SERVICE ORIENTERET ARKITEKTUR ELLER FÆLLES DATABASE.

Af
Christoffer Pagaard (cwil@)

Vejleder
Søren Lauesen (slauesen@)

31. JANUAR 2017
IT UNIVERSITETET I KØBENHAVN

Abstract

A public agency's system stores and uses large amounts of data, which then has to be accessed by both internal and external systems. In itself, this is challenging, but in a public agency they also have to consider changing policies, so the systems has to be both flexible and robust.

In this report I have been looking into which of Service Oriented Architecture or a shared database is the best approach in large and complex systems (in system integration). I have compared these two approaches on six different factors: How easy is it to develop or change a system, swap a system with another, move a system to another supplier, availability, security, and latency.

I have conducted interviews with large public agencies, their suppliers, and private consultants who have been involved in the design and development on both sides. I have also studied requirement specifications, architecture documents, and reports to get a more formal understanding of these systems and how they interact.

To illustrate the approaches I use the two public agencies SKAT and STAR as cases. The former has used Service Oriented Architecture and the latter has used a shared database accessed through services. I have then looked into an upcoming technology, OData which may alleviate some of the common problems they face by using SOAP based services.

From their experiences I have tried to extract some recommendations, which are the following:

a) Use a shared database. It reduces the amount of dependencies, easing the development and improving the availability.
b) Use a data model. It provides an overview of and shared understanding of the data, reducing the risk of misinterpreting its meaning or braking systems when changing or adding data fields.
c) Use OData, it provides much needed flexibility, reducing the resources required to change a service, and improving performance.

Indholdsfortegnelse

1 Indledning

I offentlige styrelser arbejdes der efter komplekse og foranderlige love. Dette betyder at systemerne, som understøtter den enkelte styrelses opgaver, må være robuste og fleksible for at gøre det muligt at tilpasse systemerne hurtigt ved lovændringer.

En styrelses system har behov for at dele store mængder data med en lang række af både interne og eksterne systemer, hvilket grundlæggende kan gøres på to forskellige måder: Med en fælles database eller Service Orienteret Arkitektur (SOA).

Med en fælles database deles alle systemerne om en database, som kan tilgås fx med services. Systemerne er dog ikke begrænset til at lagre al deres data i denne database, men kan have deres egen og lagre data, som de ikke har til fælles med andre systemer lokalt.

I en service orienteret arkitektur har det enkelte system sin egen database og andre systemer kan tilgå den via systemets udbudte services.

Hvilken af de to tilgange er den bedste? Det afhænger af hvilke faktorer der lægges vægt på og jeg har set nærmere på følgende.

1. Let at udvikle eller ændre et system
2. Let at udskifte et system
3. Let at flytte et system til en anden leverandør
4. Tilgængelighed
5. Sikkerhed
6. Svartider

1.1 Cases og Teknologi

Jeg har haft mulighed for at bruge hhv. STAR og SKAT som cases og ser derfor på hvordan deres respektive systemer understøtter de ovennævnte faktorer. Endvidere benytter Naturstyrelsen en ny teknologi ved navn OData, som er på vej frem og jeg ser i den forbindelse på hvordan de to systemers håndtering af faktorerne vil blive påvirket ved indførslen af denne teknologi.

1.1.1 STAR's arkitektur

Styrelsen for Arbejdsmarked og Rekruttering (STAR) har en fælles database kaldet Det Fælles Datagrundlag (DFDG), hvor det meste data lagres. Denne data tilgås ved at systemerne kalder SOAP baserede services, som udvikles i samarbejde mellem STAR og den leverandør som har brug for data. Ca. 50 leverandørsystemer trækker på det fælles data.

Let at udvikle eller ændre et system: Der skal forhandles om oprettelse af nye services eller om allerede eksisterende skal tilpasses, hvilket er tidskrævende. Deres data model er med til at gøre denne proces hurtigere, men den er ikke velbeskrevet over alt, hvilket medføre at forskellige systemer til tider tolker data på forskellig vis.

Let at udskifte et system: To ældre systemer er succesfuldt blevet erstattet af et enkelt nyt system.

Let at flytte et system til en anden leverandør: DFDG er blevet flyttet til en anden leverandør en gang tidligere og en ny udbudsrunde er pt. i gang.

Tilgængelighed: Hvis DFDG går ned kan de andre systemer ikke tilgå dens data og herved blive forhindret i at løse deres opgaver. Dette problem er blevet reduceret vha. redundante servere.

Sikkerhed: Da al data er samlet i DFDG har STAR fuld råderet over hvordan de vil implementere deres sikkerhed. De har samtidig gode muligheder for at tilrette den til deres behov. Fx kan de vælge at sikre deres data på rækkeniveau i databasens tabeller eller placere den i service laget.

Svartider: Da data sendes direkte mellem det enkelte system og DFDG er der ingen ekstra forsinkelser. Services er dog ikke nødvendigvis skræddersyet til det enkelte system og derfor kan det ske, at mere data end nødvendigt bliver sendt eller at flere services må kaldes for at få den fornødne data.

1.1.2 SKAT's arkitektur

SKAT's arkitektur er baseret på SOA styret af en strategi ved navn Connect. Denne strategi tvinger de enkelte systemer til at lagre deres data selv og have en skarp grænseflade med en SOAP baseret service-bus. Hvis andre systemer ønsker at få adgang til denne data skal de benytte den centrale sikkerhedsløsning (DCSL), der udstiller alle services på tværs af systemer og herved giver adgang til det enkelte systems services.

Let at udvikle eller ændre et system: Der er store forsinkelser og en omfattende arbejdsproces når nye systemer skal udvikles eller ved ændringer af eksisterende. Dette skyldes bl.a. høj kompleksitet og manglende overblik over data og hvilket system der har data.

Let at flytte et eller udskifte et system: SKAT har hverken formået at udskifte et system med et andet eller flytte et system til en anden leverandør.

Tilgængelighed: Der er problemer med tilgængeligheden, hvilket skyldes at hvis et system går ned sker der en dominoeffekt af nedbrud da alle er afhængige af flere systemer. Hvis DCSL går ned vil ingen af systemerne være tilgængelige fordi alt skal igennem DCSL. Connect strategien er samtidig imod replikering af data, så denne kædereaktion er svær at forebygge.

Sikkerhed: Sikkerheden er styret via DCSL, som afgør om et system må kalde services hos andre systemer. Dette centrale system er dog komplekst og ustabilt, hvilket har medført at nogle systemer har konstrueret deres eget mellemled for at forenkle og formindske interaktionen med DCSL.

Svartider: Bl.a. betyder spredningen af data at et system kan risikere at skulle foretage mange forskellige kald til forskellige systemer for at opnå det ønskede resultat, hvilket tager tid. Ydermere kan et enkelt kald sætte gang i en række kald hvilket igen skaber forsinkelser. Endnu en kilde til høj svartid er at alle kald på tværs af systemer går via DCSL, som har en lav ydelse.

1.1.3 OData

OData er en teknologi, som gør det muligt for klienten selv at specificere hvilken data der skal hentes og hvilke parametre den hentes ud fra. Dette kan gøres uden at involvere leverandøren, så længe den ønskede data er tilgængelig i OData servicens datamodel.

1.2 Resultat og Anbefaling

Ud over at give et bud på hvilken af tilgangene der er den bedste, så følger der to anbefalinger, som ville kunne forbedre et system uanset tilgang.

Hvilken af de to tilgange er så den bedste?

1) **Let at udvikle eller ændre et system**: Med den fælles database kan al fælles data findes et sted og systemleverandøren skal derfor kun forhandle med én anden part. Med SOA kan data være spredt ud over flere systemer, hvilket betyder at en systemleverandør i SOA systemet, først skal lokalisere hvor den ønskede data er og efterfølgende forhandle med hver dataejer
2) **Let at udskifte et system med et andet**: Det er ikke klart hvad den præcise årsag til succesen med den fælles database er, men det står tilbage, at de færre afhængigheder gør det nemmere at overskue hvilke konsekvenser en udskiftning af et system vil medføre end med SOA
3) **Let at flytte et system til en anden leverandør**: Intet her viser hvorfor det kun lykkedes at udskifte leverandør med en fælles database, det kan blot konstateres, at det har været muligt, selv om der I princippet ikke skulle være nogen forskel når et system sendes i udbud
4) **Tilgængelighed**: Den fælles database har langt færre afhængigheder og er heraf langt mindre udsat i forhold til manglende tilgængelighed. Den fælles database opnår en tilgængelighed på (99,5 %)^2 = 99 %, hvor SOA har en tilgængelighed på (99,5 %)^n, hvor n er antallet af afhængige systemer og minimum 2, altså i bedste fald det samme som med en fælles database
5) **Sikkerhed**: Med både fælles database og SOA kan den fornødne sikkerhed opnås
6) **Svartider**: SOA har spredt data over flere systemer og derfor kan det være nødvendigt at afvente flere service kald for at få den samme data, som med en fælles database, hvor al data er samlet ét sted

Det udslagsgivende på tværs af faktorerne er spredningen af data. SOA spreder data i en grad så det medfører manglende overblik og mange afhængigheder på tværs af systemerne. Det må derfor konkluderes at den fælles database løser problemet med deling af fælles data mellem systemer mest hensigtsmæssigt da arkitekturen løser netop dette.

