edustandaard

Edukoppeling

Asynchrone M2M gegevensuitwisseling binnen het onderwijs

Werkdocument Asynchrone communicatie via RESTful API's

Edustandaard
Datum: 11 februari 2026
Status: concept

Inhoudsopgave

1. Inleiding 4
   1.1. Doel en doelgroep 4
   1.2. Doel van Edukoppeling 4
   1.3. Positionering van Edukoppeling in het Edustandaard vijflagen model 4
   1.4. Uitgangspunten voor de Edukoppeling-standaard 5
   1.5. Aanleiding 6
      1.5.1. Ontwikkelingen 6
      1.5.2. Advies 6
2. Profiel 8
   2.1. Edukoppeling-aanbod: Notificaties / Signalen 8
   2.2. CloudEvents 8
      2.2.1. Rollen 9
      2.2.2. Interacties 9
   2.3. Event levering semantiek 10
      2.3.1. Aflevergaranties 10
      2.3.2. Consumer garanties 11
      2.3.3. End-to-end exactly-once verwerking in een EDA 11
      2.3.4. Trade-offs 12
   2.4. Uitgangspunten voor het profiel 12
   2.5. Registratie 13
   2.6. Producer 13
   2.7. Intermediary 14
   2.8. Consumer 14
   2.9. Technische contracten 14
      2.9.1. Contract register 15
   2.10. CloudEvent beveiligingsopties 16
3. Bijlage A: Uitdagingen in een gedistribueerd landschap 17
   3.1. Traditionele uitdagingen in een gedistribueerd IT-landschap 17
   3.2. Traditioneel gebruikte patronen in een gedistribueerd IT-landschap 17
      3.2.1. Patroon: Two-phase commit (2PC) 17
      3.2.2. Patroon: Saga-patrron 18
      3.2.3. Patroon: Transactioneel outbox-patroon 18
4. Bijlage B: Richtlijnen voor idempotentie 19
   4.1. Wanneer implementeren 19
   4.2. Algemene richtlijnen 19

4.2.1. Response 19
4.3. Technische implementatierichtlijnen: Idempotentie 20
4.3.1. Idempotentieconventies 20
4.3.2. Uniekheid idempotentie sleutel 20
4.3.3. Geldigheid en verloop van idempotentie sleutel 20
4.3.4. Idempotentie fingerprint 20
4.3.5. Verantwoordelijkheden client 21
4.3.6. Verantwoordelijkheden resource 21
4.3.7. Foutafhandeling 21
4.3.8. Client side implementatie (voorbeeld) 21
4.3.9. Resource implementatie (voorbeeld) 22
5. Bijlage C: Begrippen 23
6. Bijlage D: Referenties 25

1. Inleiding

1.1. Doel en doelgroep

Dit werkdocument ondersteunt de ontwikkeling van een nieuwe versie van Edukoppeling. Het bevat voorschriften om asynchrone M2M communicatie via RESTful API's (hierna Edukoppeling-profiel) te bewerkstelligen. Dit document is bedoeld voor de leden van de Edukoppeling werkgroep. Hen wordt gevraagd om dit document te reviewen en aan te passen waar nodig.

1.2. Doel van Edukoppeling

Edukoppeling schrijft voor hoe onderwijsorganisaties, publieke uitvoeringsorganisaties, leveranciers en andere ketenpartners gegevensuitwisselingen opzetten. De standaard gaat over de afhandeling van berichten (het transport) en niet over de inhoud van berichten. Het is een functioneel technische standaard, maar zal ook aansluiting moeten vinden op kaders van andere architectuurlagen. Hoe Edukoppeling aansluit op bredere afspraken is aan ketensamenwerkingen[1] waarin de standaard als onderdeel van de afspraak of het afsprakenstelsel wordt gevat.

Edukoppeling regelt de volgende ketenfuncties: identificatie, authenticatie, autorisatie en routering, op de 'uitwisselingslaag'. Dit om de zorgen dat vertrouwelijke gegevens tijdens transport van de ene naar de andere organisatie, niet ongeoorloofd worden ingezien of gemanipuleerd.

Edukoppeling heeft als scope alle werkingsgebieden vallend onder alle onderwijssectoren en moet hiermee ook tegemoetkomen aan de diversiteit in processen en technische inrichting van deze onderwijssectoren en ketens. Er wordt wel gestreefd naar uniformiteit, omdat er tussen ketens vanuit diverse werkingsgebieden steeds meer sprake kan zijn van gegevensuitwisseling, maar er moet ook voldoende flexibiliteit geboden worden om daar waar nodig andere keuzes te kunnen maken.

1.3. Positionering van Edukoppeling in het Edustandaard vijflagen model

Het Edustandaard 5-lagen model[2] onderkent de volgende lagen:

- grondslagenlaag: borgt de juridische basis en beleidskaders waarbinnen gegevensuitwisseling is toegestaan;
- organisatorische laag: ketensamenwerking afspraken over wie welke rol heeft, welke gegevensdiensten, interfaces en interactiepatronen er zijn en welke gegevens onder welke condities uitgewisseld worden;
- informatielaag: semantiek, waaronder gegevensdefinities, informatiemodellen en de gebruikte identifiers voor rechtspersonen en natuurlijke personen;

[1] Ketensamenwerkingen zijn bijvoorbeeld OKE, Edu-V, ROD ec. (zie ook: https://rosa-begrippenkader.wikixl.nl/index.php/Begrip:27a6accf-472d-4415-bc5b-1e9de17bf288#tab=Betekenis)
[2] AMIGO-methodiek-1.1.0-1.pdf en https://rosa.wikixl.nl/index.php/Interoperabiliteit_en_het_Edustandaard_lagenmodel#Opbouw_van_het_lagenmodel

- applicatielaag: API's en hun beveiligingsprofielen, berichtspecificaties, payload beveiliging, interactiepatronen en foutafhandeling;
- IT-infrastructuurlaag: transportprotocollen en technische beveiligingsmechanismen zoals TLS.

Het Edukoppeling-profiel heeft binnen het Edustandaard 5 lagen model met name betrekking op de applicatielaag, levert daarmee een belangrijke ondersteuning aan met name laag 2 en heeft een relatie met de IT-infrastructuurlaag.

