indstilling.Indstilling
indstilling)
        {
         try
         {
                        if(indstilling.SupplerendeSikkerhed != null && !indstilling.SupplerendeSikkerhed.Equals(""))
                        {
                                ShowSubReport("SubSupplerendeSikkerhed", reportDocument, false);
                        }
         }
         catch{}
        }

        #endregion
```

```
#region SubPrissaetning

/// <summary>
/// Denne metode undersøger om der er prissætning på indstillingen, og viser derefter Sub Rapporten.
/// </summary>
/// <param name="reportDocument"></param>
/// <param name="indstilling"></param>
private void ShowSubPrissaetning(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
{
 try
 {
		if(indstilling.PrissaetningArray != null && indstilling.PrissaetningArray.Length!=0)
		{
				ShowSubReport("SubPrissaetning", reportDocument, false);
		}
 }
 catch{}
}

#endregion

#region SubKonklusion

/// <summary>
/// Denne metode undersøger om der er konklusion på indstillingen, og viser derefter Sub Rapporten.
/// </summary>
/// <param name="reportDocument"></param>
/// <param name="indstilling"></param>
private void ShowSubKonklusion(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
{
 try
 {
		if(indstilling.Konklusion != null && !indstilling.Konklusion.Equals(""))
		{
				ShowSubReport("SubKonklusion", reportDocument, false);
		}
 }
 catch{}
}

#endregion

#region SubBevillingKreditKontor

/// <summary>
/// Denne metode undersøger om der er bevilling på indstillingen, og viser derefter Sub Rapporten.
/// </summary>
/// <param name="reportDocument"></param>
/// <param name="indstilling"></param>
private void ShowSubBevillingKreditKontor(ReportDocument reportDocument, Entity.Indstilling.Indstilling
indstilling)
{
 try
 {
		if(indstilling.BevillingCollection != null && indstilling.BevillingCollection.Count > 0)
		{
				foreach(Bevilling bevilling in indstilling.BevillingCollection)
						if(bevilling.BevilgerTypeId == 25)
```

```
                                                ShowSubReport("SubBevillingKreditKontor",
reportDocument, false);
                }
        }
        catch{}
       }

       #endregion

       #region SubBevillingKreditKomite

       /// <summary>
       /// Denne metode undersøger om der er bevilling på indstillingen, og viser derefter Sub Rapporten.
       /// </summary>
       /// <param name="reportDocument"></param>
       /// <param name="indstilling"></param>
       private void ShowSubBevillingKreditKomite(ReportDocument reportDocument, Entity.Indstilling.Indstilling
indstilling)
       {
        if(indstilling.BevillingCollection != null && indstilling.BevillingCollection.Count > 0)
        {
                foreach(Bevilling bevilling in indstilling.BevillingCollection)
                        if(bevilling.BevilgerTypeId == 24)
                                ShowSubReport("SubBevillingKreditKomite", reportDocument, false);
        }
       }

       #endregion

       #region SubKoncernDiagram

       /// <summary>
       /// Denne metode undersøger om der er bevilling på indstillingen, og viser derefter Sub Rapporten.
       /// </summary>
       /// <param name="reportDocument"></param>
       /// <param name="indstilling"></param>
       private void ShowSubKoncernDiagram(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
       {
        try
        {
                if(indstilling.BevillingCollection != null && indstilling.BevillingCollection.Count > 0)
                {
                        if(indstilling.Koncern.Koncerndiagram != null)
                        {
                                //Her laves vores koncerndiagram til brug i rapporten.
                                CrystalReport.DataSets.Koncerndiagram koncernDiagramDS = new
CrystalReport.DataSets.Koncerndiagram();
                                koncernDiagramDS.ReadXml(new
StringReader(ConvertObjectToDataSet(indstilling.Koncern.Koncerndiagram, koncernDiagramDS ).GetXml()));
                                ReportDocument subKoncernDiagram = new ReportDocument();
                                SubreportObject subreportObject =
(SubreportObject)reportDocument.ReportDefinition.ReportObjects["SubKoncernDiagram"];
                                //Her sætter vi vores koncerndiagram til det i rapport dokumentet.
                                subKoncernDiagram =
reportDocument.OpenSubreport(subreportObject.SubreportName);
                                subKoncernDiagram.SetDataSource(koncernDiagramDS);

