

Ordrupvej 101
2820 Gentofte

tel. +45 44 50 21 50
fax. +45 39 64 56 37
www.lector.dk

Service Orienteret Arkitektur

- en teknisk betragtning

d. 28. august 2006

Indhold

| | |
|---|----------|
| Indledning | 4 |
| Forventet publikum | 4 |
| Forkortelser | 4 |
| Introduktion til Service Orienteret Arkitektur (SOA) | 5 |
| Baggrund og formål | 5 |
| Koncepter og begreber | 5 |
| Arkitektur og modellering | 6 |
| Konceptuel arkitektur | 7 |
| Generelle guidelines | 9 |
| Sprogspecifik | 9 |
| C# | 9 |
| WCF:..... | 10 |
| Java | 11 |

Indledning

I de senere år er begrebet "Service Orienteret Arkitektur" (SOA) begyndt at optræde mere og mere i IT-branchen og i den generelle debat. SOA er et godt "brugt" begreb som desværre ikke altid fremstår lige godt defineret ensige beskrevet.

I dette dokument præsenteres Lector's opfattelse af begrebet SOA; i denne omgang set ud fra en teknisk betragtning. En kommerciel betragtning¹ på SOA eksisterer sideløbende med nærværende dokument.

Forventet publikum

Det forventede publikum til dette dokument er teknisk personale med et kendskab til .NET (C#) eller Java, som er de miljøer, der primært benyttes i Lector.

Forkortelser

| Forkortelse | Navn | Beskrivelse |
|-------------|--------------------------------|---|
| SO | Service Orienteret | |
| SOA | Service Orienteret Arkitektur | |
| WS* | WebService * | Samlet betegnelse for en række webservicestandarder |
| SCA | Service Component Architecture | |

Referencer:

1. "SOA – en kommerciel betragtning", Lector whitepaper 2006
2. Service-oriented modeling and architecture: <http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/> (September, 2006)

¹ "SOA – en teknisk betragtning", Lector 2006

Introduktion til Service Orienteret Arkitektur (SOA)

Baggrund og formål

Dette afsnit har til formål at introducere hvad SOA er for en størrelse. Vinklen er hovedsageligt teknisk. For en mere ikke-teknisk gennemgang af baggrunden for SOA se "SOA en kommerciel betragtning" 1.

En af de store fordele ved SOA er, at muligheden for at skabe platformsuafhængige systemlandskaber er tilstede. Da SOA oftest anvender 'tekst' (SOAP) som kommunikationsmiddel, vil de fleste platforme være istand til at kommunikere med hinanden da disse alle har understøttelse af tekst-begrebet (SOAP).

Koncepter og begreber

Lectors opfattelse² af begreberne **Service** og **SOA** er følgende:

Service:

*En **service** er en selvstændig funktionalitet som kan anvendes autonomt uden tvungne afhængigheder af andre eksterne systemer/forhold.*

Service Orienteret Arkitektur:

*At lave en **Service Orienteret Arkitektur** (SOA) er arbejdet med at sammensætte diskrete services til et koherent systemlandskab der understøtter de forretningsprocesser virksomheden har identificeret som værende "drivere" af forretningen.*

og/eller

*En **Service Orienteret Arkitektur** (SOA) er betegnelsen for et etableret systemlandskab som gør anvendelse af diskrete services for at tilbyde et sammenhængende og koherent IT-system til at bistå de forretningsafledte opgaver.*

² "SOA – en kommerciel betragtning", Lector WhitePaper 2006

Arkitektur og modellering

Et SOA-baseret system adskiller sig fra "traditionelle" objektorienterede (OO) systemer på flere områder.

I et traditionelt OO-system har man som udgangspunkt kendskab til placeringen af det objekt man ønsker at anvende. Dette instantieres til et konkret instans, og metoder kan herefter kaldes på denne instans. I en serviceorienteret verden, kan placeringen og implementeringen af en given service man ønsker at anvende være noget mere diffus. Hvilken teknologi denne service er implementeret med, er heller ikke nødvendigvis kendt ej heller normalt relevant set fra klientens synspunkt.

