Ministerie van Binnenlandse Zaken en Koninkrijksrelaties

Migratievoorziening Technisch ontwerp Deltabepaling

1.5

Datum 11-1-2017
Status Definitief

Inhoudsopgave

Versiehistorie

Datum	Versie	Omschrijving	Auteur
13-07-2015	0.1	Initiële versie	operatie BRP
22-07-2015	1.0	Review verwerkt	operatie BRP
28-09-2015	1.0.1	Referenties bijgewerkt	operatie BRP
18-08-2015	1.1	Aanpassingen vanwege refactor-slagen in de code.	operatie BRP
19-11-2015	1.2	Onderzoek toegevoegd, sleutels opnieuw beschreven	operatie BRP
25-11-2015	1.3	Review commentaar verwerkt	operatie BRP
26-01-2016	1.4	Versie en referenties aangepast	operatie BRP
11-01-2017	1.5	Referentie LO3 aangepast en code wijzigingen doorgevoerd in document	operatie BRP

Reviewhistorie

Datum	Versie	Omschrijving	Reviewers
14-07-2015	0.1		operatie BRP
15-07-2015	0.1		operatie BRP
24-11-2015	1.2		operatie BRP

1 Inleiding

1.1 Beknopte omschrijving

Dit technisch ontwerp beschrijft de deltabepaling software, welke zorgt voor het bepalen en toepassen van verschillen tussen twee versies van dezelfde persoonslijst. Deze software is onderdeel van de Data Access Layer en wordt aangesproken nadat het binnenkomende LO3 persoonslijst is geconverteerd naar een BRP persoonslijst. Als de persoonslijst nog niet in de BRP voorkomt, dan wordt deze niet door de deltabepaling software beschouwd.

1.2 Referenties

#	Document	Organisatie	Versie	Datum
[LO-GBA]	Logisch Ontwerp GBA		3.10	08-10-2016
[LO-BRP]	Logisch Ontwerp BRP			
[SAD]	Software Architectuur Document Migratievoorziening BRP			
[CONV]	Documentatie Bidirectionele Conversie			
[FO-DELTA]	Documentatie Deltabepaling			
[SYNC]	Documentatie Technisch Ontwerp BRP Synchronisatie Service			

1.3 Leeswijzer

Dit document is bedoeld voor ontwikkelaars of software architecten die kennis nodig hebben over de interne structuur van de deltabepaling software.
Benodigde voorkennis is een goede bekendheid met de Documentatie Bidirectionele Conversie [CONV]. Tevens is goede kennis van het technisch ontwerp BRP Synchronisatie Service nodig, zoals beschreven in [SYNC]. Ten slotte is kennis van de architectuur van de migratievoorziening wenselijk om de deltabepaling in de juiste context te plaatsen [SAD].

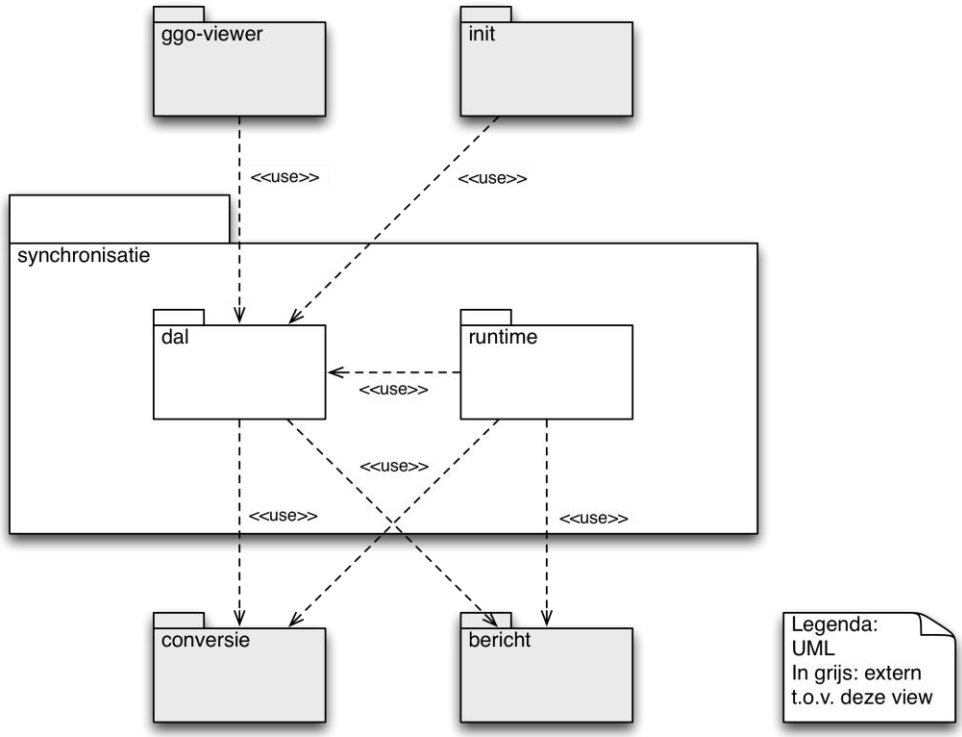
1.4 Indeling

Na de inleidende hoofdstukken begint dit document met uitleg over de onderdelen die gebruikt worden in het hoofdstuk ‘Componenten’, gevolgd door de beschrijving van hoe de verschillen worden bepaald en verwerkt in het hoofdstuk ‘Flow van Deltabepaling’.
Dit document sluit af met een overzicht van de ‘Ontwerpbeslissingen’ en een aantal opmerkingen over de ‘Test Hulpmiddelen’ die worden gebruikt.

