Op en neer stappen tussen abstractieniveaus

Een voorbeeldige classificatie

Henk Jan Pels

Met de tot nu toe gepresenteerde middelen zijn er twee manieren om een objectklasse in een model in te voeren. Enerzijds door een beperking te specificeren waarbij die klasse als eerste coördinaat genoemd wordt, en anderzijds door een lidmaatschap te specificeren waarbij die klasse als tweede coördinaat voorkomt. Evenzo zijn er twee manieren om een object in een model in te voeren: namelijk door een gegeven te specificeren waarin dat object als eerste coördinaat voorkomt, of door een lidmaatschap te specificeren waarin dat object als eerste coördinaat voorkomt.

CLASSIFICATIE

Hetzelfde unieke symbool kan zowel in de rol van object voorkomen in een model, als in de rol van klasse. Bij een object mogen beperkingen worden gespecificeerd, waardoor dat object tevens klasse wordt en bij een klasse mogen attribuutwaarden worden gespecificeerd, waardoor die klasse een object wordt.

Daarmee zijn objecten en klassen niet langer twee separate begrippen, maar worden het rollen die de symbolen ten opzichte van elkaar spelen. De betekenis van die relatie is dat:

1. de attribuutwaarden van het object moeten voldoen aan alle beperkingen die voor zijn klassen zijn gespecificeerd,
2. het object erft alle attribuutwaarden van zijn klassen.

Er blijft nog één vraag over: wat is de betekenis als we een lidmaatschap specificeren waarin zowel de eerste als de tweede coördinaat een klasse is? Dit kan worden geïllustreerd met het voorbeeld van de auto's en hun type van het vorige artikel (zie figuur 1). In figuur 1 is een stippelpijl getekend van auto naar autotype. De betekenis daarvan is de volgende beperking: elke instantie van de klasse auto moet tevens lid zijn van een instantie van de klasse autotype.

Kort geformuleerd: "elke auto is van een bepaald type". Uiteraard moet zo'n bijzondere relatie als deze een eigen naam hebben. De klasse autotype staat duidelijk 'boven' de klasse auto. De term superklasse is echter al vergeven voor de generalisatie-abstractie. Ik stel daarom voor om autotype een 'hyperklasse' van auto te noemen: auto heeft autotype als hyperclass. Formeel laat dit zich noteren als de beperking:

```
<<auto, hyperclass, autotype, 1>>
```

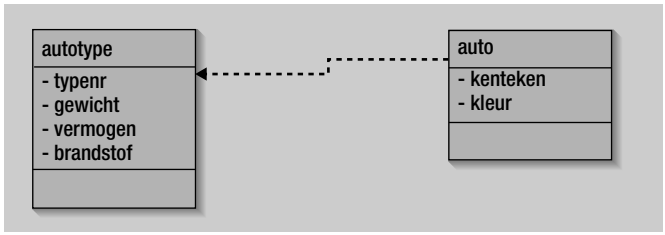
Voor de liefhebbers van symmetrie zouden we nog auto een 'hypoklasse' van autotype kunnen noemen, maar echt nodig is zo'n extra term niet. Uit de structuur van figuur 1 kunnen we nu lezen dat elke auto, behalve de attributen kenteken en kleur, ook de attributen typenr, gewicht, vermogen en brandstof heeft. Dit zijn echter impliciete attributen waarvan de waarden worden geërfd van het specifieke autotype.

DOCUMENTMANAGEMENT

Een levensecht voorbeeld van een klassenhiërarchie kunnen we vinden in de wereld van documentmanagement. Dit is een toepassingsgebied dat, met de explosie van het aantal digitaal beschikbare documenten, sterk in de belangstelling staat. Bij documentmanagement onderscheidt men vaak verschillende documenttypen waarvoor verschillende formaten en procedures bestaan. Elk document wordt ingedeeld bij zo'n type, zodat vastligt hoe het behandeld moet worden. Vaak doorlopen documenten een life

Modellering van scratch af aan (4)

In het vorige artikel heeft Henk Jan Pels de mogelijkheid voorgesteld om hiërarchieën van klassen te vormen, eenvoudig door toe te staan dat het ene object een instantie is van het andere. Deze keer wordt een voorbeeld uitgewerkt, waarin deze constructie fraai tot zijn recht komt en helpt om een (abstracte) werkelijkheid precies te beschrijven.
Hopelijk is het gegoochel met abstracties geen ontmoediging om deze artikelenreeks met interesse te blijven volgen.



