archixl

Applicatie-integratie

data overal toegankelijk

Erwin Oord, Danny Greefhorst en Jan Willem van Veen

Het is belangrijk dat iedereen toegang heeft tot de data die nodig zijn om processen goed uit te voeren. Data zitten veelal opgesloten in applicaties en zullen moeten stromen om iedereen toegang te geven tot de juiste data. Applicatie-integratie is het koppelen van applicaties zodat data beschikbaar zijn voor de gebruiksdoelen. Dat kan van alles zijn: een enkele byte waarmee wordt medegedeeld dat een bepaalde gebeurtenis heeft plaatsgevonden, een compleet video-archief van vele gigabytes, en alles daar tussenin. Deze whitepaper geeft meer inzicht in de uitdagingen en oplossingsrichtingen voor applicatie-integratie.

Applicatie-integratie is niet nieuw

Applicatie-integratie is niets nieuws: al in de jaren zestig werden applicaties al aan elkaar gekoppeld. Maar het thema wordt wel steeds belangrijker. De ketens waarin data worden verwerkt worden steeds langer en applicaties worden steeds afhankelijker van elkaar. In een modern applicatielandschap zijn data vaak gedistribueerd, dat wil zeggen dat iets dat we logischerwijs als een geheel beschouwen (bijvoorbeeld een klantdossier) verspreid en gefragmenteerd is opgeslagen in meerdere applicaties. De reden is meestal dat data ook op meerdere plaatsen worden beheerd. Applicatie-integratie is dan noodzakelijk om over alle relevante data te kunnen beschikken.

Applicatie-integratie is geen triviale exercitie. Er zijn uiteenlopende vormen van integratie beschikbaar, al dan niet gebruik makend van ondersteunende middleware-oplossingen. De kosten van integratieprojecten kunnen uiteenlopen van enkele honderden euro (voor een simpele koppeling) tot miljoenen euro (voor de implementatie van complete en grootschalige middleware-stack die een volledige integratie van een pluriform applicatielandschap moet realiseren). De kans op mislukking is reëel en afhankelijk van de keuzes die gemaakt worden. Het is daarom belangrijk op de hoogte te zijn van de soorten applicatie-integratie die bestaan, wat hun kenmerkende verschillen zijn, en waarop te letten bij het maken van keuzes.

Paradigma's

Een paradigma is een alomvattende zienswijze. Als het gaat over applicatie-integratie dan zijn er meerdere paradigma's te onderkennen. De belangrijkste zijn: service-oriëntatie, eventoriëntatie, resource-oriëntatie en data-oriëntatie. Het is verstandig om paradigma's bewust te kiezen en te bewaken. Al deze paradigma's hebben een bepaalde waarde, maar ze kunnen voor een belangrijk deel ook als vervanger van de andere paradigma's worden gezien. Een bewuste keuze, positionering en afbakening zorgt voor consistentie en voorkomt verwarring. Consistentie brengt eenvoud en rust, en helpt daarmee fouten te vermijden en kosten te beheersen. Dit soort fundamentele keuzes zijn onderdeel van enterprise-architectuur. In de navolgende paragrafen beschrijven we de belangrijkste kenmerken van actuele en veelgebruikte paradigma's. Vervolgens gaan we in op de meer specifieke integratiepatronen die je binnen deze paradigma's kunt zien en hoe deze zich tot elkaar verhouden. Vervolgens beschrijven we architectuurprincipes voor applicatie-integratie, per paradigma. Het whitepaper eindigt met een aantal adviezen voor de implementatie van applicatie-integratie.

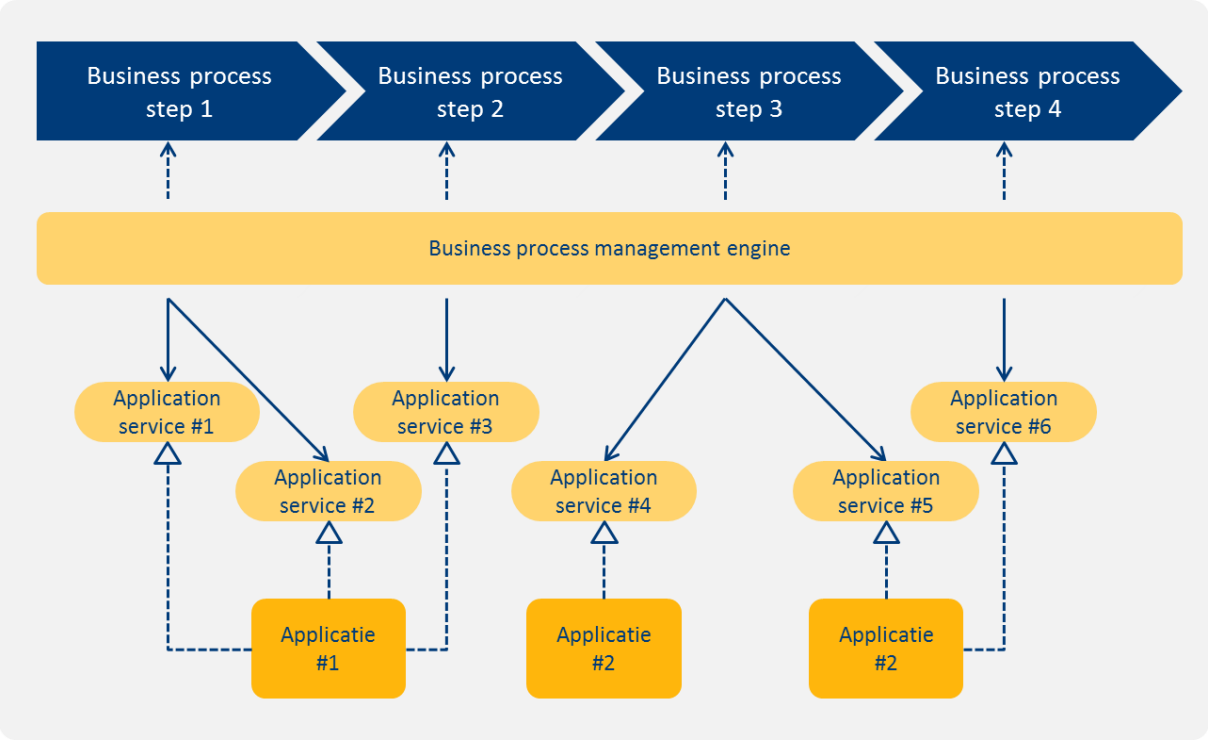
Service-oriëntatie

Bij service-oriëntatie (ook wel: Service Oriented Architecture - SOA) bieden applicaties hun functionaliteit aan in de vorm van services. Een applicatieservice is een logisch afgebakend "stuk functionaliteit" waarmee een bepaalde bedrijfsfunctie ondersteund wordt. Een service wordt altijd in zijn geheel uitgevoerd. In een volledig servicegeoriënteerde omgeving kunnen bedrijfsprocessen worden samengesteld door services te

combineren, al dan niet aangestuurd via een business process engine. Een voorbeeld van een logisch afgebakende service is “registreren nieuwe klant”. Een service als deze kan in meerdere bedrijfsprocessen gebruikt worden.

Bij een goed doorgevoerde servicearchitectuur zijn aan applicaties duidelijke logische taken toegewezen, en zijn vanuit die taken services gedefinieerd – onafhankelijk van de beoogde afnemers. Door de services te publiceren in een servicecatalogus, voorzien van kwaliteitseigenschappen zoals beschikbaarheid, wordt het assembleren van bedrijfsprocessen mogelijk.

Service-oriëntatie is een veelgebruikt paradigma. Veel softwarepakketten en clouddiensten worden geleverd met kant-en-klare servicekoppelingen. Veel ontwikkelomgevingen bieden ondersteuning voor het zelf ontwikkelen van services.

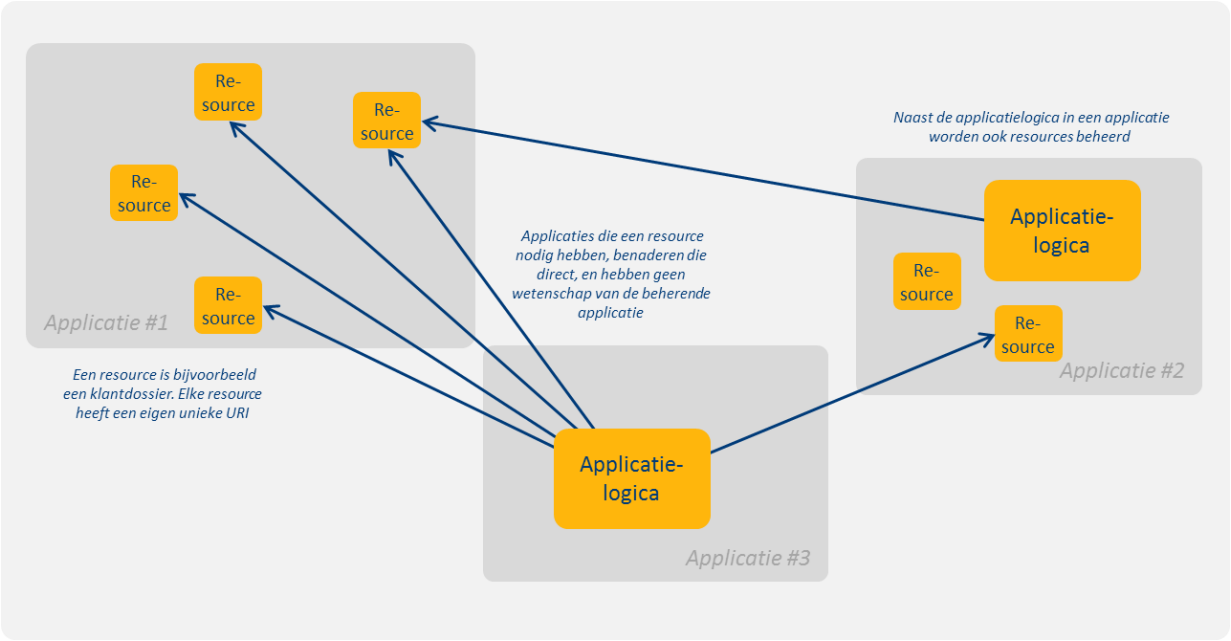


Resource-oriëntatie

Tegenwoordig is er toenemende aandacht voor integratie op basis van API’s, dat je kunt zien als onderdeel van het paradigma resource-oriëntatie. Bij resource-oriëntatie (ook wel: Resource Oriented Architecture - ROA) is niet de service maar de te benaderen resource (bijvoorbeeld een klantobject) de focus van integratie. Elke resource heeft daartoe een eigen adres (URL); welke applicatie de resource beheert is voor de vragende applicatie niet relevant. Benaderen van een resource gebeurt volgens een vast protocol, meestal HTTP, en onder vermelding van de gewenste operatie (GET/POST/CREATE/DELETE). Resource-oriëntatie gebruikt REST als principe voor applicatie-integratie en combineert dat met principes uit de wereld van Linked Data.

REST staat voor “representational state transfer” [1]. Als een applicatie via een RESTful API gekoppeld kan worden, dan betekent dat dat “resources” binnen die applicatie rechtstreeks aangeroepen kunnen worden

via een URL. In feite werkt het world wide web op die wijze: met een HTTP-get request kan bepaalde data (meestal een webpagina) opgevraagd worden; met een HTTP-post request wordt data verstuurd (denk aan een webformulier). Het belangrijkste voordeel van REST is dat het een gestandaardiseerde methode introduceert voor de uitwisseling van data.



