Standaardisatie van gegevensleveringen

De Polis Papers (8): Nostalgie naar de toekomst

René Veldwijk

Na zeven artikelen over de Polisadministratie hebben we nog maar weinig woorden vuil gemaakt aan datgene waar het in de kern om gaat: het leveren van gegevens over werk en inkomen aan grote aantallen afnemers binnen (en ook buiten) het publieke domein. Dat gemis werken we in dit artikel weg met de bespreking van een mechanisme waarmee UWV afnemers kan bedienen, zonder dat er voor elke nieuwe gegevenslevering weer opnieuw software op de PA moet worden ontwikkeld.

Dit artikel is geschreven met enige gêne. Voor het eerst beschrijven we geen werkende software maar een concept op de tekentafel waarmee we het grootste nog overgebleven probleem van de PA moeten oplossen: een standaard API waarmee gegevensafnemers op maat kunnen worden bediend en die van de kant van UWV geen programmeerinspanning vereist. Om te begrijpen hoe gênant dat wel is gaan we eerst terug in de tijd. UWV bestaat sinds 2002. Daarvoor waren er vijf zogenaamde uitvoeringsinstellingen (UVI's) die premies inden en uitkeringen verzorgden: GAK en vier andere. Die UVI's hadden allemaal een eigen basisregistratie waarin tenminste verzekerde personen, werkgevers en dienstverbanden waren opgeslagen. UWV is de opvolger van de vijf UVI's en de PA is de opvolger van die vijf basisregistraties. Natuurlijk waren de UVI-registraties bedoeld voor intern gebruik en waren ze véél kleiner dan de PA. Maar voor het overige is er niet zoveel veranderd, op één ding na. Alle UVI's hadden geleerd dat het niet slim is om voor elke gegevenslevering nieuwe programmatuur te ontwikkelen en daarom waren de meeste overgegaan tot de ontwikkeling van gestandaardiseerde gegevensleveringen op basis van een stabiele API, waar de afnemers tegenaan konden programmeren. Dat was in veel opzichten slim. Zo kon de gegevensafnemer zelf programmatuur ontwikkelen en konden wijzigingen in de structuur van de basisregistratie vaak transparant voor de afnemer worden doorgevoerd. Toen kwam de initiële PA op de tekentafel en koos men voor een model waarbij elke afnemer wordt bediend met maatwerkprogrammatuur, waardoor elke gegevenslevering niet alleen voor de afnemer maar ook voor UWV een project wordt. Bij de doorstart van de PA eind 2007 troffen we specificaties en ontwerpen of zelfs al gebouwde koppelingen van sterk wisselende kwaliteit aan. Tientallen specifieke gegevensleveringen die aansluiten op de eerdere specificaties zijn sindsdien gerealiseerd, maar ondertussen staat er een rij van afnemers voor de deur die op een dag moeten worden bediend. Dat we weer terug moeten naar het oude UVI levermodel is duidelijk en dat is dit jaar ook besloten door UWV. Helaas zijn de uitdagingen waarvoor wij nu staan veel groter dan die van de UVI's twintig jaar geleden.

Standaard leveringen: legio problemen

Waarom is het zo moeilijk om een overzichtelijk aantal standaard gegevensverstrekkingen op de PA te definiëren waarvan de afnemers verplicht gebruik moeten maken? We zagen al dat de meeste UVI's het konden in de jaren negentig en hetzelfde geldt voor de bevolkingsadministratie (GBA). Dáár hoef je als afnemer niet aan te komen met het verzoek om op andermans kosten maatwerkprogrammatuur te laten ontwikkelen. Er is gewoon een set standaard verstrekkingen waar je gebruik van kunt maken en de ICT-organisatie is er klein want de afnemer moet zelf het werk doen. Welnu, allereerst is er een organisatorisch en bestuurlijk probleem. Het is duidelijk dat afnemers die vandaag op maat worden bediend op kosten van UWV niet juichend zullen overstappen op een GBA levermodel. Als UWV 90 procent minder werk heeft en de afnemer 10 procent meer, dan is er ook geen automatische win-win. En net als gewone mensen vinden ICT'ers het ook niet leuk als het werk verdwijnt. Problemen als deze zijn belangrijk maar in deze serie richten we ons op de ICT-problemen. Ook die zijn enorm.