1.4. Uitgangspunten voor de Edukoppeling-standaard

1. Het organisatorisch werkingsgebied: onderwijs[3] waaronder alle door de overheid erkende onderwijsorganisaties die binnen de sectoren po, vo, mbo/bve en ho vallen en hun dienstverleners.
2. Het functioneel toepassingsgebied: geautomatiseerde uitwisseling van vertrouwelijke gegevens (gesloten data) tussen informatiesystemen van onderwijsorganisaties en ketenpartners (onderling, met bedrijven of met de overheid). Deze uitwisseling betreft M2M point-to-point verbinding voor uitwisseling tussen een confidential client en een gesloten API waarbij de toegang is geregeld met OAuth.
    a. De client kan zowel een computersysteem van een onderwijsorganisatie als van een dienstverlener zijn.
    b. Of en hoe mandatering is ingericht valt buiten de scope van deze versie. Wel wordt aangenomen dat de Authorization Server een rol heeft bij mandaatverificatie. Daar waar mandaten van toepassing zijn, wordt er geen access token uitgegeven als er geen valide mandaat bestaat.
3. Edukoppeling is gebaseerd op internationale open standaarden en onderwijsstandaarden geregistreerd bij Edustandaard.
4. Er wordt als mogelijk een volwassen industry standard gekozen. Dit om te voorkomen dat nieuwe/kleine partijen te maken krijgen met (te) grote integratiedrempels.
5. Edukoppeling-profielen zijn op zichzelf staande documenten. We verwijzen naar internationale open standaarden en onderwijsstandaarden (bijvoorbeeld UBV TLS[4]).
6. Als we gebruik maken van (delen van) nationale standaarden (bijvoorbeeld NL GOV of Digikoppeling) dan worden relevante aspecten overgenomen met bronvermelding.
7. De standaard volgt ontwikkelingen en wordt onder Edustandaard beheer doorontwikkeld.
8. Afhankelijk van de karakteristieken van een ketensamenwerking kan het nodig zijn om meerdere varianten/keuzes binnen het profiel te ondersteunen.
9. Edukoppeling-profielen gaan niet over de inhoud van de uitwisseling en ook niet over het design van API's.

[3] https://rosa.wikixl.nl/index.php/Werkingsgebieden
[4] https://www.edustandaard.nl/standaard_afspraken/uniforme-beveiligingsvoorschriften/

1.5. Aanleiding

Binnen het Groeifondsprogramma Npuls werken Studielink, SURF en MBO Digitaal samen aan het onderbrengen van het huidige Studielink / Cambo in één nieuwe centrale voorziening voor het Aanmelden, Inschrijven en Intekenen (ook wel project AII). Als onderdeel van dit project worden ook de koppelvlakken met sectorpartners (onderwijsinstellingen, DUO, etc.) opnieuw onder de loep genomen, zodat daar voor het onderwijs een oplossing wordt neergezet conform de modernste standaarden en die past bij de strategie van de Nederlandse overheid en het daarbinnen vallende onderwijs (i.e. semi-overheid).

Vernieuwing van koppelvlakken vereist afstemming en dit wordt vanuit Edustandaard gefaciliteerd. Deze notitie biedt richting aan het toekomstige koppelvlak op het gebied van asynchrone machine-to-machine (M2M) integraties.

1.5.1. Ontwikkelingen

In deze paragraaf worden relevante ontwikkelingen binnen de overheid en IT beschreven die inspiratie bieden voor ontwikkeling binnen het onderwijs.

- Digikoppeling (ondersteunen door Logius) voor overheidsorganisaties vernieuwd regelmatig. Zo geldt ook dat Digikoppeling Architectuur 2.1.1 (januari 2025) [5] een uitgebreider aanbod heeft voor uitwisselingen; specifieke te benoemen: 4.4.4. Notificaties en Signalen.
- patroon voor gegevens uitwisseling binnen Event Driven Architecturen. Dit is één van de nieuwe patronen binnen de Digikoppeling Architectuur 2.1.1 en is daarbij een mogelijk bruikbaar patroon in de toolbox van een architect. Dit verwijst o.a. naar een JSON Event Format [6] voor CloudEvents en Web Hooks [7] voor Event Delivery. Een definitieve versie 1.0 [8] is momenteel al beschikbaar en definitieve concept versie 1.1 [9] ligt nu nog ter voorstel voor.
- AsyncAPI specificatie initiatief voor het beschrijven van event-driven APIs [10]
- Een voorgestelde IETF-standaard, om idempotency [11] te implementeren voor HTTP methoden die niet als veilig worden gezien, die toegevoegde waarde heeft voor robuuste end-to-end Event Driven Architecturen.

1.5.2. Advies

Het Edukoppeling-profiel kent momenteel al een transactiepatroon voor asynchrone melding-bevestiging. Dit transactiepatroon maakt gebruik van REST-APIs, vaak gespecificeerd via een OpenAPI-specificatie [12], i.e. hoe ook het synchrone bevragingen/meldingen patroon wordt vormgegeven.

Dit maakt de koppeling tussen aflevering van het bericht en inhoud nauw aan elkaar verknoopt. Het gebruik van CloudEvents (zoals beschreven in 2.2) zorgt juist voor

[5] https://gitdocumentatie.logius.nl/publicatie/dk/architectuur/2.1.1/
[6] https://github.com/cloudevents/spec/blob/v1.0.1/json-format.md
[7] https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md
[8] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.0/
[9] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/
[10] https://www.asyncapi.com/en
[11] https://datatracker.ietf.org/doc/draft-ietf-httpapi-idempotency-key-header/
[12] https://www.forumstandaardisatie.nl/open-standaarden/openapi-specification

ontkoppeling van aflevering en inhoud. Voor asynchrone communicatie is daarom de voorkeur om deze vorm van uitwisseling uit het aanbod te gaan gebruiken.

Dit geldt niet alleen voor simpele notificaties en signalen (zoals de Digikoppeling standaard voorschrijft), maar aanvullend ook voor het bereiken van losgekoppelde gegevensuitwisselingen.

2. Profiel

2.1. Edukoppeling-aanbod: Notificaties / Signalen

Asynchrone communicatie via RESTful API is een uitbreiding op het Edukoppeling-profiel en biedt aanvullend aanbod voor uitwisseling van gegevens. Door toevoeging van dit patroon in de gegevensuitwisseling wordt een zogenaamde Event Driven Architecture (EDA) gerealiseerd.

Een Event Driven Architecture (EDA) biedt een aantal voordelen ten opzichte van meer synchrone architecturen, zoals een Service Oriented Architectuur (SOA). Dit biedt verschillende voordelen ten opzichte van transactiepatronen die nu worden gebruikt:

A. Ontkoppeling van systemen: Deze architectuur is van nature losjes gekoppeld, omdat applicaties met elkaar communiceren via gebeurtenissen (en een tussenlaag – ook wel Intermediary). Dat maakt het eenvoudiger om applicaties onafhankelijk van elkaar te ontwikkelen, testen en implementeren.
B. Asynchrone communicatie: In een EDA hoeven aanvragen niet op elkaar te wachten.
C. Schaalbaarheid en gemak van het toevoegen van nieuwe consumenten: EDA maakt het gemakkelijk om nieuwe applicaties of diensten te implementeren en te integreren zonder de bestaande te beïnvloeden.
D. Hoge doorvoer en low latency: Een EDA kan een groot aantal events met een lage latency verwerken.

Dit patroon is niet alleen bruikbaar voor simpele notificaties en signalen (zoals de Digikoppeling standaard voorschrijft), maar aanvullend ook voor het bereiken van losgekoppelde gegevensuitwisselingen.

2.2. CloudEvents

Dit hoofdstuk beschrijft de aanvullende voorschriften op de CloudEvents standaard die de basis vormt voor asynchrone communicatie binnen het Edukoppeling-profiel. Het Edukoppeling-profiel biedt op een aantal punten keuzemogelijkheden. Hiermee beogen we een profiel dat een verplichte basisset van voorschriften bevat, maar ook genoeg ruimte biedt voor passende configuraties om voor ketenpartners binnen een bepaalde ketensamenwerking niet onoverkomelijke drempels op te werpen. Het is aan een ketensamenwerking om te bepalen welke opties van toepassing zijn.

