GEPUBLICEERD ALS:
EEN DERDE DIMENSIE VOOR INFORMATIEMODELLEREN, H.A. PROPER,
DB/MAGAZINE, 3(10), MEI 1999, 52-55.

Een derde dimensie voor informatiemodelleren

H.A. (Erik) Proper
ID Research
Groningenweg 6
2803 PV Gouda
E.Proper@acm.org

De vraag hoe om te gaan met complexiteit is een vraag die op verschillende plekken in een software ontwikkelproces terugkomt. In dit artikel richten we ons op de complexiteit zoals deze tijdens de eerste fasen van de software ontwikkeling cyclus (de analyse fase) in de informatiemodellen naar voren komt. Het doel van dit artikel is om de lezer een simpele en praktische aanpak aan te reiken voor het omgaan met complexiteit in informatiemodellen.

Informatiemodellen
Informatiemodellen zijn er op gericht om tijdens de analyse fase inzicht te verwerven of te bieden in de belangrijkste onderliggende informatiestructuren van een domein. Voor het weergeven van deze informatiemodellen wordt doorgaans gebruik gemaakt van grafische technieken. Informatiemodellen blijken doorgaans een zeer effectief communicatiemiddel te zijn ter ondersteuning van de communicatie tussen informatieanalisten, bouwers en projectmanagers. In veel gevallen zijn ze zelfs bruikbaar bij de communicatie met de vertegenwoordigers uit de gebruikersgemeenschap.

Echter, een tekortkoming van veel technieken voor informatiemodellering is het ontbreken van adequate mechanismen om met complexiteit om te gaan. In de praktijk neemt bij grotere domeinen de complexiteit van de resulterende informatiemodellen nogal snel toe. Bij zulke domeinen heeft men doorgaans al gauw te maken met honderden, zo niet duizenden, concepten en relaties. De effectiviteit van de informatiemodellen erodeert dan veelal snel onder inwerking van deze complexiteit.

Een uit de hand lopende complexiteit, waardoor men vaak al snel het gevoel krijgt door de bomen het bos niet meer te zien, is ook een veel gehoord argument in discussies rond het gebruik van een methode zoals Object Role Modelling (ORM) [1] (in Nederland misschien beter bekend als NIAM [2]) versus de Entity Relationship (ER) [3] gebaseerde methoden. In ORM/NIAM wordt geen onderscheid gemaakt tussen de zogenaamde entiteit- en attribuuttypen. In ORM worden beide soorten typen als objecttypen aangemerkt, en gelijkelijk behandeld. Dit heeft tot voordeel dat een informatieanalist zich tijdens de analyse fase niet hoeft af te vragen of een objecttype eigenlijk als een attribuuttype of als een entiteittype moet worden gemodelleerd. Het gevolg is echter wel dat informatiemodellen in ORM nogal snel in omvang toenemen.

Het onderscheid tussen entiteittypen en attribuuttypen zoals dit in ER kan worden gemaakt, biedt een simpel mechanisme om onderscheid te maken tussen concepten die in het gemodelleerde domein als belangrijk worden ervaren (de entiteiten), en welke als minder belangrijk worden ervaren (de attributen). Door dit onderscheid kunnen in ER informatiemodellen compacter worden weergegeven dan in ORM. Omdat ORM dit onderscheid niet biedt, zal voor een gegeven domein het ER schema er dan ook minder complex uit zal zien dan het bijbehorende ORM schema.

Het principe om onderscheid te maken tussen attributen en entiteiten heeft echter slechts een beperkte reikwijdte als het gaat om het omgaan met complexiteit. Dit omdat het slechts éénmaal is toe te passen. Bij

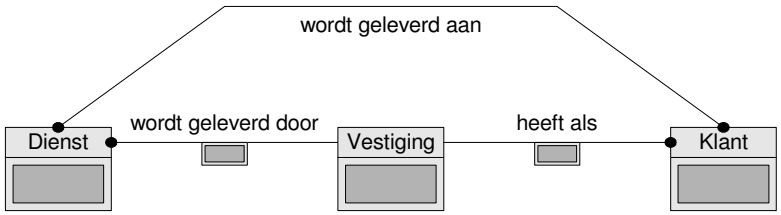
grotere domeinen neemt de complexiteit van ER schema's eveneens snel toe. In het onderscheid tussen hoofd- en bijzaken ligt echter wel de kiem voor een structurele manier om tot een betere beheersing van de complexiteit te komen.

