Denken in verschillende abstractieniveaus blijft lastig

Beschrijvingen met productfamilies

Henk Jan Pels

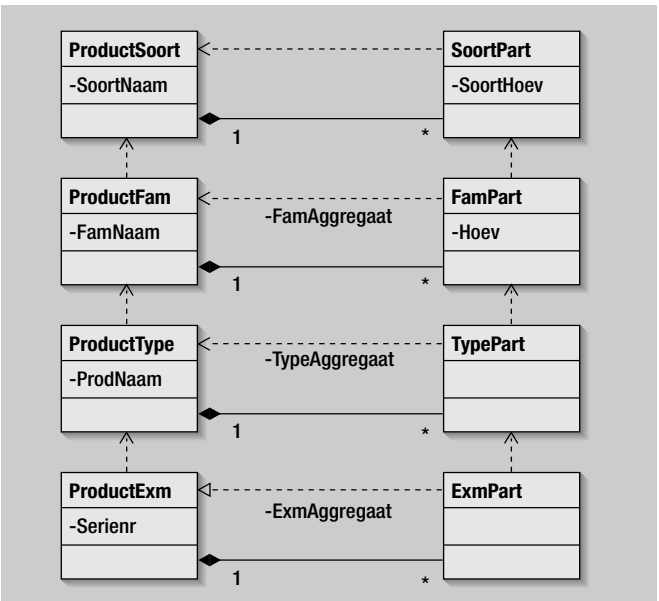
Een van de meest opvallende ontwikkelingen in de industrie gedurende de laatste dertig jaar is de toenemende variëteit, die we, net als een bekende grootgrutter, ook kunnen karakteriseren als "het feest van zoveel meer keus". Voor de tweede wereldoorlog maakte Ford zijn bekende T-model in "any colour, as long as it is black". Geen keuze dus: de consument mocht kopen wat de industrie wenste te maken. Intussen is er de computer gekomen om productieprocessen te besturen. Dat maakt het mogelijk om elk volgend product een beetje anders te maken dan het vorige. De computer zorgt voor alle nodige extra informatie en voor de complexe coördinatie van alle complexe handelingen die nodig zijn om het moderne product te assembleren. De keuzeruimte voor de klant wordt niet meer beperkt door de flexibiliteit van het productieproces, maar door de snelheid waarmee nieuwe producten bedacht en in productie genomen kunnen worden. Een typisch probleem daarbij is dat het op de markt brengen van een nieuw product altijd weer een leerproces is dat gepaard gaat met fouten en dat gebonden is aan de learning curve. Productfamilies bieden een mogelijkheid om dit probleem voor een deel op te lossen.

Een productfamilie is een verzameling verschillende producttypen, ook wel varianten genaamd, die bepaalde eigenschappen hetzelfde hebben. Men zou kunnen zeggen dat het een verzameling producten is die in concreto allemaal verschillend zijn, maar die, beschouwd vanaf een iets hoger abstractieniveau, allemaal identiek zijn. Dat abstractieniveau heet productfamilie. Door nu ontwerp, beproeving en productie-aanloop op productfamilieniveau uit te voeren, kunnen alle varianten met voorspelbare snelheid en kwaliteit gemaakt worden tegen de kosten van een massaproduct.

Om echter productfamilies te kunnen ontwerpen, moet wel heel precies gedefinieerd zijn wat een productfamilie is. Daarbij zijn gegevensmodellen weer een handig hulpmiddel, wat we verderop bespreken.

RAAMWERK

Denken in productfamilies betekent het beschrijven van producten op een aantal abstractieniveaus. Het laagste niveau is dat van het tastbare productexemplaar. Het naast hogere niveau beschrijft het producttype: het gedetailleerde model waarnaar alle identieke exemplaren van dat type gemaakt zijn. Nog een abstractieniveau



Afbeelding 1: UML-diagram voor productfamilies.

