Conceptueel modelleren met drie vormen van abstractie

Kan een object een klasse zijn?

Henk Jan Pels

Ontwerpers plegen een nieuw ontwerp te beginnen met een schets. Ook in discussies over mogelijke oplossingen worden vaak schetsen gemaakt. Zo'n mogelijkheid om te schetsen mis ik bij het datamodelleren. Een UML diagram is nuttig om overzicht te houden over een complex geheel, maar het is niet handig voor een snelle schets.

Het is al helemaal niet geschikt als taal om in een lopend verhaal een voorbeeld op te bouwen en te bediscussiëren. Daarom heb ik een notatie bedacht om beweringen over database en schema in een lopende tekst te kunnen opnemen. Van tijd tot tijd kan er dan een UML schema bij gezet worden om het overzicht te houden.
In het vorige artikel heb ik laten zien hoe je daarmee bekende zaken als attributen, sleutels en associaties kunt uitdrukken.
Tot nu toe heb ik dus alleen oude wijn in nieuwe zakken geboden.
In deze aflevering komt er werkelijk iets nieuws: hiërarchieën van klassen, die het mogelijk maken om classificatie als abstractie te modelleren.

ABSTRACTIE

Abstractie is één van de meest essentiële intellectuele instrumenten. Taal kan niet bestaan zonder abstractie. Zelfs als we, wijzend met onze vinger, zeggen: "dat ding is vies", gebruiken we twee abstracties: 'ding' als abstractie voor alles waar je naar zou kunnen wijzen en 'vies' voor alle dingen die een onprettige smaak hebben. Een abstractie gebruik je om alle niet relevante details buiten beschouwing te kunnen laten. Vaak wordt een verhaal begonnen met een abstractie om de context te plaatsen. Met "die beuk voor ons huis" wordt aangegeven dat er iets gezegd gaat worden over een boom van het soort beuk, en wel in het bijzonder over de beuk voor ons huis. Zo maakt elke zin gebruik van één of meer abstracties en in de loop van het betoog wordt voortdurend gesprongen tussen abstractieniveaus: van bomen naar beuken, van beuken naar de beuk voor ons huis, van die beuk naar andere dingen voor ons huis enzovoort. Als we willen spreken over abstracte zaken, die we niet kunnen voelen of zien en waarvan we ons dus alleen een mentaal beeld kunnen vormen, zijn abstracties zeker belangrijk. Bovendien is het dan nuttig als we ons een beetje precies kunnen uitdrukken, omdat tussen de mentale beelden van de deelnemers aan het gesprek gemakkelijk storende verschillen kunnen ontstaan.

Sinds de begintijd van het conceptueel modelleren worden drie vormen van abstractie onderscheiden: classificatie, aggregatie en generalisatie. Classificatie wordt gebruikt bij het definiëren van objectklassen: een objectklasse is een abstractie voor een verzameling soortgelijke objecten. De klasse stelt ons in staat een enkele bewering te doen die voor alle objecten van de klasse geldt. Ook maakt hij het mogelijk een context te stellen voor een bepaalde operatie, opdat die op alle objecten van de klasse wordt toegepast. Verder zien we bij classificatie af van de details van de individuele objecten teneinde ons te kunnen concentreren op zaken die voor alle objecten van die klasse van toepassing zijn. Het omgekeerde

Een UML diagram is niet handig voor een snelle schets

Modellering van scratch af aan (3)

In de voorgaande twee artikelen (DB/M 4 en 5) betoogt Henk Jan Pels dat conceptueel datamodelleren niet alleen een nuttig middel is om databases te ontwerpen, maar dat het ook een krachtig middel is om abstracte zaken te beschrijven. Zoals werktuigbouwkundige tekeningen gebruikt worden om de geometrie van fysieke voorwerpen nauwkeurig te beschrijven, zo zijn datamodellen geschikt om abstracte concepten zoals een productieplan of een productfamilie te beschrijven. In dit derde artikel presenteert de auteur een door hem bedachte notatie om beweringen over database en schema in een lopende tekst te kunnen opnemen.

van classificeren is instantiëren: je kiest een enkel object uit de klasse of voegt er één aan toe.