Note: Det skal nævnes, at der eksisterer design patterns som abstraherer denne placering også i OO-systemer.

SOA kræver en ny måde at tænke på. Dette ses blandt andet i selve servicebegrebet og de centrale aktører, som er i spil. I SOA er de primære centrale aktører:

- Serviceprovider: Denne stiller en konkret service til rådighed.
- Locationprovider: Denne stiller placeringen af en ønsket service til rådighed
- Serviceconsumer: Klienten som gør anvendelse af servicen

Som udgangspunkt skal serviceconsumeren ikke vide, hvordan servicen er implementeret. Potentielt skal denne serviceconsumer heller ikke vide hvor denne er placeret. Dette afsnit beskriver, hvordan Lector anbefaler en SOA-arkitektur konstrueret for at tilfredsstille ovenstående udsagn.

Et SOA system kræver stillingtagen til blandt andet følgende udfordringer:

- Identifikation: Hvordan identificerer vi en service?
- Specifikation/definition: Hvordan specificerer vi en service?
- Realisering/implementering: Hvordan anbefales implementering af en service (.NET/Java)?

Førend vi kommer til disse anbefalinger, vil en række karakteristika omkring SOA være gode at være bekendte med. Et SOA-system vil med fordel efterleve følgende udsagn:

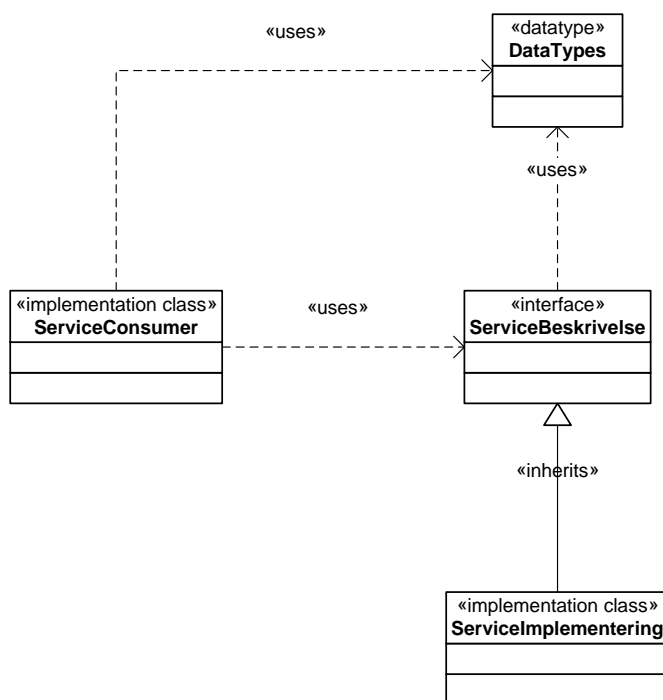
| Kendetegn | Beskrivelse |
|---|--|
| Løst koblet | Systemerne må ikke have binære referencer til andre komponenter, men skal udelukkende være afhængig af en servicebeskrivelse/interface. |
| Forretningsorienteret | De services, der stilles til rådighed skal bygge på processer i forretningen. |
| Afkobling af servicebeskrivelse fra serviceimplementation | Selve beskrivelsen af en service skal være separeret fra implementationen. Det sker for at serviceconsumeren ikke skal kende noget til hvordan servicen er implementeret. |
| Dynamisk konfiguration af services | Sammensætning af de enkelte services skal kunne konfigureres uden noget særlig kendskab til hvordan servicen er implementeret. Desuden skal placeringen af servicen være skjult for serviceconsumeren. |
| Stærk samhørighed | De muligheder en enkelt service stiller til rådighed skal hænge tæt sammen. |

