Best Practice integratiepatronen, berichten en protocollen

Inleiding

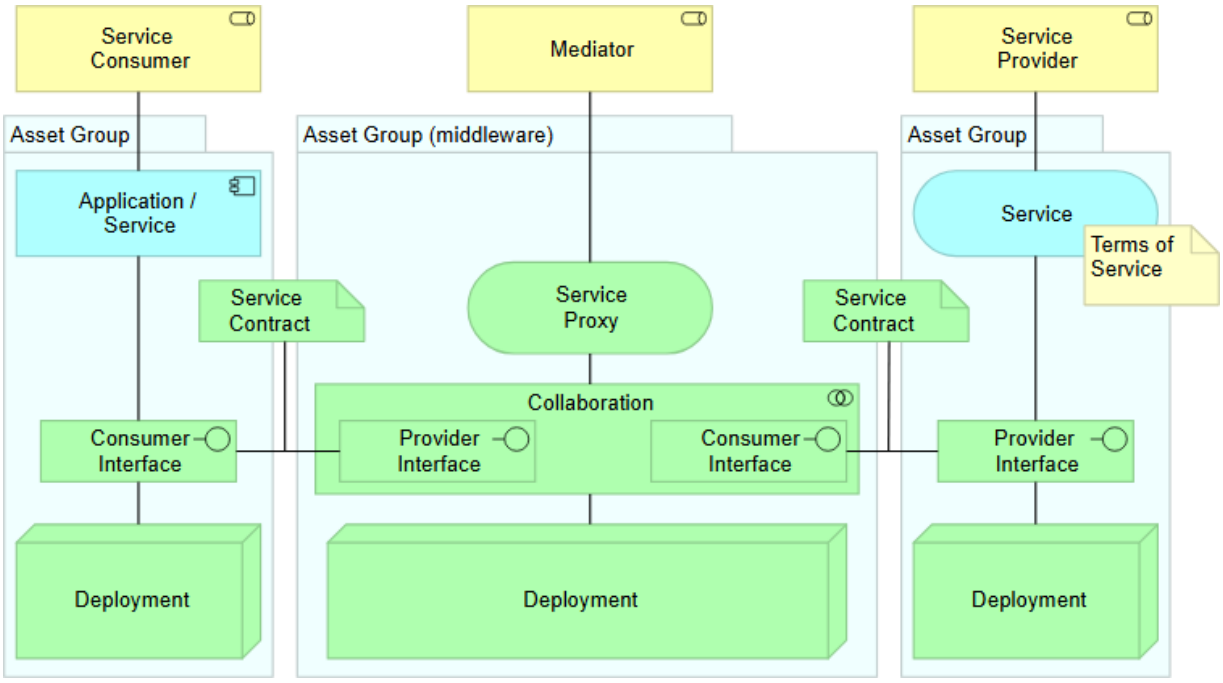
Dit document is tot stand gekomen door praktijkervaringen en best practices. Het is een extractie uit een set van documenten die worden gebruikt in de voortbrenging van integraties. Het document beschrijft 3 basispatronen die vaak in integratie context gebruikt worden. Om discussie te voorkomen tijdens de implementatiefase zijn verschillende zaken belicht die kunnen bijdragen aan het realiseren van zogenaamde robuuste implementaties en dus integraties.

Belangrijk om te weten is dat deze best practices gebaseerd op een referentie architectuur voor integratie, waarin standaard bouwblokken gedefinieerd zijn. De referentie architectuur is per bedrijf verschillend. Dus hoewel in dit document wordt verwezen naar een referentie architectuur, zal elk bedrijf haar eigen architectuur kennen met standaard bouwblokken voor integratie. Dit document beschrijft echter niet de referentie architectuur voor integratie.

Dit document is bedoeld om een discussie op te starten tussen verschillende partijen en hun ervaringen. Een van de voorbeelden het technisch overleg Digikoppeling was bijvoorbeeld dat je REST voor van alles kan inzetten maar is dit wel verstandig. Waarom zou je dat wel of niet willen doen!

SOA Metamodel

De gebruikte terminologie is gebaseerd op het onderstaande Archimate metamodel. Integraties 'leven' volgens dit model in de technologielaag. Bedrijfslogica hoort daar niet thuis en bevinden zich in applicatielaag onder service of applicaties.



Asset: Een asset is een specifiek type object dat een artefact in je SOA-omgeving vertegenwoordigt, zoals een webservice, een REST-service, een implementatie (deployment) of een bouwsteen.

Service Provider (Dienstenleverancier): De dienstenleverancier is de eigenaar van de service en verantwoordelijk voor de correcte uitvoering van zijn service(s).

Service Consumer (Dienstafnemer): De dienstafnemer is de eigenaar van de applicatie of service die een service gebruikt.

Mediator: De mediator is de beheerder van de serviceproxy-oplossing en fungeert altijd als tussenpersoon tussen de service provider en de service consumer.

Asset Group: Een assetgroep is een logische groep van assets, meestal een Configuration Item (CI) met een grote scope of gewoon de applicatie zelf.

Applicatie: Een applicatie is een bouwsteen van het type applicatie en bevat alleen consumenteninterfaces. Het vertegenwoordigt de applicatiecomponent die andere services

gebruikt, maar niet door andere services of applicaties wordt gebruikt. Het vertegenwoordigt ook de implementatie van bedrijfsfunctionaliteit en maakt deel uit van de applicatielaag.

Service: Een service is een bouwsteen van het type service en kan zowel provider- als consumenteninterfaces bevatten. Het vertegenwoordigt de implementatie van bedrijfsfunctionaliteit en biedt deze altijd aan via interfaces. Een service maakt deel uit van de applicatielaag.

Service Proxy: Een serviceproxy is een bouwsteen van het type serviceproxy en bevat altijd minimaal één provider- en consumenteninterface. Het wordt gebruikt om services en hun interfaces te ontkoppelen. Het bevindt zich in de infrastructuurlagen en bevat geen daadwerkelijke bedrijfsfunctionaliteit.

Virtual Service Proxy: Een virtuele serviceproxy is een bouwsteen van het type serviceproxy. Het is een logische bouwsteen zonder implementatie of daadwerkelijke interfaces. Het bevat de direct verbonden interfaces van de service provider en service consumer.

Collaboration (Samenwerking): Een samenwerking is een aggregatie van twee of meer elementen die samenwerken om een bepaald collectief gedrag uit te voeren.

Provider Interface: Een providerinterface is een interface die een service aanbiedt en altijd door een consumenteninterface wordt gebruikt.

Consumer Interface: Een consumenteninterface is een interface die altijd een providerinterface gebruikt.

Terms of Service: De terms of service beschrijft hoe een service gebruikt dient te worden binnen vastgelegde kaders.