De belangrijkste gedachte achter ROA is dat het er niet toe doet in welk applicatie data beheerd worden. Elk object heeft een eigen unieke adres. Zo kunnen data van een klant bijvoorbeeld bereikbaar zijn via een URL https://www.archixl.nl/customer/123456789 waarbij het nummer een unieke eigenschap van de betreffende klant is, bijvoorbeeld het BSN ingeval van personen. Via die URL kunnen data niet alleen opgevraagd worden, maar ook gewijzigd, en kunnen eventueel andere operaties uitgevoerd worden.

Van belang is dat een resource-ontologie opgesteld wordt, enigszins vergelijkbaar met een canoniek datamodel. De resource-ontologie beschrijft welke (typen) resources onderkend worden, wat de scope is (bijvoorbeeld omvat een klant-resource ook de contacten van die klant of is dat een afzonderlijke resource?), welke operaties de resource kent en wat de samenhang is met andere resources.

ROA is een vergaande doorontwikkeling van SOA. Je zou kunnen stellen dat ROA zich tot SOA verhoudt als object-georiënteerd programmeren tot procedureel programmeren. Het is een nieuwere ontwikkeling, gebaseerd op de ideeën van het world-wide web en linked data.

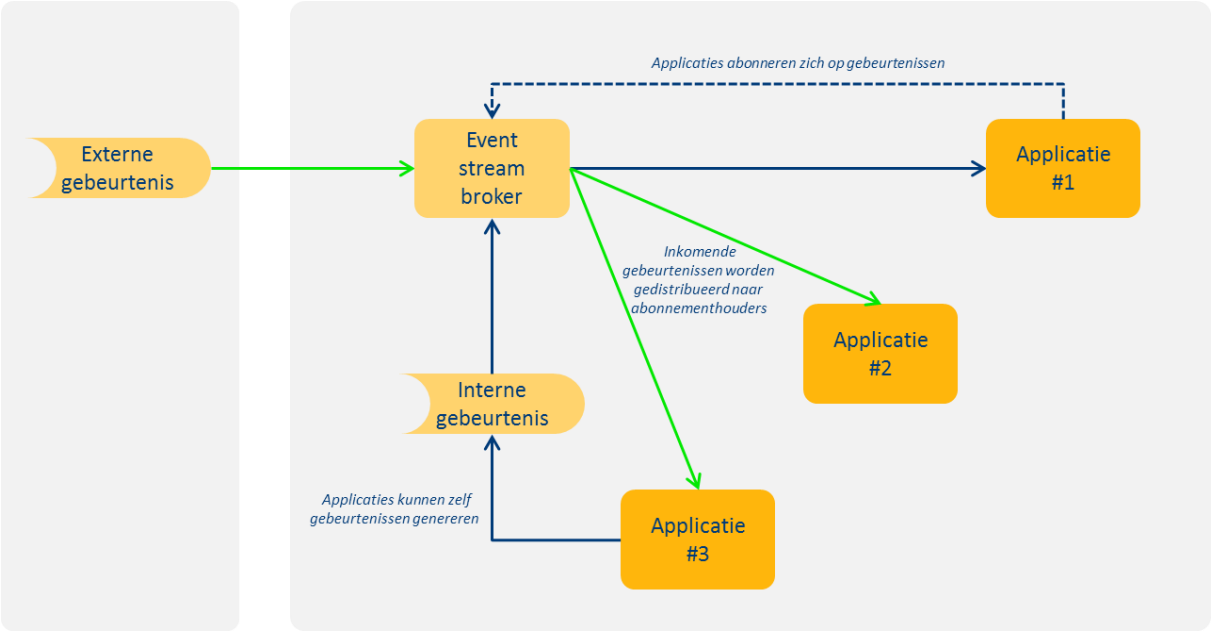
Eventoriëntatie

Bij eventoriëntatie (ook wel: Event Driven Architecture - EDA) vormen gebeurtenissen de initiator voor uitwisseling van data. Denk aan de geboorte van een kind, wat in overheidsland tot allerlei acties leidt zoals registratie in de GBA, opname in het hielprikprogramma enz. enz. Kenmerkend voor een eventgedreven architectuur is het bestaan van een gebeurtenissenkaart, dat wil zeggen een overzicht van alle mogelijke gebeurtenissen met hun samenhang en de gevolgen die zijn hebben.

Bij event-oriëntatie wordt veel gewerkt met het zogenoemde publish-and-subscribe mechanisme. Dat houdt in dat applicaties zich kunnen abonneren op bepaalde typen gebeurtenissen, waarna ze van het optreden van gebeurtenissen van het betreffende type melding krijgen. In bovengenoemd voorbeeld zal

het hielprikplanningsapplicatie graag een abonnement willen op gebeurtenissen van het type “geboorte”. Merk op dat een gebeurtenis kan leiden tot een cascade van vervolggebeurtenissen die zich door het applicatielandschap voortplant.

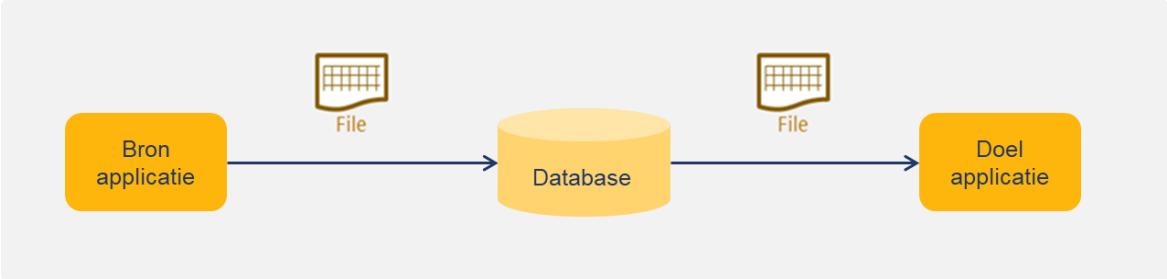
Event-oriëntatie is te zien als een doorontwikkeling van service-oriëntatie. Applicaties bieden twee soorten services: een waarmee andere applicaties zich kunnen abonneren op gebeurtenissen, en een om gebeurtenismeldingen te kunnen ontvangen. Er wordt veelal gebruik gemaakt van integratiemiddleware voor het aanbieden van een dergelijk abonnementsmechanisme.



Data-oriëntatie

Bij de integratie van applicaties kan ook de data centraal staan. Het gaat dan dus niet primair om functionaliteit maar om data. Welke rol de data precies heeft in het proces en voor welke functionaliteit de data gebruikt wordt is daarbij minder relevant. Het belangrijkste is dat er een applicatie is die data levert (bronapplicatie) en er is een applicatie die de data gebruikt (doelapplicatie). De data kan via bestanden worden uitgewisseld, maar ook via databases en eventueel ook via berichten.

Deze vorm van integratie is al heel oud. Het werd in het verleden vaak gebruikt in de context van batchverwerking. Daarbij is er typisch een dagelijks (batch)proces dat bepaalde data als input heeft en deze moet vantevoren klaar worden gezet. De data-uitwisseling kan echter ook meer realtime. Het wordt dan vaak toegepast om wijzigingen in masterdata (stabiele stamdata) ook te distribueren naar applicaties die deze masterdata nodig hebben voor hun eigen verwerking. Dat is met name relevant voor pakketapplicaties die ervan uitgaan dat alle data in hun eigen database aanwezig zijn. Tegenwoordig is het ook relevant in de context van microservices. Microservices zijn zelfstandige eenheden van functionaliteit en data die zoveel mogelijk los van elkaar moeten kunnen worden ontwikkeld en vervangen.



Aandachtspunten bij applicatie-integratie

Er is een grote diversiteit aan applicatie-integratiemogelijkheden. Veel applicaties hebben standaardkoppelvlakken en er zijn verschillende soorten integratieproducten die applicaties op elkaar kunnen aansluiten. Het is niet zo dat de een per definitie beter is dan de ander. Het zijn juist de omstandigheden die bepalen wat de beste oplossing is. Bij het ontwerpen van een koppelvlak tussen applicaties moet daarom nauwkeurig onderzocht worden welke aspecten van belang zijn. Hieronder beschrijven we (in willekeurige volgorde) de belangrijkste vragen die een ontwerper moet stellen alvorens te gaan ontwerpen:

Richting – In welke richting wordt data overgedragen? We onderscheiden unidirectionaliteit (eenrichtingsverkeer) en bidirectionaliteit (tweerichtingsverkeer). Een voorbeeld van unidirectioneel dataverkeer is het versturen van een e-mailbericht. Een voorbeeld van bidirectioneel dataverkeer is het oproepen van een pagina op een website.

Initiatief – Welke applicatie neemt het initiatief tot uitwisseling? Is dat de applicatie die data wil overdragen (“push”) of juist de applicatie die data nodig heeft (“pull”)?

Afhankelijkheid – In welke mate zijn de te koppelen applicaties afhankelijk van elkaar? Ontstaan er nieuwe afhankelijkheden? In het algemeen is dat ongewenst. Door een verkeerde integratiewijze te kiezen kunnen onnodige afhankelijkheden worden geïntroduceerd.

Datavolume – Hoeveel data zal worden uitgewisseld per transactie (dat wil zeggen, per keer dat het koppelvlak gebruikt wordt)? Dit kan enorm uiteenlopen, en is van groot belang bij het kiezen van uitwisselformaten en protocollen.

Transactiefrequentie – Met welke frequentie wordt het koppelvlak gebruikt? Bij veelvuldig gebruik wordt bijvoorbeeld de overhead van het gebruikte protocol belangrijk. Bij incidenteel gebruik zou zelfs overwogen kunnen worden om überhaupt niet te automatiseren.

Responstijd – Welke eisen gelden er ten aanzien van de responstijd? In een “realtime” omgeving móet er binnen een bepaald tijdsbestek antwoord zijn, maar in andere gevallen zit de initiërende applicatie helemaal niet op antwoord te wachten. Ook dit maakt uit voor het te kiezen architectuurpatroon.

Actualiteit – Hoe actueel moeten de data zijn die uitgewisseld worden? Voor operationele toepassingen zijn doorgaans actuelere data nodig dan voor bijvoorbeeld rapportages.

Beschikbaarheid – Wat is de beschikbaarheid van de te koppelen applicaties? Het heeft geen zin om “realtime” data op te vragen bij een batch-applicatie of andere applicatie die grote delen van de tijd niet beschikbaar is. Denk daarbij niet alleen aan het beschikbaarheidsvenster dat in de servicebeschrijving staat,

maar ook aan ongeplande onbeschikbaarheid. Wat moet het vragende applicatie doen als antwoord uitblijft?

Vertrouwelijkheid – Hoe vertrouwelijk moet de data-uitwisseling zijn? Voor het uitwisselen van gevoelige data zullen maatregelen zoals versleuteling nodig zijn om de vertrouwelijkheid te waarborgen. De vraag is in hoeverre het koppelvlak die maatregelen ondersteunt.

Integriteit – Hoe belangrijk is het dat data correct overkomen? Is het erg als een bericht niet of te laat aankomt?

Authenticatie en autorisatie – Welke eisen stelt een applicatie met betrekking tot authenticatie en autorisatie? Elk koppelvlak is in beginsel een beveiligingslek en het is belangrijk om dat lek goed te dichten

Schaalbaarheid – In hoeverre moet het koppelvlak uitbreidbaar en herbruikbaar zijn? Een koppelvlak dat masterdata beschikbaar stelt, is in potentie veelvuldig herbruikbaar. Maar dan moet wel gekozen worden voor een integratiemechanisme dat schaalbaar is.