                                ShowSubReport("SubKoncernDiagram", reportDocument, false);
```

```
                                        }
                                }
                        }
                        catch{}
                }

                #endregion

                #region Subkautionist

                /// <summary>
                /// Denne metode undersøger om der er interessenter på indstillingen, og viser derefter Sub Rapporten.
                /// </summary>
                /// <param name="reportDocument"></param>
                /// <param name="indstilling"></param>
                private void ShowSubkautionist(ReportDocument reportDocument, Entity.Indstilling.Indstilling indstilling)
                {
                        try
                        {
                                if(indstilling.Interessenter!=null && indstilling.Interessenter.Count > 0)
                                        ShowSubReport("Subkautionist", reportDocument, false);
                        }
                        catch{}
                }

                #endregion

                #region SubEjendomSkemaNyInds

                /// <summary>
                /// Denne metode undersøger om der er ejendomme på indstillingen, og viser derefter Sub Rapporten.
                /// </summary>
                /// <param name="reportDocument"></param>
                /// <param name="indstilling"></param>
                private void ShowSubEjendomSkemaNyInds(ReportDocument reportDocument, Entity.Indstilling.Indstilling
indstilling)
                {
                        try
                        {
                                if(indstilling.Ejendomme!=null && indstilling.Ejendomme.Count > 0)
                                {
                                        ShowSubReport("SubEjendomSkemaNyInds", reportDocument, false);
                                }
                        }
                        catch{}
                }

                #endregion

        #endregion
        }
}
```

8.4.2 BRF.Entity.Indstilling Indstilling.cs

```
using System;
using System.Xml;
```

```
using System.Xml.Serialization;

namespace BRF.Entity.Indstilling
{
    /// <summary>
    /// Summary description for Class1.
    /// </summary>
    [Serializable()]
    public class Indstilling : EntityBase
    {
     #region Declarations
     private string myAfdeling = string.Empty;
     private int myIndstillingId;
     private string myIndstillingNavn = string.Empty;
     private int myIndstillingTypeId;
     private string myIndstillingNr = string.Empty;
     private int myIndstillingStatusId;
     private       string myOprettetAf = string.Empty;
     private DateTime myOprettelseDato;
     private string myKonklusion = string.Empty;
     private string myKortSagsFremstilling = string.Empty;
     private string myAfdelingNr = string.Empty;
     private string mySupplerendeSikkerhed = string.Empty;
     private string myUserId = string.Empty;
     private string myIndstillingKommentar = string.Empty;
     private string myLejerKommentar = string.Empty;
     private string myKaldenavn;
     private int myKoncernId;

     private EjendomCollection myEjendomCollection;
     private Koncern myKoncern;
     private Prissaetning[] myPrissaetningArray;
     private InteressentCollection myInteressenter;
     private Entity.Indstilling.Type myIndstillingType;
     private IndstillingKilde myIndstillingKilde;
     private BevillingCollection myBevillingCollection;

     public enum Fields
     {
      IndstillingId,
      IndstillingNavn,
      IndstillingTypeId,
      IndstillingNr,
      IndstillingStatusId,
      OprettetAf,
      OprettelseDato,
      Konklusion,
      KortSagsFremstilling,
      AfdelingNr,
      SupplerendeSikkerhed,
      UserId
     }
     #endregion

     public Indstilling() : base()
     {
      myEjendomCollection = new EjendomCollection();
      myInteressenter = new InteressentCollection();
```

```
 myKoncern = new Koncern();
 myPrissaetningArray = new Prissaetning[2];//
 myIndstillingType = new Entity.Indstilling.Type();
 myIndstillingKilde = new IndstillingKilde();
 myBevillingCollection = new BevillingCollection();
}