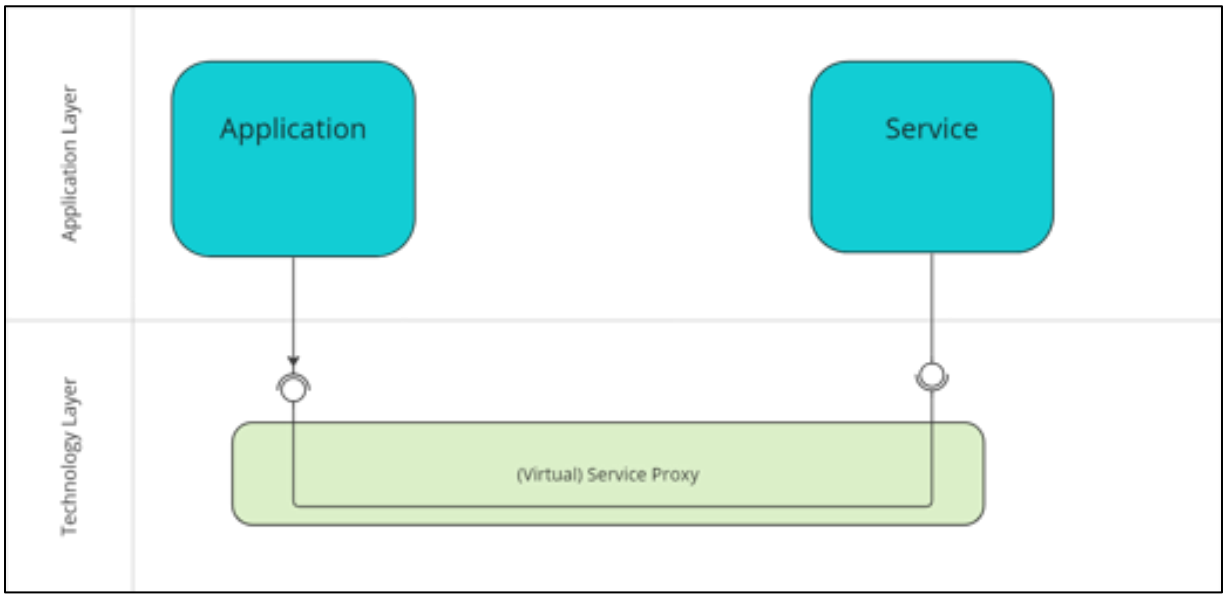
Service Contract: Een servicecontract is een beschrijving van het gebruik dat direct gerelateerd is aan de gebruiksvoorwaarden van een service.

Deployment: Een deployment is de technische identificatie van een interface op een netwerk.

Building Block (Bouwsteen): Een bouwsteen is een abstracte component die wordt gebruikt om blokken met gerelateerde interfaces te creëren. Een bouwsteen is doorgaans een applicatie, service of serviceproxy. Het bevat de relaties tussen interfaces.

Scope

De scope van robuuste implementaties gaat over de koppelvlakken (interfaces), service implementatie van de service provider en service consumers en de service proxies in relatie tot berichtverwerking. Echter beperkt de scope zich wel tot de technologielaag.



Er zijn uiteraard gecombineerde patronen (zie ook bijlage:

Ontwerp patronen) te herkennen: events (claim-check, publish/subscribe, event sourcing etc) en smart routing. Deze vallen allen buiten de scope van deze beschrijving. Dit zijn complexere integratiepatronen gefundeerd op de hier beschreven patronen en zullen apart uitgewerkt worden.

Robuuste implementaties

Een robuuste implementatie kenmerkt zich o.a. door:

- hoge beschikbaarheid
- hoge performance
- schaalbaarheid
- betrouwbaarheid
- onderhoudbaarheid
- beveiligbaarheid

- Beheerbaarheid

Uitgangspunten

- De best practice richt zich op de interface van een service of applicatie (applicatielaag) en de transport van gegevens er tussen (technologielaag). Dit betekent dat de transport is afgehandeld, wanneer er een ontvangstbevestiging is verstuurd door het protocol. Wat er verder gebeurt in de applicatie of service (bijv. een verwerkingsverslag/proces) is in deze best practice niet tot in detail meegenomen. Wel worden er een aantal ontwerpoverwegingen genoemd. Dit valt onder het applicatie- of serviceontwerp.
- De transportlaag mag nooit gebruikt worden als buffer voor het verwerken van berichten. De hoofdfunctie van deze technologische laag is uiteraard transport. Het bufferen van berichten dient te geschieden bij de service provider en/of service consumer (*applicatie- of servicebuffer*).
- Het gebruik van batch is in moderne oplossingen slechts beperkt toegestaan (en meestal niet nodig):

  Batchverwerking wordt toegepast indien dit belangrijke efficiëntie- / kostenvoordelen biedt en als de kans gering is dat een 'foutief geval" in de groep de verwerking van de overige berichten onacceptabel lang ophoudt.

  Dus realtime uitwisselingen gaan voor batching.

- De service provider dient 'Contract First' ontwikkeling te volgen.
- De service consumer moet waar mogelijk gebruik maken van de service provider interface zonder te transformeren.
- Infrastructuur wordt gemonitord: CPU, geheugen, storage, netwerk, enz. worden automatisch gemonitord. Er wordt alarm geslagen als een vooraf bepaalde limiet bereikt wordt, zodat actie ondernomen kan worden.
- Wanneer partijen niet kunnen voldoen aan de requirements voor robuuste implementaties, dan zal de referentie architectuur voor Integratie moeten worden gevolgd.
- Robuuste implementaties worden getest op de veel voorkomende foutsituaties.
- Alle koppelvlakken dienen geregistreerd te worden in een centrale Unified Repository. Hier worden ook de technische service contracten bijgehouden voor optimalisatie van capaciteitsmanagement en (voorspellen) van beschikbaarheid.

- De uitdagingen voor robuuste implementaties zijn gebaseerd op de ISO 25010 - Kwaliteitskenmerken van software en systemen .

De onderstaande kenmerken zijn bedoeld om meetbaar te maken hoe robuust de gekozen oplossing is. Indien men aangeeft dat een kenmerk niet van toepassing is, kan met een 'pas toe of leg uit' onderbouwing worden afgeweken.

De referentie architectuur kent 2 basis patronen voor het transport gehanteerd, te weten: ***raadplegen*** en ***leveren***.