Anvend OData

De to tilgange kunne begge drage fordel af at benytte OData frem for SOAP. Hvis dette skete ville de kunne opnå følgende forbedringer:

a) Mindre, men mere fokuseret, leverandør samarbejde fordi det nu ikke længere handler om nye services, men der i mod om hvilken data der skal gives adgang til
b) Lavere udviklingsomkostninger fordi der ikke skal oprettes nye services
c) Udskiftning af systemer ville blive lettere fordi et nyt system vil kunne nøjes med at gøre dets data tilgængelig frem for at oprette en masse nye services

d) En forbedret ydelse som måske ikke er nødvendig pt. men kan blive det i fremtiden med et voksende systemporteføljе

Anvend en datamodel

Med en data-model og beskrivelse opnår man et større overblik og en fælles forståelse for data og relationerne her imellem, hvilket er med til at reducere risikoen for misforståelser og at data ”forsvinder”.

2 Faktorer

For at kunne vurdere hvilken tilgang, der er den bedste, er det nødvendigt at sammenligne ud fra de samme faktorer.

Jeg har i gennemgangen af de indsamlede interviews og dokumenter stødt på mange mulige faktorer, og udvalgt følgende:

1. Let at udvikle eller ændre et system
2. Let at udskifte et system med et andet
3. Let at flytte et system til en anden leverandør
4. Tilgængelighed
5. Sikkerhed
6. Svartider

2.1 Let at udvikle eller ændre et system

Et offentligt system er til for at skabe besparelser ved at lette opgaverne for sagsbehandlerne. Da sagsbehandlerne ofte løser opgaverne ud fra bestemmelser i loven er det nødvendigt at systemet hurtigt kan ændres og udbygges efterhånden som nye lovforslag træder i kraft eller helt nye systemer kan tilføjes, uden andre systemer påvirkes.

Hvis det nye eller tilrettede system kan genbruge eksisterende services skal leverandøren ikke selv udvikle dem, hvilket åbner for potentielt færre fejl og kortere udviklingstid. Det kræver dog at der er et godt overblik over både data og eksisterende services samt deres anvendelse.

2.2 Let at udskifte et system med et andet

EU-lovgivningen kræver, at de offentlige systemer sendes i udbud hvert fjerde år for at sikre der ikke betales for meget.

2.3 Let at flytte et system til en anden leverandør

For at kunne spare penge på drift og vedligehold er det nødvendigt at kunne udskifte leverandør og herved skærpe konkurrencen.

2.4 Tilgængelighed

Et systems oppetid er af indlysende årsager noget som alle parter er enige om skal være høj. Hvis systemet ikke er tilgængeligt kan dets services ikke blive anvendt og al produktion, som er afhængig af disse må i bedste fald blive reduceret og i værste fald stoppe helt indtil systemet er tilgængeligt igen.

2.5 Sikkerhed

I offentlige systemer arbejdes der ofte med fortrolige data om den enkelte borger. Derfor er det vigtigt, at systemet er sikret imod, at uvedkommende kan få fat i denne data.

2.6 Svartider

Et IT system er til for at effektivisere arbejdsprocesserne og det er derfor vigtigt, at systemet får hurtigt svar fra de kaldte services, så medarbejderen ikke skal vente på det og det dermed bliver en hæmsko.

3 Teknikker

3.1 SOA

Service Orienteret Arkitektur(SOA) havde sin storhedstid i midt 00'erne hvor Microsoft lancerede en .NET løsning baseret på SOA, nemlig SOAP som en del af deres .NET framework. SOA bliver ofte hyldet for at følge flere principper som anses for at være vigtige inden for den objektorienterede verden. Det gælder f.eks. genbrug, fleksibilitet og indkapsling, hvilket skulle fremme besparelser og formindske time-to-market.

SOA bliver defineret på forskellige måder afhængig af hvem man henvender sig til. Ian Gorton har kigget på flere af dem, her i blandt Microsoft, IBM, W3 m.fl., og her efter samlet dem i følgende definition:

> "A service is an implementation of a well-defined piece of business functionality, with a published interface that is discoverable and can be used by service consumers when building different applications and business processes."

I de fleste definitioner af SOA indgår en form for forretningsdrevet perspektiv, men i de systemer som jeg har erfaring med fungerer de primært som en form for bindeled til systemers data og sekundært til deres funktioner. Dette afspejles i Microsofts definition af SOA

> " Service-Oriented Architecture is an approach to organizing information technology in which data, logic, and infrastructure resources are accessed by routing messages between network interfaces."

SOA er altså ikke en bestemt løsning, men en række principper, hvor de hyppigst fremhævede er følgende:

a) Genbrug
b) Formel kontrakt, som definerer hvad og hvordan der kommunikeres
c) Lav kobling
d) Indkapsling, skjuler underliggende logik

Disse principper har ofte en stærk og direkte indflydelse på ikke funktionelle krav. Fx understøtter principperne om kontrakter og indkapsling godt op omkring det ikke funktionelle krav tværgående-samarbejde. Der er dog også en række krav, som bliver negativt ramt, hvilket kan medføre store problemer.

a) Tilgængelighed og pålidelighed kan falde drastisk, da brugen af en service skal via netværket. Et andet problem kan være antallet af abstraktionslag, som en service er bygget på, hvilket øger risikoen for fejl, enten direkte i servicen eller i andre services, som den er afhængig af.
b) Ydeevnen bliver ofte ramt hvilket der er flere forskellige grunde til: Service kald foregår via netværket, hvilket skaber en direkte forsinkelse; data skal transformeres til og fra f.eks. XML og mellemliggende lag, evt via andre services. I forsøget på at gøre services mere genbrugelige, definerer man dem ofte sådan at de sender relateret data med tilbage, hvilket også øger tidsforbruget.

Et aspekt ved brugen af services er muligheden for at ignorere hvordan den bagvedliggende funktionalitet og data bliver håndteret. Dette giver en lav kobling, men det gør det også muligt at relateret data bliver

fordelt ud over forskellige systemer, hvilket betyder at overblikket over data reduceres og relationer kan blive brudt.

Et andet problem ved SOA er, at mængden af services har det med at gro, hvilket kan skyldes flere forskellige ting:

a) Hvis en service ændres risikeres det, at andre systemer, som kalder servicen, enten kalder den forkert eller får noget uventet data tilbage, og derfor er det sikrere at producere en ny.
b) Services er af natur ikke fleksible og hvis en klient ønsker at slå op i et register med et andet parameter end ellers, skal en ny service oprettes.
c) Det kan også skyldes manglende overblik hvilket medfører at en eksisterende service bliver overset og en ny oprettes.

3.1.1 SOAP og REST

Der er to fremherskende tilgange baseret på service orienteret arkitektur: SOAP og REST, og de benytter begge primært http til at kommunikere mellem udbyder og klienter.

SOAP er en protokol som benytter XML til at formatere sin data og en SOAP baseret service repræsenterer ofte en funktion på det system hvor servicen er placeret, men kan mere generelt beskrives som en måde hvorpå data kan kommunikeres mellem to systemer.

En SOAP service er beskrevet med det XML baserede sprog WSDL. En sådan beskrivelse repræsenterer en kontrakt der beskriver hvordan servicen bliver kaldt, hvilke parametre den skal kaldes med og hvad den returnerer. En sådan kontrakt kan sammenlignes med en metodes signatur i et objekt orienteret programmeringssprog.

Når en service kontrakt er skrevet via WSDL kan den efterfølgende blive kaldt. Neden for er et eksempel på et SOAP service kald og svar for at hente medarbejderen med ID 2.

```
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <m:GetEmployee xmlns:m="http://namespaces.example.com">
      <key>2</key>
    </m:GetEmployee>
  </s:Body>
</s:Envelope>
```

Det efterfølgende svar ser ud som følger.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header />
  <s:Body>
    <GetEmployeeResponse xmlns="http://tempuri.org/">
      <GetEmployeeResult z:Id="i1"
xmlns:a="http://schemas.datacontract.org/2004/07/ODataService.Models"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
        <a:Address>908 W. Capital Way</a:Address>
        <a:BirthDate>1952-02-19T00:00:00</a:BirthDate>
```

```
        <a:City>Tacoma</a:City>
        <a:Country>USA</a:Country>
        <a:EmployeeID>2</a:EmployeeID>
        <a:FirstName>Andrew</a:FirstName>
        <a:HireDate>1992-08-14T00:00:00</a:HireDate>
        <a:HomePhone>(206) 555-9482</a:HomePhone>
        <a:LastName>Fuller</a:LastName>
        <a:PostalCode>98401</a:PostalCode>
        <a:Region>WA</a:Region>
        <a:Title>Vice President, Sales</a:Title>
        <a:TitleOfCourtesy>Dr.</a:TitleOfCourtesy>
      </GetEmployeeResult>
    </GetEmployeeResponse>
  </s:Body>
</s:Envelope>
```

REST er derimod en arkitektur stil som benytter sig af http og rækken af verber som protokollen har defineret til, at kommunikere hvordan data skal behandles. Et konkret eksempel på en REST baseret arkitektur er www, som arbejder på denne måde.

Hvis den SOAP baserede service oven for blev lavet om til en REST baseret ville kaldet være noget enklere:

```
http://localhost/rest/Employees/2
```

Og svaret ville se ud som følger:

```
{   "http://localhost/rest
/#Employees/@Element","EmployeeID":2,"LastName":"Fuller","FirstName":"Andrew","T
itle":"Vice President, Sales","TitleOfCourtesy":"Dr.","BirthDate":"1952-02-
19T00:00:00","HireDate":"1992-08-14T00:00:00","Address":"908 W. Capital
Way","City":"Tacoma","Region":"WA","PostalCode":"98401","Country":"USA","HomePho
ne":"(206) 555-9482","Extension":"3457" }
```

REST kan lige som SOAP returnere svaret i XML, men ved at bruge JSON reduceres mængden af meta-data markant. Modsat REST, så er SOAP specificeret til at benytte XML og er derfor fastlåst her i.

Som det tydeligt fremgår oven for er et REST kald betydeligt enklere end SOAP og den reducerede mængde af meta-data betyder at den endelige mængde af data sendt er omkring 50 % for en REST baseret service.

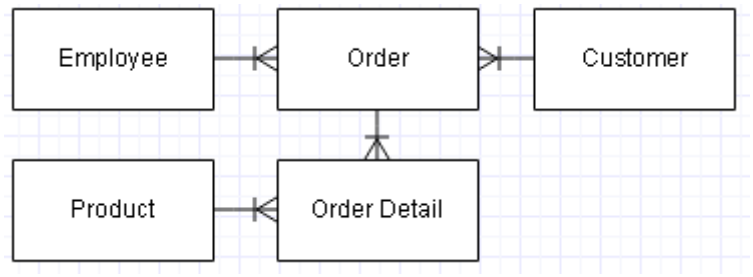
3.2 Data management og modeller

For at kunne udvikle og vedligeholde et stort system effektivt er det nødvendigt at have et overblik over hvad systemets data er, bruges til og hvordan det administreres. Til at opnå dette kan en ”Data Manager” gruppe nedsættes til at forhandle med leverandører og ajourføre en data-model.

3.2.1 Data modeller

En datamodel er en beskrivelse af den data som et system er i besiddelse af og hvordan data relaterer til hinanden. En ofte benyttet måde at illustrere data og dens sammenhæng er ved at bruge en ER model, der med en enkel grafisk notation beskriver tabellerne og hvordan de relatere til hinanden.

Neden for er et eksempel på en enkel forhandlers database. Forhandleren har ansatte, der servicere kunder og hjælper dem med at bestille forskellige produkter.



Figur 1: Eksempel på et ER diagram

Ud over den grafiske ER model er det nødvendigt at entiteterne, men også de enkelte attributter, er beskrevet, så deres betydning står klart fx hvordan de hænger sammen med domænet de benyttes i , hvor data kommer fra, hvor det benyttes osv. Det er også vigtigt at evt. formatering er beskrevet her, så et andet system ikke bryder ned eller tolker forkert på data.

Eksempelvis er det ikke givet at en kundes ID ikke er et stigende tal lige som en medarbejders eller ordrers, men der i mod en bogstav sekvens baseret på kundens navn. Et andet eksempel er at telefonnumre inkluderer landekoden. Hvis ikke disse informationer er tilgængelige ville det første eksempel ikke give mening og ophavet til bogstaverne ville være ukendt. I det andet eksempel har det en betydning i hvordan tallene bliver fortolket, fx kunne ordren blive sendt til et andet land end der hvor kunden ringede fra.

Fordelen ved at have en data model er at der kan skabes et overblik over hvilken data man er i besiddelse af og hvilke hensyn som skal tages når man har behov for at ændre entiteter eller relationer her imellem. Manglen på en sådan model betyder der imod at der ikke er noget overblik og forandringer kan derfor medføre problemer, f.eks. kan data af samme type blive gemt forskellige steder eller eksisterende funktionalitet i systemet kan blive brudt hvis ikke der er styr på hvordan data hænger sammen.

3.2.2 Normalisering

For at reducere risikoen ved forandringer af databasen benyttes teknikken ”Normalisering”. Denne teknik har som mål at sikre at fremtidige rettelser vil have mindst mulig indflydelse på det eksisterende system, hvilket opnås ved at reducere mængden af redundant data. Et eksempel her på er entiteten ”Employee”. Den ansattes informationer kunne gemmes direkte i ordren, men det ville medføre en række problemer.

a) Der ville optræde unødig information

b) Den samme information ville lagres flere gange
c) Der ville mangle information i gamle rækker, når en ny kolonne blev tilføjet
d) Det ville kræve ændringer i mange ordrer hvis den ansatte fx flyttede eller ændrede navn.

Neden for kaldes en SQL query på den ikke-normaliserede database for at finde en ansats adresse og fødselsdato ud fra hans navn. Resultaterne kan dog være forskellige afhængig af om han har skiftet navn, om flere med det navn er ansat, eller om han har flyttet adresse. Ydermere bliver SQL kaldet mere kompliceret fordi det skal kompensere for den manglende struktur og aktualitet af data.

```
SELECT EmployeeAddress, EmployeeBirthDate
FROM Order
GROUP BY EmployeeAddress, EmployeeBirthDate
WHERE EmployeeName = 'Andrew Fuller'
```

En normaliseret database reducerer altså risikoen for at få forældet eller forkert data, samtidig med at den reducerer behovet for at vedlige holde den.

Kombinationen af en normaliseret database og en grafisk datamodel som et ER diagram gør det samlet muligt at danne sig et hurtigt overblik og forståelse af store mængder af tabeller og deres relationer frem for at gennemgå de enkelte tabellers beskrivelse. Mine egne erfaringer er at komplekse data modeller beskrevet via simple tabeller kan tage flere dage at sætte sig ind i, hvor det der imod kan tage timer eller minutter at få en tilsvarende forståelse af strukturen hvis en ER model er tilgængelig.

Endnu værre er det hvis kun service specifikationer er tilgængelige. I så fald vil processen være endnu langsommere, meget information vil gå igen, den samme information vil blive præsenteret på forskellig vis, og ikke al information vil være præsenteret.

3.3 OData

Et systems data kan udbydes på flere forskellige måder. I dag foregår det oftest via SOAP baserede services, hvilket medfører dertilhørende problemer. Som et alternativ kan en service baseret på OData tilbyde en langt højere fleksibilitet, samtidig med at leverandøren får adgang til den tekniske meta-data, inklusiv relationerne mellem entiteterne.

OData gør det muligt at udføre et SQL lignende kald via http således at kun den ønskede data bliver hentet. Herved er der ikke længere behov for generelle services, som returnerer for meget data, eller en lang række af mere specialiserede services

OData definerer en datamodel og en protokol, som gør det muligt for klienter at tilgå og manipulere data, eksponeret via en REST baseret service. Den består af fire hovedkomponenter:[1]

Protokollen: Definerer et query sprog, som benyttes til at hente og manipulere data, fx vil nedenstående http kald fortolkes sådan, at ID'et på alle medarbejdere i London vil blive hentet.

```
http://localhost/odata/Employees?$select=EmployeeID&$filter=City eq 'London'
```

[1] https://msdn.microsoft.com/en-us/data/hh237663.aspx

Servicen: Den er set ude fra ikke meget mere end en REST service, men implementerer OData protokollen og kan herved transformerer http kaldet mellem servicen og datamodellen.

Datamodellen: Beskriver hvordan data er organiseret og er fx genereret ud fra database skemaet. Den står for at transformere ændringer og kald af data mellem servicen og databasen.

Klienten: Kan i realiteten bare være en internet browser da kaldet til servicen ikke er andet end et http kald.

For at hente data sendes en http forespørgsel til servicen. Denne service fortolker forespørgslen vha. protokollen og forbinder det ønskede data med modellen, som efterfølgende transformere kaldet til et database kald.

Et eksempel på et kald kan være at man ønsker at finde alle de kunder i Tyskland, som en specifik medarbejder har solgt produkter til:

```
http://localhost/odata/Orders?$filter=Employee/EmployeeID eq 2 and
Customer/Country eq
'Germany'&$expand=Customer&$select=Customer/CompanyName,Customer/ContactName
```

Ovenstående kald vil I sidste ende blive transformeret til følgende SQL query.

