Domeinmodellen

Hoofdstuk 9

Domeinmodellen

INTRODUCTIE

Dit hoofdstuk gaat over de constructie van een domeinmodel. We geven richtlijnen hoe u zo'n model kunt opstellen. We zullen in dit hoofdstuk het boek niet volgen, niet omdat er fouten in staan maar omdat de gekozen aanpak naar ons idee in de praktijk niet goed werkt.

Het domeinmodel is het belangrijkste artefact dat voortkomt uit de business modeling discipline (zie pagina 34 van het tekstboek). Het domeinmodel is daarnaast vaak het eerste duidelijk *objectgeoriënteerde* artefact (use- casebeschrijvingen zijn niet objectgeoriënteerd!).

We willen hier benadrukken dat een domeinmodel wordt opgevat als een beschrijving van alles uit het domein dat voor het systeem van belang kan zijn. Dit is ruimer dan het opstellen van een informatiemodel zoals dat bijvoorbeeld in de cursus Model Driven Development is gedaan, waarbij duidelijk werd toegewerkt naar een relationele database. In een domeinmodel worden ook klassen opgenomen die overeenkomen met elementen uit de context en waarvan het helemaal niet duidelijk is of die ooit een tegenhanger in het uiteindelijke systeem zullen krijgen. Denk bij domeinmodel dus vooral niet alleen aan een datamodel!

LEERDOELEN

Na het bestuderen van dit hoofdstuk wordt verwacht dat u

- de volgende begrippen kunt omschrijven: kapstokklasse, conceptuele klasse, domeinmodel, klassendiagram, multipliciteit, superklasse, subklasse, abstracte klasse, afleidbaar attribuut
- de rol van het domeinmodel binnen UP kunt beschrijven
- weet wat bedrijfsregels zijn en hoe u die kunt aangeven, zowel in een domeinmodel als in tekst
- weet dat een bedrijfsproces in het algemeen drie kanten heeft: aanbod, vraag en transactie
- het ouder-kind patroon, het exemplaarpatroon en het verzamelpatroon kent en kunt toepassen
- een domeinmodel kunt opstellen
- generalisatie juist kunt toepassen.

Studeeraanwijzingen

Dit hoofdstuk vervangt hoofdstuk 9 en 31 van het tekstboek. Het is niet nodig deze hoofdstukken uit het tekstboek te bestuderen.
In dit hoofdstuk gaan we ervan uit dat u over basiskennis beschikt op het gebied van OO en UML. Ter opfrissing of aanvulling van die kennis treft u op de cursussite leereenheid 5 van de cursus Objectgeoriënteerd programmeren in Java 1 aan.

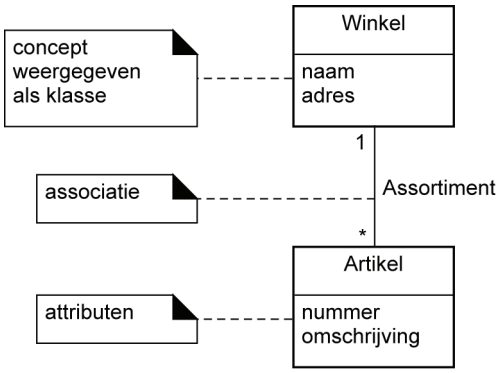
De studielast van dit hoofdstuk is ongeveer 10 uur.

LEERKERN

1 Wat is een domeinmodel?

In deze paragraaf bespreken we wat een domeinmodel is en waarom het zinvol is een domeinmodel op te stellen. Hoewel het UP niets voorschrijft en dus ook niet het opstellen van een domeinmodel, zal dit in de praktijk vrijwel altijd gedaan worden.

Een domeinmodel beschrijft betekenisvolle typen objecten in de context van het probleemdomein en hun onderlinge verband. We geven zo'n model meestal weer in de vorm van een UML-klassendiagram zonder operaties (methoden), aangevuld met tekst voor bijvoorbeeld de weergave van bedrijfsregels.
Figuur 9.1 toont een klein deel van een eenvoudig domeinmodel voor het domein van de POS-casus van Larman. Het model bestaat uit klassen met hun attributen en een associatie tussen de klassen.



FIGUUR 9.1 Een deel van het domeinmodel voor de POS-casus

Het model laat zien dat winkel in het domein van de POS-casus een betekenisvol concept is. Van een winkel leggen we de naam en het adres vast en tevens dat een winkel een assortiment van meerdere artikelen heeft waarvan we nummer en omschrijving vastleggen.

Het domeinmodel vormt een basis voor discussies tussen analisten en domeindeskundigen en is een inspiratiebron en een vertrekpunt voor het ontwerpen van softwareklassen. In dit hoofdstuk benadrukken we deze laatste functie, dat wil zeggen dat het model een goed uitgangspunt moet zijn voor het ontwerp.

Conceptuele klasse

Het domeinmodel bestaat uit *conceptuele* klassen. Zo'n klasse representeert een concept of iets tastbaars (bijvoorbeeld een artikel) uit de werkelijkheid. Als voorbeeld van een concept kunt u denken aan de verkoop van een artikel.
Een object is een zelfstandig iets uit de werkelijkheid dat van belang is voor het te ontwerpen systeem. Objecten met gelijksoortige eigenschappen, bijvoorbeeld artikelobjecten, kunnen we beschouwen als instanties van een klasse Artikel.

Hét model bestaat niet

Bedenk wel dat hét domeinmodel niet bestaat. Een domeinmodel zal altijd interpretaties bevatten en beslissingen. Een andere ontwerper kan mogelijk een andere interpretatie geven en andere beslissingen nemen. Interpretatieverschillen kunnen door een goede communicatie met opdrachtgever en gebruikers wellicht nog wel opgelost worden.