#region Public Properties

public int KoncernId
{
 get{return myKoncernId;}
 set{myKoncernId = value;}
}

public string Afdeling
{
 get
 {
  return myAfdeling;
 }
 set
 {
  if (this.myAfdeling != value)
  {
                this.myAfdeling = value;
                myIsDirty = true;
  }
 }
}

public int IndstillingId
{
 get
 {
  return myIndstillingId;
 }
 set
 {
  myIndstillingId = value;
 }
}

public string IndstillingNavn
{
 get
 {
  return myIndstillingNavn;
 }
 set
 {
  if (this.myIndstillingNavn != value)
  {
                this.myIndstillingNavn = value;
                myIsDirty = true;
  }
 }
}
```

```
public string Kaldenavn
{
 get {return myKaldenavn;}
 set
 {
  if (this.myKaldenavn != value)
  {
                this.myKaldenavn = value;
                myIsDirty = true;
  }
 }
}

public int IndstillingTypeId
{
 get
 {
  return myIndstillingTypeId;
 }
 set
 {
  myIndstillingTypeId = value;
 }
}

public string IndstillingNr
{
 get
 {
  return myIndstillingNr;
 }
 set
 {
  if (this.myIndstillingNr != value)
  {
                this.myIndstillingNr = value;
                myIsDirty = true;
  }
 }
}

public int IndstillingStatusId
{
 get
 {
  return myIndstillingStatusId;
 }
 set
 {
  if (this.myIndstillingStatusId != value)
  {
                this.myIndstillingStatusId = value;
                myIsDirty = true;
  }
 }
}
```

```
public string OprettetAf
{
 get
 {
  return myOprettetAf;
 }
 set
 {
  myOprettetAf = value;
 }
}

public DateTime OprettelseDato
{
 get
 {
  return myOprettelseDato;
 }
 set
 {
  myOprettelseDato = value;
 }
}

public string Konklusion
{
 get
 {
  return myKonklusion;
 }
 set
 {
  if (this.myKonklusion != value)
  {
                this.myKonklusion = value;
                myIsDirty = true;
  }
 }
}

public string KortSagsFremstilling
{
 get
 {
  return myKortSagsFremstilling;
 }
 set
 {
  if (this.myKortSagsFremstilling != value)
  {
                this.myKortSagsFremstilling = value;
                myIsDirty = true;
  }
 }
}

public string AfdelingNr
{
```

```
 get
 {
  return myAfdelingNr;
 }
 set
 {
  if (this.myAfdelingNr != value)
  {
                this.myAfdelingNr = value;
                myIsDirty = true;
  }
 }
}

public string SupplerendeSikkerhed
{
 get
 {
  return mySupplerendeSikkerhed;
 }
 set
 {
  if (this.mySupplerendeSikkerhed != value)
  {
                this.mySupplerendeSikkerhed = value;
                myIsDirty = true;
  }
 }
}

public string UserId
{
 get
 {
  return myUserId;
 }
 set
 {
  myUserId = value;
 }
}

public EjendomCollection Ejendomme
{
 get
 {
  return myEjendomCollection;
 }
 set
 {
  myEjendomCollection = value;
 }
}

[XmlElement(ElementName = "AndreInteressenter")]
public InteressentCollection Interessenter
{
 get
```

```
 {
  return myInteressenter;
 }
 set
 {
  myInteressenter = value;
 }
}

public Koncern Koncern
{
 get
 {
  return myKoncern;
 }
 set
 {
  myKoncern = value;
 }
}

public Prissaetning[] PrissaetningArray
{
 get
 {
  return myPrissaetningArray;
 }
 set
 {
  myPrissaetningArray = value;
 }
}

public Type IndstillingType
{
 get
 {
  return myIndstillingType;
 }
 set
 {
  myIndstillingType = value;
 }
}

public IndstillingKilde IndstillingKilde
{
 get
 {
  return myIndstillingKilde;
 }
 set
 {
  myIndstillingKilde = value;
 }
}

public BevillingCollection BevillingCollection
```

```
{
 get
 {
  return myBevillingCollection;
 }
 set
 {
  myBevillingCollection = value;
 }
}

public string IndstillingKommentar
{
 get
 {
  return myIndstillingKommentar;
 }
 set
 {
  if (this.myIndstillingKommentar != value)
  {
                this.myIndstillingKommentar = value;
                myIsDirty = true;
  }
 }
}

public string LejerKommentar
{
 get
 {
  return myLejerKommentar;
 }
 set
 {
  if (this.myLejerKommentar != value)
  {
                this.myLejerKommentar = value;
                myIsDirty = true;
  }
 }
}