Monitoring gebruik – Is het gewenst dat monitoring plaatsvindt op het gebruik en het werken van het koppelvlak? Dat kan bijvoorbeeld nodig zijn om verstoringen snel te verhelpen. In dat geval kan middleware gebruikt worden die hiervoor voorzieningen biedt.

Verstoring operatie – Wat is het risico dat het koppelvlak de operatie van een van beide applicaties verstoort? Wanneer bijvoorbeeld een applicatie data opvraagt bij een andere applicatie, en die applicatie moet daarvoor een complexe databasequery uitvoeren, dan zou dat ertoe kunnen leiden dat gebruikers van die applicatie verslechterde performance of zelfs uitval ervaren. Hoe groot is de kans daarop? En hoe erg is dat?

Hacker-bestendigheid – In hoeverre vermeerdert de introductie van het koppelvlak de “aangrijpingspunten” voor hackers om de applicaties te compromitteren? Met name in een publieke omgeving (het Internet) is dit van groot belang.

Complexiteit – Hoe complex wordt de architectuur door de introductie van het koppelvlak? Complexe koppelvlakken zijn kwetsbaar en moeilijk te onderhouden.

Leveranciersafhankelijkheid – Leidt toepassing van een bepaald koppelvlak of bepaalde software of technologie tot leveranciersafhankelijkheid? Dan moet een afweging gemaakt worden of de voordelen van het koppelvlak (althans in de gekozen vorm) tegen die afhankelijkheid opwegen.

Standaardfuncties – Welke functies en voorzieningen biedt de voor het koppelvlak te gebruiken software al “out-of-the-box”. In hoeverre moet maatwerk verricht worden om aan de functionele eisen te voldoen? Het zelf moeten realiseren van bijvoorbeeld versleutelde communicatie is complex en foutgevoelig en moet daarom zoveel mogelijk vermeden worden.

Mogelijkheden van de applicaties – Welke mogelijkheden voor integratie bieden de te koppelen applicaties? Het gaat daarbij zowel om de ondersteunde protocollen als om formaten waarin data uitgewisseld kan worden. Als die niet op elkaar aansluiten, is “middleware” nodig die een vertaling maakt.

Bestaande architectuur – Welke integratiemechanismen zijn in het applicatielandschap reeds beschikbaar? Hoewel hergebruik geen automatisme moet zijn, kan het kostenbesparingen opleveren om de bestaande integratie-infrastructuur te gebruiken. In het algemeen is het door elkaar toepassen van uiteenlopende mechanismen onverstandig.

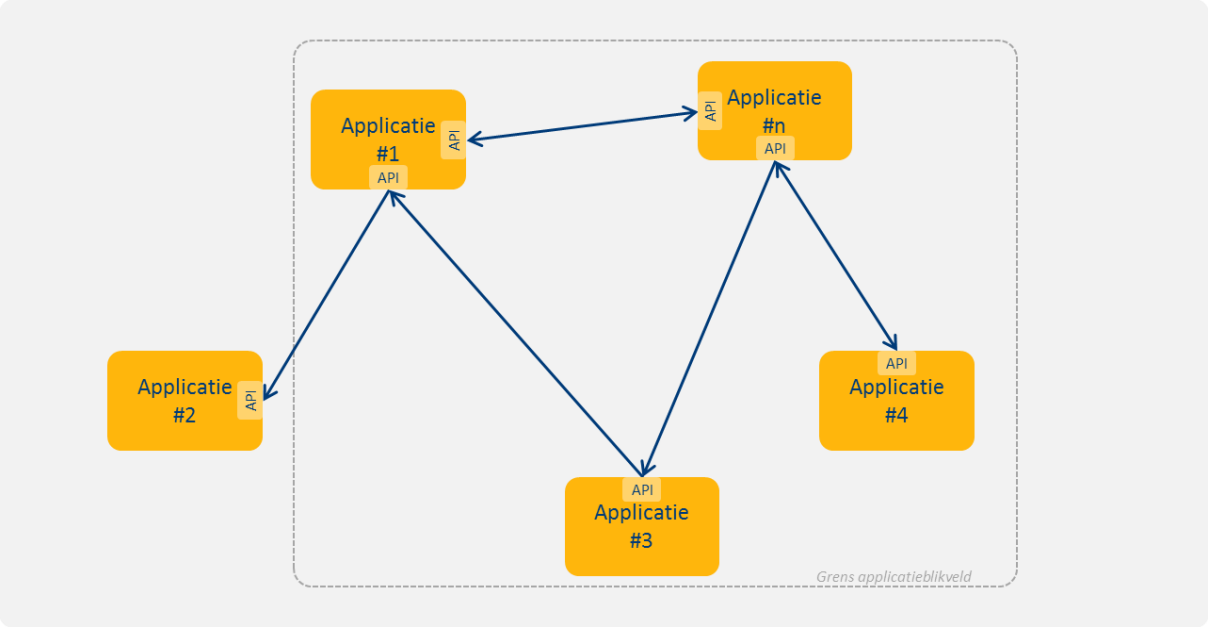
Kennis – Vereist het introduceren van een koppelvlak specifieke kennis? En is die kennis beschikbaar in de organisatie? En in de markt? Het is niet verstandig om technologie in huis te halen waar men zelf geen verstand van heeft.

Integratiepatronen

Als er gekozen is voor één of meer paradigma's en als bovengenoemde vragen zijn beantwoordt kan kunnen meer specifieke integratiepatronen worden geselecteerd. Een dergelijk patroon geeft meer specifiek aan welke componenten en interacties gebruikt worden. Daarbij wordt ook gebruik gemaakt van integratiemiddleware; software-infrastructuur die een deel van de integratie voor zijn rekening neemt. Hieronder volgt een toelichting op de meest voorkomende integratiepatronen. Elke vorm krijgt een drieletterige afkorting waarnaar later in dit stuk verwezen wordt.

Point-to-point (P2P)

Deze basisvorm van applicatie-integratie kenmerkt zich doordat applicaties direct met elkaar in verbinding staan. Daarmee worden externe afhankelijkheden van integratiemiddleware vermeden. Keerzijde is dat koppelen alleen werkt als de applicaties elkaar verstaan qua protocol en formaat en ook op alle andere eisen compatibel met elkaar zijn. In veel gevallen is dat niet zo, met name als integratie met bestaande of pakketapplicaties noodzakelijk is. Dan is al snel maatwerk nodig, zoals voor het afhandelen van uitzonderingssituaties. Een ander nadeel is dat wanneer meerdere koppelingen tussen applicaties op deze wijze gerealiseerd worden, elke applicatie veel kennis van andere applicaties moet hebben om te kunnen functioneren (groot applicatieblikveld). Point-to-point integratie kan bij alle genoemde paradigma's en met allerlei protocollen plaatsvinden. Tegenwoordig wordt meestal gebruik gemaakt van RESTful API's, maar het kunnen ook bijvoorbeeld Web Services zijn.



Voordelen:

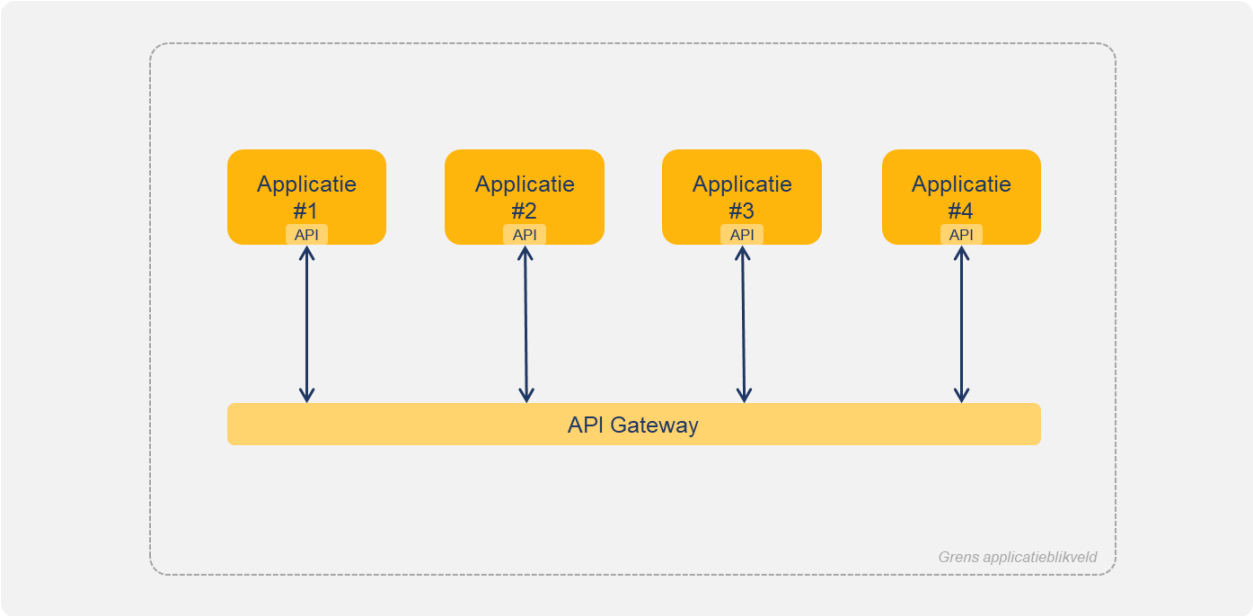
- Eenvoudige architectuur
- Snelle responsetijd door directe koppeling
- Geen afhankelijkheid van derde componenten zoals integratiemiddleware

Nadelen:

- Slecht schaalbaar/beheersbaar (bij n applicaties ontstaan 2^n koppelvlakken)
- Aanpassingen vragen relatief veel kennis van andere applicaties
- Vaak toch maatwerk of complexe oplossingen nodig, bijvoorbeeld gebaseerd op bestandsuitwisseling

Gateway (GTW)

Dit is een verdere optimalisatie van de point-to-point architectuur waarbij de applicaties van elkaar ontkoppeld worden door een gateway. Een dergelijke gateway is vooral een technische component die ervoor zorgt dat applicaties niet direct aan elkaar verbonden zijn. Hierdoor kunnen de fysieke locaties van applicaties eenvoudiger wijzigen. Daarnaast biedt een dergelijke gateway veelal ook functionaliteit voor het beheersen van de netwerkbelasting zodat applicaties niet overbelast worden. Het kan ook controles uitvoeren op de aanwezigheid van API-keys en het kan binnenkomende verzoeken authenticeren. Er wordt vaak ook eenvoudige functionaliteit geboden voor het vertalen van berichten. Dit alles ontlast applicaties van dergelijke functionaliteit. Een gateway wordt vaak gebruikt in combinatie met resource-oriëntatie en API's. Het wordt met name relevant als functionaliteit ook wordt aangeboden aan partijen buiten de organisatie. Externe partijen verwachten vaak een hoog serviceniveau.