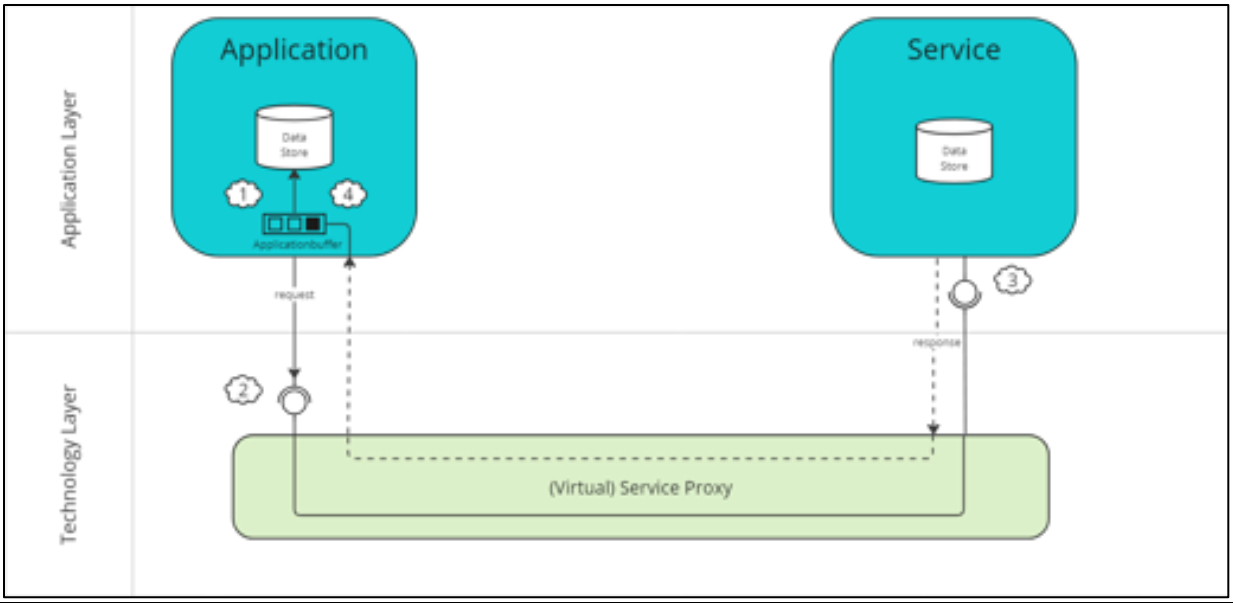
Kenmerken Robuust	Raadplegen	Leveren
hoge beschikbaarheid	De service achter de interface dient 24/7 (99.8%) beschikbaar te zijn. Indien de service onbeschikbaar is, moet de service consumer de correcte afhandeling verzorgen.	De service achter de interface dient 24/7 99.8% van de tijd beschikbaar te zijn.
hoge performance	De service provider mag niet blokkerend werken. Dit betekent dat het antwoord gemiddeld 300 ms mag duren en binnen maximaal 1 seconde geleverd dient te zijn.	De service provider/consumer die de berichten ontvangt, moet binnen 100 ms gepersisteerd te zijn. Dit kan een gebufferde oplossing zijn in de transportlaag. De service provider/consumer zal de berichten in een applicatie- of servicebuffer moeten plaatsen om te persisteren.
schaalbaarheid	De service provider dient ontworpen te zijn voor schaalbaarheid. Echter moet toepassing matchen met de schaling voor zowel *vertical scaling* als *horizontal scaling*. *Design for redundancy, Design for scalability*	De service provider dient ontworpen te zijn voor schaalbaarheid. Echter moet toepassing matchen met de schaling voor zowel *vertical scaling* als *horizontal scaling*. *Design for redundancy, Design for scalability*
betrouwbaarheid	De service provider/consumer dienen een gedegen foutafhandeling te hebben ingericht (design for failure).	De service provider/consumer dienen een gedegen foutafhandeling te hebben ingericht (design for failure).

onderhoudbaarheid	De service provider heeft een duidelijke Life Cycle.	De service provider/consumer hebben een duidelijke life cycle.
beveiligbaarheid	De service provider is voldoende beveiligd op basis van de classificatie van gegevens in relatie tot de Security Richtlijnen.	De service provider is voldoende beveiligd op basis van de classificatie van gegevens in relatie tot de Security Richtlijnen..
beheerbaarheid	De service provider/consumer heeft een duidelijk en efficiënt beheerproces. Er dient een beheerorganisatie te zijn ingericht voor het afhandelen van incidenten op de robuuste componenten.	De service provider/consumer heeft een duidelijk en efficiënt beheerproces. Er dient een beheerorganisatie te zijn ingericht voor het afhandelen van incidenten op de robuuste componenten.

Basispatroon 1: Raadplegen van gegevens

- Beperkt zich tot het raadplegen van gegevens of het verzoeken van functionaliteit (een voorbeeld hiervan is het verzoek voor het uitvoeren van een werkproces);
- Synchroon karakter;
- Bij voorkeur kleine berichten, wordt afgeraden bij hoge load voor langdurige periodes;
- Wel een functionele terugmelding vanuit de bron bij raadplegingen;
- Niet blokkerend, een verzoek leidt tot een direct antwoord zonder te wachten op de uitvoering van een (werk)proces;
- Niet persistent;
- Volgens de RA voor integratie kan dit worden geïmplementeerd met de de protocollen/standaarden SOAP/XML en REST/JSON.

Dit type uitwisseling heeft een synchroon karakter waarbij de ‘consumer' van de service een antwoord verwacht. Is de service niet beschikbaar dan faalt de uitvraag en zal de ‘consumer’ opnieuw moeten proberen op een later moment. Dit leidt veelal niet tot directe problemen, tenzij de onbeschikbaarheid uiteraard te lang duurt. Daarnaast ligt het initiatief bij de 'consumer’.



Happy Flow

Een applicatie gaat *synchroon (binnen 1 sessie)* gegevens uitvragen bij een service. *De nummers in de wolkjes refereren naar het Transport Model.*

1 De applicatie maakt een bericht aan met parameters voor het ophalen van gegevens.

2 De applicatie doet een verzoek aan de (virtuele) service proxy door de interface aan te spreken.

3 De service ontvangt het verzoek (met eventueel parameters) en levert een antwoord.

4 De applicatie ontvangt het antwoord.

Design Considerations

1 Is het nodig om te bufferen voor uitsturen? Afweging heeft vooral te maken met of de applicatie verder gaat met andere interne processen.

2 Indien de interface niet beschikbaar is, moet de applicatie een goede foutafhandeling kennen. Door hoge mate van koppeling is deze oplossing minder robuust en kunnen ketens direct hinder ondervinden van onbeschikbaarheid van een andere service(s). Identity en Access Management moet worden ingericht voor de service proxy.
Belangrijk: een service proxy biedt extra transportfunctionaliteit die niet thuishoren bij een service. Indien er gebruik gemaakt wordt van een virtuele service proxy, ben je als service provider zelf verantwoordelijk voor de integratie functionaliteit. Een virtuele service proxy is een directe koppeling tussen service consumer en provider zonder middelware.

3 De service moet de aanvragen snel kunnen verwerken of snel een bruikbare foutmelding genereren. Schaalbaarheid is een belangrijk aspect om de beschikbaarheid en snelheid te verhogen. Echter moet het achterland waarin de service opereert dit wel aankunnen. Identity en Access Management moet worden ingericht voor de service.