2 De bouwstenen van een domeinmodel

Een domeinmodel is opgebouwd uit klassen die door associaties met elkaar verbonden zijn. Een klasse wordt nader bepaald door zijn attributen die eigenschappen van zo'n klasse vastleggen.
In deze paragraaf gaan we nader in op de individuele bouwstenen van een domeinmodel zodat we daarna met het echte werk kunnen beginnen: "Hoe stel je een goed domeinmodel op ?".

2.1 KLASSEN EN ATTRIBUTEN

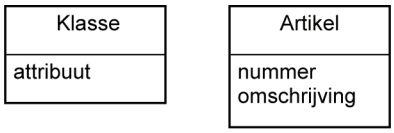
Zoals gezegd representeren klassen betekenisvolle dingen of concepten uit de werkelijkheid. Een klasse wordt gekarakteriseerd door zijn naam. In het algemeen bezitten klassen attributen die eigenschappen van zo'n klasse beschrijven. Ook attributen hebben een naam. Attributen hebben ook een gegevenstype. Hierbij worden zogenaamde primitieve gegevenstypen gebruikt zoals gehele getallen, gebroken getallen, tekst, datums, tijdstippen en booleans.

Als u vertrouwd bent met Java, dan weet u dat in die taal tekst, datums en tijdstippen geen primitieve gegevenstypen zijn. Binnen een domeinmodel benoemen we ze wel als zodanig, want daar houden we geen rekening met dergelijke implementatiedetails, zelfs niet als al gekozen is voor Java als implementatietaal.

Klasse Artikel van figuur 9.1 heeft twee attributen. Attribuut nummer is een geheel getal en omschrijving heeft als type tekst. We kiezen ervoor in deze cursus in het domeinmodel de gegevenstypen niet expliciet aan te geven, hoewel UML daartoe dus wel de mogelijkheid geeft. De keuze van een gegevenstype is in het algemeen pas van belang bij het ontwerp en natuurlijk bij de implementatie.

Naamgeving klassen

Als naam van een klasse kiezen we altijd de enkelvoudsvorm van een zelfstandig naamwoord, bijvoorbeeld Artikel en niet Artikelen, hoewel er in het systeem veel artikelobjecten zullen voorkomen. Voor het weergeven van een klasse in een diagram gebruiken we de UML-notatie van figuur 9.2.

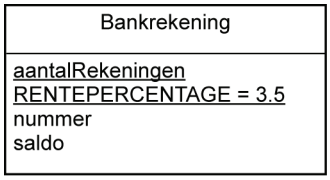


FIGUUR 9.2 UML-notatie voor een klasse met een voorbeeld

Klassenattribuut

De besproken attributen kunnen voor ieder object een andere waarde hebben; zij worden ook wel objectattributen genoemd. Dit in tegenstelling tot klassenattributen. Een klassenattribuut beschrijft een eigenschap

die voor alle instanties dezelfde waarde heeft. Als die waarde constant is, spreken we van een klassenconstante. Klassenattributen worden onderstreept in UML-diagram. Constanten worden daarnaast met hoofdletters weergegeven.
Een klasse Bankrekening kan naast de gegevens van een individuele rekening ook gegevens bevatten die voor alle rekeningen gelden: het rentepercentage en het aantal Rekening-objecten. Figuur 9.3 toont de notatievorm van UML voor deze klasse. Daarbij is aantalRekeningen een klassenattribuut en RENTEPERCENTAGE een klassenconstante.



FIGUUR 9.3 Klassenattribuut en klassenconstante in een domeinmodel

2.2 AFLEIDBARE ATTRIBUTEN

Afleidbaar attribuut

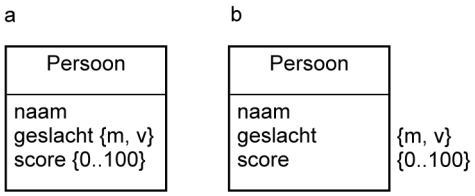
Van een klasse rechthoek willen we lengte, breedte en oppervlakte vastleggen. Attribuut oppervlakte is speciaal: de waarde ervan kunnen we bepalen op basis van een rekenregel: de oppervlakte van een rechthoek = lengte * breedte.
UML heeft voor zulke attributen een speciale notatie: we laten de naam van het attribuut voorafgaan door een slash. Figuur 9.4 toont deze notatie voor klasse Rechthoek.

FIGUUR 9.4 Klasse Rechthoek met afleidbaar attribuut oppervlakte

Het opnemen van een afleidbaar attribuut betekent niet dat dit attribuut ook altijd zal voorkomen in de implementatie. Vaak worden afleidbare attributen niet als attribuut geïmplementeerd maar wordt een methode ontworpen die de berekening uitvoert en de uitkomst daarvan als terugkeerwaarde heeft.

2.3 ATTRIBUTEN MET BEPERKTE WAARDENVERZAMELING

Soms is het wenselijk de mogelijke waarden die een attribuut kan aannemen, vast te leggen of te beperken. We kunnen dan de mogelijke waarden opsommen of de grenzen van het waardenbereik aangeven door middel van een begin en eindwaarde met daartussen twee punten, bijvoorbeeld 1..6. We plaatsen de mogelijke waarden tussen { en } en zetten ze in of vlak naast het klassendiagram. Figuur 9.5 toont een voorbeeld.