Reductie van complexiteit
Bij het modelleren van processen, workflows, of dataflows, wordt doorgaans gebruik gemaakt van een vorm van structurele decompositie als mechanisme om de complexiteit van modellen in de hand te houden. Het is bijvoorbeeld bij de meeste technieken voor procesmodellering mogelijk om processen hiërarchisch te decomponeren door deze op te bouwen uit diverse sub-processen. Dit kan men naar believen herhalen om zo steeds complexere processen samen te stellen zonder meteen met de gehele complexiteit geconfronteerd te worden. Dergelijke decompositiemechanismen komen feitelijk voort uit de wereld van gestructureerd programmeren [4].

In [5-8] zijn deze ideeën toegepast op de wereld van informatiemodellen. De kern hiervan wordt gevormd door het op meerdere niveaus van abstractie onderscheiden van de belangrijke en de minder belangrijke concepten uit een domein. Wanneer dit mechanisme wordt toegepast op een gegeven informatiemodel is het resultaat een zogenaamd drie-dimensionaal informatiemodel. In dit drie-dimensionale model worden de eerste twee dimensies opgespannen door de (traditionele) concepten en hun onderliggende relaties, en leiden de verschillende abstractielagen tot een derde dimensie.

Voorbeeld domein
In dit artikel wordt dit decompositie mechanisme als voorbeeld toegepast op een bancair domein. Het hier gepresenteerde domein kan wellicht als een 'speelgoed' domein aangemerkt worden, maar het stelt ons prima in staat om de essentie en de kracht van drie-dimensionale informatiemodellen te illustreren.

Het model in Figuur 1 geeft een overzicht van het in dit voorbeeld beschouwde bancaire domein. Dit is het model op het hoogste niveau van abstractie. Zoals te zien is zijn de drie belangrijkste concepten in het voorbeeld domein: diensten, vestigingen, en klanten. De diensten die door de bank aan de klant worden aangeboden, worden uiteindelijk geleverd via de vestigingen van de bank. De klanten doen direct zaken met een specifieke vestiging.

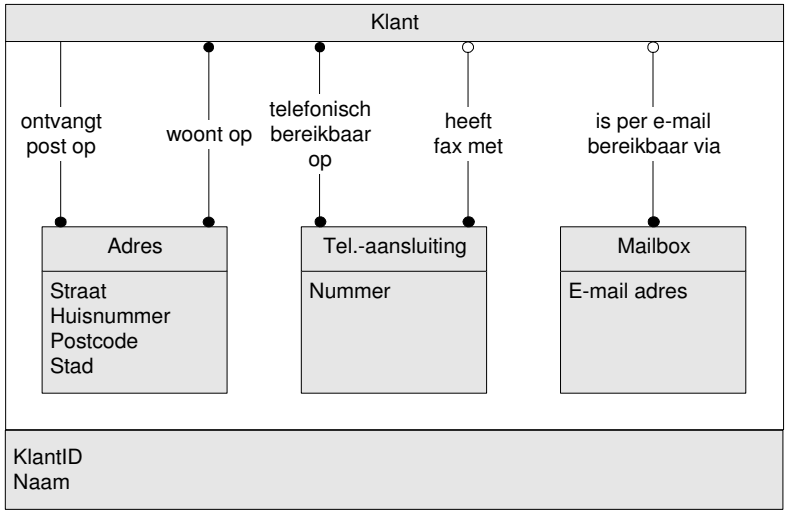


Figuur 1: Overzicht van een bank domein.

De in Figuur 1 gebruikte notatie is gebaseerd op OMT [9]. OMT is een objectgeoriënteerde modelleringstechniek, en is één van de stamvaders van de wellicht bekendere UML [10] modelleringstechniek. Hoewel het voorbeeld domein is uitgewerkt in de context van OMT [9], zijn de gebruikte abstractiemechanismen even goed toe te passen in een ER- als een ORM-context [5-7]. Kenners van de OMT notatie zullen echter niet bekend zijn met de donker gekleurde rechthoeken zoals deze in de objecttypen en op de associatietypen opduiken. Deze rechthoeken worden gebruikt om aan te duiden dat de betrokken object- of associatietypen een decompositie hebben.

De decompositie van het klant objecttype is te vinden in Figuur 2. In deze decompositie valt te lezen dat klanten op een adres wonen, terwijl dat adres niet noodzakelijkerwijze hetzelfde hoeft te zijn als waar de klant eventuele correspondentie wenst te ontvangen. Daarnaast is het zo dat een klant eventueel per telefoon, fax, of e-mail bereikbaar is. De traditionele OMT manier om een object type weer te geven is voor het Klant object type als het ware 'binnenstebuiten' gekeerd. Dit is gedaan om aan te geven dat Adres, Telefoonaansluiting, en Mailbox als complexe eigenschappen van Klant gezien moeten worden. In een CASE

tool die een OMT versie met een decompositiemechanisme zou ondersteunen zou men kunnen verwachten dat bij het aan-clicken van de donkergekleurde rechthoek uit het Klant object type van Figuur 1, de decompositie van Klant wordt weergegeven.