Figuur 1. Hyperklasse.

cycle, waarin zij nog verschillende wijzigingen kunnen ondergaan. Daardoor wordt het noodzakelijk om onderscheid te maken tussen documentversies. Eén en ander is gemodelleerd in figuur 2, een structuur die elke ware informatiemodelleur zou moeten doen watertanden.

In het midden staan vijf objectklassen vermeld. *Documenttype* is hyperklasse van *document*, dus elk document is een (instantie van een) documenttype. Document op zijn beurt is hyperklasse van

Het valt niet mee voortdurend tussen abstractieniveaus op en neer te stappen

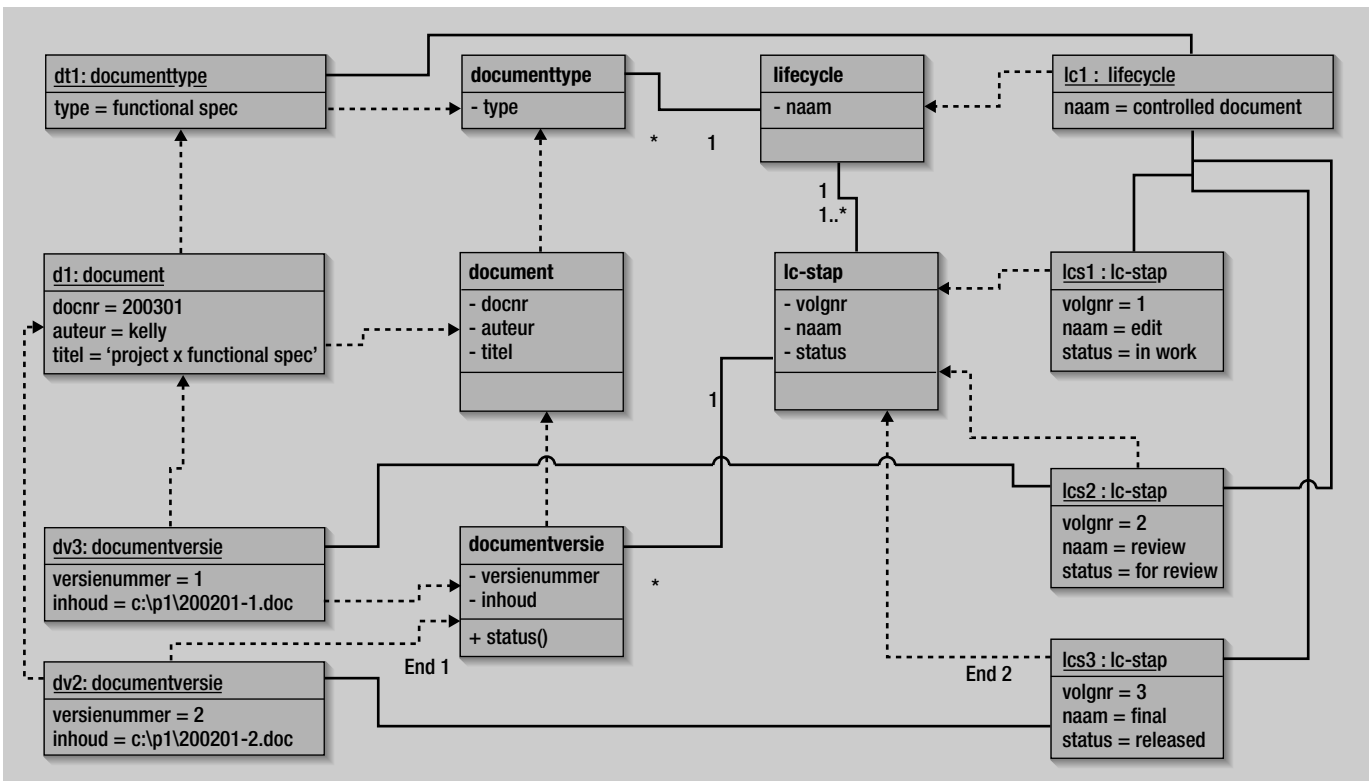
documentversie, dus elke documentversie is een document. Elk documenttype heeft een *life cycle*, die op zich weer een aantal life cycle stappen (lc-stap) heeft. Tenslotte heeft elke documentversie een link naar een *lc-stap*, waardoor de waarde van zijn status bepaald is. Dit laatste kan niet uit UML worden gelezen, maar kan wel worden uitgedrukt in de constraint:

```
<<documentversie, OCL, status =
self.lc-stap.status, 1>>
```

