Methodology

In de dagelijkse praktijk worden veel integratieprojecten uitgevoerd. Het creeëren van een SOA-infrastructuur kan daarbij problemen opleveren. We nemen de elf belangrijkste valkuilen bij een SOA-implementatie onder de loep. Het ontwijken van deze valkuilen helpt om een omgeving te creëren met meer flexibiliteit en herbruikbaarheid en die leidt tot een meer transparante IT.

Goede implementatie maakt IT transparant

Elf valkuilen om te ontwijken voor goede SOA

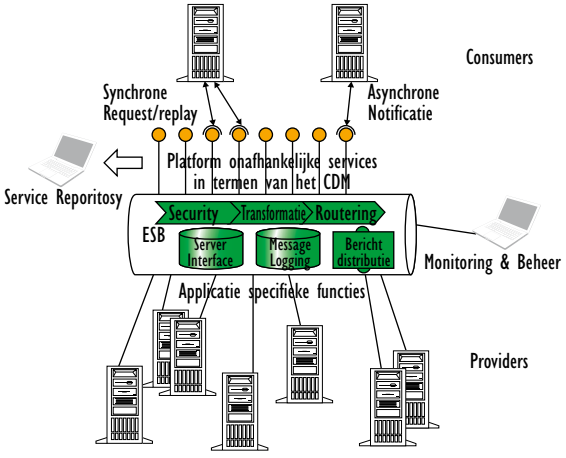
Peter van Langen
is Informatie Architect
bij Atos Origin.

Systeemkoppelingen worden vaak op een ad hoc wijze geïmplementeerd en dat levert de bekende 'spaghettiplaatjes' op. Om systeemintegratie op een meer gestructureerde wijze aan te pakken zijn veel bedrijven gestart met Enterprise Application Integration (EAI), een Enterprise Service Bus (ESB) en/of een Service Oriented Architecture (SOA). Bij EAI ligt de nadruk op asynchroon berichtenverkeer met een gegarandeerde aflevering met behulp van een message broker. Bij de ESB gaat het om de implementatie van synchrone request/replyservices. Bij SOA tenslotte staat de ontkoppeling tussen providers en consumers centraal, niet alleen in technisch, maar ook in organisatorisch opzicht.
Bij ieder van de drie varianten is een grote mate van flexibiliteit en herbruikbaarheid van de koppelingen in het vooruitzicht gesteld, maar die belofte blijkt niet altijd te worden ingelost.
We bespreken elf valkuilen die kunnen optreden bij de implementatie van een SOA-infrastructuur.

Beschouw de ESB-implementatie niet als de SOA-implementatie.
In feite is de ESB niet meer dan een technisch platform waarop een aantal technische zaken efficiënt en centraal kunnen worden geïmplementeerd, namelijk:

- Security;
- Monitoring;
- Logging;
- Routering;
- Transformatie;
- Bericht distributie (notificatie server/message broker);
- Filtering (alleen van toepassing bij bericht distributie).

Voor SOA zijn veel meer dingen van belang om tot herbruikbare services te komen. Zo is het bijvoorbeeld zonder een SOA-repository ondoenlijk om vast te stellen welke services er beschikbaar zijn voor hergebruik, wat hun status is, en wie ervoor verantwoordelijk is. Ook voor beheer- en impactanalyses is de repository onmisbaar, aangezien hier ook de contracten tussen de consumer en de provider van een service worden vastgelegd. Daarnaast is de repository van cruciaal belang voor een goede workflowimplementatie ter ondersteuning van de processen die rondom de ontwikkeling en beheer van goede softwareservices onontbeerlijk zijn.