De tweede abstractie, aggregatie, is het samenvoegen van afzonderlijke objecten tot een enkel object. De term aggregaat wordt gebruikt voor de samenvoeging van een pomp of dynamo met een motor, waardoor een samengesteld, geaggregeerd ding ontstaat met een hogere functie. Zo is het woord auto ontstaan uit automobiel, wat ruwweg staat voor: "kar met motor". Omdat het aggregaat een functie heeft die de afzonderlijke delen niet hebben, en omdat er frequent aan gerefereerd wordt, past de mens een aggregatie-abstractie toe om te kunnen afzien van de eigenschappen der afzonderlijke dingen. Het omgekeerde van aggregeren is decomponeren: daarbij daalt men af van het geheel en treedt in de details van de afzonderlijke delen.

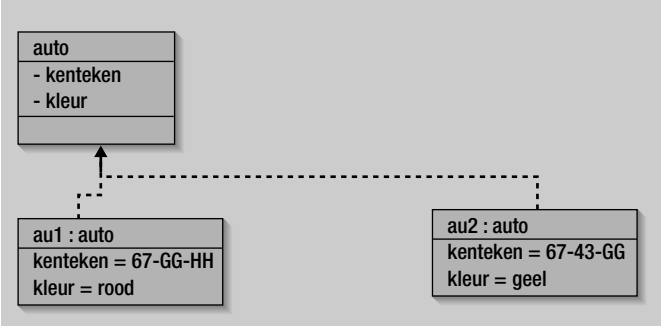
GENERALISATIE

De derde vorm van abstractie, generalisatie, past men toe om een aantal klassen bij elkaar te voegen tot een hogere orde begrip. Daarbij concentreert men zich op de eigenschappen die voor al die klassen gemeenschappelijk zijn. Het omgekeerde in specialiseren. Door hiërarchieën van generalisaties te vormen kan men zich al generaliserend verheffen tot steeds hogere niveaus met steeds algemenere eigenschappen of zich specialiserend afdalen naar steeds grotere diepten van speciale eigenschappen van bijzondere groepen objecten.

Bij nadere beschouwing is er iets vreemds aan de hand met de manier waarop de verschillende abstracties in het datamodelleren worden gebruikt. Generalisatie is het meest uitgewerkt met een eigen semantiek van overerving van eigenschappen en met de mogelijkheid van hiërarchieën. Deze vorm van abstractie wordt

Hiërarchieën van klassen zijn niet mogelijk

veel toegepast en draagt ook heel sterk bij aan het vereenvoudigen van software-ontwikkeling. Aggregatie wordt vaak alleen gezien in de vorm van het object als aggregatie van zijn attributen. In de objectgeöriënteerde benadering zijn daarbij weliswaar hiërarchieën mogelijk door als attribuutwaarde een object te nemen dat zelf weer attributen heeft, maar het vreemde is dat hier niet gesproken kan worden van het modelleren van aggregaties in de relevante wereld. Je kunt moeilijk volhouden dat een persoon beschouwd kan worden als de samenstelling van naam, adres en geboortedatum. Als je een persoon decomponeert kom je eerder uit op hoofd, romp, armen en benen. UML kan dit als aggregatie modelleren, maar kent er weer geen specifieke semantiek aan toe. Het is eigenlijk niet meer dan een bijzondere vorm van associatie met een eigen tekenwijze. Met classificatie is het ook een beetje vreemd. Als ik de objectklasse Persoon definieer als klasse van



FIGUUR 1. KLASSE AUTO MET TWEE INSTANTIATIES.