Links en rechts van deze klassen zijn verschillende instanties weergegeven, met aan de linkerkant volgens de corresponderende associaties. Zo is d1 een document van het documenttype dt1. Daaruit volgt:

```
<d1, type, functional spec, 0>
```

Document d1 heeft verder een docnr, een auteur en een titel, maar geen inhoud. De inhoud van een document verschilt namelijk per versie. Documentversie dv3 is een versie van document d1. Het is de eerste versie en de inhoud kan worden gevonden op de aangegeven bestands-locatie, kennelijk dus op een harde schijf. Confronteren we dit model met het normale spraakgebruik, dan zien we als overeenstemming dat we kunnen spreken van de titel, de auteur en het nummer van documentversie dv3. Een verschil is echter dat men in de dagelijkse praktijk vaak wijst naar een afdruk van de inhoud van een document en daarover spreekt als 'dit document'. In de praktijk haalt men dus document en documentversie vaak door elkaar. Dat komt omdat er voor de meeste gebruikers maar één versie de 'juiste' is, zodat het onderscheid voor hem niet relevant is. Voor het documentbeheersysteem, dat moet zorgen dat elke gebruiker altijd de juiste versie krijgt, is dit onderscheid



Figuur 2. Een model voor documentmanagement.

echter uiterst relevant. Een documentbeheerder mag document, versie en inhoud dus nooit door elkaar halen.

In figuur 2 is ook te lezen dat Documenttype dt1 gelinkt is met lifecyle lc. Documenten van dit type behoren dus gecontroleerd te worden, wat betekent dat er in de life cycle drie stappen worden onderscheiden: *in work* (de auteur is bezig maar het document is nog niet gereed), *for review* (het document is gereed, maar moet nog gecontroleerd worden) en *released* (het document mag aan derden ter inzage worden gegeven). Documentversie dv3 is gelinkt aan life cycle-stap lcs2. Daaruit volgt voor de status van dv3 automatisch de waarde 'for review', hetgeen voor ingewijden betekent dat de auteur het document als gereed beschouwt, maar dat het nog niet is vrijgegeven voor inzage door derden. Overigens is er een documentversie dv2 met status = released. Kennelijk is er uit de review van de eerste versie een wijziging voortgekomen die heeft geleid tot de tweede versie. Als iemand vraagt om het document te mogen inzien, moet hem deze versie worden gepresenteerd.

Het aantal digitaal beschikbare documenten explodeert

CONCLUSIE

Dit artikel heeft de lezer een beetje hersengymnastiek laten doen. Het subtiele onderscheid tussen document en documentversie laat zich zonder classificatie moeilijk beschrijven. Dit kan best verwarrend zijn, want het valt niet mee om voortdurend tussen abstractieniveaus op en neer te moeten stappen. Een beetje meer oefening is daarom gewenst en de volgende aflevering gaan we in op de problemen van productfamilies. Met name bij het modelleren daarvan komen meer situaties voor van begrippen, die eigenlijk gemodelleerd moeten worden als klassen van andere begrippen. Daarbij is classificatie een zeer handig middel. Aangezien productstructuren een centrale rol spelen bij productmodellering, zal eerst *aggregatie* moeten worden geïntroduceerd als derde abstractiemechanisme. ●

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