```
SELECT
    Orders.OrderID,
    Orders.CustomerID,
    Customers.ContactName,
    Customers.CompanyName
    FROM  Orders
    LEFT OUTER JOIN Customers AS Customers
        ON Orders.CustomerID = Customers.CustomerID
    WHERE Orders.EmployeeID = 2
        AND Customers.Country = 'Germany'
```

Med funktionalitet som dette og meget mere gør OData det muligt at tilgå data på samme måde, som hvis SQL var direkte tilgængeligt uden at servicen først skal defineres og forhandles på plads med leverandøren. Herved reduceres behovet for specialiserede services , som ellers ville være nødvendigt med SOAP

Hvis ovenstående forespørgsel blev aktuel i et SOAP baseret system ville der være flere forskellige scenarier.

a) Der skulle produceres en helt ny service, hvilket er dyrt pga. udviklingsomkostninger
b) Informationen måtte sammensættes vha. eksisterende services, hvilket ville være som at lægge et puslespil eller overflødig data ville blive hentet.
c) Den vil aldrig blive foretaget

3.3.1 OData i praksis

OData er en relativ ny teknologi og har endnu ikke fundet et solidt fodfæste inden for offentlig IT. Det har derfor været svært at finde konkrete kilder, som har arbejdet med teknologien i praksis. Hos STAR har man kort kigget på mulighederne, men har pga. manglende viden og erfaring ikke gået videre i processen.

Det er dog lykkedes at finde en enkelt leverandør, som stiller en OData løsning til rådighed for et offentligt IT system. ScanJour, nu opkøbt af KMD, leverer et dokument system som de har udviklet en OData service til og som Naturstyrelsen anvender dagligt. I 2015 stod dette system for 40 % af styrelsens sager.

Da OData endnu ikke er specielt udbredt er den udfordret af, at der er færre som har erfaring med den end med SOAP. Den har dog den fordel, at den bygger på velkendte teknikker og herved sker der en udjævning af læringskurven. ScanJour og Naturstyrelsen har erfaring med begge teknologier, og de har den oplevelse, at det ikke kræver mere af udviklerne at sætte sig ind i OData end SOAP.

De sikkerhedsmæssige udfordringer, der er med OData, adskiller sig ifølge ScanJour, ikke væsentligt fra SOAP. Begge protokoller fungerer via http og har derfor de samme fordele og ulemper når det kommer til kommunikation mellem server og klienten.

På service niveau har OData mulighed for at sikre de enkelte entiteter og attributter afhængig af klientens tilladelser. Her ved kan en klient stadig specificere hvilken data der skal hentes, men vil få afvist sit kald hvis forespørgslen beder om data der ikke er tilladelse til.

Hos ScanJour havde man placeret sikkerhedsniveauet ved rækkerne i databasen. Dette betød at uanset om adgangen blev opnået via OData eller SOAP så var behovet for sikkerhed reduceret markant på service niveau. Her ved kunne de fokusere på hvilken data klienten skulle have adgang til, frem for hvilke services som kunne kaldes

Selv med de udfordringer, der måtte være ved at indfører en ny teknologi som OData, så springer to fordele i øjnene, når man taler med ScanJour og Naturstyrelsen: Ydeevne og fleksibilitet.

Ifølge ScanJour yder OData løsningen fire gange så hurtigt som den traditionelle SOAP løsning. En af de store kilder til denne forbedring er, at JSON er et langt mere kompakt format end XML. En pakke i førstnævnte format indeholder et minimum af meta-data, hvor en pakke med XML ville indeholde omkring den dobbelte mængde. Mine egne tests af SOAP og OData bekræfter dette med en reduction mellem 45 og 55 % data sendt retur med identiske kald.

En anden kilde til den forbedrede ydeevne er den høje fleksibilitet, som muliggøres af OData. Denne fleksibilitet sikrer, at man kan nøjes med at hente og transmittere det data, som man ønsker, frem for at bruge en standard service, som henter alle data ud, som den er konfigureret til, uanset om de skal bruges eller ej.

En mulig teoretisk fordel er, at med kombinationen af den forhøjede fleksibilitet, ydelse og de lavere mængder af sendt data, bliver tilgængeligheden af systemet forøget, da risikoen for at netværket bliver en flaskehals er mindre.

Måske den vigtigste fordel ved OData er, at klienten kan nøjes med at tilpasse sit kald uden at involvere leverandøren eller påvirke andre service kald. Med SOAP er det derimod vanskeligt at ændre en service pga. risikoen for at bryde andre systemer. Derfor oprettes der som regel en ny og mere specialiseret service, hvilket skal forhandles med leverandøren.

Denne fleksibilitet kan spare mange ressourcer på udvikling og vedligeholde af systemer. Hvis en kunde hos ScanJour har brug for et nyt data felt, kan det tilføjes uden at bryde eller sløve andre systemer og de som

skal bruge det nye felt kan tage det i brug uden at andre påvirkes. Hvis kunden ønsker at filtrere data på nogle andre parametre end normalt behøver de ikke kontakte leverandøren, men blot tilpasse deres kald.

Sidst men ikke mindst så er det stadig muligt at implementere specielle metoder til at håndterer komplekse eller tunge database kald. Disse kald kan implementeres således at ODatas fleksibilitet ikke går tabt, men der imod understøtter eksisterende kald eller bliver kombineret med nye.

3.3.2 OData og faktorerne

I forhold til de udvalgte faktorer vil jeg mene OData påvirker følgende af dem målt op imod SOAP.

a) **Let at udvikle eller ændre et system:** ODatas fleksibilitet gør det lettere at udvikle et nyt system da dataleverandøren ikke behøver at være lige så meget inde over processen. Sikkerheden skal konfigureres, men ingen nye services behøver som udgangspunkt at blive udviklet.
b) **Let at udskifte et system med et andet:** Da det nye system kan tilpasse sine kald uden at dataleverandøren indblandes kan et system blive udskiftet mere smertefrit. Hvis andre systemer er afhængig af systemets data er det nødvendigt at det eksponerer de samme entiteter, men enkelte services behøves ikke at blive udviklet.
c) **Tilgængelighed:** Der er en teoretisk mulighed for at et system bygget med OData vil have en højere tilgængelighed pga. den reducerede mængde af data på netværket.
d) **Svartider:** Som nævnt yder OData op til fire gange så godt som en SOAP baseret løsning og kan her af besvare forespørgsler hurtigere.

4 STAR

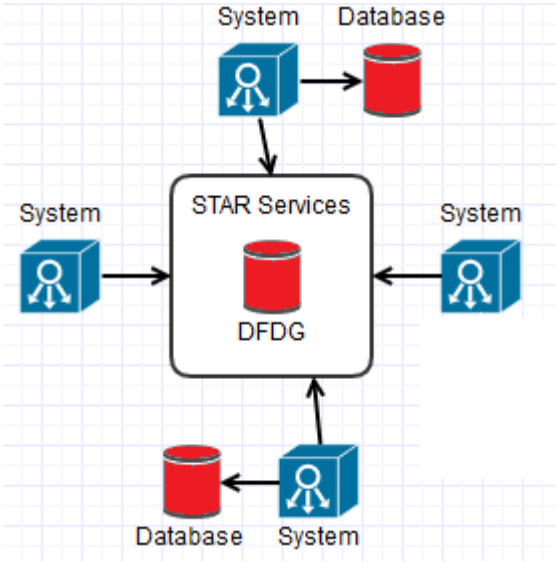
Styrelsen for Arbejdsmarked og Rekruttering (STAR) har som hovedområde den samlede beskæftigelsesindsats. Dette betyder, at de står for jobcentrene, løntilskud, kontanthjælp og web portaler som f.eks. Job-net. Det er også her, at firmaer kan søge om at få en person ansat i f.eks. løntilskud eller virksomhedspraktik.

Til at holde styr på alle disse informationer har STAR udviklet en stor central database. Databasen går under navnet Det Fælles Datagrundlag (DFDG) og bruges til at holde styr på borgerens data i beskæftigelsessystemet.

Pga. de mange forskellige aktører involveret i hele beskæftigelsesprocessen er der koblet omkring 50 forskellige systemer op i mod DFDG her iblandt syv A-kasser, de kommunale jobcentre og diverse ydelsessystemer, foruden alle STAR's egne systemer.

Alle disse systemer tilgår DFDG via SOAP baserede services, som udvikles i samarbejde mellem STAR's arkitekter og det enkelte systems leverandør.

Da data udveksles via DFDG kan hvert system opbygges forskelligt fra andre systemer da disse ikke er direkte afhængige af hinanden. Det betyder, at nogle systemer kommunikerer løbende med DFDG, andre selv har en database og derfor henter og lagrer data i batches, og endnu andre gør det på en tredje måde. Det vigtigste er dog, at den aftalte data bliver lagret i DFDG, så andre systemer kan tilgå den uden indblanding.



Figur 2: STAR's overordnede arkitektur

Som det fremgår af figur X, så kan systemerne køre udelukkende på data fra DFDG, men de kan også have deres egen database, som kun de selv kan tilgå.

Et eksempel på et system er et af sagsbehandlingssystemerne, hvor konklusionerne på diverse sager er lagret hos STAR, men de enkelte sagsakter ligger ude hos de enkelte aktører. DFDG fungerer altså som en samlende enhed mellem de forskellige systemer rundt om i kommunerne.

4.1 Let at udvikle et nyt system

Fire faste releases: For at få en så gnidningsløs udviklingsproces, som muligt, har STAR valgt at have fire releases om året. Det betyder, at de og deres leverandører ved hvilke fire datoer de skal forsøge at ramme og at de også kan planlægge andre opgaver omkring denne viden. Det kan f.eks. være det, at de ikke planlægger batch jobs i den angivne periode, da systemet kan være ustabilt, eller at medarbejderne bliver forberedt på indførslen af nye tiltag.

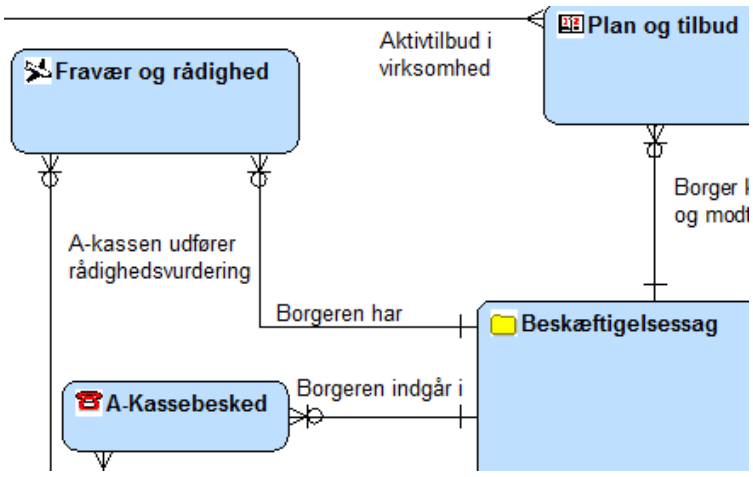
Leverandør samarbejde: I selve udviklingsprocessen indgår STAR og de involverede leverandører i en tæt dialog hvor arkitektur og eventuelle ændringer eller tilføjelser til DFDG diskuteres og konkluderes. Dette sikrer at alle parter på et hvert givent tidspunkt er enige om hvor de er, hvad der skal ske, og i hvilken retning projektet bevæger sig.

Da det enkelte system kun kommunikerer med DFDG begrænser det antallet af involverede parter, hvilket gør det lettere at forhandle om hvilke services de skal bruge, om nye skal oprettes og om der er behov for tilføjelser i databasen. Denne direkte kommunikation sikrer også, at kravene til ét system ikke kræver ændringer i andre systemer. Hvis der er behov for ændringer i en service oprettes der en ny og herved undgår man at påvirke andre systemer, men det betyder også, at der findes en række næsten identiske services, som skal vedligeholdes.