4 Het ontvangen kan indien nodig worden gepersisteerd, wanneer het proces het antwoord niet direct kan verwerken.

Exceptions

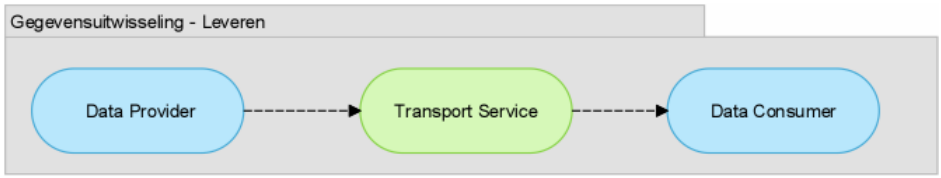
1 Kunnen gegevens worden verzameld en (eventueel) geplaatst in een buffer? Fouten die in deze stap ontstaan vallen onder applicatiefouten.

2 Wat te doen als de interface niet beschikbaar is in het geval dat het netwerk voor langere tijd down is, of de service voor langere tijd onbeschikbaar is.

3 Het bericht is niet valide en de service dient een technische foutmelding aan de consumer te geven. De Service kan de berichten niet snel genoeg verwerken, waardoor het antwoord te laat komt. De service kan hierdoor ook onbeschikbaar worden.

4 Foutscenario, bericht inhoud niet valide waardoor er functionele/technische uitval plaatsvindt. De consumer moet de uitval afhandelen middels een incidentenbeheerproces.

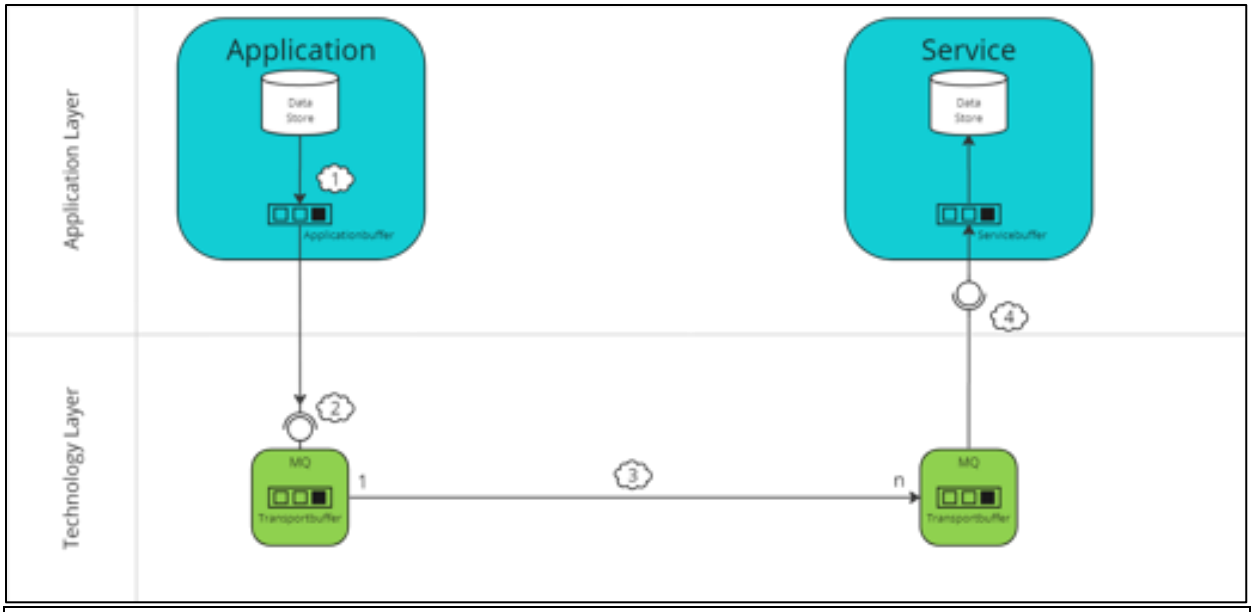
Basispatroon 2: Leveren van gegevens*



*nota bene: zowel service provider als service consumer kunnen data provider zijn.

- Beperkt zich tot mutaties, meldingen en signalen die leiden tot een verwerking van de gegevens;
- Asynchroon karakter (fire-and-forget);
- Geen functionele terugmelding (wel functionele verwerkingsterugmelding via een nieuwe service proxy);
- Ontkoppelt in technische zin de service provider en de service consumer;
- Onbeschikbaarheid van de ontvanger heeft geen direct impact op de verzender;
- Volgorde is veelal niet gegarandeerd, hiervoor is altijd een extra mechanisme noodzakelijk;

Dit type uitwisseling heeft een asynchroon karakter (fire and forget) waarbij de data provider geen functioneel antwoord verwacht. Maar de mutatie moet wel gegarandeerd worden geleverd aan de data consumer. Dit kan bijvoorbeeld met de technologie IBM MQ worden ingericht MQ Richtlijnen voor gegarandeerd transport. Het grote voordeel van een dergelijke implementatie is dat al aan een groot deel van de eisen voor robuustheid op transportniveau wordt voldaan. Het kan overigens nog steeds voorkomen dat de geleverde mutatie fout is. Hiervoor zal de data consumer een functionele melding kenbaar moeten maken aan de data provider. Dit kan een beheermelding zijn vanuit het incidentenproces, maar dit kan ook een geautomatiseerde nieuwe functionele uitwisseling zijn (melding).



Happy Flow

Een applicatie gaat *a-synchroon* gegevens leveren aan een service. *De nummers in de wolkjes refereren naar het Transport Model.*

1 De applicatie verzamelt de benodigde gegevens voordat deze worden gepersisteerd op de applicatiebuffer.

2 De gegevens worden uit de applicatiebuffer in de transportbuffer geplaatst.

3 De service ontvangt de gegevens in de transportbuffer aan service zijde.

4 De service verwerkt de ontvangen gegevens uit de transportbuffer naar de eigen persistente servicebuffer.

5 De service voert de bedrijfsregels in het werkproces uit.

Optioneel: Eventueel kan de applicatie a-synchroon een functionele (event) melding terug sturen.

Design Considerations

1 Is het nodig om te bufferen voor uitsturen of is een commando (voor bericht opbouw) voor het ophalen van data uit de data store voldoende? Afweging heeft vooral te maken met actualiteit/dynamiek van de gegevens. Moet een bericht bijvoorbeeld herverzonden kunnen worden met de toen geldende actualiteit?

2 Voor Retry mechanisme is belangrijk te weten hoe vaak, tijd tussen retries en uitvalmechanisme.

3 Het transport is zo ingericht om onbeschikbaarheid van de service af te vangen, de service haalt het bericht op als het zo ver is. Dit zou volgens de richtlijnen een only once inrichting kunnen zijn. Deze inrichting zorgt er ook voor dat de service schaalbaar kan zijn.

Exceptions

1 Kunnen gegevens worden verzameld en (eventueel) geplaatst in een buffer? Fouten die in deze stap ontstaan vallen onder applicatiefouten.

2 Kan je het bericht op de transportbuffer plaatsen? Wat doe je als het mis gaat? Fouten die in deze stap ontstaan vallen onder applicatiefouten.

3 Wat als het netwerk voor langere tijd down is, of de service voor langere tijd onbeschikbaar is. Uiteindelijk uit zich dat in Punt 2! Als de transportbuffer vol is, kan in stap 2 kan niet meer worden geplaatst.