objecten die personen representeren, dan kan ik zeggen dat objectklasse Persoon de representatie is van de abstractie 'klasse van personen' uit de relevante wereld. Ik ken echter geen talen die toestaan om vervolgens Persoon weer als instantiatie van een hogere klasse aan te wijzen. Ergo: hiërarchieën van klassen zijn niet mogelijk. Sommige modellen kennen wel zoiets als metaklassen, maar interpreteren dat bijvoorbeeld als de klasse van alle klassen, dus weer op het niveau van de modelleertaal (zoals bij object als aggregatie van attributen) en niet als modellering van een relevante wereld. Ik was benieuwd wat er gebeurt als je wel toestaat dat een objectklasse tevens object is, en dus attribuutwaarden kan hebben en lid zijn van een andere klasse. In dit artikel ga ik daar verder op in.

KLASSIFICATIEHIËRARCHIEËN

De expliciete lidmaatschapsrelatie, zoals we die in de vorige aflevering hebben voorgesteld, maakt het heel eenvoudig om een klasse te benoemen tot een instantie van een andere klasse. We nemen de auto's uit de vorige aflevering als voorbeeld (zie figuur 1):

```
<au1, kenteken, 67-GG-HH, 1>, <au1, kleur, rood>
<au2, kenteken, 67-43-GG, 1>, <au2, kleur, geel>
```

Stel nu dat we van deze auto's de brandstofsoort en het gewicht willen vastleggen. We realiseren ons echter, dat deze eigenschappen voor beide auto's gelijk zijn, omdat het auto's zijn van hetzelfde type. Het zou daarom redundant zijn om deze gegevens voor elke auto afzonderlijk vast te leggen. We voeren dus het autotype in:

```
<at1, typenr, 'LT20d', 1>, < at1, gewicht, 2150,
1>, <at1, vermogen, 125, 1>, < at1, brandstof,
diesel, 1>
```

Omdat er ook nog andere autotypes zijn die we later willen vastleggen, definiëren we ook de objectklasse autotype:

```
<<autotype, atttribuut, typenr, 1>>, <<autotype,
attribuut, gewicht, 1>>
```

en maken object at1 lid van deze klasse:

```
<(at1, autotype, 1)>
```

Nu kunnen we eenvoudig au1 en au2 lid maken maken van object at1:

```
<(au1, at1, 1)>, <(au2, at1, 1)>
```

Misschien wordt het nu een beetje moeite om alles netjes voor de geest te houden, dus het wordt tijd voor een plaatje (zie figuur 2). In figuur 2 zien we een klasse autotype met een object at1 en een klasse auto met de objecten au1 en au2. Het lidmaatschap is aangegeven met een gestippelde pijl. Tot zover heel gewoon. Bijzonder is dat er ook nog gestippelde pijlen staan van au1 en au2 naar at1. Dat is bijzonder omdat a1 een object is. Door het feit dat a1 nu leden heeft, is hij verheven tot klasse. Hij is dus nu object en klasse tegelijk.

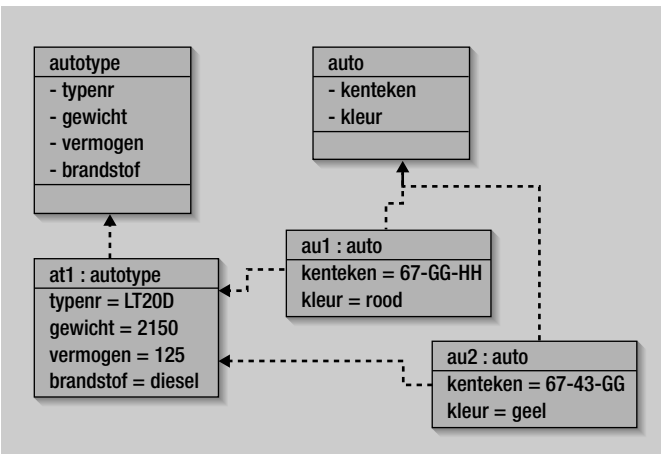
Wat moet nu de semantiek zijn van een object dat lid is van een ander object? We kwamen op deze constructie omdat beide auto's dezelfde waarden hebben voor typenr, gewicht, vermogen en brandstof, omdat ze van hetzelfde type zijn. In de relevante wereld is het kennelijk zo dat de auto's deze waarden erven van hun type. Dan moet dat in het datamodel ook zo zijn: objecten erven de attribuutwaarden van hun klasse(n).