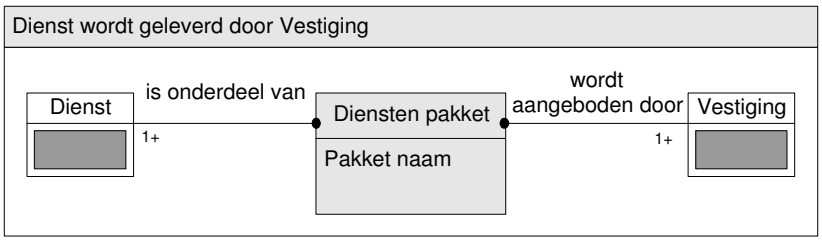
Figuur 2: Decompositie van het Klant object type.

Niet alleen objecttypen kunnen een decompositie hebben. Ook associatietypen kunnen een onderliggende decompositie bij zich dragen. In het voorbeeld domein hebben zowel het Dienst wordt geleverd door Vestiging, als het Vestiging heeft als Klant associatietype een decompositie. Deze zijn te vinden in respectievelijk Figuur 3 en Figuur 4.

Klanten kunnen hun bankzaken regelen via meerdere vestigingen, maar het is vereist dat één van deze vestigingen aangemerkt wordt als hun primaire vestiging om op zo'n manier het onderhouden van de relatie tussen klant en bank vanuit een unieke locatie te kunnen coördineren. Dit leidt tot de situatie zoals weergegeven in Figuur 3. Het attribuut "Is de primaire vestiging waar de klant bankiert" is een boolse waarde (*true* of *false*), en geeft aan of de relatie tussen een vestiging en een klant de primaire vestiging van de klant betreft of niet.

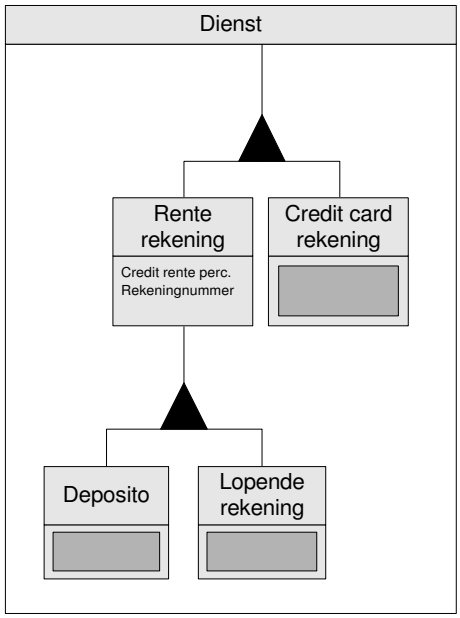
Figuur 3: Primaire vestiging van klanten.

De diensten die door een vestiging geleverd worden maken onderdeel uit van een dienstenpakket, zoals dit door een vestiging in de markt wordt gezet. Denk hierbij aan dienstenpakketten met hypothecaire producten, verzekeringsproducten, spaarproducten, etc. De associatie tussen Dienst en Vestiging kan daarom worden verfijnd tot de situatie zoals beschreven in Figuur 4.



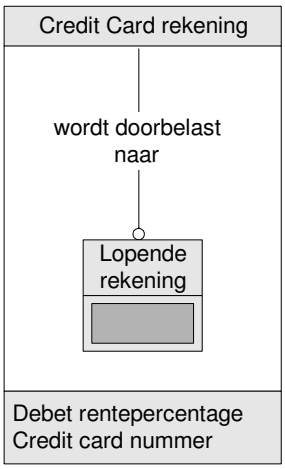
Figuur 4: Diensten pakket

Tot nog toe zijn de voorbeelden van decompositie steeds één-staps decomposities geweest. Met andere woorden, geen van de decomposities bevatte een objecttype dat zelf weer de compositie is van andere objecttypen. In Figuur 5 staat de uitwerking van het Dienst objecttype, en dit bevat een aantal objecttypen die zelf weer een nadere decompositie kennen. Merk op dat de getoonde decompositie van Dienst slechts een voorbeeld is. Hypotheekproducten, verzekeringen, effectenbeheer, etc., zouden hier bijvoorbeeld nog aan toegevoegd kunnen worden.