4 Het bericht is niet valide en dient uit te vallen in een uitvalbuffer voor afhandelen van incidenten door beheer (Technische Context). De service kan de berichten niet snel genoeg persisteren. De transportbuffer aan de service zijde loopt vol en daarmee dus ook transportbuffer aan de applicatie zijde. Uiteindelijk uit zich dat in Punt 2.

5 Foutscenario, bericht inhoud niet valide waardoor er functionele uitval plaatsvindt. Dit soort fouten kunnen lastig zijn om op te lossen omdat het een werkproces met bedrijfsregels betreft (Functionele Context).

Met de bovenstaande basispatronen kunnen complexere patronen worden gemaakt (zie ook bijlage: Ontwerppatronen voor gegevensuitwisseling)

De bovenstaande patronen verschillen wezenlijk van elkaar, waar men bij het eerste patroon wacht op antwoord (synchrone afhandeling, zowel technisch als functioneel) is dat bij het

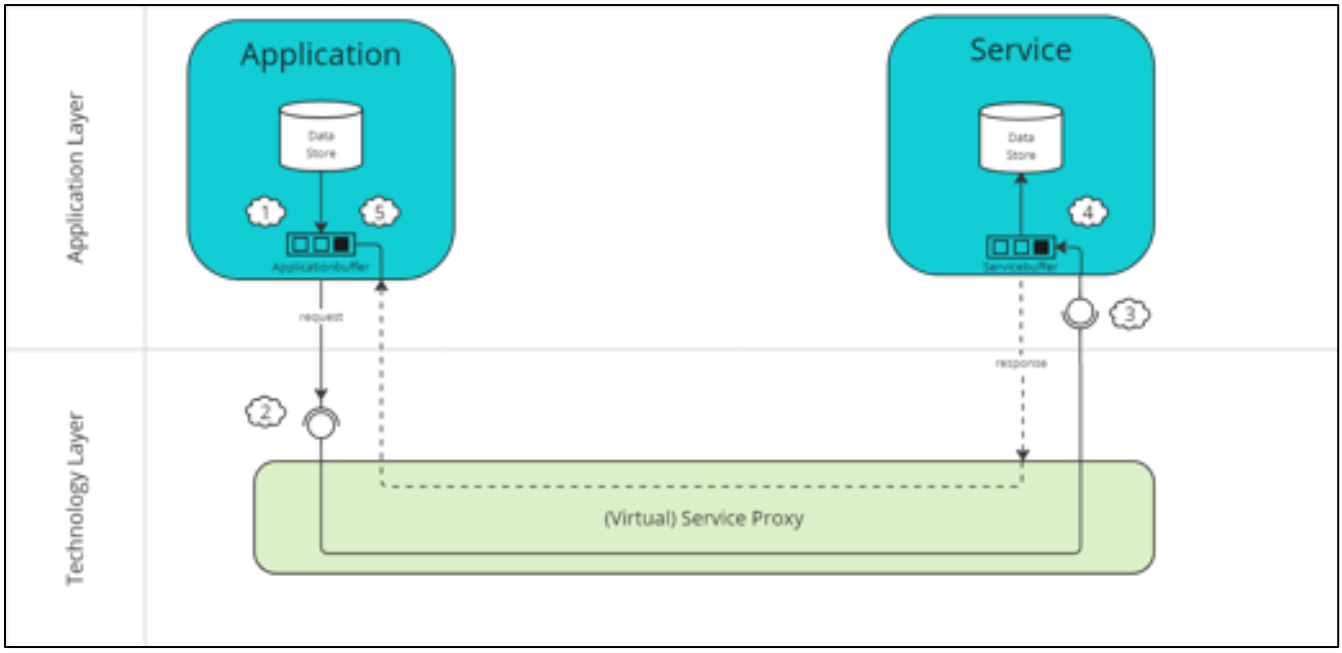
tweede patroon (a-synchrone afhandeling, alleen technisch) niet het geval. Indien nodig zal bij het tweede patroon de functionele afhandeling via een ander kanaal worden afgehandeld. Dit is bij een robuuste implementatie veelal niet nodig.

Hoewel de twee beschreven patronen de basis vormen voor de integratiepatronen, is er nog een derde patroon, dat een combinatie is van beide.

Het is natuurlijk mogelijk om met een synchrone call een mutatie te doen. Deze wijze van integreren wordt vaak afgeraden, omdat extra maatwerk en afhandeling nodig is om een robuuste koppeling over het netwerk te maken. Wanneer aan strikte eisen kan worden voldaan, kan deze oplossing worden ingezet. De huidige cloudoplossingen kunnen dit vaak makkelijk ondersteunen. Voor pakketten en custom solutions is dit meestal niet standaard.

Basispatroon 3: Synchroon leveren van gegevens

- Beperkt zich tot mutaties, meldingen en signalen die leiden tot een verwerking van de gegevens;
- Synchroon karakter;
- Wordt afgeraden bij hoge load, veel berichten over een langere periode;
- Functionele terugmelding mogelijk, indien binnen de daarvoor gestelde grenzen kan worden voldoen;
- Geen ontkoppeling in technische zin tussen de service provider en de service consumer; Service moet beschikbaar zijn voor directe afhandeling;
- Extra afhandeling nodig ten opzichte van het basispatroon 2 (a-synchrone model).



Happy Flow

Een applicatie gaat *synchroon* gegevens leveren aan een service. *De nummers in de wolkjes refereren naar het Transport Model.*

1 De applicatie verzamelt de benodigde gegevens voordat deze worden geplaatst op de applicatiebuffer.

2 De gegevens worden uit de applicatiebuffer aangeboden aan een interface voor transport.

3 De service ontvangt de gegevens.

4 De service persisteert de ontvangen gegevens naar de servicebuffer en stuurt een ontvangstbevestiging.

5 De service voert de bedrijfsregels in het werkproces uit.

NB: Eventueel kan de service synchroon een functionele/technische melding terug sturen indien dit binnen de gestelde verwerkingstijd valt.

Design Considerations

1 Na het verzamelen van de gegevens in een bericht, is het belangrijk deze te persisteren alvorens het aan te bieden voor transport. Soms is de actualiteit van de data minder relevant en is persistentie wellicht overbodig.

2 Wanneer de interface van de service (proxy) niet beschikbaar is voor het aanleveren van het bericht, zal er een Retry mechanisme moeten worden geïmplementeerd. Denk hierbij ook aan hoe vaak, tijd tussen retries en uitvalmechanisme. Voor dit punt is een buffer gewenst (zie punt 1).

3 De proxy zal niet bufferen en functioneert als een doorgeefluik. Er vindt geen persistentie plaats. Hierdoor zal punt 2 ook hier gelden.

Exceptions

(1) Kunnen gegevens worden verzameld en (eventueel) geplaatst in een buffer? Fouten die in deze stap ontstaan vallen onder applicatiefouten.