Konceptuel arkitektur

Der er, som nævnt, to mekanismer der er grundlæggende for enhver SOA-system:

- Serviceconsumer må kun kende til servicebeskrivelse og ikke den konkrete implementation
- Serviceconsumeren skal kalde servicen ved at sende en besked

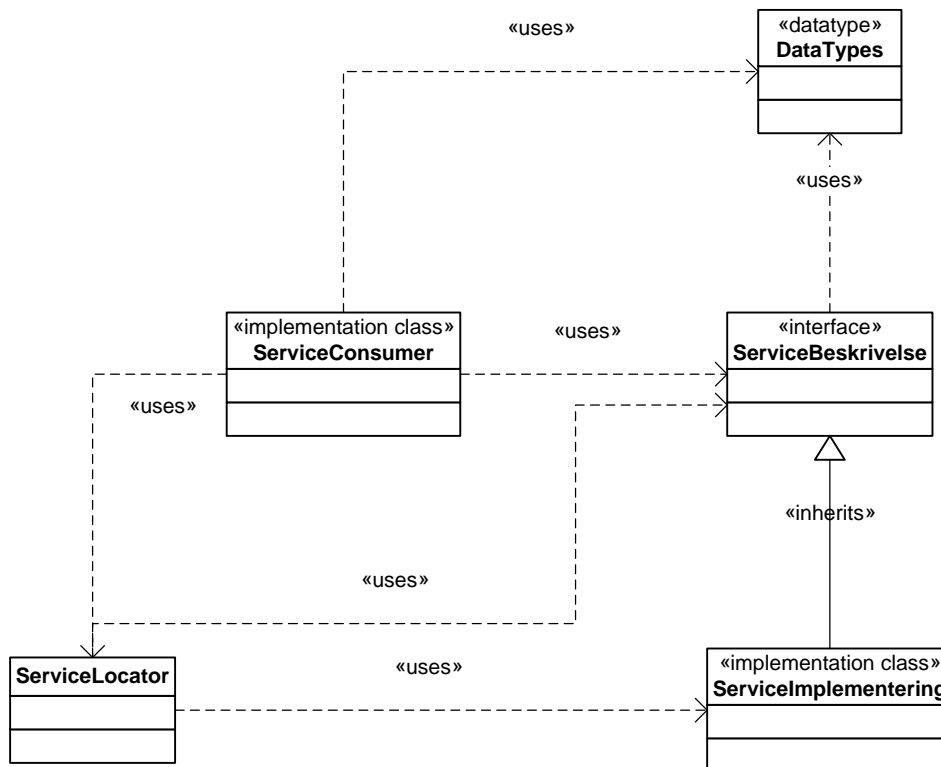
Disse to mekanismer styrer den grundlæggende arkitektur som Lector bygger et SOA-system efter:



Figur 1: Konceptuel SOA struktur (simpel)

Som det ses – er **ServiceConsumer** kun koblet til "kontrakten" (**ServiceBeskrivelse**) og de typer som denne kontrakt erklærer at bruge. I nogle situationer vil der ikke eksistere en kobling til typerne (**DataTypes**), idet disse ofte kan skabes dynamisk når koblingen skabes til ServiceBeskrivelsen. Dette understøttes af de fleste udviklingsmiljøer.

For at ServiceConsumer faktisk kan kalde en service og dernæst modtage brugbar respons; skal ServiceConsumer på et eller andet tidspunkt være vidende om adressen på en konkret service. Hvorhenne denne service befinder sig, skal klienten have at vide "udefra" via et **ServiceLocator** modul. Dette ses i nedenstående illustration.



Figur 2: Konceptuel SOA struktur (inkl. ServiceLocator)

Som det fremgår af figuren, så er det **ServiceLocator's** ansvar at levere en adresse til den konkrete service til **ServiceConsumer**.