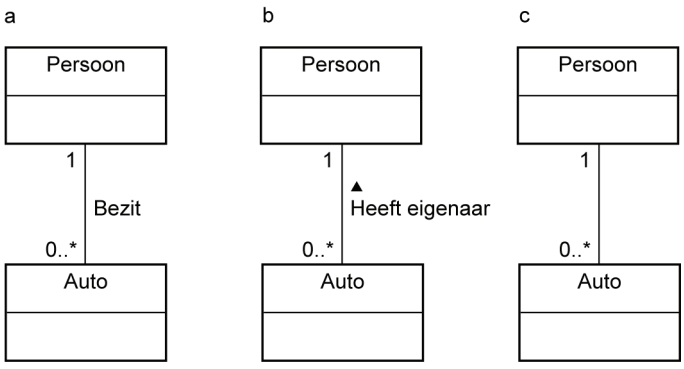
FIGUUR 9.5 Klasse Persoon met beperkingen in de klasse (a) en naast de klasse (b)

2.4 ASSOCIATIES

Een associatie is een verband tussen twee klassen. In een domeinmodel heeft een associatie geen richting: De associatie in figuur 9.1 geeft aan dat klasse Winkel een assortiment van artikelen heeft en dat een artikel behoort tot het assortiment van een winkel. In een UML-diagram geven we een associatie aan met een verbindingslijn tussen twee klassen. Associaties hebben een naam, meestal een werkwoordsvorm maar ook wel een zelfstandig naamwoord. De namen van associaties schrijven we met een hoofdletter, zie figuur 9.6a. Omdat vaak de naam van de associatie Bezit of Heeft is en de aard van de associatie duidelijk is, laten we vaak die naam weg, zoals in figuur 9.6c. Als we een naam gebruiken is de interpretatie in de leesrichting van links naar rechts of van boven naar beneden. Als we de leesrichting expliciet willen aangeven gebruiken we een pijltje, zie figuur 9.6b.

Multipliciteit

Een associatie kan ook een beperking vastleggen voor het aantal instanties van een klasse die geassocieerd kunnen zijn met een instantie van de met de associatie verbonden klasse. Deze beperking wordt *multipliciteit* genoemd. We bekijken het voorbeeld van een persoon die geen, een of meerdere auto's kan bezitten, terwijl een auto altijd bij één persoon behoort. Deze beperkingen leggen we vast met een notatie aan het begin en het einde van de verbindingslijn tussen de twee klassen, zie figuur 9.6.



FIGUUR 9.6 Associaties met verschillende notaties

Aan iedere zijde van een associatie geven we de multipliciteit aan. Deze bestaat uit twee delen: het minimaal aantal en het maximaal aantal objecten dat geassocieerd kan worden. We geven de multipliciteit zo weer:

minimale multipliciteit .. maximale multipliciteit.

In plaats van *n..n* gebruiken we meestal *n* (met name 1 komt vaak voor) en voor 0..* gebruiken we meestal alleen * waarbij de * staat voor een willekeurig getal dat groter dan of gelijk aan 0 is. Als we de bovengrens weten, kunnen we die ook expliciet aangeven, bijvoorbeeld 1..5.

Tekenconventie

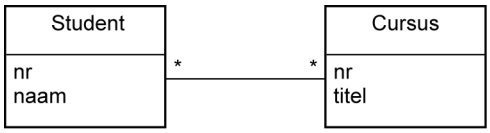
Het is van groot belang een domeinmodel goed gestructureerd te tekenen. We hanteren daarbij de conventie dat we in geval van een 1-op-veel associatie deze met de 1-kant boven tekenen zoals in figuur 9.6. Associaties met aan beiden zijden een gelijke multipliciteit zullen we vaak horizontaal tekenen.

Maximale multipliciteit

Bij het opstellen van een domeinmodel is de maximale multipliciteit het belangrijkst: gaat het om een 1-op-1 of 1-op-veel associatie? De minimale multipliciteit speelt een rol bij de implementatie, waarvoor we het ontwerpmodel gebruiken. Voorlopig gebruiken we meestal een * voor een 1-op-veel associatie en een 1 voor een 1-op-1 associatie waarbij we de minimale multipliciteit nog even in het midden laten.
Als we als context van figuur 9.6b een garage nemen, zouden we 0..1 als multipliciteit bij Persoon moeten opnemen omdat een nieuwe auto pas als deze verkocht wordt een eigenaar krijgt. Vaak echter is de keuze voor de minimale multipliciteit niet zo duidelijk.

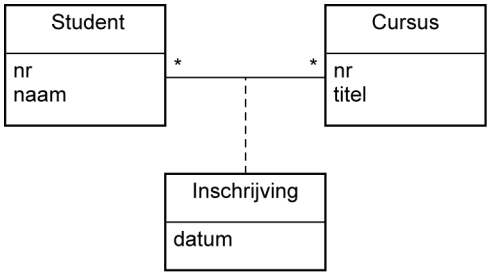
2.5 ASSOCIATIES UITBREIDEN

We bekijken de situatie waarbij we willen vastleggen dat een student ingeschreven kan zijn voor cursussen en dat bij een cursus studenten ingeschreven kunnen zijn. Figuur 9.7 toont het klassendiagram.



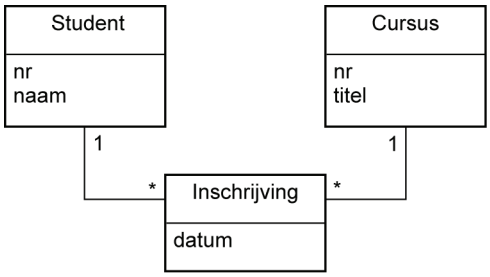
FIGUUR 9.7 Student en cursus

Stel dat we de inschrijfdatum van een student voor een cursus willen vastleggen. Attribuut inschrijfdatum kunnen we niet opnemen in klasse Student omdat deze voor meerdere cursussen ingeschreven kan zijn en attribuut inschrijfdatum maar één waarde kan hebben. Evenmin kunnen we inschrijfdatum opnemen in klasse Cursus. Wat we eigenlijk willen is een labeltje hangen aan de link tussen een Student-object en een Cursus-object. UML kent hiervoor de zogenaamde associatieve klasse. Figuur 9.8 toont het gewijzigde klassendiagram van figuur 9.7.



FIGUUR 9.8 Klassendiagram met een associatieve klasse

Omdat programmeertalen geen direct equivalent van een associatieve klasse kennen, zetten we de associatieve klasse om naar een gewone klasse. Figuur 9.9 toont het resultaat.