2 Context

De deltabepaling software is geen op zichzelf staande bibliotheek, maar is ondergebracht in de 'Data Access Layer (DAL)' module van [SYNC]. Vanwege de complexiteit en specifieke doel van deze software is het gerechtvaardigd om een eigen technisch ontwerp te hebben.

In de onderstaande afbeelding zijn ter illustratie de verschillende modules van [SYNC] weergegeven, in de paragraaf 'View-deel 4: BRP Synchronisatie Service' van de sectie 'Logical View' van het [SAD] kan meer detailinformatie hieromtrent worden gevonden.



Afbeelding 1 BRP Synchronisatie Service in context

Dit document richt zich alleen op de zaken die voor de deltabepaling software nodig zijn.

2.1 Maven Structuur

De deltabepaling software is bij de 'Data Access Layer' van [SYNC] onder gebracht en valt daardoor ook binnen dat Maven project.

Project	Subproject	Beschrijving
migr-sync	migr-synchronisatie-dal	Bevat alle functionaliteit die BRP database intensief is.

3 Componenten

Dit hoofdstuk beschrijft de componenten van de deltabepaling software.

3.1 Interfaces op bestaande entiteiten

Om deltabepaling mogelijk te maken is een aantal entiteiten uitgebreid met nieuwe interfaces.

3.1.1 AdministratiefGegeven

Het doel van deze interface is om de entiteiten aan te merken die behoren tot de administratieve gegevens in BRP. Dit zijn bijvoorbeeld *Actie* en *Document*. Deze interface is een marker-interface.

3.1.2 ALaagHistorieVerzameling

Het doel van deze interface is om de historie bij A-laag entiteiten (zoals bv *PersoonAdres*) te verzamelen zodat hier bewerkingen op gedaan kan worden. Bijvoorbeeld als het A-laag entiteit gemarkeerd is als te verwijderen, dan worden de bijbehorende historie rijen veranderd in M-rijen.

3.1.3 Entiteit

Het doel van deze interface is om alle entiteiten aan te merken als entiteit en is toegevoegd tbv onderzoek. Deze interface bevat methodes die het mogelijk maken een specifiek gegeven in de entiteit in onderzoek te plaatsen, te bevragen welke gegevens in onderzoek staan of om het onderzoek te verwijderen.

3.1.4 RootEntiteit

Het doel van deze interface is om de entiteiten aan te merken die met elkaar, op het hoogste niveau, vergeleken kunnen worden. Deze interface is een marker-interface en is een extensie op de hiervoor beschreven *ALaagHistorieVerzameling* interface.

3.1.5 SubRootEntiteit

Het doel van deze interface is om de entiteiten aan te merken die een A-laag entiteit zijn, maar niet als *RootEntiteit* aangemerkt kunnen worden. Bijvoorbeeld *PersoonAdres* is een A-laag entiteit, maar heeft *Persoon* boven zich als eigenaar. In dit voorbeeld is *Persoon* een *RootEntiteit* en *PersoonAdres* is een *SubRootEntiteit*. De interface bevat één methode om de *Persoon* entiteit op te vragen.

3.2 Decorators voor bestaande entiteiten

Voor een aantal entiteiten is er een decorator gemaakt. Deze decorators bieden extra functionaliteit die alleen nodig zijn bij het bepalen en verwerken van verschillen, waardoor de entiteiten zo compact mogelijk blijven. De decorators zijn met name nodig voor de IST- en relatiecomponenten van de persoonslijst en zijn te vinden in het package *nl.bzk.migratiebrp.synchronisatie.dal.service.impl.delta.decorators*.

3.3 Processen

De processen zijn de rode draad van deltabepaling. Deze processen hebben een gezamenlijke interface *DeltaProces* die twee methodes bevat om de verschillen te bepalen en te verwerken. Het *SamengesteldDeltaProces* is een *composite-pattern* waarmee alle processen aangestuurd worden. Het bepalen en verwerken van verschillen op een persoonslijst is opgedeeld in een aantal componenten, voor elk van deze componenten is een proces gemaakt.

3.3.1 DeltaRootEntiteitenProces

In dit proces worden alle entiteiten vergeleken die de *RootEntiteit* interface implementeert. Voor de *Persoon* entiteit is er altijd maar één instantie per persoonslijst, maar voor de IST- en relatiecomponenten moet er eerst een match gevonden worden. Deze matching wordt beschreven in hoofdstuk 3.1 van [FO-DELTA].
Bij het verwerken van de verschillen worden de verschillen op de IST-gegevens eerst verzameld en los van de andere verschillen verwerkt. Vanwege de gekozen technische sleutel voor de IST-gegevens, hebben deze verschillen invloed op elkaar en moeten deze dus allemaal in een keer verwerkt worden.

3.3.2 LoggingDeltaProces

In dit proces wordt de verantwoording conversie (verconv) verwerkt bij de persoonslijst. Hier wordt geen verschil bepaald, maar wordt de bestaande verantwoording van de persoonslijst verwijderd en de nieuwe verantwoording, die tijdens conversie wordt opgebouwd, gekoppeld aan de bestaande persoonslijst.