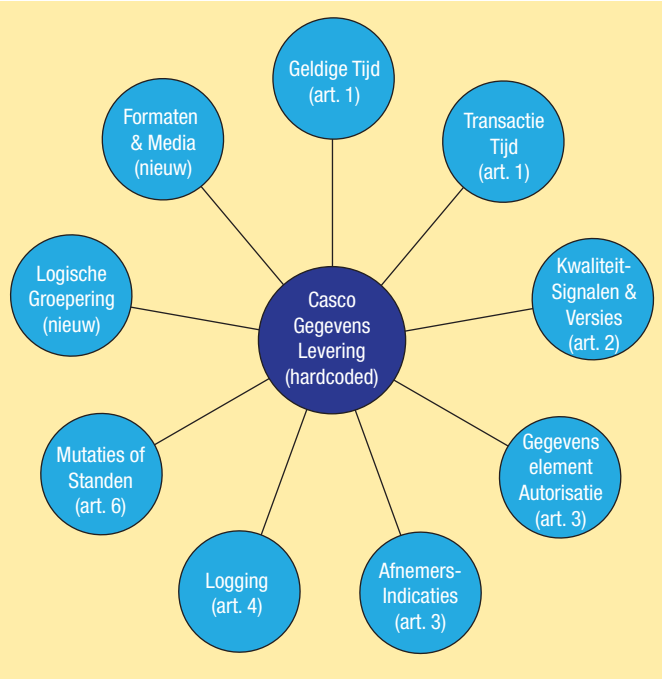
ICT-probleem nummer één is dat de PA zo groot is. Dat maakt efficiënte database access een harde noodzaak. Efficiënte bevraging houdt onder meer in dat er maximaal wordt gewerkt met *compound* vragen. Als we in één vraag gegevens over, zeg, werkgevers, inkomstenverhoudingen en personen willen hebben, dan willen we die in één SQL statement verwerkt zien. Het zou veel eenvoudiger zijn om een aantal 'atomaire' bevragingen te maken op de verschillende basisentiteiten van de PA en het afnemende programma de gegevens aan elkaar te laten knopen. Voor een database met de omvang van de PA is dat een garantie voor performanceproblemen en extra peperdure servercapaciteit. Maar als je elke vraag wilt beantwoorden met één SQL expressie op de PA database dan heb je weer maatwerk. Dat probleem moeten we dus allereerst oplossen.

Probleem twee heeft ook te maken met performance. In artikel 6 lieten we zien dat de optimale leverstrategie al bij relatief kleine bulkleveringen een *full table scan* is. In zo'n situatie maakt het enorm veel uit als je groepen afnemers in één levering kunt bedienen. Als we bijvoorbeeld gegevens moeten verstrekken aan 400 pensioenfondsen (en dat moet!) dan is het een performance nachtmerrie om 400 keer door de database heen te gaan. Je wilt in zo'n geval de gegevens ineens ophalen en pas daarna uitsplitsen. Natuurlijk moeten daar wel even afspraken over worden gemaakt met de pensioenbranche en is het ook fijn als alle pensioenfondsen precies dezelfde gegevens geleverd krijgen. In de sociaal-fiscale wereld is dat helaas nog niet vanzelfsprekend. Probleem drie is dat de situatie op ICT-gebied er sinds de jaren negentig niet eenvoudiger op is geworden. We hebben nu webservices in allerlei smaken. Simpele, beheersbare en performante batchleveringen zijn vaak vervangen door *message-based* oplossingen, gewoon omdat het kan. Er is Internet, maar ook nog steeds FTP, DVD en e-mail. Worden er schijnbaar platte bestanden geleverd dan moeten daar in het XML tijdperk allerlei hiërarchische structuren in worden aangebracht. Soms ook wil de afnemer de gegevens het liefst in een plat, ouderwets CSV-bestandje. Kortom, een hele uitdaging om dit allemaal te ondersteunen, maar voor de opvolger van de succesvolle UVI- en GBA-interfaces een harde eis.