FIGUUR 9.9 Aangepast klassendiagram zonder associatieve klasse

We zien in figuur 9.9 dat bij een student meerdere inschrijvingen kunnen horen. Bij iedere inschrijving hoort één cursus. Zo kennen we voor iedere cursus waar een student voor ingeschreven is de datum. Evenzo zijn er per cursus meerdere inschrijvingen en per inschrijving is er een bijbehorende student. Zo is voor iedere cursus de datum bekend waarop een student zich voor die cursus heeft ingeschreven. We zien dat figuur 9.9 dezelfde informatie representeert als figuur 9.8.

We zullen verder in deze cursus geen gebruik maken van associatieve klassen, maar die altijd direct omzetten naar een gewone klasse.

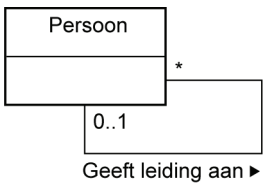
2.6 BIJZONDERE ASSOCIATIES

In deze paragraaf kijken we naar twee bijzondere vormen van associaties: de reflexieve associatie waarbij er een associatie is van een klasse met zichzelf en een meervoudige associatie tussen twee klassen.

Als voorbeeld van een associatie van een klasse met zichzelf kijken we naar een bedrijf waarin we de hiërarchie van leidinggeven willen modeleren. Een persoon kan leiding geven aan andere personen. Iedereen behalve de hoogste baas valt onder de leiding van een persoon. Deze situatie is weergeven in figuur 9.10.

Reflexieve associatie

We noemen een associatie van een klasse met zichzelf een *reflexieve associatie*.



FIGUUR 9.10 Reflexieve associatie

Merk op dat het nu nodig is de associatie te benoemen en daarbij ook expliciet de leesrichting aan te geven; anders zien we niet of een persoon leiding kan geven aan verschillende personen of een persoon leiding kan krijgen van verschillende personen. De leesrichting geven we aan met behulp van het zwarte pijltje bij de naam van de associatie.

Meervoudige associatie

Soms bestaan er tussen twee klassen verschillende relaties. Die kunnen dan met verschillende associaties worden weergegeven. Dit wordt een *meervoudige associatie* genoemd. Als voorbeeld kijken we naar de volgende situatie: op een school wordt vastgelegd voor welke vakken een docent bevoegd is. Daarnaast wordt vastgelegd welke vakken daadwerkelijk door een docent gegeven worden. Het model van figuur 9.11 toont deze situatie.

FIGUUR 9.11 Docent en vakken

Merk op dat het ook nu nodig is de associaties te benoemen om onderscheid te maken tussen de verschillende relaties.

3 Een domeinmodel ontwerpen: aanpak

In deze paragraaf tonen we een aanpak om domeinmodellen te ontwerpen. Het ontwerpen van een domeinmodel is ambachtelijk; het is niet goed mogelijk regels te geven die, indien we ze navolgen, automatisch een goed domeinmodel opleveren.
Onze aanpak is gebaseerd op het bestuderen van voorbeelden en het (her)kennen van patronen gecombineerd met het opdelen van het domein op basis van bedrijfsprocessen.

Bedrijfsprocessen

De informatiesystemen die we ontwerpen, zullen vaak ondersteuning bieden bij bedrijfsprocessen, bijvoorbeeld orderafhandeling of voorraadbeheer. Een bedrijfsproces kan gezien worden als een samenwerking van mensen en systemen binnen een bedrijf om een product of dienst te leveren aan de klant. Voor die samenwerking is coördinatie nodig. Over het proces zelf en de coördinatie kunnen gegevens geregistreerd worden in een informatiesysteem. Inzicht in bedrijfsprocessen is dan ook van groot belang voor het ontwerpen van informatiesystemen.
In de UP-discipline business modeling worden bedrijfsprocessen beschreven. Het tekstboek besteed hier verder geen aandacht aan. Voor het in kaart brengen van bedrijfsprocessen worden wel UML activity diagrams gebruikt (zie hoofdstuk 28 van het tekstboek). Deze diagrammen zullen we verder in de cursus echter niet gebruiken.

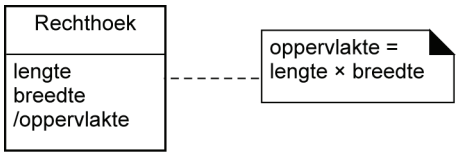
Kapstokklasse

We zullen ons ontwerp altijd beginnen met een ‘kapstokklasse’: een klasse die het hele te ontwerpen systeem representeert. Daarna breiden we ons model uit met de drie verschillende ‘kanten’ die we meestal kunnen onderscheiden in een model: de aanbodkant, de vraagkant en tenslotte de transactiekant.

Bedrijfsregels

In een domeinmodel geven we niet alleen de structuur van het domein weer maar ook de beperkingen (constraints) en afleidings- of rekenregels.
We duiden deze beperkingen en regels aan met de algemene term bedrijfsregels.

De multipliciteit van een associatie is een voorbeeld van een beperking die we in een domeinmodel kunnen weergeven Een ander voorbeeld is een beperking van de mogelijke waarden voor een attribuut.
Indien mogelijk zullen we bedrijfsregels opnemen in het domeinmodel, zoals bij multipliciteiten bij associaties en mogelijke waarden voor attributen. Ook met behulp van commentaar kunnen bedrijfsregels in het model worden opgenomen, zoals in figuur 9.12, waar een rekenregel is toegevoegd.
Echter lang niet altijd kunnen we bedrijfsregels kwijt in ons domeinmodel. In dat geval formuleren we de bedrijfsregels in natuurlijke taal. Een voorbeeld van een beperking in natuurlijke taal is de regel bij figuur 9.11: “een docent mag alleen een vak geven als hij daar ook voor bevoegd is”. Ook als er veel bedrijfsregels zijn, is het vaak overzichtelijker ze niet direct binnen het model op te nemen, maar ze apart bij het domeinmodel op te nemen in de vorm van een tekstdocument.