3.3.3 OnderzoekDeltaProces

In dit proces worden de onderzoeken verwerkt bij de persoonslijst. In dit proces wordt er geen verschil bepaald. Er kan alleen worden bepaald of een bestaand onderzoek verplaatst moet worden als de historie rij, welke in onderzoek staat, een M-rij wordt. Dit vindt plaats tijdens het *DeltaRootEntiteitenProces*.
Bij de verwerking wordt er gecontroleerd of het onderzoek toegevoegd, verwijderd of aangepast is. Dit wordt gedaan door de bestaande en nieuw gegenereerde onderzoek te matchen op basis van de *GegevenInOnderzoek* entiteiten. Als het aantal *GegevenInOnderzoek* is aangepast, dan wordt het bestaande onderzoek als verwijderd beschouwd en wordt het nieuw gegenereerde onderzoek als nieuw aangekoppeld. Een onderzoek is aangepast als de gegevens op het *Onderzoek* entiteit zijn aangepast.

3.3.4 AfgeleidAdministratiefDeltaProces

In dit proces worden geen verschillen bepaald of verwerkt, omdat er geen verschillen te bepalen zijn. Bij elke conversie wordt er maar één voorkomen afgeleid administratief aangemaakt en deze zal aan de bestaande persoonslijst worden toegevoegd.

3.3.5 ActieConsolidatieProces

Het doel van dit proces is om de BRP stapels die gevuld worden uit één en hetzelfde LO3 voorkomens, gelijk te houden qua *BRPActie* entiteiten. Dit wordt gedaan door tijdens het vergelijken van de verschillende *RootEntiteit* entiteiten een *ActieConsolidatieData* object te maken waarin wordt bijgehouden

- welke nieuwe acties aan bestaande acties zijn gekoppeld
- welke acties zijn gekoppeld aan welke historie

Tijdens dit proces wordt gekeken of er nieuwe acties naar meer dan één andere actie verwijst. Als dit zo is, dan is het mogelijk dat er meerdere acties uit hetzelfde LO3 voorkomen zijn ontstaan, wat niet is toegestaan volgens het [CONV]. Alle historie die bij de gevonden actie zijn geregistreerd, worden aangemerkt als M-rij en de nieuw geconverteerde gegevens worden als nieuw verschil toegevoegd waardoor de historie in feite wordt geresynced. Tijdens het aanmerken van de historie, wordt er gekeken naar welke acties deze historie nog meer bevat. Zo wordt er een keten gebouwd van BRP-historie wat geresynced moet gaan worden.

3.4 Vergelijkers

Waar de processen de rode draad zijn voor deltabepaling, zijn de vergelijkers het hart. De vergelijkers hebben een gezamenlijke interface *DeltaVergelijker* die één methode heeft waarbij twee entiteiten vergeleken worden. Deze methode geeft zijn resultaat object terug die de *VergelijkerResultaat* interface implementeert.

3.4.1 DeltaRootEntiteitVergelijker

Deze vergelijker vergelijkt de entiteiten die de interface *RootEntiteit* hebben geïmplementeerd. Het vergelijken van deze entiteiten vindt plaats d.m.v. reflection en gebruikt de attributen van de entiteit. Om te voorkomen dat de vergelijker niet in een oneindige loop terecht komt, zijn er beperkingen ingebouwd die er voor zorgen dat dit niet gebeurd. Deze beperkingen worden per attribuut toegepast. Als het attribuut een collectie betreft, dan wordt er naar de inhoud van deze collectie gekeken of de collectie overgeslagen kan worden.
Een attribuut wordt overgeslagen als:

- Het attribuut geannoteerd is met @Id (een database ID-veld)
- Het attribuut is de eigenaar van deze klasse. De eigenaar van een klasse is een klasse die hiërarchisch hoger staat. Bijvoorbeeld Persoon is een DeltaRootEntiteit en heeft geen eigenaar. De entiteit PersoonReisdocument heeft de eigenaar Persoon als eigenaar.
- De entiteit en het bijbehorende attribuut in de Tabel 1 - Overgeslagen attributen per entiteit voorkomt

Entiteit	Overgeslagen attributen (type entiteit)
Persoon	Betrokkenheid, Relatie, Lo3Bericht, Lo3Voorkomen, PersoonOnderzoek, Stapel, PersoonAfgeleidAdministratief
Betrokkenheid	Persoon, BetrokkenheidHistorie, Relatie
Relatie	Betrokkenheid, Stapel

Tabel 1 - Overgeslagen attributen per entiteit

3.4.2 IstStapelVergelijker

Deze vergelijker vergelijkt de *Stapel* entiteiten met elkaar nadat deze gematched zijn met de *IstStapelEnRelatieMatcher*. De entiteiten worden met elkaar vergeleken op alle attributen op *Persoon* na. Dit attribuut is de eigenaar van de *Stapel* entiteit.

3.4.3 IstStapelEnRelatieMatcher

Deze klasse is geen vergelijker, maar wel nodig om vergelijkingen te kunnen doen. Om de relaties van een persoonslijst goed te kunnen vergelijken, is het nodig om deze eerst te matchen. De entiteiten die met relaties te maken hebben, hebben echter geen identificerende gegevens behalve de IST-stapel. Deze matching wordt beschreven in hoofstuk 3.1 van [FO-DELTA]. Het resultaat van deze klasse is een collectie van *DeltaRootEntiteitMatch* objecten. Deze objecten worden vervolgens gebruikt om de vergelijkers te voeden.

3.4.4 DeltaVergelijkerResultaat

Deze klasse bevat de verschillen voor een *RootEntiteit* die zijn gevonden tijdens het vergelijken. Ook bevat deze klasse meerdere mappings die later in het proces nodig zijn voor bijvoorbeeld het opnieuw aankoppelen van de verantwoording conversie aan de acties. Bij het toevoegen van verschillen worden deze gelijk gegroepeerd per historie. Als het geen verschil op een historie rij betreft maar op een A-laag entiteit, dan wordt dit als apart verschilgroep gezien.