Data model: For at holde overblik over indholdet i DFDG er den beskrevet med en offentlig tilgængelig datamodel inklusiv en emneopdelt ER model. Med en emneopdelt ER model er der to lag, hvor det øverste lag beskriver emneområder og hvordan de interagerer og det andet lag viser det valgte emneområdes entiteter.

Et emneområde består af en række entiteter, som logisk hører sammen og viser hvordan de relaterer til hinanden. Den enkelte entitet har en beskrivelse af hvad den dækker, kilden til data og hvilke begrænsninger den er underlagt eller gennemtvinger.

Entiteternes attributter har ofte en beskrivelse af indholdet og nogle har eksempler på muligt indhold. Det er dog ikke gennemført konsekvent, hvilket betyder at indholdet kan blive fortolket forskelligt fra system til system, eller at viden om indholdet forsvinder med tiden.



Figur 3: Udklip af oversigten over emneområder.

4.2 Let at udskifte et system med et andet

Da al essentiel data er lagret i DFDG kan et system udskiftes med et andet uden risiko for tab af information. Et eksempel på dette er, at en af leverandørerne har formået at slå to ældre sagsbehandlingssystemer sammen til et nyt. Dette system fungerer primært som et præsentationslag for DFDG via de givne services, som de har fået stillet til rådighed og gemmer selv kun meget lidt data, som DFDG ellers ikke ville lagre. Det er altså påvist i praksis, at det er muligt, at udskifte et system med et andet.

4.3 Let at flytte et system til en anden leverandør

DFDG har været flyttet til en anden leverandør én gang tidligere og en ny udbudsrunde er pt. i gang, hvor den forrige leverandør har budt ind. Det er altså påvist, at en flytning er muligt, og forsøget ikke har skræmt eventuelle leverandører væk.

4.4 Tilgængelighed

Den store risiko ved at have et system med en centraliseret database er ifølge STAR, at hvis databasen går ned, går alt ned. Denne risiko kan de dog reducere ved at have redundante servere, som herved sikrer tilgængeligheden af DFDG.

Denne struktur betyder, at et system ikke behøver at være afhængig af mere end ét andet system og med et krav til et systems oppetid på 99,5 procent, vil et system afhængigt af DFDG altså kunne forventes at have en tilgængelighed på 99,0025 procent.

4.5 Sikkerhed

Da al data er centraliseret et sted kan STAR styre hvordan og hvem der kan opnå adgang til hvilken som helst data på et hvert givent tidspunkt. De har ydermere muligheden for at vælge på hvilket niveau de ønsker deres sikkerhed implementeret, hvilket kan variere fra række niveau i databasen til service laget i DFDG.

4.6 Svartider

Den anvendte service er nødvendigvis ikke skræddersyet til at løse det specifikke problem som klienten ønsker løst. Derfor kan den kaldte service returnere mere data end nødvendigt eller det kan være nødvendigt at kalde flere services for at opnå den ønskede effekt. Det betyder, at selv om DFDG er centralt placeret og systemet kan hente den ønskede data direkte, så transporteres og transformeres mere data end nødvendigt både pga. overflødig data, men også pga. ekstra service kald.

I det nye system, som primært fungerer som grænseflade til DFDG, har manglen af lokal data den betydning, at når leverandøren ønsker at beregne statistik på den viden, der er gemt i DFDG, skal de trække meget store mængder data ud, hvilket medfører en stor belastning af systemet.

For at løse dette problem, sørger STAR for, at leverandøren får stillet en delvis kopi af databasen til rådighed. Dette fungerer for begge parter, men da andre leverandører også kan se fordelen ved dette, ønsker de nu også denne mulighed, hvilket i fremtiden kan udvikle sig til en større arbejdsopgave og belastning for systemet.

4.7 Hvis de anvendte OData

Fleksibilitet: Hvis STAR anvendte OData ville det lette udviklingen, vedligeholdelsen og udskiftningen af systemer pga. ODatas fleksibilitet. STAR ville b.la. ikke behøve at oprette nye services til nye systemer, men ville kunne nøjes med at konfigurere hvilke data det nye system skulle have adgang til. Samarbejdet med leverandøren ville fokusere mere på hvilken data der skulle være adgang til og om ny data skulle lagres, frem for forhandlinger og kompromiser om services.

Sikkerhed: STAR havde en konkret bekymring i forhold til OData. De var i tvivl om, hvordan teknologien kunne håndtere de sikkerhedsmæssige udfordringer, men som beskrevet under ”OData i Praksis”, så er dette en unødig bekymring pga. STAR’s nuværende SOAP baserede løsning.

Som alternativ kunne STAR flytte deres sikkerhed ned på database niveau og herved undgå bekymringen om sikkerhed i service laget.

Svartider: Da alle STAR’s systemer er koblet op mod DFDG er der en stor belastning på systemet, da det skal transformere og kommunikere al data. OData ville med sin fleksibilitet og bedre ydelse kunne reducere mængden af data og hastigheden hvor med den vil blive transformeret.

5 SKAT

SKAT er en styrelse under Skatteministeriet, som står for inddrivelse af skatter, afgifter, told, moms, m.m., men før dette kan finde sted må SKAT først indsamle data fra mange forskellige kilder så skatten kan beregnes og efter inddrivelsen, blive registreret.

I foråret 2003 viste en analyse af SKAT's systemportefølje, at kompleksiteten udgjorde både en risiko og en barriere for administrative effektiviseringer og forbedret service[2]. På baggrund af analysen blev det anbefalet at gennemføre en modernisering af IT systemerne med følgende mål for øje:

- Forretningsmæssige:
    - At effektivisere en række centrale arbejdsprocesser
    - At give borgere og virksomheder en bedre og digital service
- IT-udvikling og drift
    - At gøre SKAT's systemportefølje mere fleksibel for forandringer
    - At skabe og fastholde konkurrence om IT-opgaverne

På baggrund af analysen fik SKAT i 2004 bevilliget midler til at igangsætte en gennemgribende modernisering af deres systemportefølje. Dette skulle gøres i faser, hvor fase ét skulle fungere som "proof of concept" for fremtidige projekter samt etablering af den centrale IT arkitektur. Denne fase inkluderede fire projekter, som bestod af det centrale system, kaldet "Infrastruktur Platformen"; deres Portal Platform, her under ToldSkat.dk; samt to projekter, som skulle illustrere håndteringen af indrapportering til og fra det samlede system.

Regeringens økonomiudvalg vedtog at den overordnede arkitektur skulle følge den såkaldte Connect strategi. Målet med denne strategi er at gøre udviklingen hurtigere, billigere og mere sikker. Visionen er, at pakke allerede eksisterende funktionalitet og data ind i services og derved benytte disse ressourcer på nye måder og i nye systemer. Nyudvikling af nye systemer skal så vidt muligt kun ske hvis der er markante forretnings eller lovmæssige ændringer, som påkræver dette, eller hvis ingen leverandører er i stand til at håndtere den anvendte teknologi. Ydermere skal fremtidige systemer være mindre og uafhængige. Tanken bag dette er, at jo mere velafgrænsede og mindre de er, desto nemmere vil det være for leverandører at overskue udvikling og vedligeholdelsesopgaven.

Efterfølgende blev den konkrete arkitektur valgt på baggrund af Videnskabsministeriets anbefaling til alle offentlige myndigheder. Valget faldt på Service Orienteret Arkitektur, der understøtter en opdelt arkitektur hvilket er i overensstemmelse med den valgte strategi.

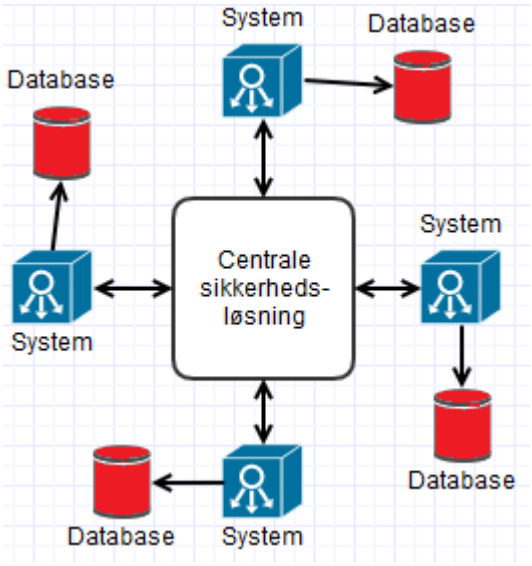
I forbindelse med valget af SOA har SKAT beskrevet en SOAP baseret Reference Arkitektur, som er idealmodellen for deres systemer. Målet med denne model er, at sikre systemerne kan implementeres, administreres og vedligeholdes fleksibelt på samme måde hen over alle systemer. Modellen beskriver en lagdelt arkitektur, samt tre vertikale søjler, som hhv. står for styring af driften, styring af udvikling og vedligehold, og et katalog over services.

Med SKAT's egne ord er service kataloget, System Architect, krumtappen i arkitekturen. Kataloget holder styr på hvilke services der er, hvordan de anvendes, hvad reglerne for brugen af dem er, samt bruger og

[2] Rigsrevisionen: Beretning til Statsrevisorerne om SKAT's systemmodernisering(2015)

rolle administration. Dette katalog skulle gøre det muligt at finde ud af hvilke services der er på tværs af alle systemerne og herved åbne muligheden for genbrug af dem, hvilket skulle reducere behovet for ny udvikling.

Hvert enkelt system har sin egen database og andre systemer må tilgå denne data ved først at kontakte den centrale sikkerhedsløsning og her fra de services som de enkelte systemer eksponerer.



Figur 4: SKAT's overordnede arkitektur

For at få et klart billede af hvordan SKAT's arkitektur fungerer, ser jeg på den via systemet Digitalt Motor Register (DMR). Dette system blev udviklet af firmaet Netcompany efter en tidligere leverandør måtte opgive. Projektet er karakteriseret ved at have en kravspecifikation med en meget høj detalje grad og et sidetal på omkring 16.000, men det lykkedes at realisere det.

DMR holder styr på alle kørertøjer i Danmark, hvilket betyder, at mange andre myndigheder og organisationer har brug for eller levere data til systemet.

5.1 Let at udvikle og ændre et system