#endregion

#region IClonable Members

public Indstilling Clone()
{
 Indstilling indstilling = new Indstilling();
 indstilling.myAfdeling = this.myAfdeling;
 indstilling.myAfdelingNr = this.myAfdelingNr;
 indstilling.myIndstillingId = this.myIndstillingId;
 indstilling.myIndstillingNavn = this.myIndstillingNavn;
 indstilling.myIndstillingNr = this.myIndstillingNr;
 indstilling.myIndstillingStatusId = this.myIndstillingStatusId;
 indstilling.myIndstillingTypeId = this.myIndstillingTypeId;
 indstilling.myKonklusion = this.myKonklusion;
```

```
        indstilling.myKortSagsFremstilling  = this.myKortSagsFremstilling;
        indstilling.myIndstillingKommentar = this.myIndstillingKommentar;
        indstilling.myLejerKommentar = this.myLejerKommentar;
        indstilling.myOprettelseDato  = this.myOprettelseDato;
        indstilling.myOprettetAf  = this.myOprettetAf;
        indstilling.Kaldenavn = this.Kaldenavn;
        indstilling.mySupplerendeSikkerhed = this.mySupplerendeSikkerhed;
        indstilling.myUserId  = this.myUserId;

        return indstilling;
       }

      #endregion
     }
}
```

8.4.3 BRF.Indstilling.BackEnd.DataAccess Indstilling.cs

```
using System;
using BRF.Standard.Database.DataMaster;
using BRF.Standard.DataMapping;

namespace BRF.Indstilling.Backend.DataAccess
{
    [ReadProcedure("IndstillingRead", typeof(Setting), "myConnectionString")]
    [WriteProcedure("IndstillingWrite", typeof(Setting), "myConnectionString")]
    [DeleteProcedure("IndstillingDelete", typeof(Setting), "myConnectionString")]
    public class Indstilling
    {
     #region Declarations

     [ReadParameter("@IndstillingId", 0)]
     [WriteParameter("@IndstillingId", 0, ParameterDirectionEnum.InputOutput)]
     [DeleteParameter("@IndstillingId", 0)]
     [IsIdentifier(true, 1)]
     [ReturnValue("IndstillingId")]
     [DataMapperField("myIndstillingId")]
     private int myIndstillingId;

     [ReturnValue("IndstillingTypeId", 0)]
     [WriteParameter("@IndstillingTypeId")]
     [DataMapperField("myIndstillingTypeId")]
     private int myIndstillingTypeId;

     [ReturnValue("KoncernId", 0)]
     [WriteParameter("@KoncernId")]
     [DataMapperField("myKoncernId")]
     private int myKoncernId;

     [WriteParameter("@Kaldenavn")]
     [ReturnValue("Kaldenavn")]
     [DataMapperField("myKaldenavn")]
     private string myKaldenavn;

     [ReturnValue("IndstillingNr")]
     [WriteParameter("@IndstillingNr")]
```

```
[DataMapperField("myIndstillingNr")]
private string myIndstillingNr = string.Empty;

[ReturnValue("Afdeling")]
[WriteParameter("@Afdeling")]
[DataMapperField("myAfdeling")]
private string myAfdeling = string.Empty;

[ReturnValue("IndstillingNavn")]
[WriteParameter("@IndstillingNavn")]
[DataMapperField("myIndstillingNavn")]
private string myIndstillingNavn = string.Empty;

[ReturnValue("IndstillingStatusId", 0)]
[WriteParameter("@IndstillingStatusId")]
[DataMapperField("myIndstillingStatusId")]
private int myIndstillingStatusId;

[ReturnValue("OprettetAf")]
[WriteParameter("@OprettetAf")]
[DataMapperField("myOprettetAf")]
private string myOprettetAf;

[ReturnValue("OprettelseDato")]
[WriteParameter("@OprettelseDato")]
[DataMapperField("myOprettelseDato")]
private DateTime myOprettelseDato;