Voordelen:

- Applicaties zijn technisch meer van elkaar ontkoppeld
- Overbelasting van applicaties wordt voorkomen
- Applicaties worden ontlast van API key management en authenticatie

Nadelen:

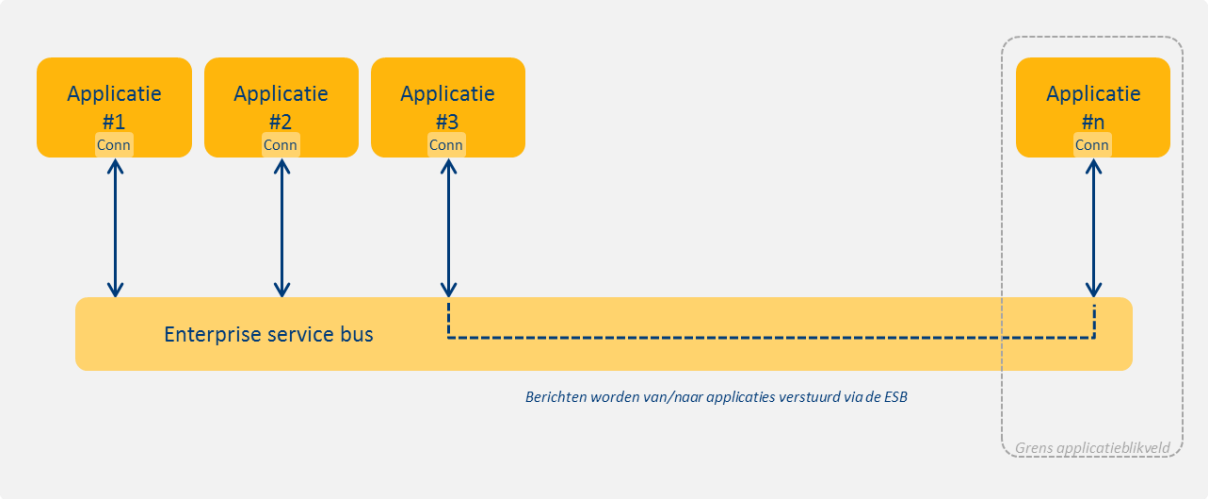
- Applicaties blijven functioneel nog aan elkaar gekoppeld
- Vraag inrichting van API management infrastructuur (API gateway)

Enterprise Service Bus (ESB)

Deze vorm van applicatie-integratie kenmerkt zich door het gebruik van een zogenoemde Enterprise Service Bus (ESB) als tussenstation voor data-uitwisseling. Data-uitwisseling via een ESB verloopt altijd op basis van berichten. Het wordt in veel gevallen vooral gebruikt om service-oriëntatie te ondersteunen. De

berichten zijn dan verzoeken tot het aanroepen van een service en het antwoord daarop. Het kan echter ook event-oriëntatie ondersteunen omdat het vaak ook publish-subscribe mechanismen biedt. De ESB vormt het aanspreekpunt voor alle aangesloten applicaties en routeert berichten naar de juiste bestemming, zodat applicaties zelf niet hoeven te weten hoe en waar elkaar te bereiken, met andere woorden: het applicatieblikveld is beperkt tot de applicatie zelf en de ESB.

De ESB kan ook gegarandeerde aflevering van berichten bewaken zodat applicaties niet afhankelijk zijn van elkaars beschikbaarheid, of zelf maatregelen moeten treffen om ontvangst van berichten te waarborgen. Verder kan de ESB berichten vertalen als aangesloten applicaties niet elkaars “taal” (formaat en protocol) spreken. Daarbij is er vaak een grote verzameling van adapters beschikbaar die allerlei technologie en pakketapplicaties snel kan koppelen. Dit alles levert een maximale ontkoppeling van de aangesloten applicaties. Veel ESB-oplossingen bieden bovendien voorzieningen waarmee centrale sturing en monitoring mogelijk is. De keerzijde is dat het ESB concept een complexe integratie-infrastructuur met zich meebrengt die actief beheerd moet worden en afhankelijk van de toepassing ook een behoorlijke investering vergt.



Voordelen:

- Ontkoppelt applicaties op het gebied van beschikbaarheid, bereikbaarheid en uitwisselformaat en protocol
- Biedt voorzieningen voor sturing en monitoring
- Breed scala aan integratiemogelijkheden door beschikbaarheid van adapters

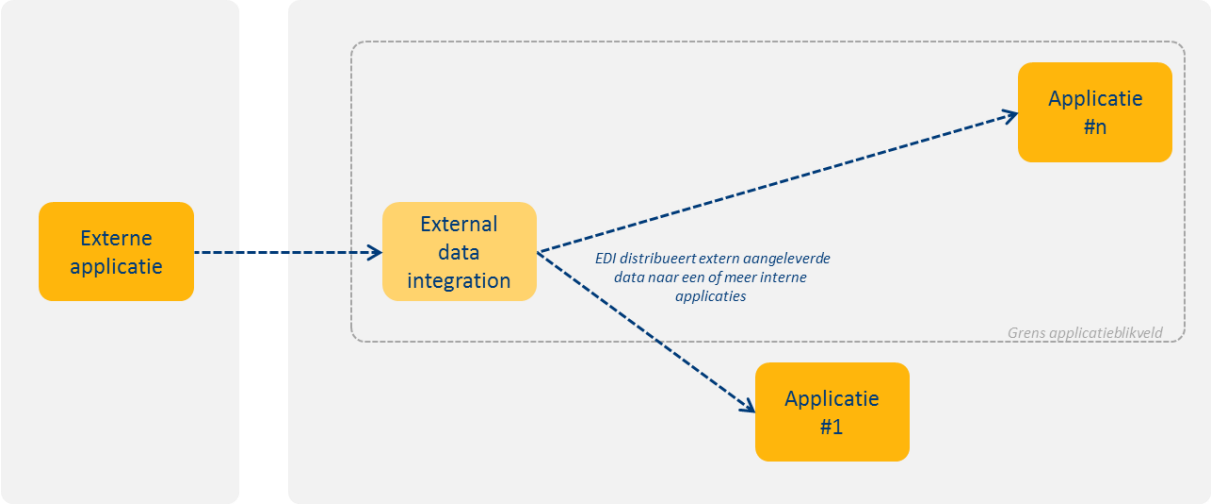
Nadelen:

- Relatief complexe infrastructuur die ook veel expertise vraagt en een investering vraagt
- Creëert de afhankelijkheid van een centraal component (en vaak ook een centrale afdeling), waardoor het minder goed past op agile ontwikkeling
- Kennis over applicatie-integratie komt in ESB die daardoor lastiger te beheren wordt

Business-to-Business integratie (B2B)

Deze vorm van applicatie-integratie kenmerkt zich door het gebruik van een apart component als tussenstation bij het uitwisselen van data met de “buitenwereld”. Dat wordt ook wel een Business-to-Business Gateway genoemd [1]. De belangrijkste redenen voor het inzetten van zo’n gateway is beveiliging en het aansluiten op specifieke uitwisselstandaarden. De gateway voorkomt dat externe applicaties rechtstreeks contact hebben met interne applicaties. Specifieke uitwisselstandaarden zijn vaak zo specifiek

dat standaard integratiemiddleware deze niet ondersteunt. Dit patroon ligt dicht aan tegen het gateway patroon. Dat patroon is echter meer gericht op ondersteuning van REST API's, terwijl dit patroon gericht is op andersoortige B2B uitwisselstandaarden zoals bijvoorbeeld ebXML. Dit patroon kan ook toegepast worden in situaties waarbij data in bulk uitgewisseld moet worden. Deze kunnen tussentijds op de gateway worden bewaard, waardoor er een vorm van ontkoppeling ontstaat.



Voordelen:

- Hogere veiligheid door ontkoppeling van interne en externe applicaties
- Ondersteuning specifieke (B2B) uitwisselstandaarden
- Kan problemen door te zware belasting verkleinen door data tijdelijk te bewaren

Nadelen:

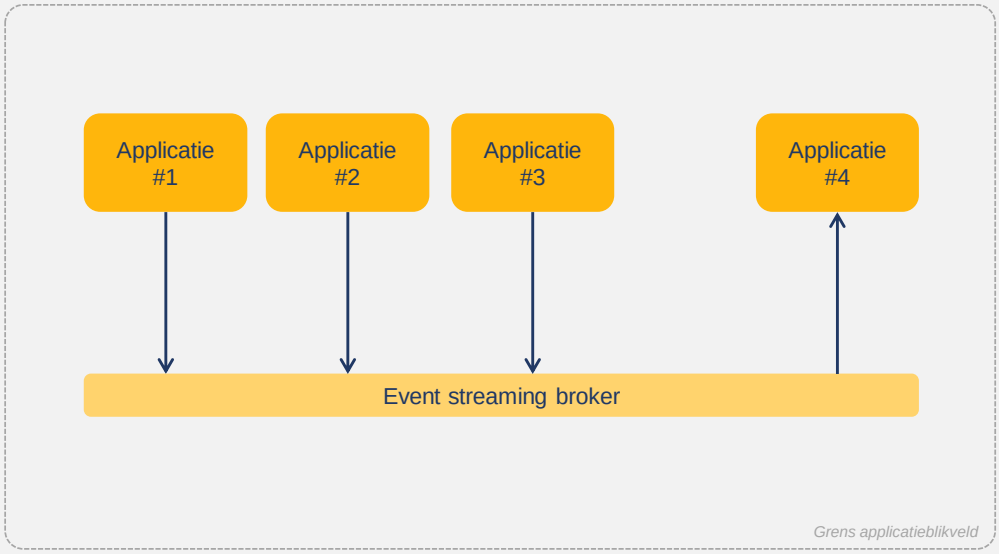
- Extra component in de architectuur
- Ondersteunt geen ad-hoc-bevragingen doordat het koppelvlak voorgedefinieerd is

Streaming (STR)

Streaming is een integatiepatroon dat specifiek is gericht op event-oriëntatie (afhandelen van gebeurtenissen). Het maakt gebruik van middleware die hier specifiek voor is geoptimaliseerd. Een *event stream broker* beheert de registers van abonnementhouders en routeert gebeurtenismeldingen naar de abonnementhouders.

Wat het streaming patroon toevoegt aan het basisidee van event-oriëntatie is dat het de gebeurtenissen zelf bundelt in streams. Het zijn expliciete dataverzamelingen die zelf ook beheerd kunnen worden. Gebeurtenissen kunnen na verspreiding uit de stream worden verwijderd, maar ze kunnen ook in de stream blijven staan. De stream kan daarmee ook een persistentie opslag van data worden. Er zijn dan ook specifieke zoektalen beschikbaar die in staat zijn om deze streams te bevragen, vergelijkbaar met wat SQL voor traditionele databases doet.

Streams kunnen ook worden gecombineerd met andere streams. Zo kan bijvoorbeeld een stream een soort samenvatting zijn van een andere stream. Deze nieuwe stream kan dan zijn geoptimaliseerd voor een specifiek gebruiksdoel. Daarnaast is het ook relatief eenvoudig om kopieën van streams beschikbaar te stellen op andere locaties. Dit is bijvoorbeeld relevant als een deel van het applicatielandschap zich op een andere locatie bevindt, zoals in de public cloud.