3.4.5 DeltaRootEntiteitMatch

Deze klasse bevat de bestaande en nieuwe *RootEntiteit* objecten die met elkaar gematched zijn. Deze klasse wordt vervolgens gebruikt om de verschillen tussen de *RootEntiteit* objecten te vinden.

3.5 Transformeerders

De transformeerders hebben als taak om de gevonden verschillen per BRP voorkomen te analyseren en verschillen toe te voegen of te verwijderen waar dit nodig is. Een voorbeeld hier van is dat het verwijderen van gegevens niet is toegestaan. Dit verschil zal dan uit de lijst van verschillen worden verwijderd en er worden verschillen toegevoegd zodat deze gegevens worden getransformeerd naar zogenoemde M-rijen.
Alle transformeerders implementeren de interface *Transformatie* die drie methodes heeft

- accept – Geeft aan of de transformeerder de verschillen kan transformeren of niet
- execute – Afhankelijk van de transformeerder worden hier verschillen toegevoegd of verwijderd.
- getDeltaWijziging – Geeft de naam van de transformeerder terug aan het proces zodat de gebruikte transformeerders gelogged kunnen worden. Dit wordt ook later in het proces gebruikt tijdens het bepalen of de bijhouding actueel of overig is (zie H0).

De transformeerders zijn beschreven in hoofdstuk 3.3 van [FO-DELTA], behalve TransformatieDw003 en TransformatieDw901. Deze transformeerders zijn ontstaan vanuit technische redenen en worden in de volgende paragrafen beschreven.

De klasse *Transformeerder* bevat een *factory-pattern* waarmee een instantie van deze klasse wordt terug gegeven waarmee de verschillen per BRP voorkomen worden verwerkt. Het resultaat van het verwerken wordt, voordat het terug gegeven wordt, aangepast door bepaalde verschillen uit de lijst te filteren.

- Alle verschillen per BRP voorkomen die alleen getransformeerd zijn door *TransformatieDw901*
- Alle verschillen die behoren bij een BRP voorkomen van de groep Bijhouding als de verschillen alleen zijn getransformeerd door *TransformatieDw023* en *TransformatieDw901*
- Alleen die verschillen die door *TransformatieDw023* zijn getransformeerd die behoren bij een BRP voorkomen van de groep Bijhouding waarbij de verschillen zijn getransformeerd door *TransformatieDw021, TransformatieDw023* en *TransformatieDw002-ACT*

3.5.1 TransformatieDw003

Deze transformeerder is het opvangnet binnen de transformeerders. Als de verschillen niet door een andere transformeerder worden geaccepteerd, dan zal deze transformeerder de verschillen wel accepteren. Deze verschillen worden vervolgens uit de lijst van verschillen gehaald en worden er verschillen toegevoegd om het bestaande voorkomen naar een M-rij te transformeren en het nieuw geconverteerde voorkomen als nieuw gegeven aan de persoonslijst te koppelen.

3.5.2 TransformatieDw901

Deze transformeerder accepteert de verschillen die ontstaan zijn doordat alleen de tijdstip registratie en de bijbehorende actie inhoud is aangepast. Dit is het geval voor de BRP voorkomens die zijn ontstaan na conversie waarbij tijdstip registratie wordt gevuld vanuit de datum/tijdstempel uit categorie 07 (Inschrijving) van LO3.
Als alleen deze verschillen zijn geconstateerd binnen een BRP voorkomen, dan worden deze verschillen genegeerd en daarmee uit de lijst van verschillen gehaald.

3.6 Verwerkers

De verwerkers passen de gevonden en getransformeerde verschillen toe op de entiteiten en hebben een gezamenlijke interface *DeltaVerschilVerwerker* die één methode heeft waarmee de verschillen worden toegepast.

3.6.1 DeltaRootEntiteitVerschilVerwerker

Past de verschillen toe op de entiteiten die de interface *RootEntiteit* hebben geïmplementeerd. Het toepassen op deze entiteiten vindt plaats d.m.v. reflection en gebruikt de attributen van de entiteit. Om te voorkomen dat de verwerker niet in een oneindige loop terecht komt, zijn er beperkingen ingebouwd die er voor zorgen dat dit niet gebeurd. Dit zijn dezelfde beperkingen als bij de *DeltaRootEntiteitVergelijker* (H3.4.1), maar om bijvoorbeeld een nieuwe *Betrokkenheid* toe te kunnen voegen aan *Persoon* worden een sommige attributen bij een entiteit niet overgeslagen bij het verwerken. De attributen die niet overgeslagen worden zijn genoemd in Tabel 2 – Attributen die wel verwerkt maar niet vergeleken worden.

Entiteit	**Overgeslagen attributen (type entiteit)**
Persoon	Betrokkenheid
Betrokkenheid	Persoon, BetrokkenheidHistorie
Relatie	Betrokkenheid

Tabel 2 – Attributen die wel verwerkt maar niet vergeleken worden

3.6.2 IstStapelVerschilVerwerker

Deze verwerker verwerkt de verschillen voor de IST-gegevens. De verwerker verwerkt deze verschillen in een specifieke volgorde vanwege de technische beperkingen die opgelegd zijn vanuit de database sleutels. IST-stapels en voorkomens worden geïdentificeerd door hun categorie, stapelnummer en voorkomennummer. Bij wijzigingen in de LO3-persoonslijst kunnen zowel de stapels als de voorkomens onderling van positie wisselen. Bij dit soort gebeurtenissen kunnen we niet de oude gegevens verwijderen en de nieuwe opslaan, aangezien er in Hibernate dan een exceptie optreedt i.v.m. het verwijderen en toevoegen van gegevens met dezelfde technische sleutels.