[ReturnValue("Konklusion")]
[WriteParameter("@Konklusion")]
[DataMapperField("myKonklusion")]
private string myKonklusion = string.Empty;

[ReturnValue("KortSagsFremstilling")]
[WriteParameter("@KortSagsFremstilling")]
[DataMapperField("myKortSagsFremstilling")]
private string myKortSagsFremstilling = string.Empty;

[ReturnValue("AfdelingNr")]
[WriteParameter("@AfdelingNr")]
[DataMapperField("myAfdelingNr")]
private string myAfdelingNr = string.Empty;

[ReturnValue("SupplerendeSikkerhed")]
[WriteParameter("@SupplerendeSikkerhed")]
[DataMapperField("mySupplerendeSikkerhed")]
private string mySupplerendeSikkerhed = string.Empty;

[ReturnValue("UserId")]
[WriteParameter("@UserId")]
[DataMapperField("myUserId")]
private string myUserId = string.Empty;

[ReturnValue("IndstillingKommentar")]
[WriteParameter("@IndstillingKommentar")]
[DataMapperField("myIndstillingKommentar")]
private string myIndstillingKommentar = string.Empty;
```

```
    [ReturnValue("LejerKommentar")]
    [WriteParameter("@LejerKommentar")]
    [DataMapperField("myLejerKommentar")]
    private string myLejerKommentar = string.Empty;

    [ReturnValue("ESDHId")]
    [WriteParameter("@ESDHId")]
    [DataMapperField("myESDHId")]
    private string myESDHId = string.Empty;

    #endregion

    #region Constructors

    public Indstilling(int indstillingId)
    {
     myIndstillingId = indstillingId;
    }

    public Indstilling()
    {
    }

    #endregion

    #region Public Properties

    public int IndstillingId
    {
     get
     {
      return myIndstillingId;
     }
    }

    public int KoncernId
    {
     get { return myKoncernId; }
     set { myKoncernId = value; }
    }

    #endregion
   }
}
```

8.4.4 BRF.Indstilling.Business Indstilling.cs

```
using System;
using BRF.Standard.Database.Common;
using BRF.Standard.Database.DataMaster;
using BRF.Standard.DataMapping;

namespace BRF.Indstilling.Backend.Business
{
    public abstract class Indstilling
    {
```

```
    public static Entity.Indstilling.IndstillingSoegeResultatCollection
SoegIndstilling(Entity.Indstilling.IndstillingSoegekriterier indstillingSoegekriterier)
    {
     // map fields fra entity søgekriterier til dataaccess indexer
     DataAccess.IndstillingSoegeResultatIndexer indstillingSoegeResultatIndexer =
      (DataAccess.IndstillingSoegeResultatIndexer) DataMapper.DataMapToDataAccess(indstillingSoegekriterier,
typeof(DataAccess.IndstillingSoegeResultatIndexer));

     // load indexer
     DataMaster.Load(indstillingSoegeResultatIndexer);

     // create entity collection
     Entity.Indstilling.IndstillingSoegeResultatCollection indstillingSoegeResultatCollection = new
Entity.Indstilling.IndstillingSoegeResultatCollection();

     foreach (DataAccess.IndstillingSoegeResultat dataaccessIndstillingSoegeResultat in
indstillingSoegeResultatIndexer.GetAll())
     {
      // map dataaccess object to entity
      Entity.Indstilling.IndstillingSoegeResultat indstillingSoegeResultat =
                (Entity.Indstilling.IndstillingSoegeResultat)
DataMapper.DataMapToEntity(dataaccessIndstillingSoegeResultat, typeof(Entity.Indstilling.IndstillingSoegeResultat));