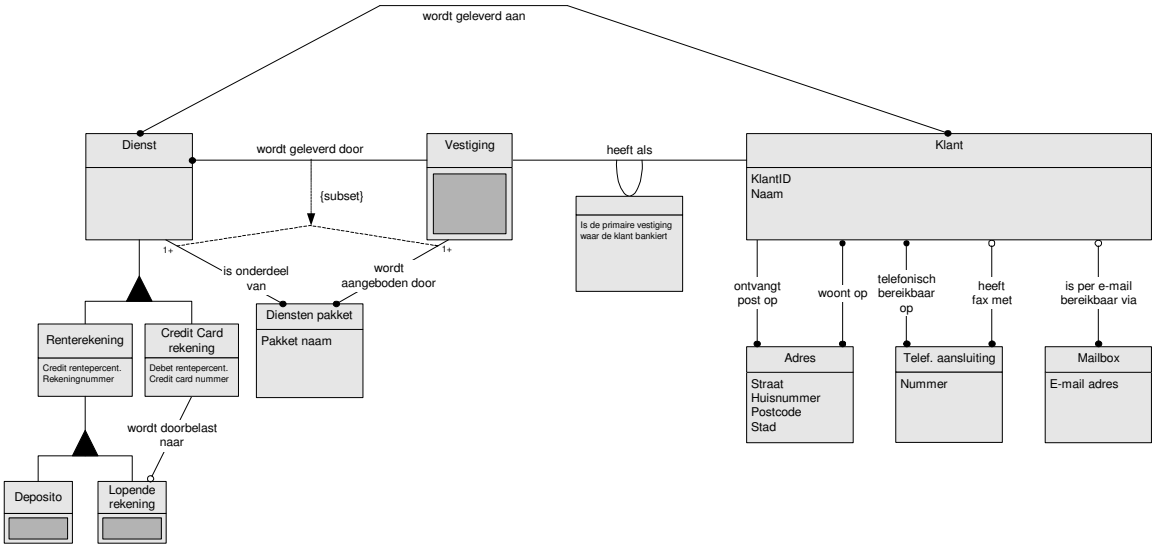
Figuur 5: Geneste decompositie

Als laatste voorbeeld van een decompositie staan in Figuur 6 de details van een Credit Card rekening. De uitgaven die gedaan zijn met een credit card worden eens per maand doorbelast naar een lopende rekening.



Figuur 6: Credit card details

Wanneer alle decomposities samengevoegd worden tot één schema, dan resulteert dit in het schema zoals dit in Figuur 7 is te vinden. De Vestiging, Deposito, en Lopende rekening objecttypen zijn in het kader van bovenstaand voorbeeld niet verder gedecomponeerd, en zijn daarom ook in het totaal beeld niet verder gedecomponeerd.



Figuur 7: Totaal beeld van het bank domein.

Het bovenstaande voorbeeld geeft een korte illustratie van de effectiviteit van het decompositiemechanisme. Wanneer Figuur 1 en Figuur 7 naast elkaar worden gelegd zal het duidelijk zijn dat het eerste beter geschikt is voor een eerste introductie tot het domein. Door vervolgens op de verschillende gedecomponeerde objecttypen in te zoomen kan op een gedoseerde en gerichte wijze een detailoverzicht van het domein verkregen worden. Bij het 'speelgoed' domein uit dit voorbeeld was dit natuurlijk niet zo hard nodig, maar in domeinen uit de praktijk zullen de informatiemodellen dermate groot zijn dat een decompositie mechanisme zeker op zijn plaats is.

Methodische inbedding

Een vraag die nu misschien bij de lezer is gaan leven is hoe men in de praktijk zou moeten bepalen wat de essentiële object- en associatietypen zijn. Analysten die veel ervaring hebben met bepaalde domeinen zullen wellicht redelijk snel kunnen inschatten wat de essentiële concepten zijn binnen een domein, om vervolgens op een top-down manier de rest van het domein vorm te geven.

Een andere aanpak is natuurlijk om bottom-up te werk te gaan. Men kan op een traditionele manier eerst een 'plat' (2-dimensionaal) informatiemodel maken, en vervolgens hierin op zoek te gaan naar de essentiële concepten en langs deze weg stapsgewijs de hogere abstractieniveaus afleiden. In [6] wordt een algoritme besproken dat op basis van een stelsel van heuristieken deze abstractieniveaus automatisch probeert af te leiden. In de praktijk zal men waarschijnlijk een mengvorm tussen de bottom-up en top-down aanpak willen gebruiken.