FIGUUR 9.12 Klasse Rechthoek met bedrijfsregels

3.1 DE BASIS

Om een goed domeinmodel te kunnen ontwerpen hebben we kennis van en inzicht in het betreffende domein nodig. Als eerste bron daarvoor komen de use-casebeschrijvingen in aanmerking. Voor het opstellen van use cases worden vaak interviews met opdrachtgever en gebruiker gehouden. Daarnaast zijn er meestal additionele documenten beschikbaar waaruit we kennis kunnen putten, bijvoorbeeld allerlei formulieren die gebruikt worden. Zo zou voor de POS-casus de kassabon een goede bron kunnen zijn.
In dit hoofdstuk zullen we domeinmodellen opstellen voor de POS-casus uit het tekstboek, voor de dvd-winkel (uit de zelftoets van hoofdstuk 6) en voor het cursusregistratie-casus (CRS) uit het werkboek. Voor het goed begrijpen van dit hoofdstuk is het raadzaam de genoemde casusbeschrijvingen met de bijbehorende use-casebeschrijvingen nogmaals te bestuderen en bij de hand te hebben.

3.2 DE KAPSTOKKLASSE

Bij het ontwerpen van een domeinmodel is het handig met een vaste klasse te beginnen. Deze klasse representeert het te ontwerpen systeem en noemen we een kapstokklasse omdat we vanuit deze klasse alle andere klassen van het model kunnen benaderen. Ook kunnen we de kapstokklasse gebruiken als beheerder van andere klassen; we verbinden deze klassen dan vaak met een 1-op-veel associatie met de kapstokklasse. Meestal betreft het een “extra” klasse zonder attributen die we later ook niet zullen implementeren maar voor het opstellen van een domeinmodel heeft zo’n kapstokklasse veel nut. Soms kunnen we één van de domeinklassen de rol van kapstokklasse laten spelen. In dat

geval zal de kapstokklasse vaak wel attributen hebben. In het geval van de POS-casus kunnen we de klasse Store als kapstokklasse nemen. Deze representeert namelijk de hele winkel, waarvan alle andere domeinklassen deel uit maken.
Voor het vinden van een kapstokklasse kijken we eerst in het domein of er een geschikte klasse is die deze rol kan spelen. Als deze er niet is, voegen we zelf zo'n klasse toe.

Studeeraanwijzing

Omdat het domeinmodel voor de POS-casus later in de cursus nog een rol speelt, blijven we bij het opstellen daarvan zo dicht mogelijk bij het tekstboek. Daarom geven we voor deze casus aan alle klassen Engelse namen.

OPGAVE 9.1
Welke klasse neemt u als kapstokklasse bij de dvd-winkel en welke bij de CRS-casus?

3.3 AANBOD, VRAAG EN TRANSACTIE

Vaak hebben bedrijfsprocessen betrekking op het aanbieden van diensten of artikelen in de meest ruime zin. We kunnen deze bedrijfsprocessen opdelen in drie delen: aanbod, vraag en transactie. De aanbodkant beschrijft het aanbod van artikelen of diensten en alles wat daar voor nodig is. De vraag kant beschrijft wie de afnemers van de dienst of artikelen zijn. De transactiekant legt transacties vast tussen afnemers en diensten of artikelen.
Door deze opdeling kunnen we de complexiteit terugbrengen door een bedrijfsproces deel voor deel te modelleren.
De POS-casus betreft het aanbod en de verkoop van artikelen. De artikelen vormen de aanbodkant. De klanten vormen in dit geval de afnemers van de dienst of artikelen en zijn de vraagkant. De transactie betreft de verkoop van artikelen aan een klant en vormt dus een koppeling tussen de vraagkant en de aanbodkant.
Bij de aanbodkant rekenen we ook alles wat nodig is om het aanbod te beheren en transacties mogelijk te maken. Voor de POS-casus zouden we hierbij aan de kassa's en het personeel dat achter de kassa zit kunnen denken.

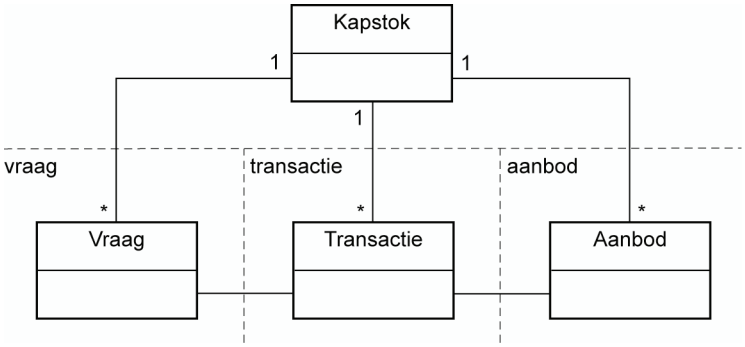
OPGAVE 9.2
Beschrijf kort de dvd-winkel en CRS-casus in termen van vraag, aanbod en transactie.

Vraag, aanbod en transactie zijn relatief

Merk op dat de begrippen vraag, aanbod en transactie relatief zijn ten opzichte van een bedrijfsproces. Bij de CRS-casus rekenen we het aanbod van cursussen met de daarbij horende docenten tot de aanbodkant. Maar natuurlijk is er ook een proces "toewijzen docenten aan cursussen" aan voorafgegaan. In dat proces vormen de cursussen de aanbodkant, de docenten de vraagkant en het vastleggen welke cursussen door welke docenten gegeven worden de transacties. Met evenveel recht kunnen we ook zeggen dat de docenten de aanbodkant vormen en de cursussen de vraagkant. De keuze wat de aanbodkant is en wat de vraagkant is niet zo relevant; via de transactie verbinden we de vraagkant, de aanbodkant en de transactiekant tot één model waarbij de aanduiding vraag, aanbod en transactie weer verdwijnen.

Op basis van een verdeling in aanbod, vraag en transactie kunnen we het domeinmodel opdelen in kleinere stukken, hetgeen het opstellen van een model weer wat eenvoudiger maakt. We maken dan gebruik van het al-oude verdeel en heers-principe. Bij het modelleren van de transacties brengen we de aanbodkant en de vraagkant met elkaar in contact en ontstaat op natuurlijke wijze het complete model.