Om deze reden worden de IST-gegevens onderling technisch verplaatst. Dit vraagt om een bepaalde volgorde van verwerking zodat er niet ongevraagd gegevens overschreven worden.

3.6.3 LoggingVerwerker

Deze verwerker implementeert niet de *DeltaVerschilVerwerker* interface en wijkt ook af in het feit dat deze geen verschillen verwerkt. Deze verwerker koppelt die nieuw verantwoording conversie aan de bestaande acties die niet aangepast zijn. Tijdens het bepalen van verschillen met de vergelijkers (zie H3.4) wordt er een map aangelegd waarbij de bestaande acties, mits deze niet gewijzigd zijn,met de nieuwe verantwoording conversie wordt vast gelegd. Deze map wordt gebruikt in deze verwerker om deze mapping vast te leggen op de persoonslijst.

3.7 Sleutels

Om goed aan te kunnen geven waar er verschillen zijn geconstateerd, zijn er *Sleutel*-klasses geïmplementeerd. Deze klasses implementeren de *Sleutel* interface.

3.7.1 EntiteitSleutel

Deze sleutel is de algemeen gebruikte sleutel binnen de Deltabepaling en bestaat uit de volgende elementen

- Entiteit - De klasse van de entiteit waar de sleutel op van toepassing is
- Veld - De naam van de attribuut binnen de entiteit
- EigenaarSleutel – de sleutel waarmee de eigenaar van entiteit wordt aangeduid
- Delen – een key/value lijst met daarin delen waardoor de sleutel uniek wordt gemaakt

Deze sleutel is zo ontworpen dat wanneer een verschil in een entiteit binnen een collectie wordt gevonden, het gegeven dat de rij uniek aanwijst als onderdeel van de sleutel wordt op genomen. In onderstaande voorbeelden wordt dit verder toegelicht. De voorbeelden zijn niet volledig voor de wijzigingen die plaats vinden (verschillen voor o.a. het aanmaken van M-rijen worden niet beschreven), maar richten zich alleen specifiek de sleutel die van toepassing is.

Voorbeeld 1: Een wijziging op de *Persoon*

Het versienummer van de LO3 persoonslijst is aangepast. Deze wijziging heeft betrekking op het versienummer veld op de Persoon entiteit zelf en op de PersoonInschrijvingHistorie.

```
EntiteitSleutel[Entiteit=class
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.Persoon,Veld=versienu
mmer,EigenaarSleutel=<null>]
```

Bovenstaande sleutel wordt aangemaakt voor de *Persoon* entiteit waar het veldnaam "versienummer" wordt ingevuld. Aangezien de *Persoon* entiteit geen eigenaar heeft, wordt de EigenaarSleutel niet ingevuld.

Naast deze sleutel zullen er ook sleutels gegenereerd zijn om

- een nieuwe *PersoonInschrijvingHistorie* rij toe te voegen
- de bestaande *PersoonInschrijvingHistorie* rij te converteren naar een M-rij.

Voorbeeld 2: Een wijziging op een historie onder een A-laag entiteit van de *Persoon*
Bij de nationaliteit wordt een reden verlies toegevoegd. Deze wijziging heeft betrekking op het redenVerliesNLNationaliteit veld van de bestaande PersoonNationaliteit en de bestaande PersoonNationaliteitHistorie.

```
EntiteitSleutel[Entiteit=class
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteitH
istorie,Veld=redenVerliesNLNationaliteit,EigenaarSleutel=EntiteitSleutel[Entiteit=
class
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteit,
Veld=persoonNationaliteitHistorieSet,EigenaarSleutel=EntiteitSleutel[Entiteit=clas
s
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.Persoon,Veld=persoonN
ationaliteitSet,EigenaarSleutel=<null>,nation=2],dataanvgel=19761030,tsreg=1976-
10-31 02:00:00.0]]
```

Bovenstaande sleutel wordt aangemaakt voor de historie van de nationaliteit waarbij het veld *redenVerliesNLNationaliteit* wordt ingevuld. Aangezien een nationaliteit meerdere historie rijen kan hebben wordt deze uniek gemaakt door het toevoegen van *datum aanvang geldigheid(dataanvgel)* en de *tijdstip registratie(tsreg)*. De historie op deze manier uniek maken is niet voldoende, omdat een persoon meerdere nationaliteiten kan hebben en deze nationaliteiten dezelfde historie kunnen bevatten. Om de historie rij toch specifiek aan te kunnen wijzen, wordt er aan de sleutel waarmee de juiste nationaliteit wordt aangemerkt, de *nationaliteitcode(nation)* toegevoegd.