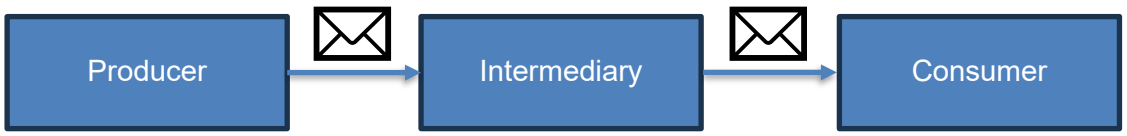
Dit profiel moet worden toegepast wanneer er binnen een ketensamenwerking *asynchrone* gegevensuitwisseling plaatsvindt tussen confidential clients en RESTful API's. De CloudEvents standaard is gebaseerd op het principe van het niet opleggen van meer eisen op de betrokken partijen dan noodzakelijk.

Het onderwijsveld bestaat uit verschillende ketenpartners die ieder verantwoordelijk zijn voor een deel van de keten en zijn systemen. De CloudEvents standaard moet daarom in de context van deze ketensamenwerking worden geplaatst.

De rollen en interacties worden hieronder verder toegelicht.

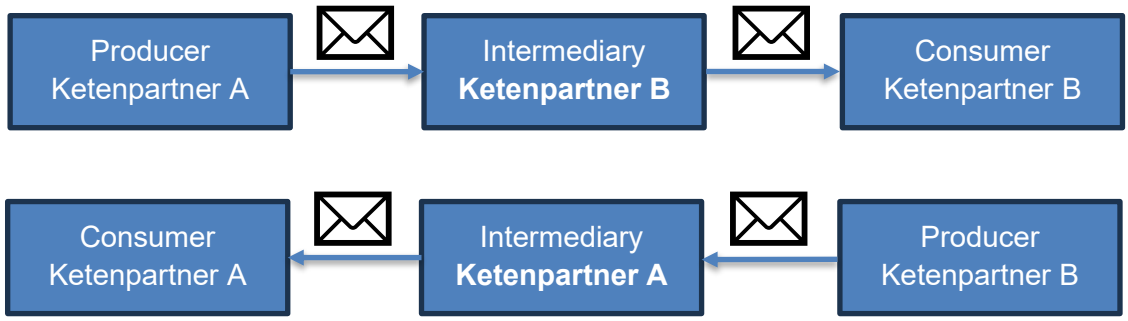
2.2.1. Rollen

Het basispatroon beschrijft een applicatie in de rol van ‘producer’ die ‘events’ publiceert: dataregistraties die een gebeurtenis en de bijbehorende context vastleggen. Gepubliceerde events kunnen worden geconsumeerd door applicaties in de rol van ‘consumer’. Consumers abonneren zich op bepaalde type events. Er kunnen één of meerdere applicaties in de rol van ‘intermediary’ zijn die zorgdragen voor het routeren van events naar consumers op basis van contextuele informatie. Dit is vergelijkbaar met het publish-subscribe-patroon:



Figuur 1 - Publish-subscribe patroon

In dit patroon is het duidelijk dat de producer wordt beheerd door de ketenpartner die berichten produceert en de consumer wordt beheerd door de ketenpartner die de berichten consumeert, en asynchroon verwerkt. Voor de Intermediary is dit niet direct duidelijk en is er momenteel ook binnen de (semi-)overheid geen één centrale partij toe te wijzen die logischerwijs dit beheer op zich kan nemen. We kiezen hiervoor om binnen de CloudEvents patronen *per ketenpartner* een Intermediary te implementeren. Dit zorgt ervoor dat elke dienstverlener deze Intermediary naar eigen inzicht kan inrichten en beheren:



Figuur 2 - Intermediary per ketenpartner

Deze aanpassing aan het patroon betekent dat elke partij een API-endpoint definieert waar CloudEvents naar verstuurd kunnen worden.

De rollen worden respectievelijk in paragrafen 2.5, 2.7 en 2.8 nader toegelicht.

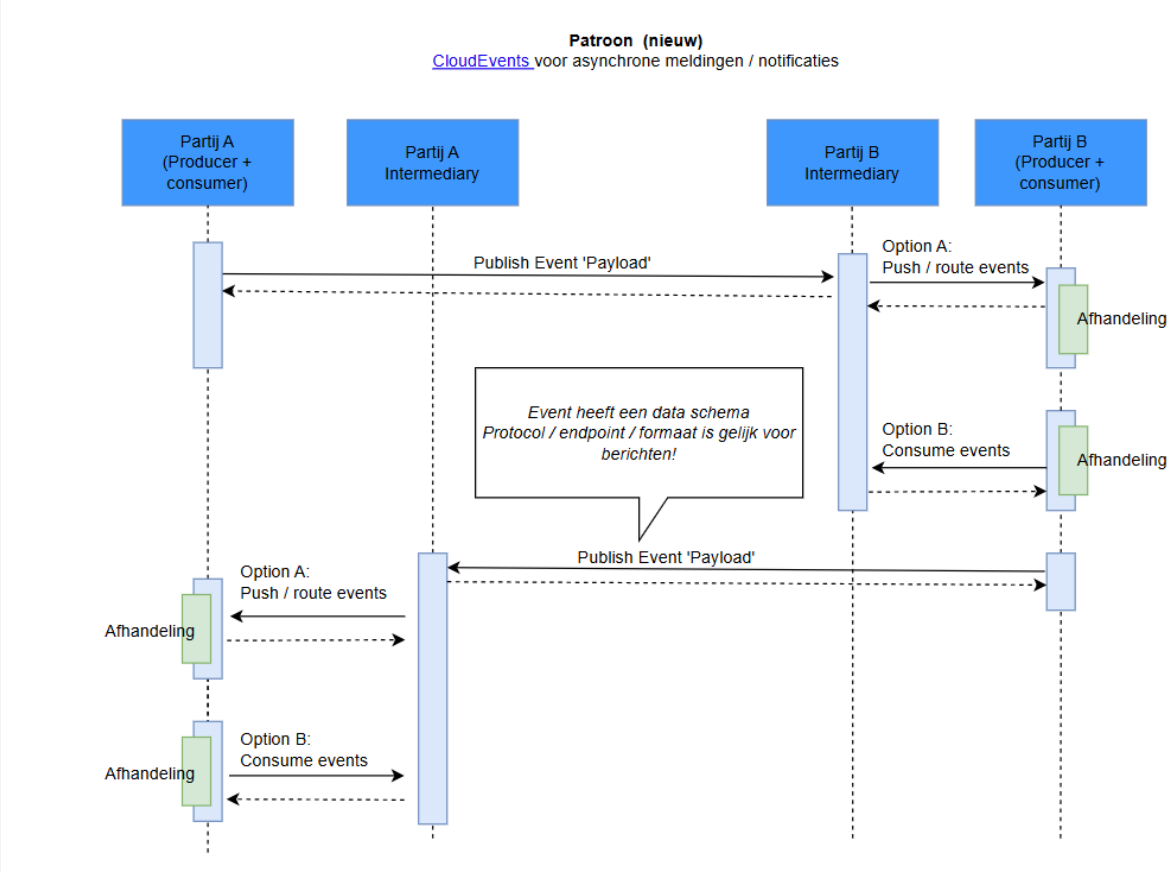
2.2.2. Interacties

Deze componenten hebben de volgende interacties:

- Berichtenverkeer partij A naar partij B:
  - De producer van partij A produceert events naar de Intermediary van partij B.
  - De Intermediary B persisteert het event zodanig dat consumer partij B deze kan consumeren óf stuurt het event direct door naar consumer partij B.
  - Consumer partij B verkrijgt het event van Intermediary B gerouteerd/gepushed óf consumeert de events van Intermediary B.
- Berichtenverkeer partij B naar partij A:
  - De producer van partij B produceert events naar de Intermediary van partij A.