      // add to collection
      indstillingSoegeResultatCollection.Add(indstillingSoegeResultat);
     }


     return indstillingSoegeResultatCollection;
    }

    public static Entity.Indstilling.Indstilling NyIndstilling(int indstillingTypeId)
    {
     Entity.Indstilling.Indstilling indstilling =
      new Entity.Indstilling.Indstilling();
     indstilling.IndstillingTypeId=indstillingTypeId;


     indstilling = GemIndstilling(indstilling);

     return indstilling;
    }

    public static Entity.Indstilling.Indstilling HentIndstilling(int indstillingId)
    {
     DataAccess.Indstilling dataAccessIndstilling = new DataAccess.Indstilling(indstillingId);
     DataMaster.Load(dataAccessIndstilling);
     Entity.Indstilling.Indstilling entityIndstilling = (Entity.Indstilling.Indstilling)
DataMapper.DataMapToEntity(dataAccessIndstilling, typeof(Entity.Indstilling.Indstilling));


     entityIndstilling.Interessenter = Interessent.HentInteressenter(indstillingId);
     entityIndstilling.Koncern = Koncern.HentKoncern(entityIndstilling.KoncernId);
     entityIndstilling.IndstillingType = Type.HentType(Entity.Indstilling.GruppeTypeEnum.IndstillingType,
entityIndstilling.IndstillingTypeId);
     entityIndstilling.BevillingCollection = HentBevilling(indstillingId);
     //entityIndstilling.IndstillingKilde = HentIndstillingKilde(indstillingId);
     entityIndstilling.PrissaetningArray = HentPrissaetning(indstillingId);
```

```
        entityIndstilling.Ejendomme=Ejendom.HentEjendomme(indstillingId);
        return entityIndstilling;
       }

       public static Entity.Indstilling.Indstilling GemIndstilling(Entity.Indstilling.Indstilling entityIndstilling)
       {
        Entity.Indstilling.Indstilling returnIndstilling = entityIndstilling;
        Entity.Indstilling.Indstilling savedIndstilling = entityIndstilling.Clone();

        ITransaction transaction = DataMaster.BeginTransaction(DataAccess.Setting.ConnectionString);

        try
        {
         // Gem koncern
         // TODO: Koncern skal ikke have et indstillingsid, men
         // det er pt. påkrævet.
         Entity.Indstilling.Koncern koncern = Koncern.GemKoncern(
                        entityIndstilling.Koncern, transaction);

         // Opdater indstillingsobjektet med den gemte koncern.
         savedIndstilling.Koncern = koncern;
         // Save indstilling (and update identifier if new)
         if (entityIndstilling.IsDirty || entityIndstilling.IsNew)
         {
                        DataAccess.Indstilling dataAccessIndstilling = (DataAccess.Indstilling)
DataMapper.DataMapToDataAccess(savedIndstilling, typeof(DataAccess.Indstilling));
                        dataAccessIndstilling.KoncernId = koncern.KoncernId;
                        DataMaster.Save(dataAccessIndstilling, transaction);
                        returnIndstilling = (Entity.Indstilling.Indstilling) DataMapper.DataMapToEntity(dataAccessIndstilling,
typeof(Entity.Indstilling.Indstilling));
         }

         foreach (Entity.Indstilling.Interessent interessent in entityIndstilling.Interessenter)
                        returnIndstilling.Interessenter.Add( Backend.Business.Interessent.GemInteressent(interessent,
transaction));

         foreach (Entity.Indstilling.Ejendom ejendom in entityIndstilling.Ejendomme)
         {
                        ejendom.IndstillingId=returnIndstilling.IndstillingId;
                        returnIndstilling.Ejendomme.Add( Backend.Business.Ejendom.GemEjendom(ejendom, transaction));
         }

         returnIndstilling.Koncern = koncern;
         DataMaster.CommitTransaction(transaction);
         return returnIndstilling;
        }
        catch
        {
         DataMaster.RollBackTransaction(transaction);
         throw;
        }

       }

       public static void SletIndstilling(int indstillingId)
       {
        DataAccess.Indstilling dataAccessIndstilling = new DataAccess.Indstilling(indstillingId);
        ITransaction transaction = DataMaster.BeginTransaction(DataAccess.Setting.ConnectionString);
```

```
        try
        {
         // Delete other objects in transaction
         DataMaster.Delete(dataAccessIndstilling, transaction);
         DataMaster.CommitTransaction(transaction);
        }
        catch
        {
         DataMaster.RollBackTransaction(transaction);
         throw;
        }
       }

       public static void GemIndstillingsRapport(Entity.Indstilling.Indstilling indstilling, byte[] rapportData)
       {
        try
        {
         Backend.ServiceAgent.ESDH.gemDokument(indstilling, rapportData);