Deze sleutel kan dan als volgt gelezen worden vanaf de *Persoon* entiteit middels de vet gedrukte stukken:

```
EntiteitSleutel[Entiteit=class
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteitH
istorie,Veld=redenVerliesNLNationaliteit,EigenaarSleutel=EntiteitSleutel[Entiteit=
class
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteit,
Veld=persoonNationaliteitHistorieSet,EigenaarSleutel=EntiteitSleutel[Entiteit=clas
s
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.Persoon,Veld=persoonN
ationaliteitSet,EigenaarSleutel=<null>,nation=2],dataanvgel=19761030,tsreg=1976-
10-31 02:00:00.0]]
```

Vanaf de *Persoon* entiteit wordt de juiste nationaliteit aangewezen in het veld *persoonNationaliteitSet* waarbij naar de nationaliteit wordt gezocht met code “2”.

```
EntiteitSleutel[Entiteit=class
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteitH
istorie,Veld=redenVerliesNLNationaliteit,EigenaarSleutel=EntiteitSleutel[Entiteit=
class
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteit,
Veld=persoonNationaliteitHistorieSet,EigenaarSleutel=EntiteitSleutel[Entiteit=clas
s
nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.Persoon,Veld=persoonN
ationaliteitSet,EigenaarSleutel=<null>,nation=2],dataanvgel=19761030,tsreg=1976-
10-31 02:00:00.0]]
```

Nadat de juiste nationaliteit is gevonden, wordt de juiste historie rij gezocht met de kenmerken die de historie rij uniek maakt.

EntiteitSleutel[Entiteit=class nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteitHistorie,Veld=redenVerliesNLNationaliteit,EigenaarSleutel=EntiteitSleutel[Entiteit=class nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.PersoonNationaliteit,Veld=persoonNationaliteitHistorieSet,EigenaarSleutel=EntiteitSleutel[Entiteit=class nl.bzk.migratiebrp.synchronisatie.dal.domein.brp.kern.entity.Persoon,Veld=persoonNationaliteitSet,EigenaarSleutel=<null>,nation=2],dataanvgel=19761030,tsreg=1976-10-31 02:00:00.0]]

He veld "redenVerliesNLNationaliteit" van de gevonden historie rij is het veld waar een verschil is gedetecteerd.

Naast deze sleutel zullen er ook sleutels gegenereerd zijn om

- een nieuwe *PersoonNationaliteitHistorie* rij toe te voegen
- de bestaande *PersoonNationaliteitHistorie* rij te converteren naar een M-rij.
- Het veld "redenVerliesNLNationaliteit"op de *PersoonNationaliteit* in te vullen

3.7.2 IstSleutel

Deze sleutel is specifiek gemaakt voor IST-stapels en wijkt qua aanwijzing waar het verschil geconstateerd is, af van de *EntiteitSleutel*. Voor deze sleutel wordt niet naar de entiteit gekeken, maar specifiek naar *Stapel* of *StapelVoorkomen*. De *ISTSleutel* bestaat uit de volgende velden

- Categorie – categorienummer van de stapel
- Stapelnummer – volgnummer van de stapel zoals gevonden op de LO3 persoonslijst. Het eerst aangetroffen stapel heeft volgnummer 0.
- isBestaandeStapel – Geeft aan of de stapel al bestaat of nieuw is
- veld – veld binnen de stapel waar het verschil gevonden is
- delen – een key/value lijst met daarin delen waardoor de sleutel uniek wordt gemaakt. Dit wordt alleen gebruikt om de relaties binnen één IST-stapel uit elkaar te houden.
- voorkomennummer – het volgnummer van het voorkomen binnen de stapel. Het actuele voorkomen heeft als nummer 0.

Voorbeeld 1: Wijziging m.b.t. een stapel
Er wordt een nieuw kind geregistreerd bij de persoonslijst.

```
IstSleutel[categorie=09,stapel=0,voorkomen=<null>,veld=<null>,isBestaandeStapel=fa
lse,delen={}]
```

Voor deze wijziging wordt er een verschil op de IST-stapels geconstateerd dat er een nieuwe stapel is bijgekomen. De sleutel die hiervoor gemaakt wordt, is een sleutel voor een **stapel**. Deze bevat de *categorie* (09), *stapelnummer* (0) en omdat deze stapel nieuw is, wordt *isBestaandeStapel* aangeduid met *false*. *Voorkomen*, *veld* en *delen* blijven leeg, aangezien deze niet nodig zijn voor dit verschil.

Voorbeeld 2: Wijziging m.b.t. een stapelvoorkomen
Een ouder wordt bijgewerkt zodat de ouder van een "punt-ouder" (onbekende ouder), een bekende ouder wordt.

```
IstSleutel[categorie=02,stapel=0,voorkomen=0,veld=stapelVoorkomens,isBestaandeStap
el=false,delen={}]
```

Voor deze wijziging wordt er een verschil binnen een IST-stapel geconstateerd, namelijk dat er een voorkomen is bijgekomen bij één van de ouder categorieën . De sleutel die hiervoor gemaakt wordt, is een sleutel voor een **stapelvoorkomen**. Deze bevat de *categorie* (02), *stapelnummer* (0), *voorkomen* (0) en omdat dit voorkomen bij de nieuw stapel is geconstateerd, wordt *isBestaandeStapel* aangeduid met **false**. *Veld* geeft aan dat deze wijziging plaats heeft gevonden in de *stapelVoorkomens* van een IST-stapel.

Voorbeeld 3: Wijziging m.b.t. meerdere relaties aan één stapel
Een huwelijk wordt omgezet naar een geregistreerd partnerschap.

```
IstSleutel[categorie=05,stapel=0,voorkomen=<null>,veld=relaties,isBestaandeStapel=
true,delen={datumAanvang=19920808, srt=GEREGISTREERD_PARTNERSCHAP}]
```

4 Bij deze wijziging ontstaat er in de BRP een nieuwe relatie. De bestaande IST-stapel krijgt er dus een relatie bij. Om deze extra relatie te onderscheiden in de sleutel, worden er extra gegevens (delen) opgenomen waardoor de sleutel uniek wordt. Ook is deze sleutel voor een bestaande IST-stapel, dus *isBestaandeStapel* is gevuld met **true**.Flow van Deltabepaling

Dit hoofdstuk beschrijft de stappen die gedaan worden om alle verschillen tussen de bestaande en de aangeboden persoonslijst te vinden en te verwerken m.b.v. de componenten uit het vorige hoofdstuk. Ook zal hier beschreven worden of het bijwerken van de persoonslijst een **bijhouding actueel, infrastructurele wijziging, afvoeren PL** of een **bijhouding overig** is.