Modellering van scratch af aan (6)

In de voorgaande artikelen heeft Henk Jan Pels laten zien hoe de classificatie gebruikt kan worden om de impliciete abstracties in de wereld van document management bespreekbaar te maken en hoe aggregatie een elegante techniek is om productstructuren te modelleren. In dit afsluitende artikel worden beide abstracties gecombineerd, om in productfamilies de gemeenschappelijke karakteristieken van verschillende productvarianten te kunnen beschrijven.

hoger bekijken we de zaak vanuit de productfamilie. Een productfamilie is een klasse van producttypen die verschillen in de waarde voor bepaalde parameters, maar die overigens gelijk zijn. Op het hoogste niveau zien we de productsoort: de klasse van alle soortgelijke producten, zoals telefoon, auto of fiets. Afbeelding 1 laat het principe van de productfamilie zien in de vorm van een UML-datadiagram, waarin gebruik gemaakt wordt van de productstructuren zoals die in het vorige artikel behandeld zijn.

FIETS

We nemen een fiets als voorbeeld om de concepten met betrekking tot productfamilies te verkennen. We beginnen ons model op het hoogste abstractieniveau. Een fiets bestaat in het algemeen uit een frame en twee wielen (waarbij we voor het gemak veronderstellen dat alle overige toeters en bellen van de fiets in het frame verwerkt zijn). Het *concept fiets* wordt gemodelleerd als:

```
<fiets1, P:roductSoort, 'fiets', 1>
```
fiets1 is een object van productsoort 'fiets',
```
<wiel1, P:roductSoort, 'wiel', 1>
```
wiel1 representeert het concept wiel,
```
<frame1, P:roductSoort, 'frame', 1>
```
frame1 staat voor het concept frame

Object	Class	Soortnaam
fiets1	ProductSoort	fiets
frame1	ProductSoort	frame
wiel1	ProductSoort	wiel

Tabel 1: Productsoorten.

Voor de overzichtelijkheid zal ik in dit artikel de tabelvormgebruiken voor groepjes gelijksoortige data-elementen (zie tabel 1):

In deze tabellen bevat de eerste kolom steeds het object, de tweede kolom de klassen waarvan dat object instance is en de overige kolomen de attribuutwaarden. Afgeleide of geërfde attributen en attribuutwaarden worden cursief geschreven. In tabel 2 staat de samenstelling van de fiets genoteerd.

In de eerste regel staat dat het object wielpart1 behoort tot zowel de klasse SoortPart als wiel1. Van wiel1 erft het de waarde voor attribuut Productsoort. De tweede regel laat zien hoe het frame in de fiets past.

FIETSFAMILIE

Nu is vastgelegd hoe een fiets in het algemeen is samengesteld, kunnen we een productfamilie gaan specificeren. In tabel 3a en 3b is de productfamilie toerfiets en zijn samenstellling gemodelleerd.

Bij de toerfiets wordt, als extra detail ten opzichte van de fiets in het algemeen, onderscheid gemaakt tussen het voorwiel en het achterwiel. Dat is nodig omdat het achterwiel een andere naaf heeft ten behoeve van de aandrijving. Bij de productfamilies valt op dat elk object naast de klasse ProductFam, ook behoort tot zijn specifieke ProductSoort. Elke familiepart is als instance toegekend aan zowel de betreffende productfamilie, als aan de betreffende SoortPart. Het overervingsmechanisme van de classificatie zorgt ervoor dat attribuutwaarden voor Famnaam, Soortnaam, Soorthoev en Aggregaat worden geërfd.

Hier stuiten we ook op een subtiel stukje database-ontwerp: men moet voor de aggregatie van Fampart een rolnaam gebruiken, om de samenstelling van de productfamilie te kunnen onderscheiden van die van de productsoort. Omdat namelijk aggregatie een

Object	Class	SoortHoev	Aggregaat	*Productsoort*
wielpart1	SoortPart, wiel1	2	fiets1	*wiel*
framepart1	SoortPart, frame1	1	fiets1	*frame*

Tabel 2: Samenstelling van productsoorten.