Voordelen:

- Afhankelijkheden worden beperkt; applicaties zijn alleen afhankelijk van gebeurtenissen
- Rijke mogelijkheden om de events ook in andere vormen en op andere locaties beschikbaar te stellen
- Geoptimaliseerd op hoge volumes

Nadelen:

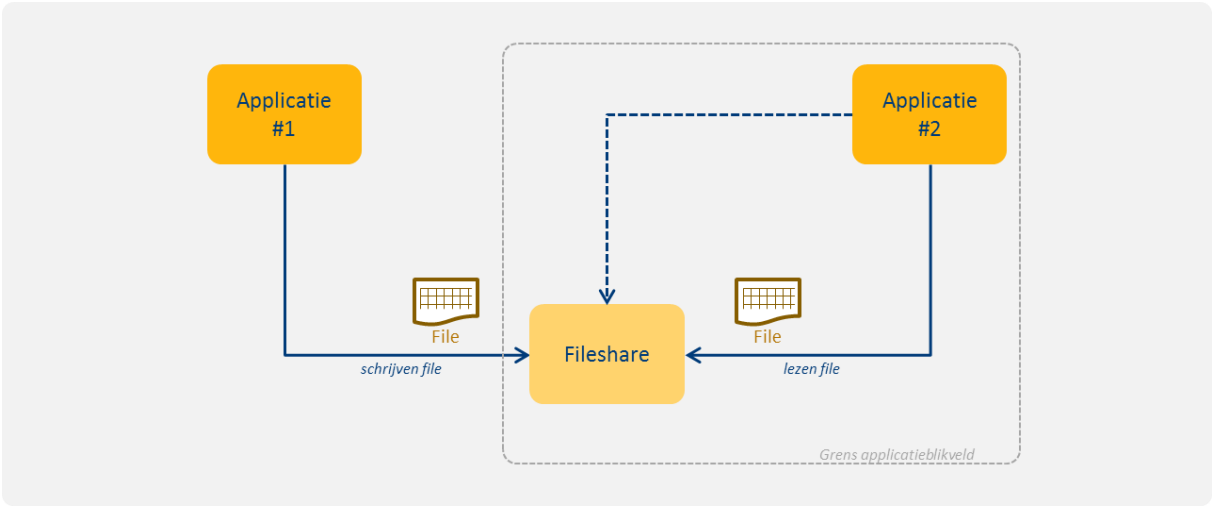
- Relatief complexe infrastructuur die ook veel expertise vraagt en een investering vraagt
- Vraagt een andere manier van denken dan meer traditionele integratiepatronen
- Past maar op een deel van de integratiescenario's

Bestandsuitwisseling (BUW)

Deze vorm van applicatie-integratie kenmerkt zich door het bestaan van een fysiek bestand als overdrachtsmiddel voor de uit te wisselen data. Deze vorm werd en wordt veelal gekozen als point-to-pointintegratie niet mogelijk is, bijvoorbeeld omdat de ene applicatie geen bruikbaar koppelvlak biedt. Bij bestandsuitwisseling exporteert de initiërende applicatie een bestand en zet dat op een afgesproken locatie neer, zodat dat bestand vervolgens door de andere applicatie wordt ingelezen en verwerkt.

Een voordeel van deze integratievorm is dat zeer grote hoeveelheden data overgedragen kunnen worden en dat de wederzijdse afhankelijkheid minimaal is. Het applicatieblikveld beperkt zich tot de applicatie zelf en de fileshare die als uitwisselpunt fungeert.

Een nadeel is dat er een tussenmedium nodig is, bijvoorbeeld een fileserver, en dat het verwerken van uitzonderingssituaties al gauw complex wordt. Als bijvoorbeeld een bestandsfout optreedt, is er geen directe methode om dat terug te koppelen naar de bron. Ook gelden hier de bij P2P genoemde nadelen van slechte schaalbaarheid en herbruikbaarheid.



Voordelen:

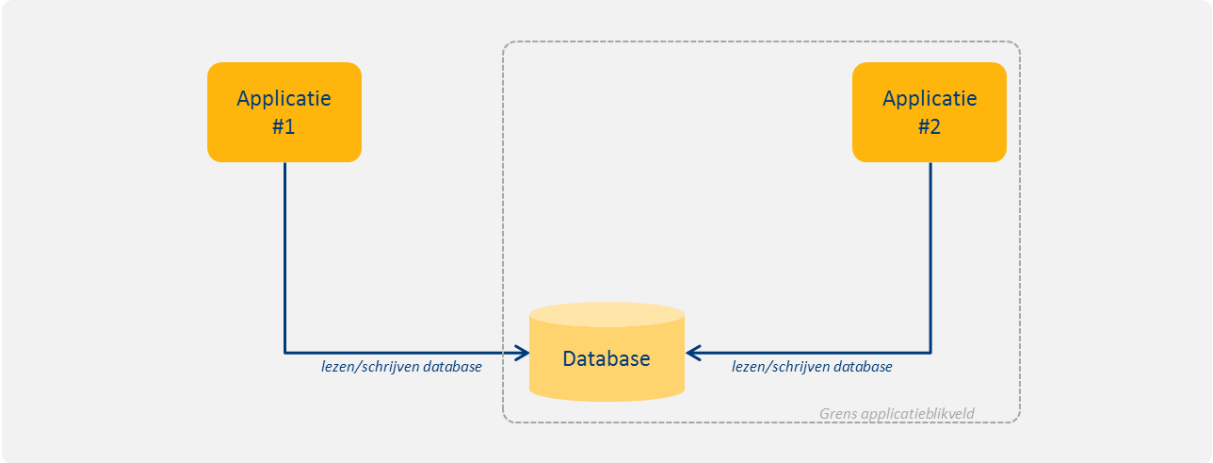
- Kan zeer grote datavolumes aan
- Minimale wederzijdse afhankelijkheid

Nadelen:

- Vraagt een fileserver als tussenstation
- Beperkte mogelijkheden voor afhandelen uitzonderingssituaties
- Slecht schaalbaar/beheerbaar (bij n applicaties ontstaan 2^n koppelvlakken)
- Slecht herbruikbaar door ontbreken van standaardisatie

Gemeenschappelijke database (GDB)

Deze vorm van applicatie-integratie kenmerkt zich door de aanwezigheid van een database die door twee of meer applicaties gebruikt wordt. Er is daarom niet zozeer sprake van overdracht van data maar van gemeenschappelijk gebruik van data. Deze integratievorm wordt al zeer lang toegepast en is relatief eenvoudig te realiseren. Dat is dan ook meteen het belangrijkste voordeel, samen met het feit dat de koppeling niet beperkt is tot twee applicaties; in beginsel kunnen oneindig veel applicaties op de gemeenschappelijke database aansluiten, als het databaseplatform dat tenminste aankan.



De wederzijdse afhankelijkheid van de gekoppelde applicaties is groot. Een databasefout treft direct alle applicaties. En wijzigingen moeten in alle applicaties tegelijk doorgevoerd worden, wat grote impact kan hebben op het beheer. Alle validaties en bedrijfsregels die van toepassing zijn op de data in de database moeten in alle aangesloten applicaties doorgevoerd en consistent gehouden worden.

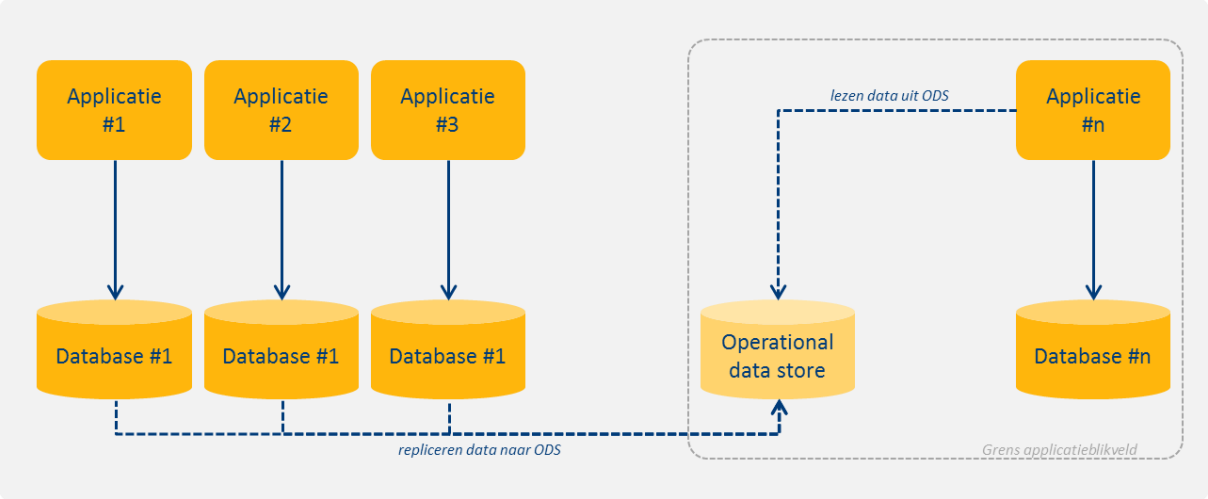
Voordelen:
- Vaak eenvoudig realiseerbaar
- Kan meerdere applicaties tegelijk koppelen

Nadelen:
- Grote wederzijdse afhankelijkheid tussen applicaties
- Vereist duplicatie van validatie- en bedrijfslogica
- Slecht beheerbaar doordat wijzigingen tegelijk doorgevoerd moeten worden

Operational datastore (ODS)

Deze vorm van applicatie-integratie kenmerkt zich door het gebruik van een operational datastore (ODS). Een ODS is een separate database die kopieën bevat van data uit andere databases, en die die data met hoge beschikbaarheid ontsluit [2]. De ODS wordt veelal gevuld met datareplicatietechnologie, waarbij data uit verschillende bronnen met elkaar gecombineerd worden tot één integrale dataverzameling.

Keerzijde van de hoge ontkoppeling en goede performance die een ODS kan bieden, is dat de architectuur complex wordt als doelapplicaties niet alleen data lezen maar ook willen toevoegen of wijzigen.



Voordelen:
- Op één plaats toegang tot data uit meerdere bronapplicaties
- Biedt veelal een hogere beschikbaarheid, capaciteit en responstijd dan bronapplicaties
- Ontkoppelt doelapplicaties van bronapplicaties

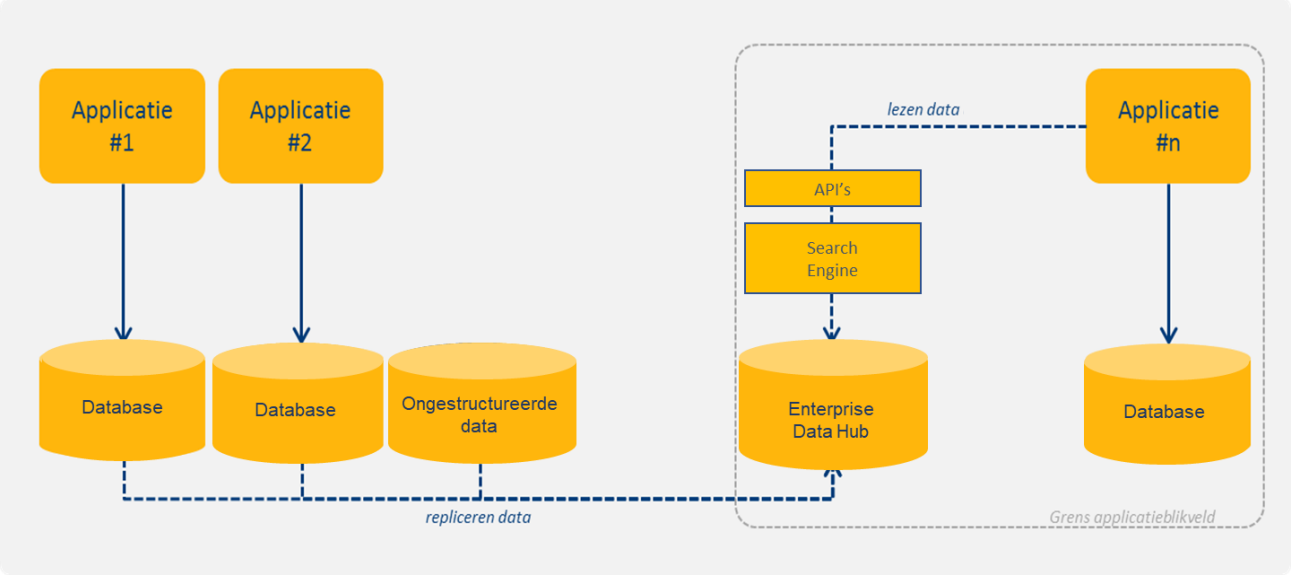
Nadelen:
- Vereist duplicatie van validatie- en bedrijfslogica
- Wordt complex bij schrijven van data

Enterprise data hub (EDH)

Deze vorm van applicatie-integratie is een doorontwikkeling van de operational data store. Het kan naast actuele data ook historie bevatten en het kan ook ongestructureerde data bevatten. Er is typisch meer

functionaliteit aan gekoppeld voor het beschikbaar stellen en delen van de data. Denk ondermeer aan API's voor het ontsluiten van de data, een search engine voor het zoeken in de data maar ook meer aandacht voor de beveiliging van de data. Het gebruik is vaak ook breder dan dat van een ODS. Zo kunnen er bijvoorbeeld ook allerlei analyses (bijvoorbeeld in het kader van data science) mee worden uitgevoerd.