- De Intermediary A persisteert het event zodanig dat consumer partij A deze kan consumeren óf stuurt het event direct door naar consumer partij A.
- Consumer partij A verkrijgt het event van Intermediary A gerouteerd/gepushed óf consumeert de events van Intermediary A.

De componenten en interacties worden weergegeven in Figuur 3 - CloudEvents uitwisseling.



Figuur 3 - CloudEvents uitwisseling

2.3. Event levering semantiek

In een gedistribueerd landschap heb je traditioneel te maken met de uitdaging van *atomiciteit*. Dit geldt ook voor het event-uitwisseling.

2.3.1. Aflevergaranties

Als het gaat om aflevergaranties richting een Intermediary zijn er drie opties:

- At most once: een bericht kan worden afgeleverd, maar nooit meer dan één keer. Dit kan leiden tot verlies van berichten en wordt daarom zelden, zo niet nooit, gebruikt.
- At least once: een bericht wordt afgeleverd, maar kan meer dan één keer worden afgeleverd. Dit kan leiden tot dubbele berichten.
- Exactly once: een bericht wordt precies één keer afgeleverd.

De reikwijdte van deze garanties ligt echter alleen binnen de event-broker en niet daarbuiten, zoals in een gedistribueerd IT-landschap. In een volledig robuust IT-landschap moeten we rekening houden met end-to-end robuuste aflevering en uitgaan van het worstcasescenario.

Een worstcasescenario is wanneer een producer crasht / herstart / stopt tijdens de verwerking van een event. Met name in een schaalbare infrastructuur is dit een realistisch scenario. Afhankelijk van het exacte moment van de crash kunnen zich verschillende situaties voordoen:

- Als de producer het event nog niet heeft geproduceerd: de producer zal het event opnieuw proberen te leveren. Dit vormt geen probleem.
- Als de producer het event heeft geproduceerd en geen andere status hoeft bij te houden (meestal wanneer de event-broker de enige bron van waarheid is): dit vormt geen probleem.
- Als de producer het event heeft geproduceerd, maar het event nog niet als afgeleverd heeft gemarkeerd: de producer zal het event opnieuw proberen te produceren, wat leidt tot dubbele events. Zonder mitigerende maatregelen aan de consumer-zijde kan dit tot problemen leiden.

2.3.2. Consumer garanties

Wat betreft consumer-garanties vanuit een event-broker zijn er drie opties:

- At most once: een bericht kan worden geconsumeerd, maar nooit meer dan één keer. Dit kan leiden tot verlies van berichten en wordt daarom zelden, zo niet nooit, gebruikt. Dit treedt typisch op wanneer een bericht als verwerkt wordt gemarkeerd vóór de verwerking, maar de verwerking daarna faalt.
- At least once: een bericht wordt geconsumeerd, maar kan meer dan één keer worden geconsumeerd. Dit kan leiden tot dubbele verwerking. Dit treedt typisch op wanneer een bericht pas ná de verwerking als verwerkt wordt gemarkeerd, maar het markeren faalt en het bericht opnieuw wordt geconsumeerd.
- Exactly once: een bericht wordt exact één keer verwerkt. Een bericht kan vaker dan één keer worden geconsumeerd, maar de verwerking moet exact één keer plaatsvinden. In onze gedistribueerde opzet is dit zonder aanvullende maatregelen per definitie niet mogelijk, vanwege de uitdagingen rond atomiciteit.

2.3.3. End-to-end exactly-once verwerking in een EDA

In deze sectie bekijken we hoe echte end-to-end exactly-once verwerking kan worden gerealiseerd binnen een Event-Driven Architecture (EDA). Daarbij zorgen we ervoor dat:

A. Een **producer** events **at least once** produceert. In normale omstandigheden produceert een producer altijd exactly once, maar in foutscenario's kan dit vaker gebeuren.
B. Een **consumer** events **at least once** consumeert. In normale omstandigheden consumeert een consumer altijd exactly once, maar in foutscenario's kan dit vaker gebeuren.
C. De verwerking **exactly once** wordt uitgevoerd door middel van idempotentie (zie bijlage B) in de ontvangende service.

Dit zorgt voor een minimale extra impact op de performance aan de verwerkingskant en waarborgt uiteindelijke atomiciteit zonder dataverlies of duplicatie.

2.3.4. Trade-offs

We gaan uit van een moderne EDA en vergelijken traditionele patronen (zie ook bijlage A) 2PC, Saga en idempotentie:

	2PC	Saga	Idempotentie
Schaalbaarheid	---	+	+++
Complexiteit	- (coördinatie)	--- (compenserende acties, orkestratie)	+
Latency	---	-	+++
EDA-vriendelijk	---	+++	+++
Impact op leveranciers/keten	---	---	-
Eindscore	---	+-	++

In bijlage B worden richtlijnen voor het implementeren van idempotentie beschreven.

2.4. Uitgangspunten voor het profiel

1. Dit Edukoppeling-profiel is gebaseerd op de internationale open standaarden rond CloudEvents. Het fundament is de CloudEvents specificatie, opgesteld door Cloud Native Computing Foundation (CNCF).

2. Dit Edukoppeling-profiel volgt de richtlijnen zoals beschreven in het NL GOV profile for CloudEvents [13]:
   a. Context attributen (H3);
   b. Event Data (H4);
   c. Size Limits (H5);
   d. Gebruik van JSON, HTTP en webhook (bijlage A) zoals ook beschreven in de Guidelines for NL-GOV profile CloudEvents [14].

   Met uitzondering van:
   e. CloudEvent Security Options (H6);
   f. Abuse protection as described in the guidelines [15].

   In paragraaf 2.10 CloudEvent beveiligingsopties worden de beveiligingsopties binnen dit profiel beschreven.

3. Dit Edukoppeling-profiel is alleen van toepassing in de context van confidential clients.

4. Dit Edukoppeling-profiel volgt het OAuth client credentials profiel voor RESTful API's voor machine-to-machine (M2M) gegevensuitwisseling binnen het onderwijs.

[13] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.1/
[14] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/
[15] https://gitdocumentatie.logius.nl/publicatie/notificatieservices/guidelines/1.0/

5. Dit Edukoppeling-profiel schrijft het gebruik van Transport Layer Security (TLS) voor conform UBV TLS.

2.5. Registratie

De ketenpartner die verantwoordelijk is voor de producer (hierna: producer) is ervoor verantwoordelijk dat een portaal en/of proces beschikbaar gesteld voor registratie van de ketenpartners die verantwoordelijk is voor de consumer (hierna: consumer).