For nu ikke at gøre dette unødigt kompliceret, så vil implementingen af denne ServiceLocator oftest være en simpel konfigurationsfil som læses direkte af klienten. Denne konfigurationsfil vil så indeholde den konkrete adresse for den service som klienten skal kalde.

I større systemer hvor muligheden for, under systemafvikling at kunne udskifte adressen på en service, vil en decideret **ServiceLocator** være anvendt. Dette ses f.eks. ved systemer hvor man har redundante services som tilbyder samme funktioner (ServiceBeskrivelse). Dette kan være grundet ønsket om en belastningsfordeling (loadbalancing) eller blot at kunne lukke ned for een service, mens en anden overtager opgaverne. I de fleste systemer og i praksis er dette dog ikke nødvendigt og man anvender derfor en konfigurationsfil.

Generelle guidelines

Generelt præsenteres disse anbefalinger (på tværs af .NET og Java).

| Guideline | Beskrivelse |
|--|--|
| Benyt interfaces | Generelt anbefaler Lector, at der i så høj grad som muligt kodes mod interfaces (også kaldet kontrakter). Dette giver mulighed for at lade implementeringen blive skabt af en 'factory' for derved yderligere at afkoble implementering fra beskrivelsen (interfacet) og forbrugeren |
| Anvend unikke namespaces | For at tilsiere at datatyper og metoder er globalt unikke for enhver service, anvend unikke namespaces ihht. denne syntax: <a href="http://schemas.<firma>.com/<year>/<version>/<name>">http://schemas.<firma>.com/<year>/<version>/<name> f.eks. http://schemas.lector.com/2006/01/OrderServices |
| Erklær fejlmuligheder i interfacet | For at klienten kan agere proaktivt og teste for givne fejlmuligheder præsenteret af servicen, erklær da i serviceinterfacet hvilke fejl der potentielt kan kastes fra servicen |
| Erklær "fulde" data-strukturer i servicen | At kalde en webservice er alt andet lige langsommere end at kalde et object direkte. Derfor skal der IKKE laves metodesignaturer som lægger op til "smalltalk", men som lægger op til at lave en større opgave serverside. Med andre ord så skal en service ikke levere f.eks. en int tilbage på et metodekald, men en komplet datastruktur som typisk repræsenterer en større operation. |
| Sikkerhed | Erklær sikkerhed som en kontekstuel parameter i et servicekald (ServiceContext). Lad IKKE sikkerhedsrelateret information (f.eks. username/password) tage del i den normale forretningsdrevne metodesignatur. |

Sprogspecifik

C#

WebServices:

Faktorer komponenter således:

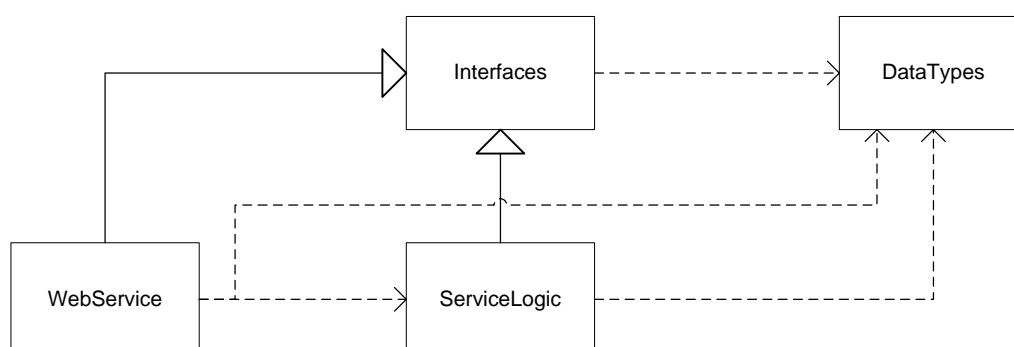


Figure 1: Servicefaktorering (webservices)

Fordelen ved ovenstående arkitektur som anviser komponenterne som indgår i en "serverside" implementering, er at det er muligt at anvende servicen's logik (ServiceLogic) direkte skulle dette være ønskeligt. WebService's formål i denne opsætning er at eksponere servicelogikkomponentens funktionaliteter (beskrevet via interfacet) til omverdenen **som** en service.