Deze semantiek betekent dat in de database de auto's deze attribuutwaarden in afgeleide vorm hebben:

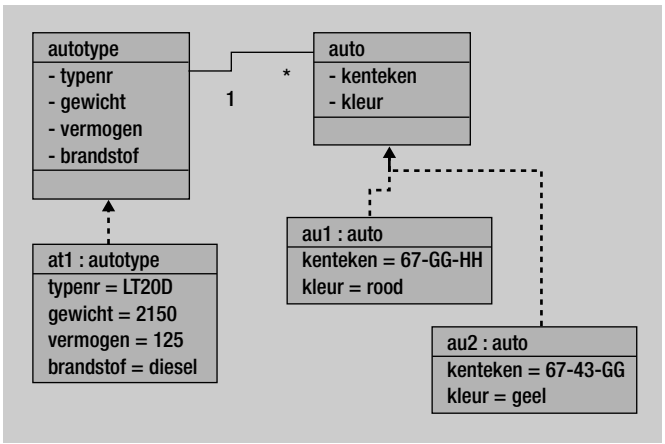
```
<au1, typenr, 'LT20d', 0>, < au1, gewicht, 2150,
0>, <au1, vermogen, 125, 0>, < au1, brandstof,
diesel, 0>

<au2, typenr, 'LT20d', 0>, < au2, gewicht, 2150,
0>, <au2, vermogen, 125, 0>, < au2, brandstof,
diesel, 0>
```

Daarmee hebben we ons doel bereikt: de auto's hebben een typenummer, een gewicht, een vermogen en een brandstof. Ondanks dat deze gegevens voor beide auto's gelijk zijn, is er geen sprake



FIGUUR 2: OBJECT ALS KLASSE.



FIGUUR 3. ALTERNATIEF VOOR FIGUUR 2 MET ASSOCIATIES.

van redundantie, omdat het immers afgeleide gegevens zijn. Uiteraard is het ook mogelijk om deze situatie redundantievrij te modelleren met de bestaande middelen. Een voor de hand liggende oplossing is om een associatie tussen auto en autotype te leggen, zoals in figuur 3.

In figuur 3 kan bijvoorbeeld het gewicht van auto au1 gevonden worden met de expressie

```
au1.autotype.gewicht
```

Dat is minder elegant dan de oplossing met classificatie volgens figuur 2, waarin het gewicht direct gevonden wordt met

```
au1.gewicht
```

Dit komt beter overeen met de relevante wereld waarin men immers ook spreekt over het gewicht van de auto, en niet over het gewicht van het type van de auto.

CONCLUSIE

Het toestaan dat objecten instantiaties hebben en dus tegelijk klassen zijn blijkt eenvoudig te realiseren. Het is bovendien nuttig omdat het de situatie waarin objecten eigenschappen met hun waarde erven van andere objecten elegant kan modelleren. In een volgende aflevering zal ik een voorbeeld behandelen uit de wereld van documentbeheer. Daarin wil ik laten zien dat deze manier van modelleren, beter dan natuurlijke taal, in staat stelt om uit te drukken hoe de relevante wereld in elkaar zit. ●

Henk Jan Pels (H.J.Pels@tm.tue.nl) is werkzaam aan de Faculteit Technologie Management, Capaciteitsgroep Informatie & Technologie van de Technische Universiteit Eindhoven.