Probleem vier is dat de loonaangifte waarmee de PA wordt gevuld inherent ambigu is. In artikel 1 lieten we zien dat de PA, ontdaan van alle niet essentiële zaken, uit maar vier entiteittypen bestaat: Persoon, Werkgever, Inkomstenverhouding en Inkomstenopgave, plus natuurlijk allerlei referentietabelletjes. Je zou dus kunnen volstaan met een klein aantal standaard leveringen, misschien maar een stuk of tien, waarmee de PA vanuit allerlei standen kan worden bevraagd. Het probleem is echter dat de bedenkers van de loonaangifte het hebben toegestaan dat uw werkgever meldt dat u van 1 januari tot 15 januari 2009 wel verzekerd bent voor de WW en daarna niet meer. De loonopgave is echter over de maand januari 2009. Welke verzekeringstatus hoort nu bij uw loon over januari? De situatie van 1 januari? Die van de 15e of de 31e? Een pro rata loonbedrag wellicht? Alles kan, want er is geen eenduidige interpretatie voorgeschreven. Dat standaardiseert dus weer lekker.

De oplossing: dimensies afpellen

Ondanks alle problemen denken we dat het mogelijk moet zijn om, uitgaande van een overzichtelijk aantal standaard routines, de wereld op maat te bedienen. De basis daarvoor ligt in het leidende architectuurbeginsel van de PA: orthogonaliteit. In bijna alle voorgaande artikelen zagen we hoe we problemen zoals tijd, kwaliteit, autorisatie, logging en performance aanpakten door ze uiteen te rafelen in van elkaar onafhankelijke concepten. Dat idee vormt ook de basis voor een oplossing van het probleem van standaardisatie van gegevensleveringen. Afbeelding 1 visualiseert dat idee.



Afbeelding 1: Dimensies van gegevensleveringen.

In de kern is een standaard levering natuurlijk een stuk programmacode: net zoals bij de verdwenen UVI basisregistraties en de nog bestaande, stokoude GBA. Maar dat stukje programmacode is heel klein en eenvoudig, terwijl de hoeveelheid functionaliteit die wordt geboden, zoals we zullen zien, enorm is. We gebruiken afbeelding 1 als handleiding bij wat we hierna gaan bespreken en we beginnen dan met een stukje code die een levering beschrijft. We nemen als casus de levering van gegevens over inkomens en verzekeringen voor de medewerkers van bedrijven in éénzelfde bedrijvensector. Dat stukje SQL-code zou er gestyleerd uit kunnen zien zoals in afbeelding 2.
Dit stukje code zou niet werken op de PA. Loongegevens zitten bijvoorbeeld niet in een tabel 'Inkomstenverhouding' maar in een tabel 'Inkomstenopgave', maar dat is alleen maar zo omdat er elke maand een nieuwe opgave binnenkomt. Als de PA alleen de laatste opgave zou registreren (lees: als er geen tijdsdimensie in PA zat) dan zou er geen aparte tabel met repeterende inkomstenopgaven bestaan. Maar omdat tijdaspecten in de PA systematisch zijn gemodelleerd (zie artikel 1) en we beschikken over een krachtige datadictionary, zou je uitgaande van het stukje code een uitgebreider SQL statement op de fysieke tabellen kunnen genereren dat *wel* werkt. Wat dan natuurlijk nodig is, is informatie over zowel de geldige tijd als de beschouwingstijd. Zo zouden we voor de geldige tijd het interval 1/1/2008 tot 31/12/2008 kunnen meegeven en voor de transactietijd 1/7/2009 om 0:00 uur. We krijgen dan voor het geselecteerde bedrijf de medewerkers en hun inkomens- en verzekeringsgegevens over het jaar 2008 voor zover bekend op 1 juli 2009. Een andere variant is een geldig tijdstip 1/1/2009 en een transactietijd interval 1/4/2009 om 0:00 uur tot heden. We krijgen dan alle mutaties die betrekking hebben op 1 januari 2009 die vanaf 1 april zijn

binnengekomen. Zo hebben we opeens met één klein programmaatje de mogelijkheid om in vele schijnbaar verschillende informatiebehoeften te voorzien. Natuurlijk valt daar meer over op te schrijven dan de ruimte hier toelaat. Gelukkig is dat al gebeurd in een serie artikelen over Tijd in Databases die u kunt downloaden op de DB/M site. De ideeën die daar zijn uitgewerkt kunnen volledig worden toegepast op de gegevensleveringen van de PA.