4.1 Initialisatie deltabepaling

De verwerking door deltabepaling wordt gestart door een nieuw *SamengesteldDeltaProces* te maken door de methode *newInstanceMetAlleProcessen* op deze klasse aan te roepen. Deze methode geeft een instantie van de eigen klasse terug waar vervolgens de aanroepen worden gedaan voor *bepaalVerschillen* en *verwerkVerschillen*. De processen die hiermee (in volgorde) aangestuurd worden zijn

- OnderzoekDeltaProces
- DeltaRootEntiteitenProces
- ActieConsolidatieProcesLoggingDeltaProces
- AfgeleidAdministratiefDeltaProces

Als er geen verschillen op de persoonslijst zijn bepaald, dan worden alleen de volgende processen uitgevoerd

- OnderzoekDeltaProces
- LoggingDeltaProces
- AfgeleidAdministratiefDeltaProces

Deze processen zijn nodig om de eventuele verschillen op onderzoek te verwerken en de nieuwe administratieve handeling en verantwoording conversie aan de persoonslijst te koppelen.

4.2 Bepaal verschillen (bepaalVerschillen-methode)

4.2.1 OnderzoekDeltaProces

Deze methode is in dit proces niet geïmplementeerd.

4.2.2 DeltaRootEntiteitenProces

In dit proces worden de verschillen gedetecteerd die er zijn tussen de bestaande en de aangeboden persoonslijst. Zoals beschreven in hoofdstuk 3.3.1 wordt er eerst de matches gezocht voor de IST/relatie gegevens. Nadat de bestaande en nieuwe IST/relatie gegevens aan elkaar zijn gekoppeld, wordt de rest van de beide persoonslijsten aan elkaar gekoppeld.
Alle matches worden vervolgens één voor één door de DeltaRootEntiteitVergelijker vergeleken behalve voor de IST-matches. Deze wordt door de IstStapelVergelijker vergeleken.

Nadat de verschillen zijn gedetecteerd, worden deze, behalve de IST verschillen, indien nodig, getransformeerd. Het resultaat van dit proces is een collectie met alle gevonden entiteit matches (*DeltaRootEntiteitMatch)*. De gevonden verschillen zijn opgeslagen in de match entiteit.

Bepalen soort bijhouding

De keuze of de bijhouding **actueel** of een **overig** wordt, wordt bepaald door welke transformeerders er per BRP voorkomen zijn geraakt. Dit is beschreven in [FO-DELTA].
Als de bijhouding **actueel** is en het betreft een infrastructurele wijziging (zie [FO-DELTA]), dan wordt de bijhouding **infrastructurele wijziging**. Als de bijhouding **actueel** is en het betreft het afvoeren van de persoonslijst (zie [FO-DELTA]), dan wordt de bijhouding **afvoeren PL**.

4.2.3 ActieConsolidatieProces

In dit proces wordt er gekeken of er door de gevonden verschillen geen verschillende acties uit hetzelfde LO3 voorkomen worden gemaakt. Indien dit wel het geval is, worden de stapels die hier voor verantwoordelijk zijn, vervangen door de nieuw geconverteerde stapels.

Bepalen soort bijhouding

Als er één of meerdere stapels vervangen gaan worden, dan wordt de bijhouding **overig**.

4.2.4 LoggingDeltaProces

In dit proces worden er geen verschillen gedetecteerd, maar wordt de bestaande verantwoording conversie verwijderd bij de persoonslijst.

Bepalen soort bijhouding

Dit proces heeft geen invloed op de keuze **actueel** of **overig**.

4.2.5 AfgeleidAdministratiefProces

Deze methode is in dit proces niet geïmplementeerd.

4.3 Verwerk verschillen (verwerkVerschillen-methode)

4.3.1 OnderzoekDeltaProces

In dit proces worden de bestaande onderzoek en de nieuw gegenereerde onderzoeken met elkaar gematched op basis van de *GegevenInOnderzoek* entiteiten. Tijdens het bepalen van de verschillen wordt een mapping opgebouwd tussen de bestaande en nieuwe entiteiten. Deze mapping wordt gebruikt om te bepalen of het *GegevenInOnderzoek* nog steeds naar dezelfde entiteit wijst. Als het aantal nieuwe *GegevenInOnderzoek* entiteiten niet overeenkomt met het aantal bestaande, dan is het onderzoek per definitie niet gelijk. Uit deze matching kunnen 4 situaties ontstaan

- Onderzoek is nieuw
- Onderzoek is verwijderd
- Onderzoek is gematched en bevat verschillen
- Onderzoek is gematched maar bevat geen verschillen

Nadat de matching is bepaald, worden bovenstaande situaties toegepast op de persoonslijst.