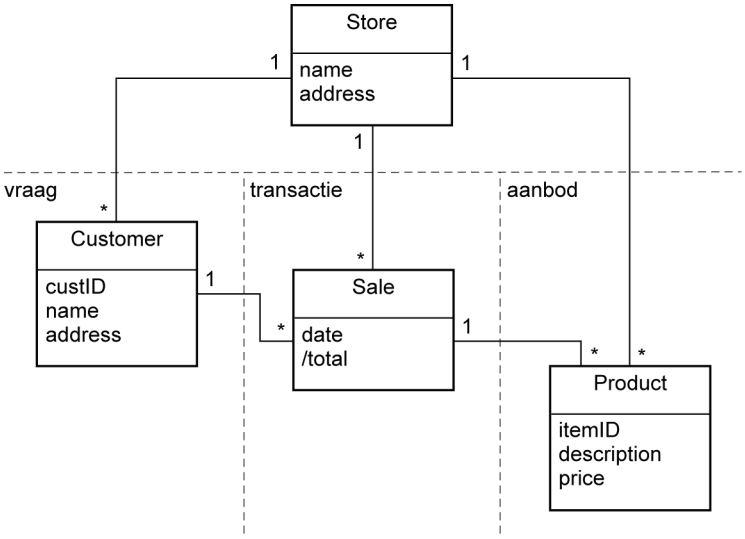
In combinatie met de kapstokklasse kunnen we op basis van de genoemde verdeling een generiek domeinmodel opstellen, zie figuur 9.13.



FIGUUR 9.13 Generiek domeinmodel

De kapstokklasse is dus “kant-overstijgend”, dat wil zeggen dat deze niet tot een specifieke kant behoort.

Een eerste opzet van een domeinmodel voor de POS-casus ziet u in figuur 9.14. Hoewel er nog veel aan het model valt te verbeteren en uit te breiden, toont het de belangrijkste klassen en de plaats van die klassen met betrekking tot aanbod, vraag en transactie. Voor het opstellen van het model hebben we gebruik gemaakt van de casusbeschrijving en vooral van de use-casebeschrijving “process sale”. Voor de klant hebben we voorlopig wat fictieve gegevens gebruikt .



FIGUUR 9.14 Voorlopig domeinmodel voor de POS-casus

De klassen Customer, Sale en Product zijn niet op dezelfde hoogte getekend. Dit komt overeen met de tekenconventie beschreven in paragraaf 2.4. Bij een 1-op-veel relatie wordt de klasse aan de 1-kant hoger getekend dan de klasse aan de veel-kant

OPGAVE 9.3

Ontwerp voor de dvd-winkel een zelfde globaal model als in figuur 9.15.

3.4 PATRONEN

Wie regelmatig domeinmodellen opstelt, ervaart dat ieder domeinmodel weliswaar zijn eigen specifieke problemen kent maar dat vaak dezelfde patronen opduiken. De boeken 'Analysis patterns' van Martin Fowler en 'Data model patterns' van David Hay bevatten domeinmodellen die specifiek zijn voor verschillende toepassingsgebieden. Deze modellen zouden we als uitgangspunt kunnen nemen. Bij nadere beschouwing zijn binnen deze modellen echter ook weer (eenvoudigere) patronen te herkennen. In paragraaf 4 bespreken we een aantal van die eenvoudige patronen die essentieel zijn voor het opstellen van een domeinmodel.

3.5 UML-ONTWERPOMGEVINGEN

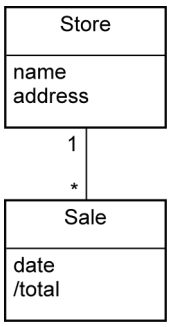
Het ontwerpen van een domeinmodel is een creatief en iteratief proces. Tijdens dit proces zal het model voortdurend gewijzigd en uitgebreid worden. Onze ervaring hierbij is dat u het beste papier, potlood en gum of een whiteboard als ontwerpgereedschap kunt gebruiken. Pas als het model stabiel geworden is, loont het de moeite om het in te voeren in een geautomatiseerde ontwerpomgeving. Er bestaan veel van dergelijke omgevingen, waarbij we twee soorten kunnen onderscheiden. Dat zijn tekenomgevingen en ontwikkelomgevingen die vaak ook code genereren. De laatste categorie dwingt u vaak al tot allerlei implementatiekeuzes zoals bijvoorbeeld het gegevenstype van een attribuut. Een tekenomgeving biedt de ontwerper veel meer vrijheid en is dan ook te prefereren voor het vastleggen van een domeinmodel.
In het boek van Larman ziet u dat bij de Monopoly-casus veelvuldig het whiteboard in combinatie met een digitale camera gebruikt wordt als gereedschap.

4 Eenvoudige patronen

In deze paragraaf bespreken we een aantal eenvoudige patronen die in de praktijk veel voorkomen. Achtereenvolgens besteden we aandacht aan het ouder-kindpatroon, het verzamelpatroon en het exemplaarpatroon

4.1 HET OUDER-KINDPATROON

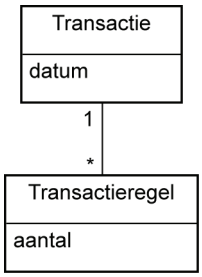
Bij het ouder-kindpatroon gaat het om niet veel anders dan een benaming voor iedere 1-op-veel associatie tussen twee klassen. Onze tekenconventie zegt ons om de ouder boven het kind te tekenen. Vaak zal de ouderklasse de kindklasse beheren. We hebben al gezien dat de kapstokklasse als beheerder optreedt van andere domeinklassen.
In figuur 9.15, een deel van figuur 9.14, ziet u dat kapstokklasse Store de verschillende Sales beheert.