Alternatieve aanpakken

Het decompositiemechanisme zoals hier gepresenteerd is niet de enige manier om met complexiteit in informatiemodellen om te gaan. In dit artikel is al eerder melding gemaakt van het onderscheid tussen attributen en entiteittypen zoals dit is terug te vinden in de op ER gebaseerde modelleringstechnieken (zoals bijvoorbeeld OMT [9]). De reikwijdte hiervan is echter beperkt, omdat dit onderscheid niet recursief is toe te passen.

Daarnaast bestaan er in diverse informatiemodelleringstechnieken constructies zoals generalisatie en aggregatie. Generalisatie biedt de mogelijkheid om verschillende objecttypen met gelijke eigenschappen samen te nemen; te generaliseren. In het bancaire voorbeeld is Dienst een generalisatie van Rentrerekening en Credit Card rekening. Echter, bij een generalisatie hiërarchie blijven de onderliggende details zichtbaar. Zie, bijvoorbeeld, Figuur 7. Dus datgene waarover gegeneraliseerd wordt, blijft in de resulterende modellen gewoon zichtbaar.

Aggregatie is een constructiemechanisme waarbij een modelleur kan aangeven dat objecten een fysiek onderdeel vormen van een groter geheel. Bijvoorbeeld, een auto bestaat uit een motor, een carrosserie, wielen, etc. De meeste in gebruik zijnde aggregatie mechanismen bieden van zichzelf echter geen mogelijkheid om de details van de geaggregeerde elementen te verbergen, en langs die weg de complexiteit van het resulterende informatiemodel te verminderen. Er worden door de generalisatie en aggregatie constructies zelf geen mogelijkheden geboden om op een hoger niveau te kunnen abstraheren van de onderliggende structuur van een aggregatie of een generalisatie.

Sommige CASE tools bieden inmiddels ook de mogelijkheid om (platte) informatiemodellen op te delen in 'sheets' of 'modules'. Feitelijk komt dit erop neer dat een groot informatiemodel wordt verdeeld over meerdere pagina's. Dergelijke mechanismen bieden echter niet de mogelijkheid om de voor een domein essentiële objecttypen te identificeren en op verschillende niveaus van detail te presenteren teneinde op deze manier een overzicht te verkrijgen. Men wordt uiteindelijk nog steeds met alle details geconfronteerd.

Behapbaar

Dit artikel heeft een korte introductie gegeven van een decompositiemechanisme om complexe informatie-modellen op een meer behapbare wijze te presenteren. Het mechanisme is geïllustreerd aan de hand van een voorbeeld. Hoewel er alternatieven zijn, bieden deze niet dezelfde krachtige abstractiemogelijkheden. Wat momenteel nog een open issue lijkt te zijn is het bieden van tool support voor dergelijke mechanismen.

Over de auteur

Dr. H.A. (Erik) Proper is werkzaam als Research Consultant bij ID Research.

Referenties

1. Halpin, T.A., *Conceptual Schema and Relational Database Design*. 1995, Sydney, Australia: Prentice-Hall.
2. Nijssen, G.M. and T.A. Halpin, *Conceptual Schema and Relational Database Design: a fact oriented approach*. 1989, Sydney, Australia: Prentice-Hall.
3. Batini, C., S. Ceri, and S.B. Navathe, *Conceptual Database Design - An Entity-Relationship Approach*. 1992, Redwood City, California: Benjamin Cummings.
4. Koster, C.H.A., *Top-Down Programmering*. Vol. 1. 1988, Schoonhoven, The Netherlands: Academic Service.
5. Bronts, G.H.W.M., *et al.*, *A Unifying Object Role Modelling Approach.* Information Systems, 1995. **20**(3): p. 213-235.
6. Campbell, L.J., T.A. Halpin, and H.A. Proper, *Conceptual Schemas with Abstractions - Making flat conceptual schemas more comprehensible.* Data & Knowledge Engineering, 1996. **20**(1): p. 39-85.
7. Creasy, P.N. and H.A. Proper, *A Generic Model for 3-Dimensional Conceptual Modelling.* Data & Knowledge Engineering, 1996. **20**(2): p. 119-162.

8. Embley, D.W., B.D. Kurtz, and S.N. Woodfield, *Object-Oriented Systems Analysis*. 1992, Englewood Cliffs, New Jersey: Yourdon Press.
9. Rumbaugh, J., *et al.*, *Object-Oriented Modeling and Design*. 1991, Englewood Cliffs, New Jersey: Prentice-Hall.
10. Booch, G., I. Jacobson, and J. Rumbaugh, *UML Notation Guide*, . 1997, Rational Rose Corporation.