Leverandør samarbejde: Arkitekturen og strategien betyder til sammen at det enkelte system har mange grænseflader til andre systemer. Dette medfører, at ændringer af og i endnu højere grad ved udviklingen af et nyt system involveres en lang række af aktører, da de involveredes systemers leverandører skal inddrages direkte i processen.

Som udgangspunkt foregik disse møder uden SKAT's folk blev involveret, men da hvert systems leverandør havde sine egne interesser at varetage, skete det ofte, at samarbejdsvanskeligheder i sidste ende tvang SKAT til, at gå ind og diktere hvad der skulle gøres i forhold til grænsefladerne.

Manglende datamodel: Den dokumentation der er på SKAT's data er produceret efter i idriftsættelse af en ekstern konsulent, så betydningen af data kan være gået tabt. Her ud over er viden om denne

dokumentations eksistens ikke blevet udbredt og derfor kender leverandørerne ikke nødvendigvis til den. Ydermere er det usikkert, om dokumentationen efterfølgende er blevet vedligeholdt.

Proxy Services: En af grundtankerne er, at genbruge så meget som muligt for at reducere behovet for ny udvikling og herved spare tid og penge. Omfanget af genbrug er dog begrænset og gjort meget komplekst pga. kravet om en meget skarp adskillelse. Dette giver sig til udtryk ved, at når et system skal kalde en service hos et andet system, skal det første system oprette sin egen service og kalde den. Denne nye service skal efterfølgende kalde den oprindelig ønskede service.

Begrundelsen for dette er, at de to systemer skal kunne eksistere helt for sig. Denne meget høje adskillelse er altså med til at forøge kompleksiteten. I praksis "genbruger" man godt nok servicen, men det forøger behovet for ny udvikling og vedligeholdelse og i realiteten er det ene system stadig afhængig af det andet for at løse opgaven. Der ud over bliver denne udefra sete identiske proxy-service også registreret i SKAT's System Architect og kommer derfor til at optræde flere gange, hvilket igen gør det mindre gennemsigtigt ift. hvilken service, der skal kaldes.

Uvarslede serviceændringer: Netcompany oplevede, at services kunne blive ændret, tilføjet eller taget ud af drift uden varsel. Derfor udviklede de et abstraktionslag, som skulle bearbejde sådanne pludselige ændringer uden at systemet brød sammen. Dette har resulteret i at Netcompany kan overkomme disse ændringer, men de beretter at andre, som benytter sig af deres services, f.eks. forsikringsselskaber ofte står med det problem, at deres systemer ikke virker.

Sikkerhedsløsningen: Fordi DCSL, ifølge Netcompany, både er ustabil og over-engineered valgte de at implementere deres egen simplere udgave i DMR for at reducere den indvirkning som DCSL havde på både udviklingen og driften af det nye system.

5.2 Let at udskifte et system med et andet

Pga. systemernes høje kompleksitet og indbyrdes afhængighed af hinanden, har der ikke været nogen udskiftninger. Der ud over er der ikke et komplet overblik over data og dets relationer, samt hvilke systemer der løser hvilke opgaver. Til sammen gør dette det svært at sende systemer i udbud.

5.3 Let at flytte et system til en anden leverandør

Jeg har ikke kunnet finde oplysninger om, at det skulle være blevet forsøgt at flytte et eneste af SKAT's systemer. Jeg antager derfor, at den føromtalte indbyrdes afhængighed og manglende overblik har stået i vejen for dette.

5.4 Tilgængelighed

Dominoeffekt af nedbrud: Pga. de mange indbyrdes afhængigheder kan et enkelt systemnedbrud starte en dominoeffekt af nedbrud. Dette sker fordi et kald til en service ikke nødvendigvis slutter med det første kald, men starter en række af efterfølgende kald til endnu flere services for at samle den fornødne data, som er spredt ud over hele systemporteføljen.

Da alle service kald skal godkendes af DCSL mindst én gang, også selv om det konkrete system ligger inde med den ønskede data, skal alle kald ind over DCSL, så hvis dette system ikke er tilgængeligt , vil ingen services fungere.

Det enkelte systems tilgængelighed er altså afgjort af en række systemers tilgængelighed og kan hurtigt falde drastisk. Med et krav til systemets oppetid på 99,5 % og med det krav at alle service kald skal godkendes af DCSL, som vi må antage har samme krav til sin oppetid, så vil et systems tilgængelighed være 0,995^2 = 99,00 %. Systemets tilgængelighed er altså fra start af under de ønskede 99,5 % og dette er hvis systemet kun kalder en service hos sig selv. Hvis et service kald der i mod starter en række af kald, som f.eks. skal inden om politiet og efterfølgende CPR registret, så er vi oppe på to ekstra systemer og vi ender med en tilgængelighed på 99,5^4 = 98,01 %.

Manglende stabilitet: Et stort problem har været at DCSL har været meget ustabil. Netcompany oplevede at da DMR blev sat i drift fik de ikke svar på omkring 33 % af deres kald til DCSL. Dette har selvfølgelig betydet at mange opgaver har taget lang tid eller har fejlet helt og forsøgt startet forfra.

I forsøget på at forenkle brugen og minimere afhængigheden af DCSL, konstruerede Netcompany en mindre udgave af systemet, som kunne håndtere brugere og roller så vidt muligt. Der ud over kunne de efter at have indsamlet dokumentation for problemet kontakte SKAT og leverandøren af DCSL og få dem til at forbedre systemet.

Kopiering af registre: De mange problemer med manglende tilgængelighed har haft flere forskellige konsekvenser, men et af dem, som koster mange ressourcer er det at flere leverandører ikke har tillid til, at de systemer de er afhængige af, er tilgængelige. Dette har medført en tendens til at registre bliver kopieret for at minimere risiciene ved afhængighederne.

Et eksempel på dette er, at systemer, som er afhængige af DMR i løbet af natten kopierer hele registret. Det betyder, at selv om DMR skulle være nede, så risikerer de andre systemer ikke at de ikke kan levere deres services. Dette løser selvfølgelig problemet med oppetid, men det skaber andre problemer. Det mest håndgribelige af disse er at det belaster DMR enormt at trække sig selv ud og sende en kopi til andre systemer. DMR er givet vis mindre aktiv om natten, men på trods af dette kører en del jobs på dette tidspunkt og er pt. På et stadie hvor disse kopieringer knap kan nå at blive gennemført før natten er omme.

Det har også den betydning, at hvis data hos DMR bliver ændret efter at et system har lavet et udtræk, har det pludselig data, som ikke længere er aktuelt. Det har her af den konsekvens, at to systemer kan komme til at handle på data af forskellige værdier.

Uvarslet service ændringer: Netcompany oplevede, at services kunne blive ændret, tilføjet eller taget ud af drift uden varsel. Derfor udviklede de et abstraktionslag, som skulle bearbejde sådanne pludselige ændringer uden at systemet brød sammen. Dette har resulteret i at Netcompany kan overkomme disse ændringer, men de beretter at andre, som benytter sig af deres services, f.eks. forsikringsselskaber ofte stod med det problem, at deres systemer ikke virkede.

5.5 Sikkerhed

SKAT har bygget deres system op sådan, at et kald til en service skal godkendes af DCSL som det første. Her ved har SKAT altid kontrol over adgangen til services og kan igennem DCSL kontrollere, om kalderen har de fornødne tilladelser, før kaldet går igennem.

Pga. kompleksiteten og ustabiliteten af DCSL har Netcompany implementeret deres egen miniudgave, som forenkler bruger/rolle strukturen. Dette har umiddelbart ikke medført sikkerhedsproblemer, men hvis det

er kutyme, at alle leverandørerne gør dette, må det anses som en sikkerhedsrisiko da der kan ske fejl i implementeringen.

5.6 Svartider

Ifølge SKAT's egne arkitekter er der ikke problemer med svartiderne, men konsulenter og Netcompany er af en anden opfattelse.

Laveste fællesnævner: Et service kald skal først godkendes af DCSL før det kan gå videre til den ønskede service. Her efter kan det starte en række af service kald, som spreder sig som ringe i vandet. Dvs. at hvis bare en af de kaldte systemer, inklusiv DCSL, har en dårlig ydelse vil dette resultere i en høj svartid.

Kopiering af registre:. Den lange række af service kald og det faktum, at Netcompany ofte må foretage det samme kald flere gange før de får svar, har betydet at de har set sig nødsaget til at tage kopier af andre systemers registre for at kunne overholde de kontraktmæssigt bestemte svartider.

Kopieringen af registre har øjensynligt udviklet sig til en generel nødvendighed og derfor kopieres DMR's register også af mange forskellige systemer hver nat. Dette har udviklet sig i en sådan grad, at Netcompany nu er ved at blive bekymret for, om systemet kan nå at færdiggøre de natlige processer i tide til at kunne levere optimalt i dagtimerne.

Ikke specifikke services: Den anvendte service er nødvendigvis ikke skræddersyet til at løse det specifikke problem som klienten ønsker løst. Derfor kan den kaldte service returnere mere data end nødvendigt eller det kan være nødvendigt at kalde flere services for at opnå den ønskede effekt. Det betyder, at selv om et system indeholder al den ønskede data, så kommunikeres og transformeres mere data end nødvendigt både pga. overflødig data, men også pga. ekstra service kald.

Teori, kald af forkerte services: Pga. de mange proxy services i SysArc antager jeg det må være en mulighed at komme til at kalde en proxy i den vildfarelse af, at det er den rigtige service. Dette vil forsinke et kald yderligere, da denne proxy må sende kaldet videre til den oprindelige service.