Een Enterprise Data Hub (EDH) is ondermeer ontstaan door een toenemende aandacht voor Big Data; andersoortige data zoals ongestructureerde data waar ook allerlei waardevolle informatie uit gehaald kan worden. In die zin lijkt het ook wel op een data lake, maar die laatste is meer gericht op het verzamelen van externe databronnen terwijl de EDH ook tussen applicaties een rol heeft (en dus als een vorm van applicatie-integratie kan worden gezien). Door dit soort andere data gebruikt een EDH vaak ook andersoortige databases (NoSQL). Daarnaast is de data in een EDH vaak minder getransformeerd en bevat het dus meer ruwe data. In die context wordt ook wel gesproken over Extract-Load-Transform (ELT) i.p.v. Extract-Transform-Load (ETL). Hierdoor kan sneller toegang worden gekregen tot data (hogere actualiteit).



Voordelen:

- Op één plaats toegang tot data uit meerdere bronnen (gestructureerd en ongestructureerd)
- Biedt veelal een hogere beschikbaarheid, capaciteit en responstijd dan bronapplicaties
- Ontkoppelt doelapplicaties van bronapplicaties
- Sneller toegang tot data (hogere actualiteit)

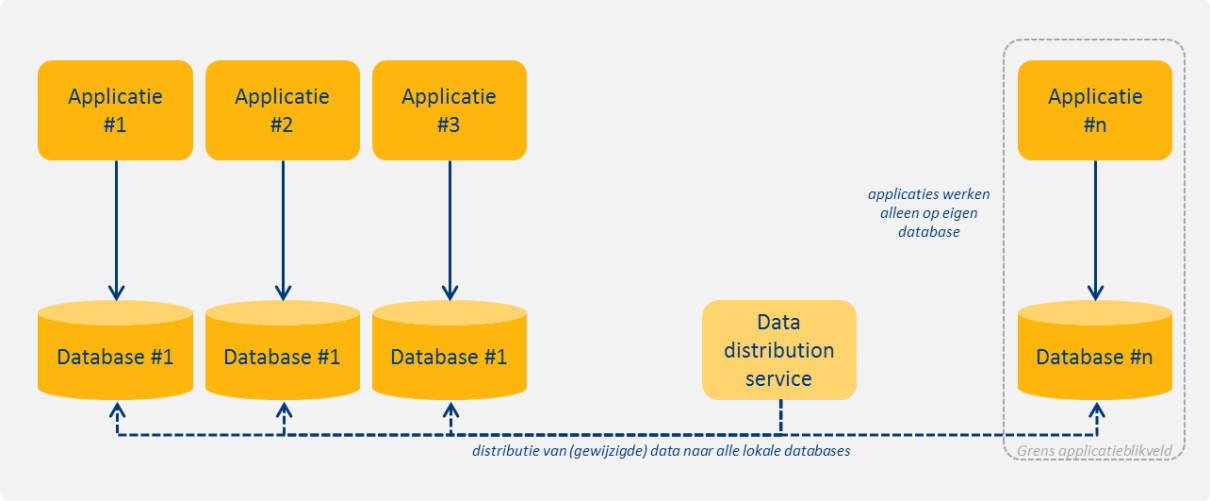
Nadelen:

- Vraagt kennis van nieuwere technologie die mogelijk nog niet aanwezig is
- Vereist duplicatie van validatie- en bedrijfslogica
- Wordt complex bij schrijven van data

Datadistributieservice (DDS)

Deze vorm van applicatie-integratie kenmerkt zich door het gebruik van een datadistributiemechanisme [3]. Deze bewaakt de distributie van data die in verschillende bronapplicaties wordt beheerd. Zodra de data wijzigt zorgt het mechanisme ervoor dat alle afnemers (doelapplicaties) de wijzigingen ontvangen. Het is daardoor voor de applicaties net alsof alle data lokaal staat, met andere woorden: het applicatieblikveld is minimaal. Applicaties hoeven zich daardoor geen rekenschap te geven van integratie-aspecten. Deze vorm

van integratie heeft interessante voordelen. Belangrijk is dat integratieproblematiek wordt weggetrokken bij de applicaties en op één enkele plaats belegd.



Voordelen:

- Belegt integratieproblematiek op één plaats, buiten de applicaties
- Biedt een hogere beschikbaarheid door een (technische) kopie te maken van data
- Biedt hoge performance en schaalbaarheid

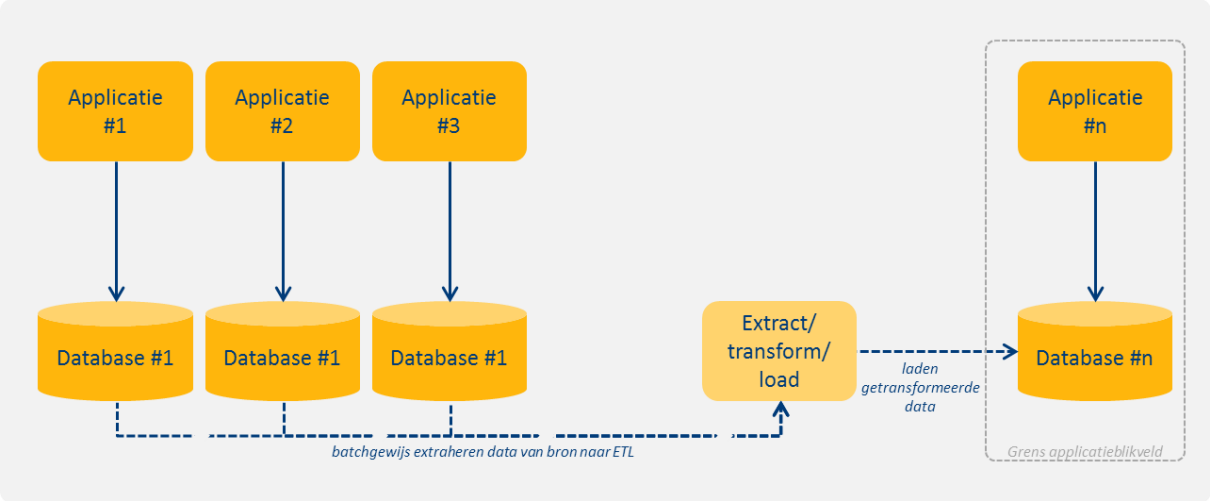
Nadelen:

- Biedt geen oplossing voor “on-demand” informatiebehoefte
- Moeilijk te realiseren bij gesloten applicaties (zoals pakketapplicaties en clouddiensten)
- Er zijn relatief minder standaardproducten voor beschikbaar

Extract-Transform-Load (ETL)

Deze vorm van applicatie-integratie kenmerkt zich door het gebruik van een apart component dat data uit de ene applicatie extraheert, daarop transformaties uitvoert, en het resultaat vervolgens laadt in een andere database (van een doelapplicatie). Deze vorm van integratie is vooral bekend vanwege de softwareproducten die gebruikt worden om data uit een operationele applicatie te verplaatsen naar een datawarehouse. Het is echter breder toepasbaar en ook inzetbaar tussen databases van individuele applicaties. Veelal wordt de ETL-procedure op gezette tijdstippen gestart, maar dat ook naar aanleiding van specifieke gebeurtenissen.

Belangrijke eigenschappen van ETL zijn dat het minimaal verstorend werkt op de bronapplicatie omdat de data daar ongewijzigd uit wordt geëxtraheerd en alle transformaties buiten die bronapplicatie worden uitgevoerd, en dat de transformaties declaratief worden gedefinieerd. Daardoor blijft een en ander goed beheersbaar. Ook zijn grote datavolumes geen probleem. Als gevolg van de opkomst van business intelligence is er veel ETL-tooling beschikbaar, soms als onderdeel van het databaseplatform. ETL is typisch een unidirectionele vorm van applicatie-integratie, die geen oplossing biedt voor het realtime opvragen van data uit een applicatie.



Voordelen:

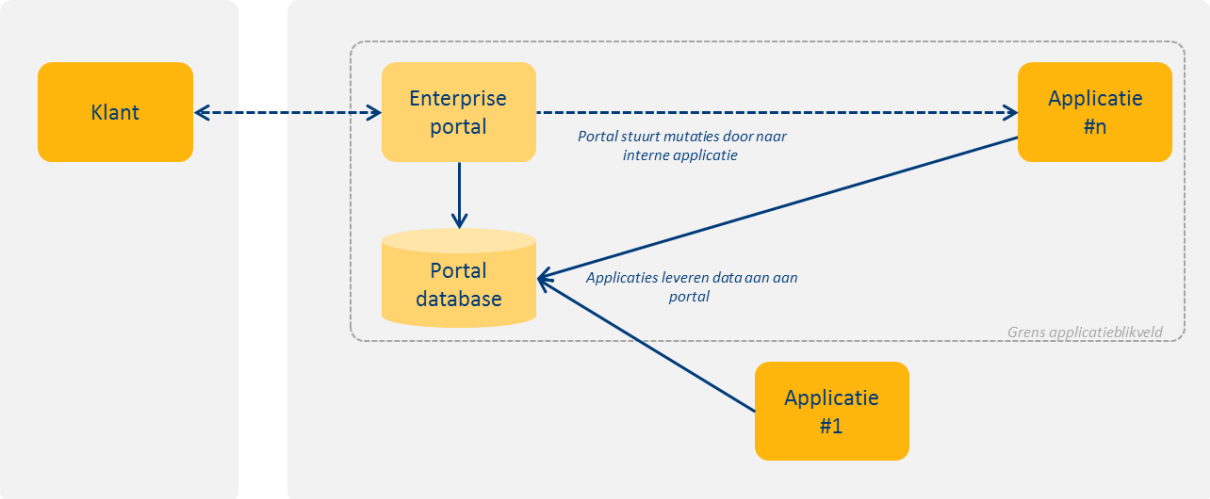
- Weinig impact op applicatie doordat transformaties buiten de applicatie zijn gedefinieerd
- Goed beheersbaar door separate, declaratieve definitie van extractie- en transformatieprocedures
- Veel en geavanceerde tooling beschikbaar
- Kan zeer grote datavolumes aan

Nadelen:

- Levert meestal geen realtime-integratie

Enterpriseportaal (EPS)

Deze vorm van applicatie-integratie kenmerkt zich door het gebruik van een portaalplatform voor het beschikbaar stellen van data en functionaliteit. Belangrijk is dat door het inzetten van een enterpriseportaal gebruikers de beleving hebben van één geïntegreerde applicatie, terwijl de applicaties op de achtergrond helemaal niet geïntegreerd hoeven te zijn. Ook het portaal ervoor zorgen dat alles er op eenzelfde manier uit ziet (één look-and-feel). Belangrijk vanuit het perspectief van informatiebeveiliging is dat de achterliggende applicaties ontkoppeld blijven van de buitenwereld, wat een extra verdediging geeft tegen hackers. Daarnaast kan een portaal ook eenvoudiger een hogere beschikbaarheid garanderen dan andere applicaties.