De registratie volgt een aantal stappen:
- Ketenpartners spreken het voornemen uit naar elkaar om berichtuitwisseling via CloudEvents op te zetten.
- De consumer accepteert en registreert dat.

Bij registratie is het van belang dat de producer alleen gegevensleveringen naar consumers mogelijk maakt die toegang mogen hebben tot de data die wordt uitgewisseld. Dit stelt de producer zeker. De producer maakt het mogelijk via een portaal/proces om een afleverlocatie te configureren / aan te leveren.

Bij registratie van het webhook-endpoint bij de producer, controleert de producer.

Het is de verantwoordelijkheid van de consumer om zich te registreren bij de producer voor de berichten van gegevensleveringen die noodzakelijk zijn. De consumer levert hiervoor een webhook-endpoint aan.

2.6. Producer

Een producer is een systeem dat voor één of meer gegevensuitwisselingen (in verschillende ketensamenwerkingen) CloudEvents produceert en aflevert bij geregistreerde Intermediaries.

De producer houdt bij het opstellen van requests rekening met aan welke geregistreerde partijen berichten dienen te worden aangeleverd (conform registratie) en stuurt de berichten naar het registreerde afleverpunt (i.e. een webhook) van de Intermediary. Hierbij houdt de producer rekening met eventuele verschillende versies die een bericht kan hebben.

Indien de producer eigenaar is van de berichtspecificatie, registreert de producer de technische contracten bij het contract register (zie 2.9 Technische contracten).
Voor de producer gelden aanvullend de volgende richtlijnen:
- Een producer *moet* best-practices in event-driven architecturen ondersteunen om at-least once produceren van events en exactly-once verwerking mogelijk te maken, door middel van bijv. het transactioneel outbox patroon (paragraaf 2.12.3) en idempotentie (hoofdstuk 3).
- Een producer *moet* rekening houden met rate-limiting van de intermediary en past throttling toe waar nodig.
- Een producer *mag* CloudEvents bufferen en als batch aanleveren, indien de ontvanger CloudEvents in batch ondersteunt.
- Een producer *mag* circuit breaking toepassen bij ongeplande niet-beschikbaarheid.

2.7. Intermediary

De intermediary uit zich naar de buitenwereld als een beveiligd endpoint. Het endpoint verwacht een payload in lijn met de CloudEvents standaard. Dit betreft daardoor twee varianten:

1. Endpoint voor individuele CloudEvents berichten (verplicht)
2. Endpoint voor CloudEvents in batch (optioneel)

De intermediary registreert het technische contract van de webhook bij het contract register (zie 2.9 Technische contracten).

De intermediary heeft een aantal taken:

a. accepteert het verzoek;
b. valideert het verzoek (minimaal de CloudEvents structuur);
c. persisteert het verzoek;
d. initieert verwerking bij de consumer (optioneel). Dit vindt plaats in geval van optie A uit Figuur 3 - CloudEvents uitwisseling;
e. geeft een technische bevestiging dat het bericht is ontvangen. Functionele verwerking vindt asynchroon plaats aan de kant van de consumer.

De intermediary *mag* rate-limiting toepassen om zijn systemen weerbaarder te maken. In het algemeen wordt aangeraden om rate-limiting toe te passen per producer.

2.8. Consumer

De consumerende partij bepaalt hoe de ontvanger geïmplementeerd wordt. Hier kunnen verschillende patronen gebruikt worden zoals beschreven in Figuur 3 - CloudEvents uitwisseling:

A. De consumer wacht op berichten voor verwerking die actief door de Intermediary aangeleverd kunnen worden (PUSH).
B. De consumer haalt de berichten actief bij de intermediary (PULL).

Een consumer *moet* best-practices in event-driven architecturen ondersteunen om at-least once consumeren van events en exactly-once verwerking mogelijk te maken, door middel van bijv. idempotentie (hoofdstuk 3), retries, DLQ en replays.

2.9. Technische contracten

Ketenpartner bepalen altijd gezamenlijk wie welke contracten opstelt. Desondanks, stellen we belangrijk richtlijnen op die kunnen worden gehanteerd. We hanteren de volgende richtlijnen:

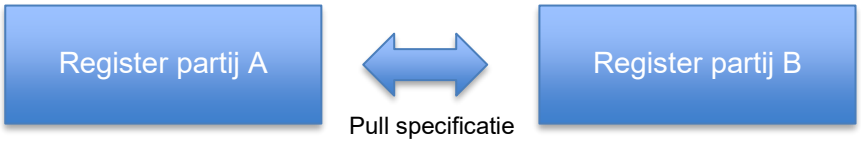
- De OpenAPI-specificatie (OAS) van de webhook wordt gedefinieerd door de ontvangende partij.
- De AsyncAPI-specificatie (AAS), i.e. de business data, wordt gedefinieerd door de producer, omdat deze eigenaar is van data en het gedrag. Hierbij stemt de producer de specificatie af op de eisen van de mogelijke consumers. Datacontracten hanteren idealiter waar mogelijk sectorstandaarden voor gegevensstructuren.

Zoals elke richtlijn kan daar met goede argumentatie van afgeweken worden, indien noodzakelijk. Zo kan het wenselijk zijn dat een partij die een centrale voorziening levert afspraken maakt over welke gegevens ontvangen kunnen worden. Daarom bepalen ketenpartners altijd gezamenlijk definitief *wie* de specificatie opstelt en onderhoudt.

2.9.1. Contract-register

Contracten worden op één plek geregistreerd door de partij die de specificatie definieert; elke ketenpartner zal zo'n contract-register aanbieden. Deze plek, een contract-register, is de bron van waarheid voor de betreffende specificatie.

Het is mogelijk om vanuit beveiligingsoogpunt eerst de specificatie over te zetten naar het 'eigen' intern register. Dit is weergegeven hieronder:

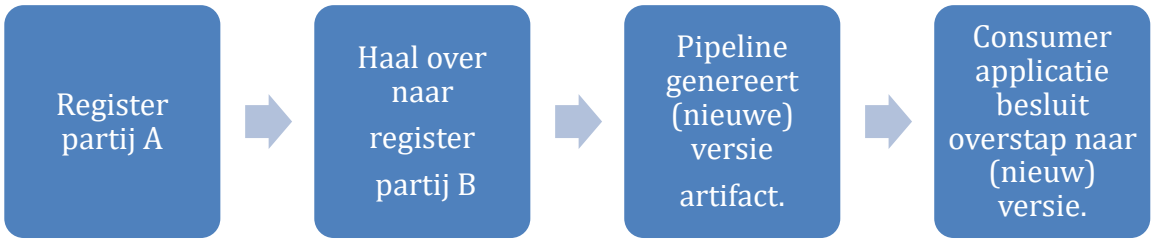


Figuur 4: Interactie tussen registers partijen.

Andere partijen kunnen deze specificatie dan handmatig en/of geautomatiseerd ophalen om de andere kant van de integratie op te zetten. Automatisering kan helpen om de berichtspecificatie op te halen uit de bron en daaruit direct, of via een intern register dat een kopie opslaat, automatisch code te generen.