WebService komponentens ansvar er desuden at genkende (authenticate) og autorisere indkommende kald for hvorvidt disse har lov til at anvende funktionaliteten denne service præsenterer. WebService komponenten kommer derfor til at fungere som en "sikkerhedsvagt" som afviser folk ved porten, hvis ikke disse har rettigheder til at anvende servicen's funktionaliteter.

Ved at trække datadefinitioner ud i en selvstændig assembly, vil denne kunne genbruges på tværs af forskellige services, således at disse anvender samme definition af f.eks. et "document" eller andre forretningsentiteter. En yderligere fordel ved dette, er at det nu er muligt at distribuere denne datatype + interfacekomponenten til klienter således at disse klienter anvender samme signatur som servicen.

WCF:

Faktorer komponenter således:

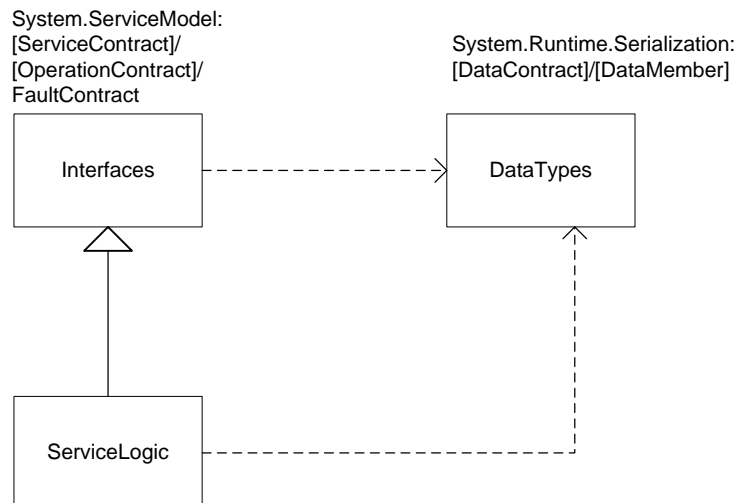


Figure 2: Servicefaktorering (WCF)

Når man anvender WCF, vil ovenstående arkitektur eller faktorering være hensigtsmæssig. Som netop gennemgået vil interfaces skulle trækkes ud i en selvstændig assembly ligesom datatyperne for derved at opnå samme fordele som i foregående sektion.

Grundet WCF's interne arkitektur er det ikke nødvendigt at have en yderligere komponent som tilbyder interfacets funktionaliteter som en service; **ServiceLogic ER en service** grundet den nedarver fra et interface med `[ServiceContract]` attributer påtrykt.

Set fra .NET's vedkommende er ServiceLogic dog stadig blot en klasse som implementerer et interface og kan derfor anvendes på helt almindelig vis.

Java

SOA arkitekturer i Java følger i store træk den struktur, som er beskrevet i afsnittet om .NET (WebServices). Dvs. at servicedefinitioner i form service-interface og datatyper placeres i selvstændig jar-fil, der kan bruges af både serviceimplementationen og klienten.

Lector benytter ofte Oracle's platform til implementering af java-aprojekt. Derfor vil det være naturligt at benytte deres SOA-værktøjer:

- Enterprise Service Bus: Bl.a. virtualisering af eksisterende systemer, såsom pl/sql procedurer. Kan også benyttes til at udvikle integrationsløsninger
- Oracle Workflow Manager: Giver mulighed for at oprette og vedligeholde processer.
- Oracle Webservices Manager: Implementer diverse sikkerheds politikker på de enkelte webservices. Giver en central styring af sikkerhedspolitikken.

Ellers benyttes standardfunktionalitet fra Java EE 5.