        }
        catch
        {
         throw;
        }
       }


       #region Private methods

       public static Entity.Indstilling.Prissaetning[] HentPrissaetning(int indstillingId)
       {
        try
        {
         DataAccess.PrissaetningIndexer prissaetningIndexer = new DataAccess.PrissaetningIndexer(indstillingId);
         DataMaster.Load(prissaetningIndexer);

         Entity.Indstilling.Prissaetning[] prissaetningArray = new
Entity.Indstilling.Prissaetning[prissaetningIndexer.Count];
         object[] objectArray = prissaetningIndexer.GetAllArray();

         for (int i = 0; i < objectArray.Length; i++)
                        prissaetningArray[i] = (Entity.Indstilling.Prissaetning) DataMapper.DataMapToEntity(objectArray[i],
typeof(Entity.Indstilling.Prissaetning));

         return prissaetningArray;
        }
        catch(Exception exception)
        {
         //throw new Exception("HentPrissaetning fejlede med følgede indstillingId: "+indstillingId, exception);
         return null;
        }
       }

       public static Entity.Indstilling.IndstillingKilde HentIndstillingKilde(int indstillingKildeId)
       {
        try
        {
```

```
        DataAccess.IndstillingKilde dataAccessIndstillingKilde = new DataAccess.IndstillingKilde(indstillingKildeId);
        DataMaster.Load(dataAccessIndstillingKilde);

        Entity.Indstilling.IndstillingKilde entityIndstillingKilde = new Entity.Indstilling.IndstillingKilde();
        entityIndstillingKilde = (Entity.Indstilling.IndstillingKilde)
DataMapper.DataMapToEntity(dataAccessIndstillingKilde,typeof(Entity.Indstilling.IndstillingKilde));

        return entityIndstillingKilde;
       }
       catch(Exception exception)
       {
        throw new Exception("HentIndstillingKilde fejlede med følgede indstillingKildeId: "+indstillingKildeId,
exception);
       }
      }

      public static Entity.Indstilling.BevillingCollection HentBevilling(int indstillingId)
      {
       try
       {
        DataAccess.BevillingIndexer bevillingIndexer = new DataAccess.BevillingIndexer(indstillingId);
        DataMaster.Load(bevillingIndexer);

        Entity.Indstilling.BevillingCollection bevillingCollection = new Entity.Indstilling.BevillingCollection();
        foreach(DataAccess.Bevilling dataAccessBevilling in bevillingIndexer.GetAll())
        {
                Entity.Indstilling.Bevilling entityBevilling = new Entity.Indstilling.Bevilling();
                entityBevilling = (Entity.Indstilling.Bevilling)
DataMapper.DataMapToEntity(dataAccessBevilling,typeof(Entity.Indstilling.Bevilling));
                entityBevilling.BevillingKommentar = HentBevillingKommentar(entityBevilling.BevillingId);
                bevillingCollection.Add(entityBevilling);
        }
        return bevillingCollection;
       }
       catch(Exception exception)
       {
        throw new Exception("HentBevilling fejlede med følgende indstillingsid: "+indstillingId, exception);
       }
      }

      public static Entity.Indstilling.BevillingKommentar HentBevillingKommentar(int bevillingId)
      {
       try
       {
        DataAccess.BevillingKommentar dataAccessBevillingKommentar = new
DataAccess.BevillingKommentar(bevillingId);
        DataMaster.Load(dataAccessBevillingKommentar);

        Entity.Indstilling.BevillingKommentar entityBevillingKommentar = new
Entity.Indstilling.BevillingKommentar();
        entityBevillingKommentar = (Entity.Indstilling.BevillingKommentar)
DataMapper.DataMapToEntity(dataAccessBevillingKommentar,typeof(Entity.Indstilling.BevillingKommentar));

        return entityBevillingKommentar;
       }
       catch(Exception exception)
       {
        throw new Exception("HentBevillingKommentar fejlede med følgede bevillingId: "+bevillingId, exception);
```