Bijvoorbeeld als volgt:

- Register partij A bevat het AsyncAPI contract.
- Partij B haalt het contract over register A naar een eigen immutable register B.
- Een pipeline draagt zorg voor het ophalen van dit contract uit het interne register, genereren, compileren, packaging en publicatie van het artifact naar een interne repository.
- Een automatische dependency updater kan detecteren dat er een nieuwe versie beschikbaar is, de versie ophogen en een ontwikkelaar laten besluiten of dit direct doorgevoerd kan worden. Belangrijk: Een consumer applicatie beslist zelf wanneer een upgrade plaatsvindt.



Figuur 5: Voorbeeld proces over rol register en automatisering.

2.10. CloudEvent beveiligingsopties

De beveiliging vindt plaats op twee niveaus:

- Applicatielaag: Maakt gebruik van het OAuth client credentials profiel voor RESTful API's.
- IT-infrastructuurlaag: Maakt gebruik van Transport Layer Security (TLS) conform UBV TLS.

Dit Edukoppeling-profiel definieert geen aanvullende beveiligingsmechanismen. Zo geldt:

- De payload wordt niet aanvullend beveiligd. Dat is niet nodig omdat het uitgangspunt machine-to-machine (M2M) is. Vooralsnog wordt de payload alleen beveiligd in transport (TLS).
- Abuse protection zoals beschreven in de webhook standaard[16] is niet nodig, omdat OAuth2 al de toegang regelt tot het webhook-endpoint. Indien de consumer toegang verleent aan de producer via OAuth2, wordt daarmee ook de producer goedgekeurd om berichten naar de consumer te sturen.

De volgende voorschriften gelden wel voor CloudEvents:

- Context-attributen: Sensitieve informatie zou niet opgenomen moeten worden in context-attributen aangezien producers, consumers en intermediairs deze attributen mogen loggen.
- Domein specifieke data (data-attribuut): Domein specifieke event data wordt niet versleuteld.

Aanvullende beveiliging afspraken worden, waar noodzakelijk, afgestemd tussen de producers en consumers. Zo geldt:

a. Versleuteling is alleen noodzakelijk in scenario's waar (niet vertrouwde) tussenliggende componenten voor datalekken kunnen zorgen. Versleutelen is rekenintensief en het maakt het bovendien moeilijker voor beveiligingsmechanismen, zoals API-gateways, de payload te valideren en transformeren (indien nodig).
b. Ondertekening van gegevens is alleen nodig indien er een risico is dat de integriteit van de gegevens aangetast kan worden of als onweerlegbare overdracht vereist wordt.

[16] https://github.com/cloudevents/spec/blob/v1.0.2/cloudevents/http-webhook.md

Bijlage A: Uitdagingen in een gedistribueerd landschap

Een gedistribueerd IT-landschap bestaat uit meerdere systemen die met elkaar communiceren door het uitwisselen van berichten. In die zin is er altijd sprake van een zender en een ontvanger van een bericht. We moeten ervoor zorgen dat er geen berichten verloren gaan of dubbel worden verwerkt. Dit zou namelijk leiden tot informatieverlies of dubbele data.

Bijvoorbeeld: in een IT-landschap met een ordersysteem en een betaalsysteem moeten we ervoor zorgen dat een order die in het ordersysteem wordt aangemaakt exact één keer financieel wordt verwerkt.

Robuuste afhandeling betekent dat alle berichten (verzonden door een zender) exact één keer door de ontvanger worden verwerkt, zonder aanvullende neveneffecten.

2.11. Traditionele uitdagingen in een gedistribueerd IT-landschap

Binnen de grenzen van één enkel systeem zorgen transacties ervoor dat óf alle wijzigingen óf geen van de wijzigingen worden opgeslagen. Dit concept van atomiciteit zorgt ervoor dat een transactie “alles of niets” is.

In een gedistribueerd IT-landschap is het echter, zonder aanvullende maatregelen die een gedistribueerde transactie over systeemgrenzen heen waarborgen, niet mogelijk om twee (of meer) systemen atomair bij te werken, omdat er in wezen sprake is van twee afzonderlijke transacties.

Consistentie over gedistribueerde systemen kan wel worden geïmplementeerd, maar dit gaat ten koste van de schaalbaarheid. In de volgende paragrafen verkennen we traditionele patronen om deze uitdagingen op te lossen binnen een microservices-architectuur of andere systeem-tot-systeemkoppelingen.

In paragraaf beschrijven we traditioneel gebruikte patronen in een traditioneel gedistribueerd IT-landschap.

2.12. Traditioneel gebruikte patronen in een gedistribueerd IT-landschap

2.12.1. Patroon: Two-phase commit (2PC)

Het two-phase commit-patroon wordt gebruikt om data op meerdere nodes in een gedistribueerd systeem atomair, volgens het alles-of-niets-principe, op te slaan. De twee fasen in dit patroon worden gecoördineerd door één centrale coördinator:

- Voorbereidingsfase (preparation phase): De coördinator stuurt een verzoek naar alle nodes en vraagt elke deelnemer te controleren of de transactie kan worden voltooid.
- Commitfase (commit phase): Als alle deelnemers positief hebben geantwoord, stuurt de coördinator een commit naar alle deelnemers. Als ten minste één deelnemer negatief heeft geantwoord, stuurt de coördinator een abort naar alle deelnemers.

Belangrijk in dit patroon is dat elke deelnemer in de voorbereidingsfase de duurzaamheid (durability) van de beslissing waarborgt.

2.12.2. Patroon: Saga-patroon

Het Saga-patroon is in essentie een reeks lokale transacties in afzonderlijke systemen. Deze reeks kan op twee manieren worden gecoördineerd:

- Elke lokale transactie publiceert berichten die lokale transacties in andere services activeren; een choreografie van berichtensequenties.
- Een orchestrator instrueert de deelnemers welke lokale transacties zij in welke volgorde moeten uitvoeren.

Binnen dit patroon moeten maatregelen worden genomen om een transactie “ongedaan te maken” wanneer een vervolgstap door een deelnemer niet succesvol kan worden verwerkt. Het ongedaan maken van een transactie is een compenserende actie, en geen strikte rollback.

2.12.3. Patroon: Transactioneel outbox-patroon

Tijdens de verwerking van een businessoperatie schrijft een microservice het ‘uitgaande bericht’ weg in een outbox-tabel binnen dezelfde transactie als de overige datawijzigingen. Een achtergrondproces leest de outbox en publiceert de berichten op betrouwbare wijze naar een message broker.

Dit zorgt ervoor dat er geen berichten verloren gaan wanneer de volledige businesslogica niet succesvol wordt afgerond. Hierdoor worden atomaire writes mogelijk binnen het systeem waarin de businessoperatie plaatsvond.

3. Bijlage B: Richtlijnen voor idempotentie

In deze bijlage kijken we naar de praktische richtlijnen voor het implementeren van idempotentie. Eerst bespreken we *wanneer* dit moet worden toegepast en vervolgens *hoe*. Als laatst worden technische richtlijnen beschreven voor de implementatie.