Figuur 1

Laat een service-implementatie niet over aan de eerste consumer van de service.
Vaak wordt een service ontwikkeld door het eerste businessproject dat de betreffende service nodig heeft. Vanwege de projectdoelstellingen met een beperkte scope leidt dit al gauw tot services die alleen goed bruikbaar zijn binnen de projectcontext. Het is daarom beter om de requirements van de service te laten opstellen door het project, en de specificatie en implementatie ervan over te laten aan de eigenaar van het domein waarbinnen die service valt. Een domein is een goed afgebakend functioneel deelgebied binnen een bedrijf. Bij een goedgekozen opdeling van het bedrijf in domeinen is iedere functie en informatieobject dat gebruikt wordt binnen het bedrijf eenvoudig toe te kennen aan één van de gedefinieerde domeinen. Domeingrenzen zijn van belang voor zowel de IT-architectuur als voor de business. Veelal zullen de domeinen daarom overeenkomen met de organisatorische divisies of afdelingen, maar dit is niet noodzakelijk. Binnen de SOA-context is een domeineigenaar verantwoordelijk voor de serviceportfolio en de implementatie ervan binnen een dergelijk domein. Bij de specificatie van de nieuwe service zal een domeineigenaar veel meer zaken laten meewegen dan die van het aanvragende project alleen. Denk hierbij bijvoorbeeld aan domeinspecifieke richtlijnen of KPI's. Het domein waarin het project valt zal daarbij veelal afwijken van het domein van de software service.

Vergeet de organisatorische aspecten niet
Om tot een goede SOA-implementatie te komen is governance noodzakelijk met een goed kostenmodel; wie betaalt de additionele kosten die gepaard gaan met het algemeen bruikbaar maken van een projectspecifieke service. Er moet voorkomen worden dat het project dat een nog niet bestaande service nodig heeft, opdraait voor alle initiële kosten van die service, en dat alleen de projecten die een service hergebruiken er de voordelen van plukken. (De eerste passagier zal niet de hele bus willen betalen). Iedere service kan daarom in eerste instantie het beste worden betaald en gebouwd door het domein waarin de service thuishoort. Zelfs voor de eerst gebruikende projecten levert dit dan een vermindering op van de kosten, waardoor deze projecten worden gemotiveerd om de functionaliteit generiek geïmplementeerd te krijgen. Eventueel zou het verantwoordelijke domein een percentage van de totale kosten bij de afnemer in rekening kunnen brengen. Door dit ook te doen voor alle afnemers van de service kunnen de kosten in sommige gevallen dan weer worden teruggverdiend. Kern van de strategie moet zijn dat functionaliteit wordt ontwikkeld en beheerd door de partij die daarvoor eindverantwoordelijk is, namelijk het betreffende domein!

Beschouw de Web Services Description Language (WSDL) niet als de service specificatie.
Hoewel WSDL een belangrijk onderdeel vormt voor interface standaardisatie, is het ook niet meer dan dat. Het beschrijft exact het formaat van de velden die in het bericht zitten en de veldnaam geeft alleen maar een hint over de betekenis ervan. Voor een volledige specificatie zijn meer zaken nodig zoals:

- Beschrijving van de functionaliteit / effect van de service;
- Nauwkeurige beschrijving van de betekenis van de berichtvelden;
- Beperkingregels tav de inhoud van de berichtvelden. (Alleen de meest simpele beperkingregels op veld niveau kunnen in de WSDL zelf worden opgenomen);
- Precondities waaronder de service kan worden gebruikt;
- Foutafhandeling.

Een veld met een specifieke betekenis, zoals de klantnaam, komt vaak in verschillende berichten voor. Het is daarom handig om de naam en betekenis van zo'n veld op een centrale plek vast te leggen, samen met de relaties die dat veld heeft met andere bedrijfsinformatie: het Common Data Model (CDM). Zonder een CDM is het in feite niet mogelijk om de services op een eenduidige wijze bedrijfsbreed en onderling consistent te specificeren. Daarnaast is het dan veel moeilijker om services te combineren tot samengestelde services. Zonder onderliggend datamodel zouden er transformaties moeten plaatsvinden tussen de velden van de verschillende berichten. Als het ene bericht bijvoorbeeld een veld in Fahrenheit doorgeeft en de andere in Celsius, dan moet er gerekend worden om deze met elkaar te kunnen combineren. Ook de functionaliteit, de precondities en foutafhandeling van een service kunnen het best worden beschreven in termen van het CDM.