(2) Kan het bericht aangeboden worden aan de interface ? Wat doe je als het mis gaat? Fouten in deze stap moeten worden afgehandeld door de applicatie. Wat als het netwerk voor langere down is, of de service voor langere tijd onbeschikbaar.

(3) Het bericht is niet valide en dient uit te vallen voor afhandeling van incidenten door beheer. De service kan de berichten niet snel genoeg persisteren.

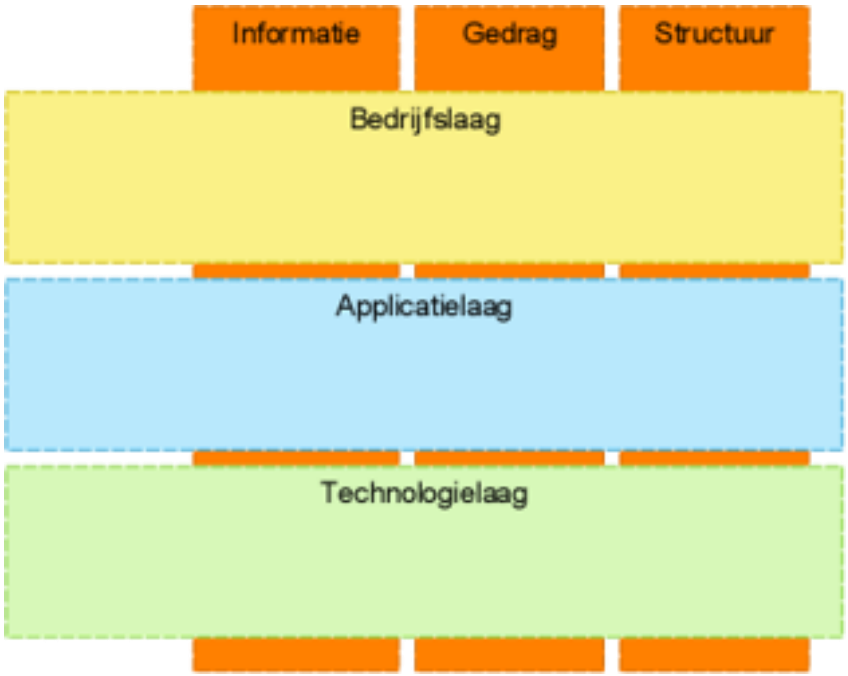
(4) Foutscenario, bericht inhoud niet valide waardoor er functionele uitval plaatsvindt. Dit soort fouten kunnen lastig zijn om op te lossen omdat het een werkproces met bedrijfsregels betreft (Functionele Context).

Bijlage 1. Ontwerppatronen voor gegevensuitwisseling

Ontwerppatronen voor gegevensuitwisseling beschrijven de Enterprise integratiepatronen voor gegevensuitwisselingen met een functionele focus en bevinden zich op het snijvlak tussen applicaties en infrastructuur. Een patroon is een beschrijving van een oplossing van een vaker voorkomend probleem binnen een specifieke context. Deze ontwerppatronen beschrijven vooral de wat-context van een probleem en de passende oplossing. Het beschrijft niet de hoe-context.

Door deze ontwerppatronen toe te passen tussen applicaties, kan in het applicatiedomein de gegevensuitwisseling sterk vereenvoudigd worden. Deze relatief simpele basispatronen helpen daarnaast de verschillende disciplines, met verschillende achtergronden zonder te veel inhoudelijke kennis, om vanuit de logica van de ontwerppatronen te redeneren over de wat-vraag. Hieruit zullen dan ook de requirements voor de specifieke oplossingen moeten komen.

Hier beschrijven we de ontwerppatronen voor de gegevensuitwisseling, maar op alle lagen zijn dergelijke ontwerppatronen te herkennen.



Figuur: Positionering ontwerppatronen gegevensuitwisseling binnen Archimate.

Ontwerp patronen

De hierna beschreven ontwerppatronen zijn een vertaling van patronen afkomstig van verschillende bronnen, waaronder:

- Enterprise Integration Patterns
- Microsoft Azure Patterns

De ontwerppatronen hebben een hoger abstractieniveau en zijn geen één-op-één vertaling, maar toegespitst op de inrichtingsprincipes van de referentie architectuur. De ontwerppatronen kunnen in sommige situaties gecombineerd worden om te komen tot het gewenste concept.

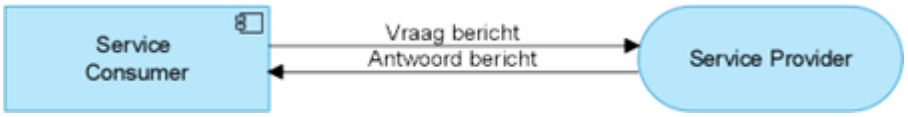
Messaging

Patroon	Messaging
Context	De context waarin dit patroon wordt geschetst is een gedistribueerde omgeving bestaande uit meerdere applicaties die berichten met elkaar uitwisselen met als doel gegevens te delen. Veelal zijn de applicaties in verschillende technologieën gebouwd en op verschillende platformen geplaatst.
Probleemstelling	Hoe kan een applicatie berichten met data versturen naar andere applicaties?
Conceptuele oplossing	De service consumer plaatst een bericht in een gepersisteerde wachtrij (queue) die tevens de berichten opslaat. Het bericht blijft hierdoor beschikbaar totdat de service provider beschikbaar is en het bericht kan ophalen of ontvangen.  De service provider is via het messaging system verbonden met de service consumer. Dit kan op vele verschillende wijze worden

	gerealiseerd, bijvoorbeeld met een point-to-point channel, message bus of message broker.
Eigenschappen	• De service consumer heeft kennis van de service provider. • De service provider heeft kennis van de service consumer. • De applicaties hoeven niet dezelfde technologie te hebben. • De applicaties hoeven niet op hetzelfde platform te staan. • Het bericht is gepersisteerd.
Implicaties	• Deze oplossing is een directe functionele koppeling tussen service provider en service consumer. • Verhoogde ontkoppeling tussen service provider en service consumer. Beide hoeven niet te allen tijde beschikbaar te zijn. • Door hoge mate van ontkoppeling is deze oplossing zeer robuust en ondervinden ketens niet direct hinder van onbeschikbaarheid van een andere applicatie(s). • Het gebruikte applicatiecomponent moet een retry-mechanisme ondersteunen. • Het asynchroon communiceren bemoeilijkt het correleren van de vraag- en antwoordberichten, omdat deze niet in dezelfde sessie worden verwerkt.
Aandachtspunten	• Hoeveelheid te versturen berichten. • Grootte van berichten. • Rationale: De wijze van versturen is het efficiëntst voor messaging wanneer berichten frequent worden verstuurd en niet in een bulk in een korte periode.

Request/Response (synchrone vraag/antwoord)

Patroon	Request/Response (synchrone vraag/antwoord)
Context	Een service consumer is een applicatie of service die een service gebruikt die beschikbaar wordt gesteld door een service provider.