FIGUUR 9.15 Ouder-kindpatroon

4.2 HET VERZAMELPATROON

Een bijzondere vorm van het ouder-kindpatroon is het verzamelpatroon. Het meest bekende voorbeeld is een transactie die uit meerdere onderdelen bestaat, die we transactieregels noemen. Zo'n transactie wordt gekenmerkt door algemene gegevens bijvoorbeeld de datum en de klant voor wie de transactie bestemd is. Een transactieregel correspondeert met een onderdeel van de transactie en bevat specifieke informatie over dat onderdeel zoals aantal, artikelsoort enzovoort. Figuur 9.16 toont de kern van deze situatie.



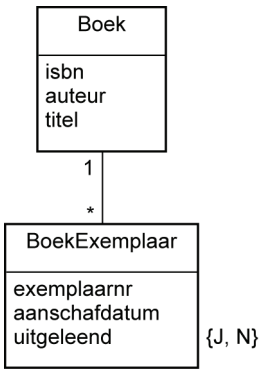
FIGUUR 9.16 Verzamelpatroon

De transactie uit de werkelijkheid wordt dus gemodelleerd met behulp twee samenhangende klassen.

OPGAVE 9.4
Kunt u bij de CRS-casus en de dvd-winkel ook een voorbeeld van het verzamelpatroon noemen?

4.3 HET EXEMPLAARPATROON

Het zal u niet verbazen dat in het domeinmodel van een bibliotheek een klasse Boek zal voorkomen met bijvoorbeeld als attributen isbn, auteur en titel. Deze klasse modelleert echter *niet* het fysieke boek dat u leent. Een bibliotheek heeft van een boek immers vaak meer dan één exemplaar en al die exemplaren hebben dezelfde isbn, titel en auteur.
We kunnen ook de exemplaren van het boek in de bibliotheek modelleren. Zo'n boekexemplaar wordt bijvoorbeeld geïdentificeerd door een apart exemplaarnummer (bijvoorbeeld gekoppeld aan de streepjescode). De juiste modellering hiervan ziet u in figuur 9.17.

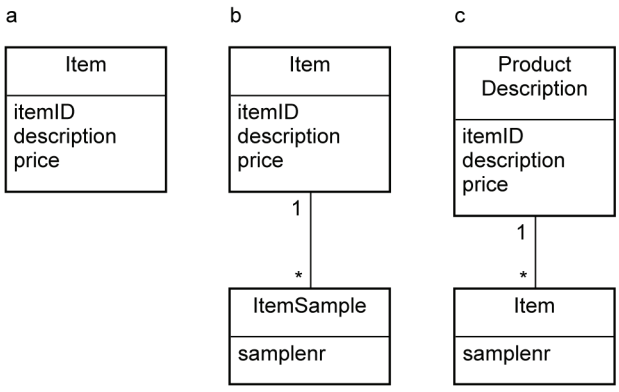


FIGUUR 9.17 Exemplaarpatroon

Beschrijvende klasse

De klasse Boek beschrijft de gegevens die voor alle exemplaren hetzelfde zijn, de individuele gegevens staan in klasse BoekExemplaar. Klasse Boek wordt ook wel een *beschrijvende klasse* genoemd.
Merk op dat het model van figuur 9.17 ook voor voorraadbeheer gebruikt kan worden: Het aantal instanties van BoekExemplaar bij een bepaalde instantie van Boek, geeft aan hoeveel exemplaren er van dat Boek in de bibliotheek aanwezig zijn, al dan niet uitgeleend.

Het exemplaarpatroon komt in de praktijk verassend vaak voor. Zo kunt u bijvoorbeeld denken aan een klasse Film met bijbehorende klasse Vertoning. Ook bij de POS-casus komt het exemplaarpatroon voor en wel bij de klasse Item. Item kan zowel gebruikt worden voor een individueel item (bijvoorbeeld dit pak melk) of voor alle exemplaren van een bepaald soort item (bijvoorbeeld een pak melk). Bij het modelleren moeten we nauwkeurig aangeven wat we bedoelen en natuurlijk rekening houden met het doel waarvoor de applicatie ontworpen wordt.



FIGUUR 9.18 Exemplaarpatroon bij de POS-casus

Figuur 9.18a toont alleen een klasse Item. Deze klasse representeert dus alle exemplaren van een bepaald Item. In figuur 9.18b zien we het exemplaarpatroon toegepast: ItemSample bevat nu de individuele streepjescode van ieder Item-exemplaar. Hier is Item de beschrijvende klasse en ItemSample de exemplaarklasse. Figuur 9.18c toont tenslotte de klassen met de naamgeving zoals Larman die gebruikt. Nu is Item de exemplaarklasse en ProductDescription de beschrijvende klasse.

OPGAVE 9.5
Zowel bij de CRS-casus als bij de dvd-winkel is een exemplaarpatroon te onderscheiden. Modelleer voor beide alleen dit deel van het domeinmodel. U mag de attributen van de klassen nog weglaten.

OPGAVE 9.6
Bij het modelleren van een domeinmodel voor een luchtvaartmaatschappij gebruiken we de tabel van figuur 9.19. Stel op basis van deze tabel een domeinmodel op.

vertrektijden

heenvlucht Amsterdam - Barcelona Vlucht TA231

← week eerder

zo 19 okt	ma 20 okt	di 21 okt	wo 22 okt	do 23 okt	vr 24 okt
06:15	06:15	06:15	06:15	06:15	06:15
12:25	12:25	12:25	12:25	12:25	12:25
18:40	18:40	18:40	18:40	18:40	18:40

terugvlucht Barcelona - Amsterdam Vlucht TA232

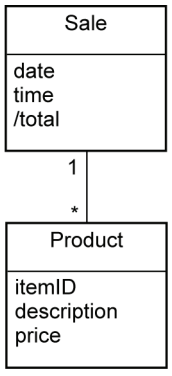
← week eerder

zo 19 okt	ma 20 okt	di 21 okt	wo 22 okt	do 23 okt	vr 24 okt
09:05	09:05	09:05	09:05	09:05	09:05
15:20	15:20	15:20	15:20	15:20	15:20
21:35	21:35	21:35	21:35	21:35	21:35