Fast forward: kwaliteit, autorisatie ...

Met de generieke ondersteuning van de twee tijdsdimensies hebben we artikel 1 nogmaals behandeld, maar nu vanuit het perspectief van gegevensleveringen. Voor artikel 2 over kwaliteitsinfo kunnen we dat ook doen. De afnemer kan vandaag al aangeven aan welke van de vastgestelde verzameling kwaliteitsignaleringen de gegevens moeten voldoen. Wil of mag de afnemer gegevens niet zien waarop een bepaald kwaliteitsignaal is geplaatst dan worden die gegevens weggefilterd. Omdat de PA een generiek kwaliteitmechanisme kent, is het uitfilteren van kwalitatief slechte gegevens, performance issues daargelaten, een fluitje van een cent.

Het alternatief is dat kwaliteitsignalen worden meegeleverd met de gegevens. Ook dit is vrij eenvoudig te realiseren. Tenslotte kent de PA ook nog een mechanisme waarin voor een betwist gegeven alvast een alternatieve waarde wordt voorgesteld, dus het is ook mogelijk om waar van toepassing die alternatieve versie meteen meteen mee te leveren en daarmee een zekere mate van multirealiteit te ondersteunen. Binnen overheidsland is dit (terug)melden van dubieuze gegevens overigens een *hot issue*.

Ook het in een gegenereerde query opnemen van autorisatieinformatie (artikel 3) is eenvoudig. Afhankelijk van de gegevensautorisaties kunnen elementen in de set geselecteerde gegevens eenvoudig worden toegevoegd of verwijderd. En ook het toevoegen van een restrictie die garandeert dat alleen die personen of bedrijven worden geleverd waarvoor de afnemer een indicatie heeft staan is vrij eenvoudig. Het is dus niet heel moeilijk om een generator te schrijven die dit allemaal verzorgt, maar het zal duidelijk zijn dat de resulterende (SQL) code al niet meer op een A4'tje past. Logging (artikel 4), tenslotte, is een buitenbeentje.

Het gaat bij logging niet om iets dat in de code moet worden gegenereerd, maar om een generieke voorziening die ervoor zorgt dat we achteraf nog kunnen zeggen wat er precies aan gegevens is verstrekt. Ook dat is niet heel complex.