Baseer de berichten niet op een bibliotheek van XML-schema's.
Vaak bestaat de neiging om in plaats van een CDM gebruik te maken van een bibliotheek van Extended Markup Language (XML) schema's om de berichten mee te definiëren. Hierbij worden de XML Schema Definitions (XSD's) dan geïmporteerd in de WSDL's. Veldbeschrijvingen zijn dan opgenomen in de XSD's als annotaties. Een probleem hierbij is dat een XSD ook beperkingregels bevat, en die zijn dan automatisch van toepassing op ieder bericht dat gebruik maakt van de betreffende XSD. Dit is niet correct, want beperkingregels zijn afhankelijk van de context waarin het veld wordt gebruikt. Daarnaast zijn XSD's slecht leesbaar en niet geschikt voor documentatiedoeleinden. Veel beter is het om het CDM (bijvoorbeeld in de vorm van Unified Modeling Language (UML) klasse diagrammen) als

Governance met een goed kostenmodel is nodig om SOA goed te implementeren.

In één keer een goede generieke service specificeren is in de praktijk nog niet zo eenvoudig!

uitgangspunt te nemen, en de WSDL's vanuit die omgeving te genereren.
Indien de WSDL's gegenereerd worden, is er geen noodzaak om gebruik te maken van gemeenschappelijk geïmporteerde XSD's. Het wijzigen van een XSD heeft dan alleen maar impact op die ene WSDL die de XSD importeert. Dit in tegenstelling tot de gemeenschappelijke XSD uit de bibliotheek waarbij alle WSDL's wijzigen die gebruik maken van die XSD. Dit is vervelend, want een XSD wordt in de runtime-omgeving gebruikt en in een SOA-omgeving streven we nu juist naar het zoveel mogelijk ontkoppelen van de verschillende stukken functionaliteit.

Laat de service specificatie niet over aan de softwareontwikkelafdeling.
De specificatie van een service wordt nogal eens gemaakt door een software-ontwikkelaar. Dit vergroot de kans dat de service in hoge mate wordt beïnvloed door de technische interface van het systeem waar de ontwikkelaar mee bezig is en blijft de scope beperkt tot die van het project waarbinnen de service wordt ontwikkeld. Veel beter is het als de service wordt gespecificeerd door een domeinarchitect, die een brede kennis heeft van zijn complete domein. Eventuele wensen ten aanzien van de service kunnen zowel van de ontwikkelaars uit de verschillende IT projecten als uit de lijnorganisatie komen. Op die manier ontstaan services die beter aansluiten bij de business en wordt de specificatie gemaakt voordat met de implementatie wordt begonnen. De architect hoeft daarbij geen detailkennis te hebben van WSDLs /XSDs/XMLs indien de WSDL uit een UML-diagram kan worden gegenereerd.

Schuif het versiebeheer van services niet op de lange baan.
Versiebeheer wordt nogal eens afgedaan als iets waar later nog wel eens naar gekeken kan worden bij het opzetten van de SOA-infrastructuur. In de praktijk blijkt dat er al heel snel nieuwe versies van een service nodig zijn. Dit is natuurlijk niet zo verwonderlijk, omdat een service door meerdere consumers afgenomen gaat worden en die consumers vaak net andere wensen hebben. In één keer een goede generieke service specificeren is in de praktijk nog niet zo eenvoudig!
Bij versiebeheer dient ook expliciet te worden gekeken hoe om te gaan met de verschillende versies die gedurende het implementeren van een versie van een service ontstaan. Een gespecificeerde service dient namelijk nogal eens aangepast te worden gedurende het bouw- en testtraject.
Bij specificatiewijzigingen zijn er veel partijen die worden geraakt. Vandaar dat workflowondersteuning in de SOA-repository bij versiebeheer onontbeerlijk is.