Aandachtspunt bij portalen is hoe gebruikers terugkoppeling krijgen op mutaties die zij doorgeven. Immers, mutaties moeten aan de interne applicaties doorgezonden worden, verwerkt, en vervolgens moeten de resultaten vanuit de interne applicaties weer teruggekoppeld worden aan het portaal.

Voordelen:

- Geïntegreerde gebruikerservaring, met één look-and-feel
- Ontkoppeling van portaal en achterliggende applicaties en daarmee hogere beschikbaarheid en veiligheid
- Ontkoppeling maakt het mogelijk dat portalen een hogere beschikbaarheid hebben dan de achterliggende interne applicaties
- Ontkoppeling voorkomt problemen met onder andere beschikbaarheid van interne applicaties als gevolg van zware belasting door portaalgebruikers.

Nadelen:

- Extra component in de architectuur

Vergelijking van integratiepatronen

De volgende tabel toont een vergelijking van de bovenbeschreven applicatie-integratiepatronen. Om de integratiepatronen goed te kunnen beoordelen is bij elk aspect een eis beschreven; de scores in de kolommen geven telkens aan in hoeverre het patroon kan voldoen aan de eis. Daarbij staat een ++ score altijd voor de gunstigste en - - voor minst gunstigste score. Een vraagteken geeft aan dat het antwoord zo afhankelijk is van de specifieke situatie dat geen algemene score gegeven kan worden.

Aspect	P2P	GTW	ESB	B2B	STR	BUW	GDB	ODS	EDH	DDS	ETL	EPS
Richting – ondersteunt bidirectioneel dataverkeer	++	++	++	+	+/-	+/-	++	+/-	+/-	++	- -	+/-
Initiatief – ondersteunt push én pull dataverkeer	++	+	+	+	- -	- -	++	-	+	++	- -	+/-
Afhankelijkheid – leidt niet tot meer afhankelijkheden tussen applicaties	- -	+	++	+	++	-	- -	+	+	++	++	+
Datavolume – kan grote datavolumes per keer verwerken	+/-	+/-	-	+/-	-	++	++	+	++	+/-	++	-
Transactiefrequentie – kan grote aantallen transacties verwerken	++	+	+	+/-	++	-	++	++	++	+	-	+/-
Responstijd – transacties verwerken met korte responstijd	++	+	+/-	+/-	+	- -	++	+	++	++	- -	+/-
Actualiteit – leveren actuele data	++	++	+	+/-	+	+/-	++	+/-	+/-	+	-	+/-
Beschikbaarheid – hoge beschikbaar	+/-	+/-	+	++	++	++	+	++	++	++	++	++
Vertrouwelijkheid – waarborgen vertrouwelijkheid van data	+	++	+	+/-	+	+	-	-	+/-	-	-	+/-
Integriteit – waarborgt integriteit van data	+/-	+/-	++	+/-	++	-	++	+	+	+	-	+/-
Authenticatie en autorisatie – ondersteunt authenticatie	+	++	-	++	+	- -	-	-	++	+/-	-	++
Schaalbaarheid – eenvoudig op te schalen	-	+	++	+	++	+	++	++	++	+/-	++	+
Monitoring gebruik – kan gemonitord en gestuurd worden	- -	++	++	+	++	-	- -	-	++	+/-	+	+
Verstoring operatie – leidt niet tot verstoring van bronapplicaties	- -	- -	+	+	++	-	- -	+	++	++	++	+
Hacker-bestendigheid – voorkomt hack-kwetsbaarheden	-	+	+	++	+	+/-	- -	-	+/-	++	++	++
Leveranciersafhankelijkheid – leidt niet tot afhankelijkheid	+	+	-	+/-	+/-	+	+	+/-	+/-	+/-	-	+/-
Standaardfuncties – “out-of-the-box” functies	-	+	++	+	+/-	- -	+/-	+	++	++	++	+
Mogelijkheden applicaties – hoeveelheid mogelijkheden voor integratie	++	- -	- -	- -	- -	-	++	-	+	- -	+/-	- -

Architectuurprincipes

Architectuurprincipes zijn een belangrijke basis voor architectuur. Om die reden is het verstandig om ook architectuurprincipes te beschrijven voor applicatie-integratie. In dit whitepaper beschrijven we daarom een aantal van dat soort architectuurprincipes. Merk op dat deze lijst niet als volledig is bedoeld, maar slechts enkele van de belangrijkste principes presenteert.

Architectuurprincipes voor service-oriëntatie

Services worden zo gedefinieerd dat doel en afbakening functioneel logisch zijn

Rationale
- Alleen dan ontstaat een landschap van functionele bouwstenen
- Maakt het mogelijk om bedrijfsprocessen te reorganiseren zonder dat altijd ook softwareaanpassingen nodig zijn

Implicaties
- Koppelvlakken tussen applicaties zijn functioneel georiënteerd
- Onderdeel van elke servicedefinitie is ook een kwaliteitscontract (“SLA”)
- Services zijn toestandsloos en hebben geen “workflow” aspecten in zich

Services worden gedefinieerd door de aanbieder, niet door de afnemer

Rationale
- Alleen dan wordt hergebruik mogelijk

Implicaties
- Services en data zijn gebaseerd op domeinkennis van aanbieder
- Afnemers zijn zelf verantwoordelijk voor eventuele conversie van data of protocol (opm. in dit geval zien we een ESB ook als afnemer)
- Afnemers passen hun applicatielogica aan aan de aangeboden services ipv. andersom

Services worden met hun kwaliteitskenmerken opgenomen in een servicecatalogus

Rationale
- Alleen zo wordt het mogelijk om services als bouwstenen te gebruiken bij het samenstellen van bedrijfsprocessen

Implicaties
- Er is een servicecatalogus beschikbaar
- Er is een proces om services (en wijzigingen) aan te melden voor de catalogus

Services zijn zelf verantwoordelijk voor het technisch valideren van hun invoer

Rationale
- Alleen zo kan de achterliggende applicatie zijn (data)integriteit waarborgen

Implicaties
- Elke service heeft een technische validatieroutine
- Als data ook aan de kant van de afnemer gevalideerd worden, vindt dubbele validatie plaats

Services zijn zelf verantwoordelijk voor het authenticeren en autoriseren van afnemers

Rationale
- Uitgaan van veronderstelde authenticatie en autorisatie door een derde component zou een veiligheidsrisico introduceren

Implicaties
- Elke service die niet-publieke data ontsluit heeft voert een authenticatie en autorisatie uit

Een serviceaanbieder beslist autonoom over wijzigen van de implementatie van een service

Rationale
- Waarborgt de onafhankelijkheid van de serviceverlener (en “loose-coupling”) en reduceert daarmee complexiteit

Implicaties
- De implementatie van een service heeft geen invloed op de definitie van die service

Services voorzien in soepele versieovergangen door ook de voorgaande versie te ondersteunen

Rationale
- Ondersteunen van de voorgaande versie biedt afnemers tijd om over te stappen en voorkomt plotseling niet meer werken van applicaties

Implicaties
- Nieuwe versies worden tijdig aangekondigd en volgen niet te snel op eerdere versies
- Afnemers worden verplicht om binnen redelijke termijn over te gaan op de nieuwe versie van een service

Services ondersteunen logging en tracing op ketenniveau

Rationale
- Als processen worden opgebouwd uit services, is tracing op ketenniveau alleen mogelijk als de gehele keten dat ondersteunt

Implicaties
- Services hebben een ketenflow-id als invoerparameter gebruiken die in de eigen logging
- Services geven bij het aanroepen van sub-services bovenliggende ketenflow-id’s mee als parameter

Architectuurprincipes voor event-oriëntatie

Integratie tussen applicaties geschiedt uitsluitend via gebeurtenisgerelateerde berichten

Rationale
- Consistent volgen van dit principe vormt de basis voor een event driven architecture

Implicaties
- Alle applicaties bieden koppelvlakken voor het abonneren op gebeurtenissen en het sturen van gebeurtenismeldingen.
- Gebeurtenissen (functioneel, niet technisch) zijn leidend bij applicatie-ontwerp
- Applicaties onderkennen en melden het optreden van gebeurtenissen direct aan abonnementhouders

Gebeurtenissen worden met hun meldingsberichten en kwaliteitseigenschappen opgenomen in een gebeurteniscatalogus

Rationale
- Om te kunnen abonneren op gebeurtenissen moet bekend zijn welke gebeurtenissen kunnen plaatsvinden
- Om gebeurtenissen te kunnen verwerken moet bekend zijn hoe die gemeld worden en met welke kwaliteitskenmerken

Implicaties
- Er is een gebeurtenissen-catalogus beschikbaar
- Er is een proces om gebeurtenissen (en wijzigingen) aan te melden voor de catalogus

Applicaties bieden de mogelijkheid om te abonneren op in het applicatie optredende gebeurtenissen

Rationale
- Dit is noodzakelijk: abonneren is voor andere applicaties de enige manier om van gebeurtenissen op de hoogte gesteld te worden

Implicaties
- Applicaties bieden een koppelvlak waarmee andere applicaties zich kunnen abonneren
- Applicaties houden een register bij van abonnementhouders (dit kan eventueel gedelegeerd worden naar een event stream broker)

- Applicaties onderkennen gebeurtenissen die kunnen optreden, met welke data daarbij relevant zijn
- Applicaties sturen bij het optreden van gebeurtenissen een melding aan alle op die gebeurtenis geabonneerde applicaties (of aan de event stream broker), met in die melding alle relevante informatie

---

Applicaties abonneren zich op alle voor hen relevante gebeurtenissen

Rationale
- Dit is noodzakelijk: zonder abonnement krijgt een applicatie geen meldingen van gebeurtenissen

Implicaties
- Applicaties onderkennen welke (typen) gebeurtenissen voor hen relevant zijn
- Applicaties abonneren zich op voor hen relevante gebeurtenissen
- Applicaties kunnen gebeurtenismeldingen ontvangen en verwerken

---

Applicaties zijn zelf verantwoordelijk voor autoriseren van abonnementhouders

Rationale
- Informatiebeveiligingsrichtlijnen vereisen dat alleen geautoriseerde applicaties een abonnement kunnen nemen op bepaalde gebeurtenissen
- Applicaties kunnen er niet van uitgaan dat autorisatie extern plaatsvindt

Implicaties
- Applicaties die vertrouwelijke gebeurtenissen melden, hebben een mechanisme voor autorisatie van (potentiële) abonnementhouders (kan eventueel ook gedelegeerd worden aan een event stream broker)

Applicaties zijn intern gestructureerd op basis van de gebeurtenissen die ze verwerken

Rationale
- Maakt applicaties beter beheerbaar

Implicaties
- Event-oriëntatie is het leidende paradigma bij het ontwerpen van applicaties

Architectuurprincipes voor resource-oriëntatie

Integratie tussen applicaties geschiedt uitsluitend door directe benadering van resources

Rationale
- Consistent volgen van dit principe vormt de basis voor een resourcegeoriënteerde architectuur

Implicaties
- Applicaties stellen de door hen beheerde resources rechtstreeks beschikbaar aan andere applicaties
- Resources zijn via een URI-strategie rechtstreeks benaderbaar buiten de beherende applicatie

---

Resources worden uniek geïdentificeerd via een eenduidige, applicatieonafhankelijke URI-strategie

Rationale
- Zonder eenduidige URI-strategie is het niet mogelijk alle resources rechtstreeks te benaderen
- Applicatieonafhankelijkheid is een van de hoekstenen van ROA

Implicaties
- Er is een URI-strategie opgesteld die voor elke denkbare resource definieert met welke URI de resource gedefinieerd wordt
- Applicaties implementeren de URI-strategie, zowel om de resources die zij zelf beheren voor andere applicaties beschikbaar te stellen, alsook om resources in andere applicaties te benaderen.