	 De service provider levert in dit patroon een service interface die een synchrone communicatie over het netwerk ondersteunt.
Eigenschappen	• De service consumer heeft kennis van de service provider. • De service provider heeft geen 'kennis' van de service consumer. • De service provider en een service consumer kennen een eenduidig onderling contract naast de interface. • De applicaties hoeven niet dezelfde technologie te hebben. • De applicaties hoeven niet op hetzelfde platform te staan.
Implicaties	• Dit patroon leidt tot een directe koppeling tussen service consumer en service provider. • Het synchroon communiceren vereenvoudigt het correleren van de vraag- en antwoordberichten, omdat deze in dezelfde sessie worden verwerkt. • Verhoogde koppeling tussen Consumer en Provider. Beide moeten beschikbaar zijn om te kunnen functioneren. • Door hoge mate van koppeling is deze oplossing minder robuust en kunnen ketens direct hinder ondervinden van onbeschikbaarheid van een andere applicatie(s). • Dit patroon kan ingezet worden voor zowel het delen van een kleine set gegevens als het gebruiken van functionaliteit van een applicatie.
Aandachtspunten	• Bij veel synchrone communicatie kan de service van een service provider onbeschikbaar raken. Dit kan doordat Consumers de service te vaak aanroepen. In praktijk zal hiervoor een beschermingsmechanisme moeten worden geïntroduceerd. Een API Gateway kan hier een uitkomst bieden. • Bij synchrone communicatie waarbij veel gegevens worden opgehaald, kan het leiden tot 'blokken' als de service van de service provider te veel concurrent verbindingen kent.
Opmerkingen	Veelal worden synchrone services middels de webservice technologie uitgewerkt.

One-way (Levering)

Patroon	One-way (asynchrone Levering)
Context	Een service consumer is een applicatie of service die een service gebruikt die beschikbaar wordt gesteld door een service provider.  De service provider levert in dit patroon een service interface die een asynchrone communicatie over een netwerk ondersteunt.
Eigenschappen	• De service consumer heeft kennis van de service provider. • De service provider heeft in de regel geen kennis van de service consumer. • De applicaties hoeven niet dezelfde technologie te hebben. • De applicaties hoeven niet op hetzelfde platform te staan. • De communicatie verloopt asynchroon. • Het bericht wordt gepersisteerd.
Implicaties	• Deze oplossing is een indirecte technische koppeling tussen service consumer en service provider. • Verhoogde ontkoppeling tussen verzender en ontvanger. Beide hoeven niet beschikbaar te zijn. • Door hoge mate van ontkoppeling is deze oplossing zeer robuust en zullen ketens niet direct hinder ondervinden van onbeschikbaarheid van een andere applicatie(s). • Deze oplossing wordt vooral ingezet wanneer gegevens worden gemuteerd. • Het gebruikte communicatieprotocol moet een retry-mechanisme ondersteunen.
Aandachtspunten	• Grootte van berichten. • Wijze van versturen, het efficiëntst voor messaging is het zeer frequent versturen van berichten en niet in een bulk in een korte periode.

	• Wanneer berichten via twee one-way verbindingen worden verstuurd (asynchrone request/response) zullen de berichten expliciet moeten worden gecorreleerd.
Opmerkingen	Een levering zou ook synchroon kunnen maar dan vervallen alle messaging eigenschappen, ergo persistentie.

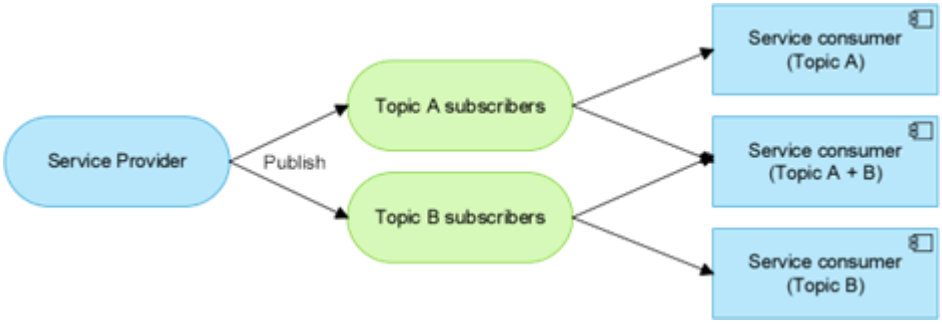
Robuuste aflevering van berichten

Patroon	Robuuste aflevering van berichten
Context	De context waarin dit patroon wordt geschetst is een gedistribueerde omgeving bestaande uit meerdere applicaties die berichten met elkaar uitwisselen met als doel gegevens te delen. Mogelijk zijn applicaties niet altijd beschikbaar.
Probleemstelling	Hoe kan een applicatie berichten met data versturen naar andere applicaties met een gegarandeerde aflevering?
Conceptuele oplossing	De verzendende partij plaatst een bericht in een zogenaamde wachtrij (queue) die tevens de berichten opslaat. Wanneer de ontvanger beschikbaar is, kan het de berichten op halen. 
Eigenschappen	• De applicaties hoeven niet op hetzelfde platform te staan. • De berichten worden gegarandeerd afgeleverd. • De berichten worden tijdelijk opgeslagen. • De uitwisseling van gegevens heeft een a-synchroon karakter.

Implicaties	• Deze oplossing is een directe koppeling tussen ontvanger en verzender. • Verhoogde ontkoppeling tussen verzender en ontvanger. Beide hoeven niet te allen tijde beschikbaar te zijn. • Door hoge mate van ontkoppeling is deze oplossing zeer robuust ondervinden ketens niet direct hinder van onbeschikbaarheid van een andere applicatie(s).
Aandachtspunten	Gegarandeerde aflevering is alleen mogelijk wanneer het bericht is vastgelegd in de wachtrij. Wanneer berichten niet kunnen worden vastgelegd, zal de verzender het bericht later nogmaals moeten aanbieden. De verantwoordelijkheid voor deze foutafhandeling vervalt pas wanneer het bericht daadwerkelijk in de wachtrij is geplaatst. Wanneer berichten zijn afgeleverd bij de ontvanger, kan er nog steeds uitval plaatsvinden wanneer berichten door de ontvanger worden ingelezen vanuit de wachtrij. In het geval dat dit bericht niet kan worden verwerkt, is de ontvanger verantwoordelijk voor de juiste foutafhandeling. Gegarandeerde aflevering heeft betrekking op het transport van de berichtuitwisseling. Alle tussenstations waar een bericht langs kan komen, vallen niet binnen de context van gegarandeerde aflevering.
Opmerkingen	Message Queue producten ondersteunen standaard out-of-the-box transacties, gegarandeerde aflevering en asynchrone communicatie. Uiteraard kunnen de eigenschappen van message queueing met webservices ook bereikt worden. Het protocol ebMS is hiervan een voorbeeld, maar voor intern gebruik een protocol met veel overhead. Bij implementatie op een HTTP webservice dient men deze functionaliteit zelf toegevoegd te worden. In een grote organisaties met veel leveranciers (en maatwerk), kan dit leiden tot een zeer diffuse uitwerkingen met betrekking tot de extra transactiefunctionaliteit. Voor leveringen is in de doelarchitectuur MQ aangewezen als bewezen standaard. Vaste MQ Richtlijnen beschrijven hoe deze technologie Enterprise breed geïmplementeerd moeten worden om berichten middels transacties uit

	te wisselen. Dit zorgt ervoor dat de implementatie verschillen per partij, voldoen aan een gedragen standaard en daarmee een zekere voorspelbare kwaliteit kennen.