```
        }
       }



      #endregion
     }

     #region UnitTest
     namespace UnitTest
     {
      [csUnit.TestFixture()]
      public class Indstilling
      {
       [csUnit.Test()]
       public static void TestSoegIndstilling()
       {
        BRF.Indstilling.Shared.SettingToken.SettingTokenSingleton.InitializeForUnitTesting();

        // Hent en eksisterende indstilling
        Entity.Indstilling.IndstillingSoegeResultatCollection indstillingSoegeResultatCollection =
Business.Indstilling.SoegIndstilling(new Entity.Indstilling.IndstillingSoegekriterier());

        csUnit.Assert.NotNull(indstillingSoegeResultatCollection);
        //csUnit.Assert.NotNull(entityIndstilling1.Navn);
       }
       [csUnit.Test()]
       public static void TestGemEksisterendeIndstilling()
       {
        BRF.Indstilling.Shared.SettingToken.SettingTokenSingleton.InitializeForUnitTesting();

        int indstillingId = 71;

        // Hent en eksisterende indstilling
        Entity.Indstilling.Indstilling entityIndstilling1 = Business.Indstilling.HentIndstilling(indstillingId);

        // Gem indstillingen
        Business.Indstilling.GemIndstilling(entityIndstilling1);

        csUnit.Assert.Equals(entityIndstilling1.IndstillingId, indstillingId);
        //csUnit.Assert.Equals(entityIndstilling1.IndstillingId, entityIndstilling1.Koncern.Koncerndiagram.IndstillingId);
       }

       [csUnit.Test()]
       public static void TestGemNyIndstillingIndstilling()
       {
        BRF.Indstilling.Shared.SettingToken.SettingTokenSingleton.InitializeForUnitTesting();

        Entity.Indstilling.Indstilling entityIndstilling1 = Business.Indstilling.NyIndstilling(1);

        // Hent den nyligt gemte indstilling
        Entity.Indstilling.Indstilling entityIndstilling2 =
Business.Indstilling.HentIndstilling(entityIndstilling1.IndstillingId);

        csUnit.Assert.Equals(entityIndstilling1.IndstillingId, entityIndstilling2.IndstillingId);

       }
```

```
        [csUnit.Test()]
        public static void TestHentPrissaetning()
        {
         BRF.Indstilling.Shared.SettingToken.SettingTokenSingleton.InitializeForUnitTesting();

         // Hent en eksisterende indstilling
         Entity.Indstilling.Prissaetning[] prissaetningArray = Business.Indstilling.HentPrissaetning(71);

         csUnit.Assert.NotNull(prissaetningArray);
         csUnit.Assert.Greater(prissaetningArray.Length, 0);
         csUnit.Assert.Greater(prissaetningArray[0].PrissaetningId, 0);
        }

        [csUnit.Test()]
        public static void TestHentIndstillingKilde()
        {
         BRF.Indstilling.Shared.SettingToken.SettingTokenSingleton.InitializeForUnitTesting();

         // Hent en eksisterende indstilling
         Entity.Indstilling.IndstillingKilde indstillingKilde = Business.Indstilling.HentIndstillingKilde(1);

         csUnit.Assert.NotNull(indstillingKilde);
        }

        [csUnit.Test()]
        public static void TestHentBevilling()
        {
         BRF.Indstilling.Shared.SettingToken.SettingTokenSingleton.InitializeForUnitTesting();

         // Hent en eksisterende indstilling
         Entity.Indstilling.BevillingCollection bevillingCollection = Business.Indstilling.HentBevilling(71);

         csUnit.Assert.NotNull(bevillingCollection);
         csUnit.Assert.Greater(bevillingCollection.Count, 0);
         csUnit.Assert.Greater(bevillingCollection[0].BevillingId, 0);
        }

        [csUnit.Test()]
        public static void TestHentBevillingKommentar()
        {
         BRF.Indstilling.Shared.SettingToken.SettingTokenSingleton.InitializeForUnitTesting();

         // Hent en eksisterende indstilling
         Entity.Indstilling.BevillingKommentar bevillingKommentar = Business.Indstilling.HentBevillingKommentar(1);

         csUnit.Assert.NotNull(bevillingKommentar);
         csUnit.Assert.Greater(bevillingKommentar.BevillingKommentarId, 0);
        }
       }
      }
      #endregion
}
```