3.1. Wanneer implementeren

Idempotentie zorgt ervoor dat het meerdere keren verwerken van hetzelfde bericht geen schadelijke of onbedoelde neveneffecten of wijzigingen in de systeemtoestand veroorzaakt en dat een deterministisch resultaat wordt gegarandeerd. Dit betekent ook dat niet alle service-operaties hierdoor worden beïnvloed. Sommige service-operaties zijn per definitie idempotent. Enkele voorbeelden:

- Elke GET-operatie is idempotent, omdat deze geen toestand wijzigt.
- Elke PUT-operatie (volgens de specificaties) vervangt of wijzigt een resource. *Voorbeeld:* PUT /account/123 met {balance: 100} moet het saldo van de rekening op 100 zetten. Opnieuw toepassen levert hetzelfde resultaat op. Order guarantee is hierbij belangrijk.
- Elke DELETE-operatie is idempotent, omdat een entiteit niet meer dan één keer kan worden verwijderd. Aandachtspunt is de response – zie sectie 3.2.1.
- POST-operaties kunnen wel of niet idempotent zijn, afhankelijk van de functionaliteit. *Voorbeeld:* POST /transactions met { amount: 100, account: 123 } kan bij herhaling opnieuw geld toevoegen aan rekening 123, wat leidt tot een onjuist saldo. *Voorbeeld:* POST /email/send zal opnieuw een e-mail versturen (er is dus een neveneffect).

3.2. Algemene richtlijnen

De aanbevolen manier om idempotentie te implementeren is door middel van een unieke event-ID in elk bericht. De consumer kan dan de volgende eenvoudige logica toepassen:

- Kent de consumer het message-ID al? → Verwerp het bericht.
- Kent de consumer het message-ID nog niet? → Verwerk het bericht en sla het message-ID op.

Om volledige atomiciteit te garanderen, wordt het message-ID opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.

Een goede message-ID moet globaal uniek, stabiel, deterministisch en compact genoeg zijn. Dit message-ID moet door de producer worden gegenereerd.

3.2.1. Response

Elke idempotente service zorgt ervoor dat de response bij de eerste, tweede en derde poging identiek is. Dit is met name relevant voor services die:

- Data aanmaken: de client wil doorgaans het identificatienummer van de aangemaakte entiteit ontvangen.
- Data verwijderen: de client moet correcte foutafhandeling kunnen implementeren. Dit zou niet mogelijk zijn als bij een tweede DELETE-aanroep een foutresponse wordt teruggegeven.

3.3. Technische implementatierichtlijnen: Idempotentie

Idempotentie zorgt ervoor dat meerdere identieke verzoeken hetzelfde effect hebben als één enkel verzoek. Dit is cruciaal voor scenario's waarin clients verzoeken opnieuw proberen te versturen vanwege time-outs, netwerkfouten of onherstelbare fouten aan de clientzijde.

Deze technische richtlijnen zijn gebaseerd op een bestaand Internet Engineering Task Force (IETF)-concept [17] en bieden aanvullende richtlijnen om consistente implementatie en efficiënte systeemintegratie binnen de onderwijssector te ondersteunen. Eerst worden de idempotentieconventies binnen de sector beschreven, daarna volgt een gedetailleerd implementatievoorbeeld op basis van deze conventies.

3.3.1. Idempotentieconventies

Alle operaties die resources aanmaken of muteren, typisch POST- en PATCH-operaties, *moeten* een Idempotency-Key-header gebruiken. Hoewel DELETE-operaties per definitie [18] idempotent zijn, kunnen clients in een gedistribueerd systeem met retries inconsistente responses krijgen. Daarom *zouden* DELETE-operaties ook een idempotency key moeten gebruiken.

Elke operatie die niet voldoet aan de standaarden voor het gebruik van HTTP-methoden [19] en resources aanmaakt of wijzigt, *moet* eveneens worden meegenomen.

3.3.2. Uniekheid idempotentie sleutel

Een UUIDv4 [20] (RFC4122) *moet* worden gebruikt als idempotentie-sleutel.

3.3.3. Geldigheid en verloop van idempotentie sleutel

De geldigheid en vervaldatum van de idempotency key *zouden* rekening moeten houden met de tijd die nodig is voor automatische en/of handmatige retry-mechanismen en met de mogelijkheid van herhalingen (worstcasescenario na een restore door de client).

De time-to-live (TTL) voor een idempotency-entry *zouden* gebaseerd moeten zijn op het maximaal verwachte replay-venster: met andere woorden, het maximale tijdsvenster waarin een bericht opnieuw kan worden aangeboden. Een vervalperiode van 7 dagen wordt *aanbevolen*.

3.3.4. Idempotentie fingerprint

Voor extra veiligheid *moet* een idempotentie fingerprint worden aangemaakt. Aanbevolen wordt om de volledige request-payload te gebruiken om deze fingerprint te genereren.

[17] draft-ietf-httpapi-idempotency-key-header-07 - The Idempotency-Key HTTP Header Field
[18] RFC 9110: HTTP Semantics
[19] RFC 9114: HTTP/3
[20] RFC 4122 - A Universally Unique IDentifier (UUID) URN Namespace

3.3.5. Verantwoordelijkheden client

Clients *moeten* de idempotentie sleutel opslaan voor de volledige levensduur van de operatie (oftewel de TTL die door de resource is gespecificeerd). Dit garandeert dat de sleutel uniek, stabiel, deterministisch en constant blijft gedurende de operatie. De client MAG hiervoor het transactional outbox-patroon gebruiken (zie bijlage A).

Clients *moeten* voor één idempotency key dezelfde request-payload versturen om HTTP 422-fouten van de resource te voorkomen.

Clients *zouden* automatische retries moeten stoppen nadat de TTL van de resource is verlopen. Daarna kan de client er niet langer van uitgaan dat de resource de idempotentie sleutel nog heeft opgeslagen. Handmatige bevestiging binnen de clientapplicatie *zou* dan moeten worden gestart en een nieuw verzoek *moet* een nieuwe sleutel bevatten.

3.3.6. Verantwoordelijkheden resource

De resource *moet* bij een duplicaatverzoek de response retourneren van de eerder voltooide operatie. Met name DELETE-verzoeken *zouden* ook dezelfde response moeten retourneren als eerder.

Als de resource meerdere clients ondersteunt, *moet* de opslag van idempotency keys per client worden gescheiden om het risico op botsingen te elimineren.

Om volledige atomiciteit te waarborgen, *zou* de idempotency key moeten worden opgeslagen in dezelfde transactie als de daadwerkelijke businessverwerking.

3.3.7. Foutafhandeling

De resource *moet* de idempotency key valideren voordat verdere verwerking plaatsvindt. Als een ongeldige idempotency key wordt aangeleverd, *zou* de resource moeten antwoorden met een HTTP 400-statuscode.

3.3.8. Client side implementatie (voorbeeld)

De volgende informatie kan in een outbox worden vastgelegd volgens het transactional outbox-patroon:

- event_id
- event_type
- payload
- schema_version
- created_at
- expires_at (moment waarop het event niet langer geldig wordt verklaard, bijvoorbeeld na het verstrijken van de TTL van de resource)
- idempotency_key (de UUIDv4)
- status (pending, acknowledged, failed)

De logica aan de clientzijde kan er als volgt uitzien:

1. Een business-event wordt vastgelegd inclusief alle metadata.
2. Zoek business-events met status *pending*.

3. Als expires_at > now(), zet de status op *failed*.
4. Anders:
   a. Lees het business-event.
   b. Exporteer het business-event en wacht op de response.
   c. Als de response een 4XX-fout aangeeft, los het probleem in de applicatie op.
   d. Als de response een 5XX-fout aangeeft, zet de status op *failed*.
   e. Als de response succesvolle verwerking aangeeft, verwijder het business-event of zet de status op *acknowledged*.