Bepalen soort bijhouding
Het soort bijhouding is **actueel** als het voorkomen dat in onderzoek staat in BRP

- Niet behoort tot de entiteiten die aangemerkt is met de *AdministratieGegeven* interface.
- Een actueel historisch voorkomen is
- Een A-laag entiteit is

4.3.2 DeltaRootEntiteitenProces

In dit proces worden de verschillen die gevonden zijn per *DeltaRootEntiteitMatch* verwerkt op de bestaande persoonslijst m.b.v DeltaRootEntiteitVerschilVerwerker. Voordat de verschillen per match worden verwerkt, worden eerst de verschillen voor de IST-gegevens verzameld en verwerkt door IstStapelVerschilVerwerker.

4.3.3 ActieConsolidatieProces

4.3.4 Dit proces heeft geen implementatie van deze methode.LoggingDeltaProces

In dit proces wordt de verantwoording conversie die opnieuw bepaald is tijdens conversie, aan de bestaande persoon aangekoppeld m.b.v LoggingVerwerker.

4.3.5 AfgeleidAdministratiefProces

In dit proces wordt de BRP groep *AfgeleidAdministratief* bijgewerkt. In deze groep is terug te vinden welke administratieve handelingen zijn toegepast op de persoonslijst.

5 Ontwerpbeslissingen

5.1 Reflectie voor vergelijker/verwerken

Er is gekozen voor het gebruik van reflectie bij het vergelijken en verwerken omdat de persoonslijst een grote hiërarchische lijst is. Dit kan uitgeprogrammeerd worden, maar dit levert een veel code op, waarin onderdelen vergeten kunnen worden. Ook is met reflectie de mogelijkheid er om de persoonslijst uit te breiden, zonder dat er stil gestaan hoeft te worden bij de deltabepaling.
Nadeel van de reflectie is dat het langzaam is en dat alle entiteiten die uit de database komen, eerst van hun *HibernateProxy* klasse ontdaan worden voordat er reflectie kan worden toegepast.

5.2 Sleutels

Om de plaats van het gevonden verschil opnieuw terug te vinden, is er gekozen om sleutels te maken. De sleutels zouden nog aangepast kunnen worden zodat je niet alleen van het gevonden verschil terug kan naar het *RootEntiteit*, maar ook van het *RootEntiteit* naar het verschil kan. In dat geval kan namelijk het toepassen van de verschillen op basis van de gevonden verschillen plaats vinden i.p.v. opnieuw de hele persoonslijst met reflectie door te lopen.

5.3 Transformeerders

De transformeerders zijn ontstaan nadat het duidelijk werd dat de historie opbouw m.b.v deltabepaling niet tot de juiste resultaten leidde om vanuit BRP de gegevens te kunnen leveren. Zo mogen er bijvoorbeeld geen database gegevens verwijderd worden, terwijl de gegevens niet meer op de persoonslijst staan. Om deze verschillen goed toe te passen, worden deze verschillen vervangen (getransformeerd) zodat er M-rijen ontstaan. BRP kan hierdoor wel leveren en als de levering plaats heeft gevonden, de gegevens verwijderen. Idealiter waren de transformers niet nodig en worden de gewenste verschillen opgenomen op het moment dat ze gevonden worden, maar omdat de deltabepaling al in een vorm bestond, is er gekozen om de transformeerders toe te passen.

5.4 Bi-directionele relatie tussen Entiteit en GegevenInOnderzoek

In het huidige databasemodel is er wel een koppeling tussen *GegevenInOnderzoek* en de desbetreffende entiteit, maar deze bestaat niet andersom. Om dit toch mogelijk te maken is de interface *Entiteit* gemaakt. Deze maakt het mogelijk dat de bi-directionele relatie tot stand komt. De relatie wordt gelegd op het moment dat het *GegevenInOnderzoek* wordt geladen. Dit wordt gedaan door een methode de *@PostLoad* annotatie heeft.

Als de *Entiteit* opgeslagen wordt, dan wordt de relatie opnieuw gelegd als blijkt dat het *GegevenInOnderzoek* dat bij het entiteit is geregistreerd niet meer terug verwijst naar de opgeslagen entiteit. Dit wordt gedaan dmv een methode die geannoteerd is met @PostPersist en @PostUpdate.

Deze beide methodes zijn geïmplementeerd in de *GegevenInOnderzoekListener* welke d.m.v. de @EntityListeners annotatie aan alle *Entiteit* entiteiten is gekoppeld.

6 Test Hulpmiddelen

6.1 Test Projecten

Voor de deltabepaling software zijn test hulpmiddelen ontwikkeld die zowel door ontwikkelaars als door testers gebruikt worden. Het primaire test project is **migr-test-persoon-database**. Dit project wordt gebruikt voor het inlezen van LO3 persoonslijsten in excel formaat, het converteren van deze persoonslijsten naar BRP en het terug converteren van de BRP persoonslijst naar LO3. Bij alle stappen worden vergelijkingen uitgevoerd om te verifiëren dat het resultaat aan de verwachtingen voldoet, of dat het overeenkomt met de oorspronkelijke gegevens voor de conversie. Om te controleren of de verschillen goed zijn toegepast op de bestaande persoonslijst ondersteund dit project het uitvoeren van SQL-gebaseerde tests op de opgeslagen resultaten van een testgeval.

6.2 Gebruik

De tests in deze projecten kunnen zowel met een in-memory database gedraaid worden, als met een echte PostgreSQL database.
Beide projecten hebben een RegressieTest klasse die een volledige regressie test draait. Ook zijn er extra Test klassen die andere test sets in de projecten gebruiken. In de Test klassen wordt ingesteld of de in-memory database of een externe database gebruikt wordt.
De configuratie van de externe database is te vinden in src/test/non-packaged-resources.
De test klassen kunnen gedraaid worden als een gewone JUnit testklasse.