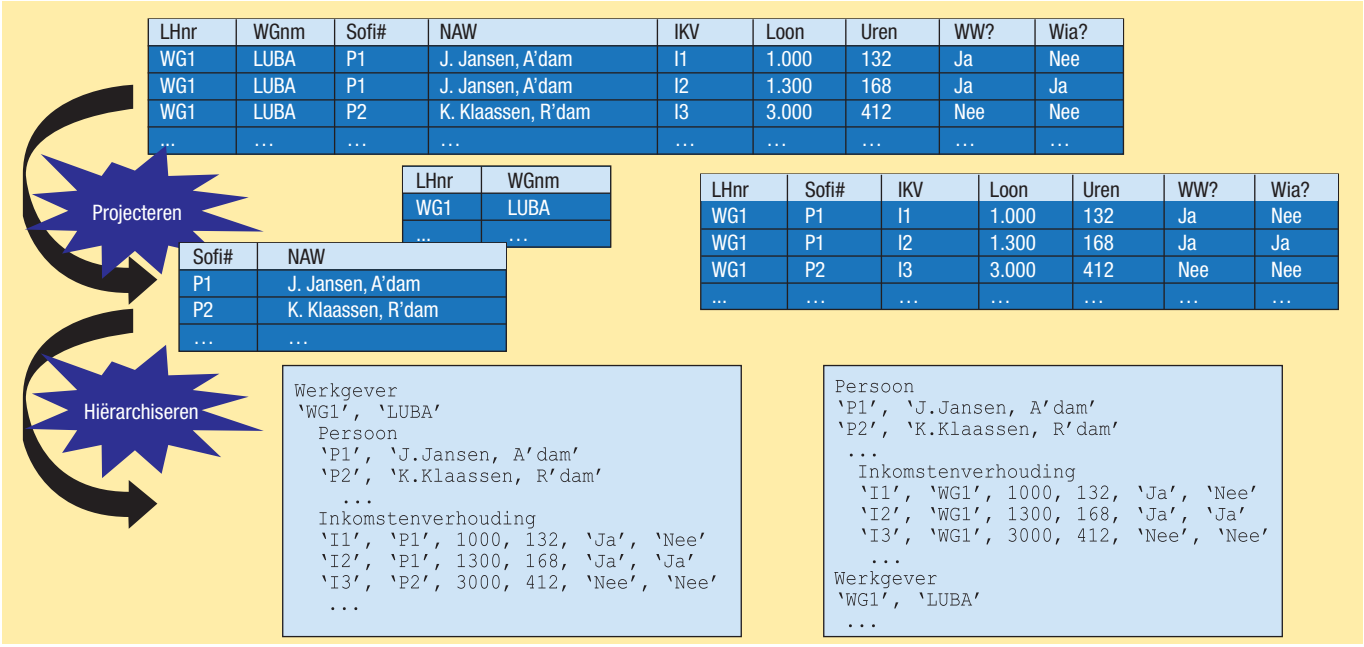
De formatendierentuin

We zijn intussen aangeland bij het linkerdeel van afbeelding 1. Allereerst zien we dan de dimensie 'mutaties-versus-standen'. Dat lijkt al besproken onder het kopje 'tijdsdimensies', maar schijn bedriegt. Bij de twee tijdsdimensies ging het erom welke gegevens we selecteren. Hoe we de gegevens tonen is een andere vraag. We kunnen bijvoorbeeld selecteren op een bepaald interval in de transactietijd en dan krijgen we mutaties terug. Maar vervolgens kunnen die mutaties worden getoond in oude waarde/nieuwe waarde vorm maar ook gewoon als stand-na-mutatie of zelfs alleen als eindstand tussen begin en einde van het transactietijd interval. Het is allemaal niet fundamenteel maar o zo praktisch voor de afnemer. We schreven het al in artikel 6: de manier waarop afnemers van de PA omgaan met tijdinformatie verschilt van primitief tot zeer geavanceerd. Er zijn afnemers die eerdere informatie uit de PA overschrijven en die altijd standen willen. Er zijn afnemers die hun systeem op tijdgebied net zo geavanceerd opzetten als de PA. En er zijn ook afnemers die standen willen krijgen om daar dan weer mutaties uit te destilleren. Al die afnemers wil je in afwachting van het voortschrijden van hun inzicht zoveel mogelijk op maat bedienen en daarmee wil je 'gedoe' voorkomen.

Nog veel belangrijker is het volgende bolletje in afbeelding 1, dat van de logische groeperingen. We hebben eerder gesteld dat we in verband met een performance issue de database zoveel mogelijk willen benaderen met *compound* SQL. Kijken we terug naar het voorbeeld in afbeelding 2 dan zien we ook een join tussen drie logische entiteiten. Vertaald naar de PA database met allerlei historische tabellen en restricties op de tabellen met kwaliteit- en autorisatie-informatie, loopt het aantal tabellen in één gegenereerd SQL statement nog verder op. Daarmee laten we de database optimizer het werk doen, zoals dat moet bij complexe vragen op grote databases. Maar het resultaat is dan zoals altijd een platte output-tabel met heel veel redundantie. In het voor-

```
Select WG.LHNR, WG."NAW" WG.Sector /* Lhnr = id werkgever */
      , P.Sofi#, P."NAW"
      , IKV.IKV#, IKV."Loongegevens", IKV."Verzekeringsgegevens"
From Persoon p
      JOIN Inkomstenverhouding ikv ON p.Sofi#=ikv.Sofi#
      JOIN Werkgever wg ON wg.Lhnr=ikv.Lhnr
Where WG.Sector = <variabele> /* Variabele restrictie*/
Order By <variabele> /* Variabele ordering */
```

Afbeelding 2: Voorbeeld casco gegevenslevering.