- Er is een naming-mechanisme zijn dat URI's vertaalt naar fysieke locaties (zoals DNS dat doet voor internetadresssen)

Er is een resource-ontologie die alle (typen) resources definieert met hun scope, mogelijke operaties en samenhang met andere resources

Rationale
- Nodig omdat resources via één URI ontsloten worden. Scope e.d. moeten daarom helder zijn

Implicaties
- Er is een resource-ontologie beschikbaar
- De resource-ontologie is leidend bij applicatieontwerpen
- Bewerkingen op de in applicaties beheerde data zijn onderdeel gemaakt van de operaties die in de resource-ontologie zijn gedefinieerd

---

Benaderen van resources gebeurt via een gestandaardiseerd RESTful koppelvlak

Rationale
- Realiseert standaardisatie in het benaderen van resources
- Sluit aan bij opkomende industriestandaard

Implicaties
- Applicaties ondersteunen RESTful koppelingen

Architectuurprincipes voor data-oriëntatie

Er is geen directe integratie tussen applicaties

Rationale
- Dit voorkomt afhankelijkheden tijdens ontwikkeling en beheer

Implicaties
- Alle benodigde data wordt in lokale databases opgeslagen
- Wijzigingen in data worden direct gerepliceerd naar de databases van andere applicaties die die data gebruiken

---

Er is een mechanisme dat data tussen databases repliceert en transformeert

Rationale
- Replicatie is nodig om applicaties over actuele data te laten beschikken
- Transformatie is nodig wanneer datamodellen van twee databases verschillen

Implicaties
- Er is inzicht in welke data door welke applicaties nodig is
- Er is inzicht in de verschillen tussen datamodellen in databases
- Er is een datadistributiemechanisme aanwezig dat ook transformatie ondersteunt

---

Er zijn bronapplicaties aangewezen voor data

Rationale
- Anders is er een grote kans op inconsistenties in de data

Implicaties
- Voor alle data is aangewezen welke applicatie de bronapplicatie is

Implementatie

Infrastructurele aanpak

Bij de implementatie van applicatie-integratiemechanismen komen twee denkrichtingen bij elkaar:

- Welke integratiepatronen sluit het beste aan bij de concrete casus, m.a.w. hoe kunnen de twee (of meer) applicaties snel en betaalbaar gekoppeld worden zodat het project verder kan?
- Welke integratiepatronen sluiten het beste aan bij de langetermijn doelstellingen van business en architectuur, met name beheerbaarheid, reductie van complexiteit, en kostenbeheersing?

Aangezien een project altijd zal zoeken naar een oplossing die past binnen de projectkaders (doorlooptijd, projectbudget), zijn sommige integratiepatronen (bijvoorbeeld ESB) alleen succesvol wanneer ze als infrastructureel project geïmplementeerd worden. Een project zal immers niet snel vragen om een ESB om twee applicaties “even” te koppelen. De keerzijde van een infrastructurele benadering is echter dat daarvoor moeilijk budget en mensen vrij te maken zijn, en dat zo’n project al gauw als “IT-feestje” beschouwd kan worden.

Om een applicatie-integratie-infrastructuur te implementeren, zullen de architecten daarom rekening moeten houden met het volgende:

- Koppel de introductie van de integratie-infrastructuur altijd aan ten minste één concreet businessproject dat van die voorziening gaat profiteren, zodat de “benefits” voor iedereen duidelijk zijn.
- Maak een inventarisatie van alle locaties/situaties waar de integratie-infrastructuur van waarde kan zijn. Maak daarbij onderscheid tussen kostenbesparing enerzijds en het creëren van nieuwe mogelijkheden anderzijds. Met name dat laatste is waardevol. Maak het resultaat van de inventarisatie zichtbaar, bijvoorbeeld in een plot van het applicatielandschap met de nieuwe applicatiekoppelingen er op geprojecteerd.
- Maak een businesscase (lees bijvoorbeeld [6] als het een SOA betreft) om budget en prioriteit voor het project te krijgen, en om later de waarde ervan te kunnen controleren en aantonen.
- Plan de implementatie in fasen, zo dat de eerste fase al meteen het eerste businessproject kan ondersteunen. Een snel succes, hoe klein ook, werkt aanstekelijk. Maak daarom een roadmap die niet alleen toont welke onderdelen of verbeteringen er gepland staan, maar ook welke resultaten daarmee gehaald (kunnen) worden.

Hybride situatie

De kans is klein dat het gehele applicatielandschap het beste geïntegreerd kan worden met behulp van een enkel integratiepatroon. Meestal bestaat een applicatielandschap uit een veelheid van soorten applicaties en platformen, waaronder maatwerk, standaardapplicaties en clouddiensten.

De vraag is dan of het verstandig is om al die applicaties in één stramien te persen. Het voordeel is natuurlijk de eenvoud en de besparing die voortvloeit uit het niet hoeven implementeren en ondersteunen van meerdere voorzieningen. Maar daar tegenover staan de meerkosten die nodig zijn om connectors te bouwen en te onderhouden, en de eventuele stabiliteits- of securityproblemen die het gevolg kunnen zijn van niet-passende koppelvlakken.

Verstandig kan in zo’n situatie zijn om te kiezen voor een “leidend paradigma”, dat wil zeggen een integratieaanpak die in beginsel gevolgd moet worden, maar waarvan afgeweken kan worden als daar goede redenen voor zijn. Dat afwijken moet dan een architectuurbesluit zijn, te nemen door de

architecture board in de organisatie. Ook moet het leidend paradigma als architectuurrichtlijn worden meegenomen in nieuwe oplossingen. Daartoe moeten de architecten heldere criteria opstellen waarmee bepaald kan worden wanneer afwijken is toegestaan. Zo'n leidendparadigma-aanpak garandeert dat het applicatielandschap langzaam maar zeker in de goede richting evolueert, terwijl het geen onmogelijke eisen oplevert voor bestaande applicaties.

iPaaS

Cloud-applicaties zijn over het algemeen prima te integreren met interne en/of andere cloudapplicaties. De integratiemogelijkheden zijn meestal echter beperkt tot integratie op basis van API's. Indien integratie met cloud-applicaties belangrijk is, dan is het zaak daar goed naar te kijken en eventueel een "cloud integration adapter" op te nemen in de architectuurroadmap.

Een andere mogelijkheid is het gebruik maken van clouddiensten voor integratie. Men spreekt dan wel van iPaaS: integration platform as a service. Er komen steeds meer cloudleveranciers die dergelijke platformfunctionaliteit aanbieden. De kracht daarvan ligt in de flexibiliteit die de leverancier kan bieden en die normaliter niet haalbaar is voor gewone organisaties, voor wie integratie immers alleen een hulpmiddel is. De opkomst van iPaaS zal leiden tot verdere standaardisatie van integratieprotocollen en formaten in de richting van resource oriented architectures gebaseerd op REST en JSON, ten koste van SOAP en XML.

Volgens sommigen zullen deze ontwikkelingen over enkele jaren zelfs leiden tot het uitsterven van ESB-architecturen [10]. Zo'n vaart zal het waarschijnlijk niet lopen, met name omdat in de IT ontwikkelingen (in tegenstelling tot het beeld dat veel mensen hebben) niet zo snel gaan. Desondanks is de vraag gerechtvaardigd of grote investeringen in ESB-infrastructuur nog verantwoord zijn nu deze integratievorm over zijn hoogtepunt heen is. In elk geval dient de terugverdientijd een aandachtspunt te zijn bij nieuwe implementaties, evenals de ondersteuning van nieuwe ontwikkelingen zoals ROA/iPaaS.

Links

[1] https://en.wikipedia.org/wiki/Representational_state_transfer

[2] https://en.wikipedia.org/wiki/Operational_data_store

[3] http://www.omg.org/spec/DDS/1.4/PDF/

[4] https://en.wikipedia.org/wiki/Enterprise_service_bus

[5] http://www.referentiearchitectuur.nl/index.php/Thema_Applicatie-integratie

[6] http://www.informatie.nl/Artikelen/2005/november/Eenservice-orientedarchitectureverdienteenbusinesscase.aspx

[7] https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern

[8] https://en.wikipedia.org/wiki/Dereferenceable_Uniform_Resource_Identifier

[9] http://www.sciedupress.com/journal/index.php/bmr/article/view/674/387

[10] http://insights.wired.com/profiles/blogs/why-buses-don-t-fly-in-the-cloud-thoughts-on-esbs

Over ArchiXL

ArchiXL is een onafhankelijk adviesbureau, gespecialiseerd in enterprise- en informatie-architectuur. Wij adviseren organisaties bij het operationaliseren van hun strategie. De naam ArchiXL is een samenvoeging van "Architectuur" en "XL", waarbij XL staat voor “excelleren”. Wij helpen organisaties om hun doelen te bereiken waardoor zij kunnen excelleren. Onderscheidend daarbij is onze pragmatische en doelgerichte werkwijze. Dat zorgt dat we sterk gericht zijn op het leveren van toegevoegde waarde, passend bij de context van de organisatie. Als specialist op het gebied van architectuur kennen we alle relevante methoden en technieken en weten we als geen ander wat de valkuilen zijn. Onze medewerkers onderscheiden zich door hun communicatieve vaardigheden, resultaatgerichtheid, en hun abstractie- en inlevingsvermogen.

Het is onze passie om de doelmatigheid en effectiviteit van veranderingen en de wijze waarop architectuur en kennis daarbij worden toegepast te verbeteren. Wij denken dat mensen en hun kennis daarin een centrale rol spelen. Het is belangrijk om de specifieke kennis, vaardigheden en talenten van mensen te zien en maximaal in te zetten voor de doelstellingen van de organisatie. De basis daarvoor is een goed gesprek en een goed luistervermogen. In onze visie wordt architectuur nog onvoldoende effectief ingezet om de organisatie te ondersteunen. Symptomen hiervan zijn ontoegankelijke architectuurdocumenten, abstracte modellen die niet aansluiten bij de praktijk en architecten die zich afzonderen van de organisatie. Door kennis te mobiliseren zet je anderen in hun kracht en kom je samen tot grote hoogte.

Onze principes

- Standaard methode – onze aanpak is gebaseerd op standaard methoden en technieken zoals ArchiMate en TOGAF, en daarmee op uitgebreide kennis en ervaring van anderen.
- Hergebruik – organisaties lijken in veel opzichten op elkaar en hergebruik van kennis en architecturen is daarom verstandig.
- Iteratief werken – het is belangrijk om snel antwoord te geven op vragen vanuit de organisatie; dit hoeft niet altijd een volledig antwoord te zijn.
- Concrete en bruikbare resultaten – architectuurproducten moeten direct bruikbaar zijn en waarde opleveren voor de organisatie.
- Samenwerking – veranderen doe je samen, daarmee bundel je ook de kennis en denkvermogen en ontstaat draagvlak voor de verandering.
- ”Just enough” architectuur – architectuurdocumenten moeten bijdragen aan de doelstellingen en niet meer beschrijven dan noodzakelijk.
- Mobiliseren kennis – architecten moeten zich vooral richten op het verzamelen, analyseren, genereren en verspreiden van kennis.

Meer weten?

telefoon: 033-2585545

e-mail: info@archixl.nl

website: http://www.archixl.nl