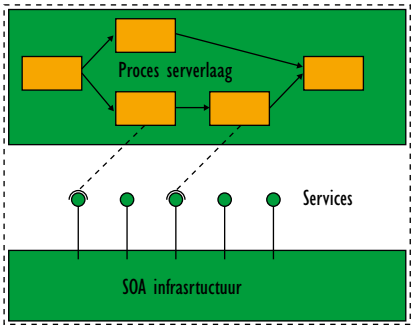
Verwacht niet al te veel hergebruik van typische EAI koppelingen.
In het verleden (het EAI tijdperk) is er vooral gebruik gemaakt van notificatieservices (ook wel asynchrone 'fire-and-forget' koppelingen genoemd), waarbij het zendende systeem een bericht verstuurt zonder dat deze een respons terug verwacht. Een message broker zorgt er vervolgens voor dat het verzonden bericht op een gegarandeerde wijze bij de ontvangende systemen wordt afgeleverd. Gebruik van services in de vorm van request/reply waren relatief zeldzaam. Koppelingen kwamen vooral tot stand indien hier een noodzaak toe was, terwijl hergebruik laag op de agenda stond. De noodzaak van een notificatieservice bestaat eigenlijk altijd uit het voorkomen van dubbele invoer indien een bedrijfsproces over meerdere systemen loopt en waarbij het doelsysteem het proces voortzet op het moment dat het bronsysteem met het eerste deel van het proces klaar is.
In feite is het binnenkomende bericht bij het doelsysteem een event dat het systeem proces triggered en bevat het bericht alle contextinformatie die nodig is om het betreffende systeemproces te starten.
Het moment van versturen en de inhoud van het bericht wordt dus volledig bepaald door het te ondersteunen proces en de verdeling van de businesslogica tussen bron- en doelsysteem. De event-notificatie is daardoor weinig geschikt voor hergebruik, want het te ondersteunen proces en de consumer van de service zullen bij hergebruik tenslotte een andere zijn.
Naast de event-notificatie is er een ander type notificatie dat veel meer potentie heeft voor hergebruik: de informatienotificatie. Bij zo'n notificatie wordt informatie ter beschikking gesteld door het leidende systeem op het moment dat het daar beschikbaar komt. Hierbij wordt de verstuurde informatie en het moment van versturen dus volledig bepaald door het zendende systeem onafhankelijk van het te ondersteunen proces of de ontvangende systemen.

Daardoor is er een echte functionele ontkoppeling tussen zender en ontvanger. Ontvangende systemen kunnen op die manier een lokale kopie bijhouden van de gegevens. In zijn zuiverste vorm komt dit type bericht weinig voor vanwege efficiencyredenen. Het bericht wordt immers verstuurd zonder dat bekend is wie het voor welke reden gaat afnemen.
Aangezien de event-notificatie de standaard vormt bij notificatieberichten, is de toegevoegde waarde van de message broker daardoor wat beperkt, omdat de meeste notificatiekoppelingen dus 1-op-1 koppelingen blijken te zijn.
Het migreren van een dergelijke bestaande koppeling naar de ESB kan overigens wel nuttig zijn in verband met standaardisatie, monitoring en beheer.

Ondersteun geen asynchrone request/reply in de ESB.
Bij een asynchrone request/reply gaat het zendende softwareproces door met de executie van de programmacode zonder te wachten op het antwoord. Als het antwoord vervolgens arriveert, wordt dit door een gescheiden softwareproces afgehandeld. Probleem bij deze aanpak is dat als het antwoord arriveert, er geen status meer beschikbaar is zoals die oorspronkelijk was opgeslagen in de lokale variabelen van het zendende softwareproces. Er is daarom een manier nodig om die status te herstellen zodat het antwoord correct kan worden geïnterpreteerd en afgehandeld.
Dit kan gebeuren door de volledige consumer context mee te sturen in het verzoek. De provider zal dan vervolgens die consumer context weer mee terugsturen in het antwoord. In plaats van de consumer context kan ook een correlatie ID worden meegestuurd die als een referentie dient naar de contextinformatie zoals die door de consumer op disk is opgeslagen. Bij een asynchrone request/reply is naast de context tevens een expliciet return adres nodig, waar de leverende partij de reply naar toe moet sturen.
Het implementeren van een asynchrone request/reply is dus een stuk lastiger dan een synchrone. Vandaar dat deze alleen interessant is als het antwoord lang op zich kan laten wachten. Dat komt eigenlijk alleen voor bij workflowsystemen, waar delen van het werk niet geautomatiseerd worden uitgevoerd. Het is daarom beter om dit soort niet-geautomatiseerde services uit te sluiten van de SOA-context. Iedere service wordt dan volledig automatisch uitgevoerd. Alle workflowgerelateerde zaken worden dan in de proces/workflow laag afgehandeld die speciaal is toegerust voor dit soort asynchrone processtappen. Ook Business-to-Business (B2B) koppelingen worden nogal eens asynchroon uitgevoerd, waarbij dezelfde strategie als hierboven geschetst kan worden gevolgd.