3.3.9. Resource implementatie (voorbeeld)

Alle API's die resources aanmaken of muteren, typisch POST, DELETE en PATCH-operaties, MOETEN een Idempotency-Key-header ondersteunen.

De volgende informatie kan worden opgeslagen in een idempotency key-tabel:

- idempotency_key
- idempotency_fingerprint (hash van het request)
- resource_id (indien van toepassing)
- client_id (indien van toepassing)
- response_status_code
- response_body
- created_at
- expires_at

De validatie aan de resourcezijde kan er als volgt uitzien:

1. Valideer de aanwezigheid én het formaat van de idempotency key. Indien deze ontbreekt of ongeldig is, retourneer een HTTP 400-fout.
2. Zoek de idempotency_key op voor resource_id + client_id in de tabel.
   a. Indien aanwezig én de idempotency fingerprint niet overeenkomt, retourneer een HTTP 422-fout.
   b. Indien aanwezig, retourneer de opgeslagen response.
3. Probeer een idempotency-lockrecord te verkrijgen inclusief vervaltijd. Als dit mislukt, verwerkt een ander proces deze key op dat moment en wordt een HTTP 409-fout geretourneerd.
4. Start een transactie:
   a. Verwerk de businesslogica.
   b. Voeg het idempotency-record inclusief response toe aan de tabel.
5. Commit de transactie en geef de idempotency-lock vrij.
6. Retourneer de response van de businesslogica.

Daarnaast kan een automatisch proces de idempotency key-entries opschonen waarvoor expires_at > now().

4. Bijlage C: Begrippen

Antwoord: Een antwoord is de inhoudelijke reactie op een verzoek: het resultaat van een operatie of vraag.

API aanbieder (ook wel API provider): Ketenpartner die binnen een ketensamenwerking een RESTful API aanbiedt.

API afnemer (ook wel API provider): Ketenpartner die binnen een ketensamenwerking met een systeem een RESTful API afneemt.

Asynchroon: Bij asynchrone communicatie stuurt een computersysteem een bericht of event en ontvangt hoogstens een *bevestiging* van ontvangst. De *verwerking* vindt losgekoppeld en later plaats en levert geen direct antwoord op.

Atomiciteit: Binnen de grenzen van één enkel computersysteem zorgen transacties ervoor dat óf alle wijzigingen óf geen van de wijzigingen worden opgeslagen. Dit concept van atomiciteit zorgt ervoor dat een transactie “alles of niets” is.

Bevestiging: Een bevestiging van ontvangst geeft alleen aan dat het bericht technisch correct is ontvangen, niet dat het al is verwerkt.

Client secret: Een vertrouwelijke sleutel die alleen bekend is bij de client en afhankelijk van de vorm (symmetrisch (wachtwoord) of asymmetrisch (PKI)) ook bij de Authorization Server. Het wordt gebruikt om de client bij de Authorization Server te kunnen authenticeren via het token endpoint.

Confidential client[21]: Een confidential client is een client die draait in een omgeving waar vertrouwelijke gegevens (waaronder het client secret) veilig bewaard kunnen worden (bijvoorbeeld server-side webapplicaties).

Consumer: Een “consumer” ontvangt het event en onderneemt actie op basis daarvan. De consument gebruikt de context en de data om logica uit te voeren, wat kan leiden tot het optreden van nieuwe events.

Consumeren: Het lezen/verkrijgen van een event vanuit de Intermediary.

Context: Contextmetadata zijn vastgelegd in de contextattributen. Tools en applicatiecode kunnen deze informatie gebruiken om de relatie van events met onderdelen van het computersysteem of met andere events te identificeren.

Data: Domeinspecifieke informatie (oftewel de payload). Dit kan informatie bevatten over de gebeurtenis zelf, details over gewijzigde gegevens of andere relevante gegevens.

[21] https://datatracker.ietf.org/doc/html/rfc6749#section-2.1

DLQ (Dead Letter Queue): een aparte queue waarin events terechtkomen die na meerdere retries niet succesvol verwerkt konden worden.

Deterministisch: Het resultaat is hetzelfde, ongeacht hoe vaak de operatie wordt herhaald.

Idempotentie: Het meerdere keren verwerken van een identiek verzoek veroorzaakt geen schadelijke of onbedoelde neveneffecten of wijzigingen in de systeemtoestand en garandeert een deterministisch resultaat.

Intermediary: Een “intermediary” ontvangt een bericht dat een event bevat met als doel dit door te sturen naar de volgende ontvanger. Dit kan een andere intermediair of een consument zijn. Een typische taak van een intermediair is het routeren van events naar ontvangers op basis van de informatie in de context.

Order guarantee: Order guarantee is het concept dat berichten worden verwerkt in de volgorde waarin ze zijn geproduceerd. Dit voorkomt dat een status wordt teruggezet naar een oude waarde terwijl een nieuwere waarde al is verwerkt.

Producer: De “producer” is een specifieke instantie, proces of apparaat dat de datastructuur aanmaakt die het CloudEvent beschrijft.
Produceren: Het creëren van een event en het afleveren ervan bij de Intermediary.

Replay: Het opnieuw aanbieden van eerder opgeslagen events om ze (opnieuw of alsnog) te verwerken.

Retry: Het opnieuw proberen te verwerken van een event nadat de eerste verwerking is mislukt, vaak vanuit een apart gezette queue.

RESTful API: Een RESTful API is een application programmable interface (API) die HTTP-methoden, zoals GET, POST, PUT, PATCH en DELETE, gebruikt om resources te beheren. Resources worden geïdentificeerd via URIs (Uniform Resource Identifiers) en worden doorgaans geretourneerd in JSON- of XML-formaat. We noemen ze RESTful omdat ze niet aan alle REST[22] principes[23] hoeven te voldoen.

Synchroon: Bij synchrone communicatie stuurt een systeem een verzoek en wacht op het *antwoord* (response). Dit antwoord wordt pas teruggestuurd nadat de *verwerking* is afgerond.

Verwerking: Verwerking is het daadwerkelijk uitvoeren van de businesslogica op basis van het ontvangen bericht of verzoek.

[22] REST - Wikipedia
[23] Dit Edukoppeling-profiel gaat uit van vertrouwelijke gegevens waarbij ketenpartners weten wat ze van elke vragen binnen een bepaalde ketensamenwerking. Ondersteuning van HATEOAS lijkt niet noodzakelijk.

5. Bijlage D: Referenties

- **CloudEvents standaard:** https://github.com/cloudevents/spec/blob/v1.0.1/spec.md
- **CloudEvents profiel Logius:** https://gitdocumentatie.logius.nl/publicatie/notificatieservices/cloudevents-nl/1.0/
- **JSON Event Format:** https://github.com/cloudevents/spec/blob/v1.0.1/json-format.md
- **Webhook:** https://github.com/cloudevents/spec/blob/v1.0.1/http-webhook.md