Object	Class	FamNaam	*Soortnaam*
toerfiets1	ProductFam, fiets1	toerfiets	*fiets*
toerframe1	ProductFam, frame1	toerframe	*frame*
voorwiel1	ProductFam, wiel1	voorwiel	*wiel*
achterwiel1	ProductFam, wiel1	achterwiel	*wiel*

Tabel 3a: Productfamilies.

Object	Class	hoev	FamAggregaat	*Famnaam*	*Soortnaam*	*Soorthoev*	*Aggregaat*
tfp1	FamPart, toerframe1, framepart1	1	toerfiets1	*toerframe*	*frame*	*1*	*fiets1*
vwp1	FamPart, voorwiel1, wielpart1	1	toerfiets1	*voorwiel*	*wiel*	*2*	*fiets1*
awp1	FamPart, achterwiel1, wielpart1	1	toerfiets1	*achterwiel*	*wiel*	*2*	*fiets1*

Tabel 3b: Samenstelling van productfamilie toerfiets.

vorm van associatie is en omdat associaties worden vastgelegd met associatie-attributen, die dus worden geërfd in classificaties, erft een FamPart als tfp1 attribuut Aggregaat met waarde fiets1. Om toch de eigen samenstelling van de productfamilie te kunnen onderscheiden, hebben we de aggregatie hier een eigen rolnaam gegeven, net als ProductType en ProductExm.

De fietsen binnen de familie toerfiets zullen worden onderscheiden door de kleur en het aantal versnellingen en wielmaat. Dat wordt vastgelegd als *constraint* voor de familie toerfiets1:

```
<<toerfiets1, attribuut,
  kleur: {zwart, groen, geel}, 1>>,
<<toerfiets1, attribuut,
  versnellingen: {1, 3, 7}, 1>>
<<toerfiets1, attribuut, maat: {18, 21, 26}, 1>>.
```

Met het domein wordt ook vastgelegd welke waarden voor de varianten mogelijk zijn. Merk op dat deze constraint alleen geldt voor toerfietsen, en niet voor productfamilies en het algemeen. Hij kan dus niet als attribuut in het UML-diagram worden weergegeven. Het is uiteraard ook mogelijk om parameters afhankelijk van elkaar te maken:

```
<<toerfiets1, OCL, self.kleur = geel ⇒
  self.versnellingen = 7, 1>>.
```

Met andere woorden: gele fietsen worden alleen met een 7 versnellingsnaaf gemaakt.

De kleur van een fiets wordt bepaald door de kleur van het frame, dat dus ook een attribuut kleur moet hebben:

```
<<toerframe1, attribuut, kleur, 1>>.
```

Evenzo moet het achterwiel een attribuut versnellingen hebben en alle onderdelen een attribuut maat:

```
<<achterwiel1, attribuut, maat, 1>>,
<<voorwiel1, attribuut, maat, 1>>,
<<toerframe1, attribuut, maat, 1>>.
```

We bedenken nu dat een fiets zijn kleur in feite erft van het frame, en de maat van de wielen. Nu komen de aggregatieattributen van pas:

```
<<toerfiets1, attribuut kleur: kopie, 1>>,
<<toerfiets1, attribuut maat: kopie, 1>>,
<<toerfiets1, attribuut versnellingen:
  kopie, 1>>.
```

De fiets erft nu zijn parameterwaarden van zijn onderdelen. Op deze manier kunnen meerdere productfamilies worden beschreven. Zo zou er een familie racefiets kunnen zijn, waarbij voor- en achterwiel gelijk zijn om ze snel uitwisselbaar te maken. Zo'n racefiets zou uiteraard andere parameters hebben. Zo zou er in plaats van een enkele versnellingsnaaf, een derailleur zijn met apart te kiezen aantal tandwielen voor en achter. Het vullen van het databasemodel voor deze en andere denkbare fietsfamilies is een mooie oefening voor de lezer.