5.7 Hvis de anvendte OData

Leverandør samarbejde: Da SKAT har haft en del problemer med serviceforhandlinger mellem leverandørerne ville OData som det første kunne lette dette problem pga. at dataleverandøren ikke behøver at være lige så meget inden over processen pga. OData's fleksibilitet. Skat ville ganske enkelt kunne give leverandøren adgang via DCSL uden dataleverandørens indblanding.

Formindsket kompleksitet: OData ville gøre det muligt for en leverandør at udgive sin data til andre systemer uden at skulle udvikle og vedligeholde en lang række af services. Dette ville gøre det nemmere at udskifte systemer, da det nye system kun skulle sørge for at en tilsvarende OData datamodel, hvor efter de andre systemer vil kunne trække data fra systemet.

Fleksibilitet: De problemer som Netcompany og forsikringsselskaberne oplevede med konstant forandrende services kunne blive reduceret vha. OData's fleksibilitet. Reduktionen ville dog være afhængig af om SKAT ville fortsætte med at ændre og fjerne datafelter.

Ydelse: Da systemerne allerede er pressede for at kunne levere de kontraktmæssigt bestemte svartider, ville OData's forbedrede ydelse kunne give et større pusterum.

Opfyldelse af målsætning: Ved at anvende OData ville det potentielt blive muligt at opnå de tekniske mål, som var begrundelse for systemmoderniseringen. Det ville blive nemmere at tilføje nye eller ændre eksisterende systemer, samtidig med det potentielt ville blive muligt at udskifte eksisterende systemer. Dette er dog kun en mulighed, da systemerne stadig er stærkt afhængig af hinanden og der ikke automatisk følger et overblik med.

6 Konklusion

For at komme frem til hvilken af de to tilgange der er den bedste, Vil de to cases blive sammenholdt for hver faktor, herved vil det blive belyst hvilken af de to arkitekturere, som er den mest hensigtsmæssige.

6.1 Faktorerne

6.1.1 Let at udvikle og ændre

STAR's udviklingsproces har faste tidsrammer samt et begrænset antal interessenter, hvilket gør deres proces relativ enkel og smertefri, da det reducere risikoen for konfliktende interesser. Dette står i skarp kontrast til SKAT, hvor udviklingen af et nyt system kan inkludere en lang række forskellige leverandører, som skal samarbejde, uden nødvendigvis at have nogen interesse i at indgå i kompromiser. Denne mangel på kompromisvillighed har ofte tvunget SKAT til at overtage forhandlingen og gennemtvinge de fornødne ændringer.

STAR's datamodel gør det muligt for leverandøren at forberede sig og komme med forandrings forslag tidligt i processen og kan fra start have en god forståelse for hvordan data relatere til hinanden. Hos SKAT er det nødvendigt at leverandøren først finder ud af hvilket system indeholder den ønskede data og der efter indgår i forhandlinger med dette systems leverandør. Her til kommer, at hvis intet system indeholder den ønskede data vil det nye system selv lagre denne data efterhånden som den bliver tilføjet. Dette betyder at relateret data kan blive spredt ud over mange forskellige systemer, hvilket medføre en række problemer.

a) Med data spredt over flere systemer vil udviklingen af nye systemer kræve, at flere leverandører involveres i processen
b) Det bliver svære at lokalisere data og risikoen for at det bliver lagret flere gange opstår
c) Forståelsen for data og relationer her imellem risikeres at gå tabt pga. manglende overblik

Der ud over har SKAT stramme krag til kald af services og en meget kompleks sikkerhedsløsning som forøger kompleksiteten af udviklingsprocessen, hvilket har betydet at nogle leverandører har set sig nødsaget til at implementere et ekstra lag alene til at håndtere SKAT's sikkerhedsløsning.

Konklusion

Med den fælles database kan al fælles data findes et sted og systemleverandøren skal derfor kun forhandle med én anden part. Med SOA kan data være spredt ud over flere systemer, hvilket betyder at en systemleverandør i SOA systemet, først skal lokalisere hvor den ønskede data er og efterfølgende forhandle med hver dataejer

6.1.2 Let at udskifte et system med et andet

Hos SKAT har man ikke formået at udskifte nogle systemer, hvilket skyldes mange afhængigheder og manglende overblik over data og opgaver. Der i mod har STAR formået at udskifte systemer uden større problemer.

Konklusion

Det er ikke klart hvad den præcise årsag til succesen med den fælles database er, men det står tilbage, at de færre afhængigheder gør det nemmere at overskue hvilke konsekvenser en udskiftning af et system vil medføre end med SOA.

6.1.3 Let at flytte et system til en anden leverandør

Hos STAR har både systemer og DFDG flyttet leverandør og en udbudsrunde af sidstnævnte er pt. i gang. Som diametral modsætning, har SKAT ikke flyttet et eneste system, hvilket jeg antager er pga. det manglende overblik.

Konklusion

Intet her viser hvorfor det kun lykkedes at udskifte leverandør med en fælles database, det kan blot konstateres, at det har været muligt, selv om der I princippet ikke skulle være nogen forskel når et system sendes i udbud.

6.1.4 Tilgængelighed

Hos både SKAT og STAR har man en flaskehals hhv. i form af DCSL og DFDG. Begge disse punkters tilgængelighed kan sikres vha. redundans, men her ophører ligheden også. Hvis DCSL skulle gå ned hos SKAT vil intet fungere da selv servicekald til systemets egne services skal godkendes af DCSL. Hos STAR vil et tilsvarende nedbrud blokere for al tilgang til det fælles data, men systemet vil stadig kunne fungere så længe det kun skulle benytte sin lokale data.

Med STAR's fælles database er et system højest afhængig af ét andet system, hvorimod et system hos SKAT minimum er afhængig af et andet og muligvis flere. Der ud over kan en kaldt service yderligere være afhængig af andre services. Dvs., at et eneste nedbrud kan medføre en dominoeffekt af nedbrud. Sat på formel betyder det, at tilgængeligheden hos STAR er (99,5 %)^2 = 99 %, hvor den hos SKAT er (99,5 %)^n, hvor n er antallet af systemer og minimum 2. Det betyder, at det værst mulige for STAR er det samme som det bedst mulige for SKAT.

Hos både STAR og SKAT oplever man kopiering af registre, men hvor det hos STAR er en undtagelse er det en nødvendighed hos SKAT. I STAR's tilfælde drejer det sig om et system, som ikke selv gemmer data lokalt, men har brug for at indsamle data til statistik. Hos SKAT der i mod er det bl.a. et spørgsmål om systemets tilgængelighed pga. de mange afhængigheder.

Hos SKAT introducerer denne kopiering et nyt problem: systemer risikere at handle på forældet data, hvilket kan resultere i fejlberegninger eller opkrævninger hos de forkerte personer.

Konklusion

Den fælles database har langt færre afhængigheder og er heraf langt mindre udsat i forhold til manglende tilgængelighed. Den fælles database opnår en tilgængelighed på (99,5 %)^2 = 99 %, hvor SOA har en tilgængelighed på (99,5 %)^n, hvor n er antallet af afhængige systemer og minimum 2, altså i bedste fald det samme som med en fælles database.

6.1.5 Sikkerhed

Hos både SKAT og STAR har man centraliseret sikkerheden. Hos SKAT vha. DCSL, som skal godkende alle service kald og hos STAR har man den fulde kontrol over adgangen til DFDG.

Hos SKAT har en systemleverandør dog oplevet, at DCSL har givet så mange problemer, at de selv har implementeret deres egen forenklede sikkerhedsløsning. Dette har pt. ikke givet nogle problemer, men hvis det udvikler sig til en trend må det anses som en potentiel risiko.

Konklusion

Med både fælles database og SOA kan den fornødne sikkerhed opnås

6.1.6 Svartider

Hos STAR kan et system opnå adgang til data uden mellemled direkte fra DFDG. Hos SKAT kan et servicekald resultere i en lang række efterfølgende kald, som skal besvares før det endelige svar kan returneres. Dette forøger i sig selv svartiden, men hvis et eller flere systemer yder dårligt, vil alle kald her til blive forsinket.

Pga. de høje svartider er der opstået en tendens blandt SKAT's leverandører til at kopiere andre systemers registre. Dette sker for at kunne overholde de kontraktmæssigt bestemte svartider, men omfanget er ved at vokse sig til en størrelse hvor det enkelte system ikke kan nå at færdiggøre de natlige processer før arbejdsdagen begynder. Hos STAR bliver der også foretaget kopiering af dele af DFDG. Dette sker dog pga. statistik og er endnu ikke et problem, men noget man er opmærksom på.

Konklusion

SOA har spredt data over flere systemer og derfor kan det være nødvendigt at afvente flere service kald for at få den samme data, som med en fælles database, hvor al data er samlet ét sted.

6.1.7 Samlet Konklusion

Det udslagsgivende på tværs af faktorerne er spredningen af data. SOA spreder data i en grad så det medfører manglende overblik og mange afhængigheder på tværs af systemerne. Det må derfor konkluderes at den fælles database løser problemet med deling af fælles data mellem systemer mest hensigtsmæssigt da arkitekturen løser netop dette.

6.2 Anvend OData

Den primære fordel ved OData er, at den enkelte leverandør selv kan specificere de services, der er brug for. Denne fleksibilitet medfører så en række afledte fordele, som både SKAT og STAR kan drage nytte af.

Leverandør samarbejde: Det reducerede behov for forhandlinger mellem dataejer og databruger vil styrke både SKAT og STAR, men specielt SKAT vil kunne drage fordel af dette da de pt. har konflikter på området.