Publish/subscribe (publiceren/abonneren)

Patroon	Publish/subscribe (publiceren/abonneren)
Context	De context voor dit patroon is een omgeving bestaande uit meerdere applicaties, die berichten met elkaar uitwisselen. Sommige applicaties versturen verschillende berichttypes en sommige applicaties zijn geïnteresseerd in één of meerdere berichttypes. 
Probleemstelling	Hoe kan een applicatie berichten versturen naar geïnteresseerd ontvangers zonder te weten welke afnemers het betreft? *Bijvoorbeeld: een mutatie heeft plaatsgevonden en de geïnteresseerde afnemers moeten op de hoogte gesteld worden.*
Conceptuele oplossing	Breidt de communicatie infrastructuur uit met een Publish/subscribe Service waar applicaties zich op berichttypen kunnen abonneren. De zogenaamde verzender publiceert een bericht aan de service (message broker of event bus), die de berichten op zijn beurt op basis van afgesloten abonnementen berichten verstuurt naar de ontvangers. Abonnementen vinden plaats op onderwerp, inhoud of berichttype.
Eigenschappen	• De verzender hoeft geen kennis te hebben van de afnemers. • De ontvanger hoeft geen kennis te hebben van de zender, maar alleen van de gebeurtenis waar het in geïnteresseerd is. • Dit patroon heeft een asynchroon karakter. • Is een vorm van Messaging en de bijbehorende eigenschappen.

Implicaties	• Deze oplossing vereist een tussenpartij die de berichten ontvangt en verspreid naar de geïnteresseerde ontvangers. • Verhoogde ontkoppeling tussen verzender en ontvanger. Beide hoeven niet te allen tijde beschikbaar te zijn of kennis te hebben van elkaars bestaan. • Door hoge mate van ontkoppeling is deze oplossing zeer schaalbaar. • In dit patroon heeft men minder controle over de volgorde van berichten. • Er is een zekere minimale performance penalty wegens het abonnementen distributie.
Aandachtspunten	• Berichten mogen niet verloren gaan en alle abonnees dienen te allen tijde hun berichten te krijgen. • Bij dit patroon is het van belang dat er een goede en eenvoudige inzage is van wat er is verstuurd en verstuurt moet worden om problemen snel te kunnen opsporen. • Volgorde van berichten. • De ontvanger bepaalt hoe snel de berichten worden verwerkt. • Denk goed na over het type en granualariteit van de notificatie, omdat het aantal berichten snel kan groeien.

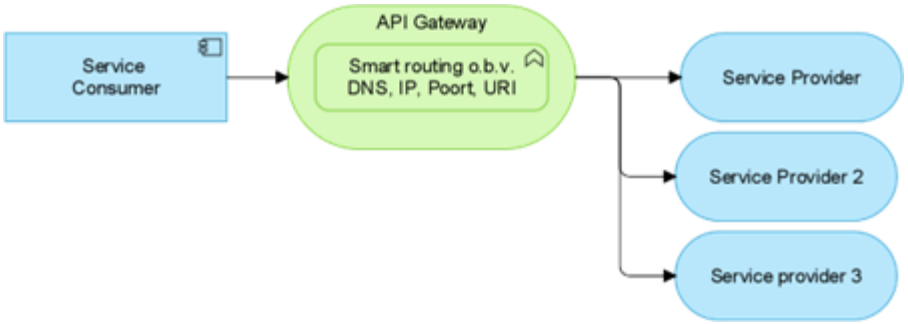
Claim check

Patroon	Claim check
Context	De context voor dit patroon is een gedistribueerde omgeving bestaande uit meerdere applicaties die bestanden met elkaar uitwisselen. De bestanden zijn veelal tussen enkele MB's tot enkele GB's groot.
Probleemstelling	Hoe kan het datavolume van gegevensuitwisselingen geminimaliseerd worden zonder informatie te verliezen?

Oplossing	Het grootste gedeelte van het bericht wordt opgeslagen en vervangen door een zogenaamde 'claim check' ofwel een unieke identificatie. Het overgebleven (meta)bericht wordt verstuurd naar de ontvanger van het gehele bericht. Deze kan middels de claim check het resterende bestand ophalen. 
Eigenschappen	• Het voorkomt het onnodig opslaan en rondsturen van grote berichten. • De ontvanger kan het bestand ophalen wanneer het mogelijk is. • In combinatie met het one-way pattern is deze oplossing in hoge mate ontkoppeld. • In combinatie met het publish/subscribe pattern kunnen meerdere afnemers efficiënt met elkaar communiceren.
Implicaties	• Niet elk tussenstation in de keten hoeft maateregelen te treffen als limieten worden bereikt door toedoen van berichtgrootte. • Het verwerken van grote berichten wordt aanzienlijk vereenvoudigt en sneller.

Smart Routing

Patroon	Smart Routing
Context	Een service consumer gebruikt meerdere services in een gedistribueerde omgeving.
Probleemstelling	De service provider dient voor elke service een endpoint te creëren en te managen. Wanneer er veranderingen optreden op de verschillende

	services zal dus ook de service consumer mee moeten. Is dit echt altijd noodzakelijk?
Oplossing	Zet tussen de service consumer en de service provider een 'gateway' die op basis van bijvoorbeeld URI, DNS, IP, port, header, payload, of beveiligingsrestrictie routeert. Vereenvoudigt de consumer interface zoveel mogelijk en laat de gateway routeren naar de juiste voorziening. 
Eigenschappen	• De consumer heeft slechts 1 aanspreekpunt maar kan meerdere services gebruiken. • De consumer interface is veelal vele malen eenvoudiger dan de werkelijk service-interface. • Bij veranderingen in het achterland hoeven de consumers niet per se mee te veranderen. • De applicaties hoeven niet dezelfde technologie te hebben. • De applicaties hoeven niet op hetzelfde platform te staan.
Implicaties	• Dit patroon kan schaalbaarheid enorm verhogen als dat nodig is. • Dit patroon kan de deployment van services aanzienlijk flexibelere maken, maar ook complexer. Denk bijvoorbeeld aan 'canary releasing' en continuous integration/deployment.
Aandachtspunten	• Het aantal services achter de gateway moet niet te groot worden, het kiezen van de juiste granulariteit is essentieel. • Routing moet simpel en doeltreffend zijn, er mag geen performance degradatie ontstaan voor de gehele voorziening en er mogen geen verborgen bedrijfsregels ontstaan.
Opmerking	• Welke functionaliteit geboden wordt ligt vast in de Richtlijnen van de Policy Service in de referentie architectuur.