FIETSTYPE

Als voorbeeld van een fietstype gaan we nu een gele toerfiets ontwerpen met maat 21. We werken deze keer bottom-up en beginnen met de onderdelen. Het voorwiel krijgt als klassen ProductType en voorwiel1. De ProdNaam wordt een droge technische code. Het krijgt 21 als waarde voor de maat en erft de FamNaam voorwiel en de SoortNaam wiel. We ontwerpen een achterwiel maat 21 met bovendien 7 versnellingen. Het frame krijgt eveneens maat 21. De fiets zelf tenslotte krijgt de beeldende naam Gelderland. Het resultaat staat in tabel 4a.

Tabel 4b geeft de samenstelling van de fiets weer. Niet alle geërfde attributen worden getoond, omdat de tabel dan te breed zou worden. Er valt wel een aantal opmerkingen te maken. Omdat het attribuut kleur van de fiets als aggregaat-kopie is gespecificeerd, erft toerfiets Gelderland de parameterwaarden kleur geel, maat 21 en 7 versnellingen van zijn componenten. Omdat deze attributen als eenwaardig zijn gespecificeerd, is het onmogelijk om een fiets te maken met twee verschillende wielen. Als gevolg van de betreffende constraint is bij de keuze geel voor de framekleur alleen de variant 7 versnellingen voor het achterwiel mogelijk.

Op dezelfde manier kunnen meerdere varianten van de toerfiets gespecificeerd worden. Voor de eenvoud blijft het hier bij deze ene.

EEN FIETSEXEMPLAAR

Het wordt nu tijd om een echte fiets te bouwen. Het resultaat staat in Tabel 5. Eerst is een exemplaar gemaakt van voorwieltype vwt1. Het krijgt een uniek serienr toegewezen en erft van zijn type de maat. Het achterwiel volgt op dezelfde manier. Dit erft bovendien 7 versnellingen. Het frame brengt, als instance van frmt1 de kleur geel mee. Hiermee zijn alle onderdelen beschikbaar en kan de

Object	Class	ProdNaam	Kleur	Versnellingen	Maat	*FamNaam*	*Soortnaam*
vwt1	ProductType, voorwiel1	vw21			21	*voorwiel*	*wiel*
awt1	ProductType, achterwiel1	aw21-7		7	21	*achterwiel*	*wiel*
frmt1	ProductType, toerframe1	tf21g	geel		21	*toerframe*	*frame*
ft1	ProductType, toerfiets1	Gelderland	*geel*	7	*21*	*toerfiets*	*fiets*

TABEL 4A: TOERFIETS TYPE GELDERLAND MAAT 21, 7 VERSNELLINGEN, GEEL.

Object	Class	TypeAggregaat	ProdNaam	Kleur	Versnellingen	Maat	FamNaam	Soortnaam	Hoev
vwtp1	TypePart, vwt1, vwp1	ft1	*vw21*			*21*	*voorwiel*	*wiel*	1
awtp1	TypePart, awt1, awp1	*ft1*	*aw21-7*		*7*	*21*	*achterwiel*	wiel	1
frmtp1	TypePart, frmt1, tfp1	*ft1*	*tf21g*	*geel*		*21*	*toerframe*	*frame*	1

TABEL 4B: SAMENSTELLING TOERFIETS TYPE GELDERLAND, MAAT 21, 7 VERSNELLINGEN, GEEL.