Ondersteun geen expliciete adressering in de ESB.
Voor een goede ontkoppeling tussen service consumer en provider is het onder andere belangrijk dat de locatie van de provider niet van belang is voor de consumer. De consumer hoeft dus geen adres mee te sturen. Het is nu juist de taak van de ESB om aan de hand van het bericht dat binnenkomt te bepalen waar deze naar toe moet. Natuurlijk moet het wel mogelijk zijn om een bericht naar een specifieke URL te sturen, maar in zo'n geval is de URL onderdeel van de payload van het bericht. Er moet dus een service bestaan die in staat is om het bericht naar dat specifieke adres te sturen. Dit gebeurt dan buiten de ESB om, want voor de ESB is alleen de gestandaardiseerde header van een bericht van belang en niet de payload.

Probeer niet altijd alle bestaande koppelingen te migreren naar WSDL interfaces.
Niet alle bestaande koppelingen zijn geschikt om te worden gemigreerd naar een WSDL-interface via de ESB. Hier kunnen twee redenen voor zijn. Ten eerste kan het een technische fijnkorrelige koppeling betreffen die zeer frequent door één specifieke consumer wordt gebruikt. De omzetting hiervan kan tot performanceproblemen leiden, terwijl het hergebruik ervan toch zeer beperkt is. Ten tweede kan het een specifieke pakketkoppeling (zoals ERP) betreffen, waarbij veel informatie wordt overgestuurd die functioneel niet van belang is, maar alleen nodig is voor een correcte werking van de gekoppelde pakketten. Dit soort koppelingen voldoen absoluut niet aan de eisen die binnen een SOA (Zie bijvoorbeeld ' www.soaprinciples.com') worden gesteld.



Figuur 2: De integratielaag.

Ga de ESB niet als applicatie platform gebruiken.
De ESB wordt naast als integratieplatform ook nogal eens gebruikt als applicatieplatform, vooral voor BPM-gerelateerde zaken. Dit wordt mede veroorzaakt door de ESB-platformleveranciers die ook graag BPM-producten willen afzetten, waarbij er een vervaging tussen de twee ontstaat. Hoewel SOA een heel goed fundament vormt voor BPM, zijn het twee heel verschillende platformen die aan hun eigen criteria moeten voldoen. In een goede IT-architectuur moeten scope en functie van de verschillende onderdelen helder gescheiden zijn, zodat het toewijzen van verantwoordelijkheden en het beheer van de onderdelen worden vereenvoudigd. Alleen vanwege performanceredenen zou het moeten zijn toegestaan om applicatie logica zoals orchestrations binnen de ESB te implementeren.

Conclusie
Het ontwijken van de bovengenoemde valkuilen zal leiden tot een betere SOA-infrastructuur. Naast meer flexibiliteit en herbruikbaarheid leidt het ook tot een meer transparante IT. SOA dwingt tot het beter in kaart brengen van welke functionaliteit door wie waar wordt gebruikt en wie er verantwoordelijk voor is. Alleen dit laatste zou al voldoende motivatie moeten zijn om (op een goede manier) SOA binnen het bedrijf in te voeren. «

SOA dwingt tot het in kaart brengen waar de verantwoordelijkheid precies ligt.