Lavere udviklingsomkostninger: Hos både SKAT og STAR ville OData's fleksibilitet betyde, at den omkostningstunge udvikling af nye services ville blive mindsket. I skat's tilfælde pga. føromtalte måde at håndtere ny data på og hos STAR skulle nye datafelter muligvis tilføjes i DFDG, men det ville ikke medføre udvikling af nye services.

Udskiftning af systemer: Hos både SKAT og STAR ville OData gøre det nemmere at udskifte et system. I begge tilfælde gælder, at hvis et nyt system sørger for at der er adgang til den samme OData datamodel, ville det ikke være nødvendigt at udvikle en lang række af specifikke services.

Ydelse: Den forbedrede ydelse, som OData kan levere, vil næppe løse nogle af de direkte problemer, men da hverken SKAT eller STAR formentlig vil få færre systemer med tiden, så vil den være med til at fremtidssikre systemerne. Der til vil den give SKAT's leverandører et tiltrængt pusterum på de pressede svartider.

7 Refleksion

Et generelt tilbud fra SKAT til IT studerende om at skrive speciale omkring SKAT´s arkitektur, var udgangspunktet for denne rapport. Dette tilbud ledte til et møde med SKAT og styrelsens chefarkitekt, som beskrev en vellykket arkitektur med få problemer.

Kort tid efter kontakten med SKAT var blevet etableret, fik jeg mulighed for at tale med en konsulent, som tidligere havde været centralt placeret hos SKAT. Dette interview gav et noget anderledes perspektiv på tilblivelsen af SKAT´s arkitektur og centrale systemer, samt udviklingen af nye systemer.

Med henvisninger fra indledende kilder og heldige sammentræf blev det muligt at få adgang til dokumenter og nye kontakter, som understøttede føromtalte konsulents perspektiv. Bl.a. fik jeg fat i en rapport fra Rigsrevisionen, som muligvis ikke nævnte arkitekturen, men gav et noget broget billede af projektledelsen og udviklingsprocesserne, hvilket understøttede konsulentens påstande.

En af de nye kilder var hos Netcompany, en større IT-virksomhed, som havde været underleverandør til bl.a. CSC på nogle af systemerne i SKAT´s portefølje. Der ud over er de også leverandør af systemet det Digitale Motor Register, som de overtog efter at IBM opgav.

Det samlede billede som var begyndt at tone frem var ikke videre flatterende og det var svært at dreje det i en konstruktiv retning. Derfor fik jeg via min vejleder mulighed for at mødes med STAR, og høre hvordan de havde sammensat deres arkitektur.

STAR's arkitektur med en fælles database, gjorde det muligt at spejle arkitekturen hos SKAT og herved se hvilke konsekvenser SOA kan få inden for udvalgte områder.

Ud fra begrundelsen for SKAT´s systemfornyelse, samt gennemgående problemstillinger i de to cases, udvalgte jeg nogle faktorer som kunne vise hvor der var nogle tydelige fordele og ulemper, samtidig med at begrundelsen kunne blive set mere generelt, da flere af argumenterne kunne genanvendes i anskaffelsen af nye systemer.

Med fokus på faktorerne kunne jeg se at SKAT og STAR havde nogle sammenfaldne problemer og jeg kiggede derfor nærmere på teknologien OData. Med denne teknologi blev det muligt at se hvor svagheder i SOA kunne løses eller afhjælpes, ved at tilgå problemet på en anden måde end ved traditionelle services.

For at få et praktisk perspektiv på OData, forsøgte jeg at få fat i nogle virksomheder som leverede OData løsninger til et offentligt system. Efter flere forsøg fik jeg ved et tilfælde kontakt til Naturstyrelsen og ScanJour. Naturstyrelsen anvender OData i et af deres systemer, som er leveret af den nu KMD opkøbte virksomhed, ScanJour.

7.1 SKAT, hvordan gik det så vidt?

En af de ting jeg er forundret over er, at man ikke har set de mange problemer komme på lang afstand. Der er så mange og så stærkt koblede services at det er helt åbenlyst, at der må opstå nogle gevaldige problemer, ikke mindst fordi de enkelte leverandører ikke har nogen grund til at indgå kompromiser.

Så hvordan kunne det gå så galt og hvor ligger ansvaret?

Efterhånden som jeg har gennemgået dokumenter og talt med både arkitekter i SKAT, Netcompany og eksterne konsulenter får jeg fornemmelsen af, der er kommet et teoretisk baseret diktat fra toppen og den efterfølende stab har forsøgt at tvinge virkeligheden til at passe til teorien.

Et af dokumenterne viser da også at tidspunktet hvorpå beslutningen om strategien og SOA er blevet taget ikke overraskende falder oven i det tidspunkt hvor hypen om SOA var stor[3], men stadig relativ utestet, specielt i den skala, som SKAT's systemportefølje udgøre.

Men måden hvorpå SKAT benytter og håndterer SOA på, kommer ikke af et valg foretaget af deres egne IT folk. Når jeg gennemgår tidslinjen fremgår det, at Regeringens Økonomiudvalg har vedtaget strategien og det daværende Videnskabsministerium generelt har anbefalet SOA. Hvem der foretog det endelige valg kunne ikke engang SKAT's chef arkitekt svare på, for han blev ansat efter valget var blevet taget. SKAT har altså hyret folkene med den praktiske viden til at implementere systemet, men dem med ansvaret for beslutningen har ikke været en del af den efterfølgende proces.

Uanset hvilke forklaringer man ender med, hvorpå på de mange problemer kunne opstå, så ligger de to tekniske begrundelser for systemmoderniseringen tilbage som en ironisk beskrivelse af hvad man ikke opnåede. En af de mulige forklaringer er dog, at der ikke har været det fornødne overblik. SKAT har bl.a. ikke benyttet sig af datamodeller til at fastholde overblikket fra starten af, men først fået produceret dem flere år efter. Der ud over lod SKAT leverandørerne forhandle services på egen hånd og der var derfor ikke nogen tilstede, som vidste hvordan relationerne mellem data var og herved sikre den nødvendige sammenhæng.

7.2 SKAT i fremtiden

Det er meget nemt at pege fingre, men hvad mener jeg man burde gøre? Med baggrund i denne rapport er det ikke overraskende, at jeg ville starte med at danne mig et overblik over data og hvilke opgaver de forskellige systemer løser.

Først når et sådan overblik er dannet kan løsningen af problemerne blive påbegyndt og systemerne kan sendes i udbud, hvilket var en af begrundelserne for moderniseringen, men også et krav ifølge EU regler. Med det nye overblik og på baggrund af udbudsreglerne, ville jeg gradvist sende systemer i udbud og løbende samle fælles data i en fælles database. Denne fælles database ville efterfølgende udbyde data via OData.

Den gradvise transition af systemporteføljen var en tilgang som STAR berettede, at de i øvrigt anvendte da de afviklede systemet "Amanda" og opbyggede deres nuværende system.

Udvælgelsen af hvilke systemer der burde sendes i udbud først, kunne ske på baggrund af f.eks. antallet af afhængigheder. Det kunne være de systemer, som færrest er afhængige af og herved sikre at færrest af de nuværende systemer ville have brug for at tilpasse sig til den nye struktur. På den anden side kunne det være de systemer, som flest var afhængige af. Dette ville medføre at mange skulle tilpasse sig, men antallet af afhængigheder ville falde hurtigt.

[3] http://www.datamation.com/entdev/article.php/3671061/SOA-Hype-vs-Reality.htm

Til slut ville jeg nok finde en anden måde at sende systemerne i udbud på. Efter at have set på den 16000 siders lange kravspecifikation til DMR, er jeg af den overbevisning, at man burde fokusere på hvilken data der er brug for, hvilke opgaver man ønsker løst og mindre på hvordan de løses.

7.3 OData, næste generations hype?

Efter min mening, nærmer OData sig i dag omtrent det sted som SOAP befandt sig på i midt 0'erne, men er langt mere anonym. Teknologien er begyndt at finde indpas i virksomhederne, men jeg havde svært ved at finde offentlige styrelser, som benyttede den. Hvad denne noget mere tilbagetrukne udbredelse og omtale skyldes, så kan jeg ikke lade være med at tænke om en af mulighederne er, at erfaringerne fra SOA og der til hørende hype, har gjort arkitekterne mere konservative og forsigtige med at springe til nye teknologier.

I denne rapport ender jeg med at anbefale OData, men hvordan adskiller min anbefaling sig egentlig fra alle de konsulenter, som op igennem 0'erne anbefalede f.eks. SKAT at bruge SOAP? Er OData ikke bare den næste generations hypede teknologi?

For det første strider min anbefaling af OData ikke imod SKAT's behov. Som jeg kommer ind på i kapitel 3.1 SOA, så fremhæves det ofte, at "services er forretningsbaserede", men dette følger SKA ikke, hvilket fremgår ved, at systemerne kopiere andre systemers registre og udtrækker data direkte. Derimod er denne selektive udvælgelse af data i realiteten det OData gør muligt, bare uden kopiering af hele registre.

For det andet, så bygger OData på velkendte teknologier her i blandt arkitekturstilen REST og formatet JSON, hvilket giver den teknisk ballast. Det succesfulde eksempel med Naturstyrelsen understøtter også op om anbefalingen. På trods af at systemet ikke er i samme skala som systemerne hos SKAT og STAR, så er det ikke et lille, isoleret system og det fungerer derfor godt som et skridt på vej til de helt store systemer.

På den anden side er det ikke undersøgt hvordan teknologiens medfølgende fleksibilitet vil påvirke klienternes adfærd og her af systemarkitekturen, i den skala som f.eks. SKAT's systemportefølje repræsentere. Jeg vil dog mene at forudsætningerne for at lave denne undersøgelse er på plads og at modsat SOAP, så introducere OData ikke nogen åbenlyse problemer.