Object	Class	Serienr	ExmAggregaat	Kleur	Versnellingen	Maat	Soortnaam
vwe1	ProductExm, vwt1, ExmPart, vwtp1	6001214	*fe1*			*21*	*wiel*
awe1	ProductExm, awt1, ExmPart, awtp1	6001256	*fe1*		*7*	*21*	wiel
frme1	ProductExm, frmt1, ExmPart, frmtp1	219031452711	*fe1*	*geel*		*21*	*frame*
fe1	ProductExm, ft1	*219031452711*		*geel*	*7*	*21*	fiets

TABEL 5: EEN GELE TOERFIETS MAAT 21.

fiets geassembleerd worden. Daarvoor moet echter eerst de fiets gedefinieerd: fe1. Als instance van fiettype ft1 erft hij alle parameters.

Omdat bij productexemplaren sprake is van fysieke aggregatie, zodat een component maar in één samenstelling gebruikt kan worden, gaat de samenstelling iets anders in zijn werk. Uit het UML-diagram (zie afbeelding 1) blijkt dat ExmPart een specialisatie is van ProductExm en niet een instanciatie zoals bij de catalogus-aggregaties op de hogere niveaus. Dit betekent dat het Exmpart niet een apart object is, maar hetzelfde object als het betreffende productexemplaar. Het is voldoende om ExmPart en het betreffende TypePart object als klassen toe te voegen.

Er is nu nog één wens over: het is gebruikelijk dat het serienummer van de fiets gelijk is aan dat van het frame. Dat lukt niet met een aggregaat-attribuut, want dan zouden de serienummers van alle componenten geërfd worden. Er is dus een aparte constraint nodig, die we op het niveau van het ProductType moeten specificeren:

```
<<ProductType, OCL, ∃ o ∈
self.Typepart[o.soortnaam = frame] ⇒
self.Serienr = o.Serienr, 1>>
```

Omdat SoortNaam via de klassificatiehierarchie geërfd wordt, is het mogelijk om de constraint heel algemeen te specificeren: elk producttype dat een frame als component heeft, erft zijn serienummer van dat frame.

CONCLUSIE

Op de hierboven beschreven fiets komen we aan het einde van dit artikel en tevens aan het einde van de serie. Het doel was om te laten zien dat de abstractievormen classificatie en aggregatie, mits voorzien van een handig gekozen semantiek, behulpzaam zijn bij het analyseren en specificeren van productfamilies.

Het voorbeeld van de fiets laat zien dat het mogelijk is om met betrekkelijk weinig woorden en ook met nauwelijks complexe constraints, de belangrijkste concepten te verklaren die voor het denken in productfamilies nodig zijn. Er zijn zelfs families op twee abstractieniveaus: productfamilie en productsoort. Door de overerving is het gemakkelijk om eigenschappen van lagere niveaus, heel generiek op hogere niveaus vast te leggen. Net zo makkelijk is het om hele specifieke eigenschappen van bepaalde productfamilies voor uitsluitend die familie vast te leggen. Alles past in een uiterst eenvoudige datastructuur met slechts acht objectklassen.

Dat het model eenvoudig blijft in de aantallen specificaties, wil nog niet zeggen dat het makkelijk te lezen is. Denken in verschillende abstractieniveaus is voor mensen lastig. Bij het ontwerpen merkt men echter wel veel houvast te hebben aan de notatie. De consistentie en de mechanismen van overerving zijn gemakkelijk stap voor stap te volgen. Dat maakt het mogelijk deze complexe materie nauwkeurig te beschrijven. ●

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

Gaarne reacties

De hier ontwikkelde concepten zijn nieuw en in de laatste jaren geleidelijk, als een soort nevenactiviteit bij ander onderzoek, ontwikkeld. Ze zijn nog niet echt rijp, want daarvoor moeten nog veel meer voorbeelden worden uitgewerkt. Als er dus lezers zijn die werken in een omgeving waar men worstelt met productfamilies, en als die lezers dan ook nog de worsteling om deze serie door te werken met blijvende interesse hebben doorstaan, dan zou ik het leuk vinden om van hen reacties te ontvangen (h.j.pels@tm.tue.nl).