Afbeelding 3: Van platte query naar gegevenshiërarchie.

beeld van afbeelding 1 zien we bijvoorbeeld een herhaling van persoonsgegevens als het resultaat meerdere inkomstenverhoudingen of inkomstenopgaven voor eenzelfde persoon bevat. Dat is het klassieke probleem van SQL en het botst frontaal met wat nu de norm is: een *XML-style* multiniveau hiërarchie van gegevensgroepen. De oplossing is een mechanisme waarmee de platte resultaattabel van een query kan worden omgezet in een door de afnemer zelf te parametriseren hiërarchie. Dat is veel minder moeilijk dan het lijkt. Het enige dat vereist is, is het maken van een aantal projecties op het platte query-resultaat. Afbeelding 3 toont een voorbeeld uitgaande van de query van afbeelding 2. We nemen een plat queryresultaat en converteren dat naar een aantal projecties die we vervolgens desgewenst weer op diverse wijzen kunnen omzetten in hiërarchische structuren – alles op basis van parametrisering natuurlijk (zie artikel 5) en zoveel mogelijk door de afnemer in te regelen. De stap om XML of een comma separated bestand te maken van het voorbeeld in afbeelding 3 is nu eenvoudig voor te stellen.

Performance en soorten leveringen

We hebben alle dimensies van afbeelding 1 nu kort de revue laten passeren. Hopelijk is het nu voorstelbaar hoe het mogelijk is om met een handvol brokjes code en een krachtig parametriseringsmechanisme voor tijdaspecten, autorisaties, presentatievormen, structuren en formaten vrijwel elke afnemer van PA gegevens te bedienen en dan vrijwel op maat. Essentieel is zoals altijd dat alle besproken dimensies onafhankelijk van elkaar zijn. Deze orthogonaliteit maakt het mogelijk om iets dat heel complex is stap voor stap te ontwikkelen en niet te verdrinken in complexe broncode. Het enige waar we ons echt zorgen over maken is het performanceprobleem. Zo willen we het stukje code van afbeelding 2 zowel kunnen toepassen op een request/response webservice die een handvol records oplevert als op een 100 Gigabyte bulklevering. Het is naïef om te denken dat de Oracle optimizer alle problemen wel voor ons oplost. Ook moeten voor die webservice de transformaties van afbeelding 3 in het interne geheugen plaatsvinden, terwijl we voor de bulklevering output tabellen gaan genereren waarop we vervolgens weer projecties loslaten. Dat heeft natuurlijk wel wat voeten in de aarde. Waar we ook niets over hebben gezegd is het transportmechanisme als zodanig en de manier waarop we in één opvraging vele afnemers ineens bedienen.

Er is, kortom, nog wel wat werk te verrichten, maar de beloning aan de eindstreep is fors: een enorme kostenbesparing voor de exploitant van de PA en dat op een manier die de klanten van de PA minimaal belast en die vermoedelijk toekomstige ICT-ontwikkelingen eenvoudig kan incorporeren. Dat is des te aantrekkelijker omdat het alternatief een toename van kosten is met elke nieuwe afnemer en levering. In de ICT-wereld van 2009 werken de oplossingen van de jaren negentig niet meer, maar conceptueel waren ze sterker dan wat we nu hebben. Als het lukt om die jaren negentig concepten weer werkend te krijgen voor de 'formatendierentuin' van nu, dan is niet alleen UWV spekkoper.

In het volgende artikel bespreken we hoe je moet omgaan met de verschillende manieren waarop gegevens vanuit de PA worden geleverd aan afnemers en hoezeer die afnemers daarmee worstelen en zoeken naar houvast. Dat houvast gaan we proberen te bieden met onze Tien Geboden voor Gegevenslogistiek.

René Veldwijk is partner bij FAA Partners, onderdeel van de Ockham Groep.