FIGUUR 9.19 Tabel met vluchtgegevens

4.4 PATRONEN BIJ DE POS-CASUS

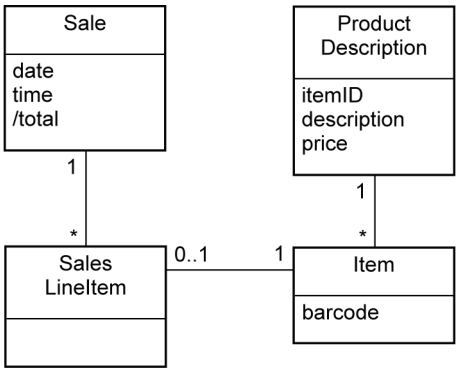
Oorspronkelijk (zie figuur 9.14) hadden we het verband tussen Sale en Product gemodelleerd zoals in figuur 9.20 te zien is.



FIGUUR 9.20 Eerste deelmodel voor Sale en Product

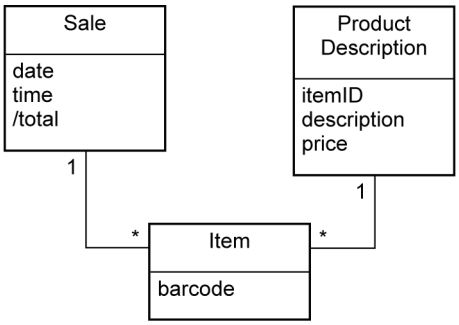
Product representeert een exemplaar van een bepaald product. Het toegepaste patroon is het ouder-kindpatroon. Op basis van onze kennis van patronen, kunnen we een beter model maken. We moeten in ieder geval het verzamelpatroon toepassen: Sale met SalesLineItems.

Bovendien kunnen we het exemplaarpatroon toepassen, zoals we in figuur 9.18c hebben gezien. Toepassing van deze patronen levert het model op van figuur 9.21.



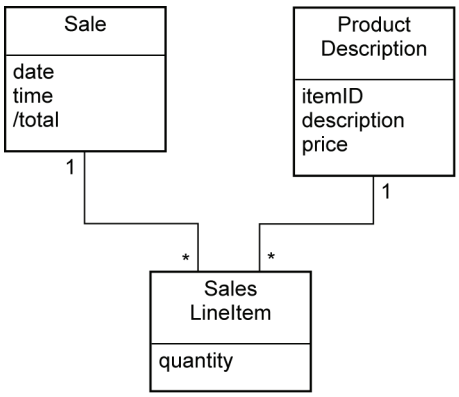
FIGUUR 9.21 Verzamelpatroon en exemplaarpatroon toegepast op POS-casus

Omdat Item een bepaald exemplaar representeert moet de multipliciteit van SalesLineItem ten opzichte van Item 1 zijn: we registreren dus de verkoop van elk artikel afzonderlijk. We zien dat SalesLineItem geen attributen bevat en omdat Item een multipliciteit 0..1 heeft ten opzichte van SalesLineItem, kunnen we deze laatste klasse weglaten. We krijgen dan de situatie van figuur 9.22.



FIGUUR 9.22 Vereenvoudigd model van figuur 9.21

In de context van een supermarkt is het de vraag of het toepassen van het exemplaarpatroon nodig is. Willen we ieder verkocht artikel afzonderlijk te registreren of stellen we ons tevreden met de registratie van het soort artikel en het aantal. Voor een supermarkt kunnen we laatste vraag met ja beantwoorden: het gaat er niet om of we precies weten welke pakken melk er zijn verkocht, maar het gaat erom dat we weten hoeveel pakken melk er zijn verkocht. We krijgen dan een model met het verzamelpatroon maar niet met het exemplaarpatroon, zoals in figuur 9.23.

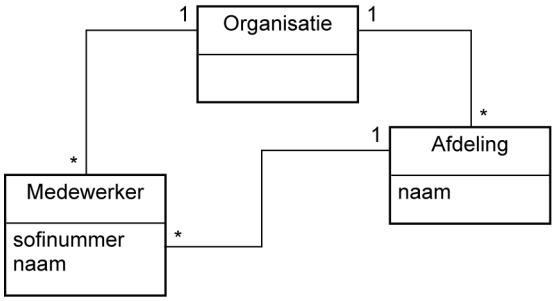


FIGUUR 9.23 Verzamelpatroon zonder exemplaarpatroon voor de POS-casus

Aan het voorgaande voorbeeld kunt u zien dat het bij het modelleren niet gaat om het zo nauwkeurig mogelijk representeren van de werkelijkheid maar om het modelleren van een geschikte interpretatie van die werkelijkheid.

5 Geschiedenis modelleren

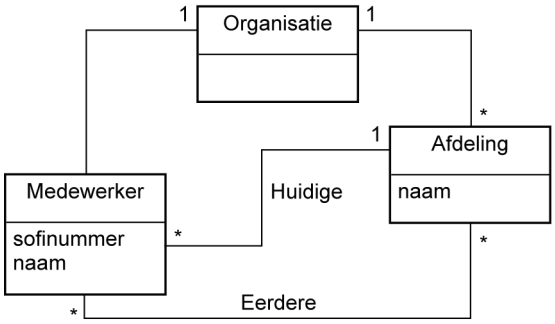
Bij het ontwerpen van een domeinmodel is het van belang te weten of het systeem op ieder moment een 'momentopname' moet zijn of dat ook de geschiedenis meegenomen moet worden. We illustreren dit aan de hand van een voorbeeld.
Een organisatie kent medewerkers en afdelingen. Iedere medewerker werkt op een afdeling en iedere afdeling heeft meerdere medewerkers. Figuur 9.24 toont een sterk vereenvoudigd domeinmodel voor deze situatie.



FIGUUR 9.24 Domeinmodel voor een organisatie

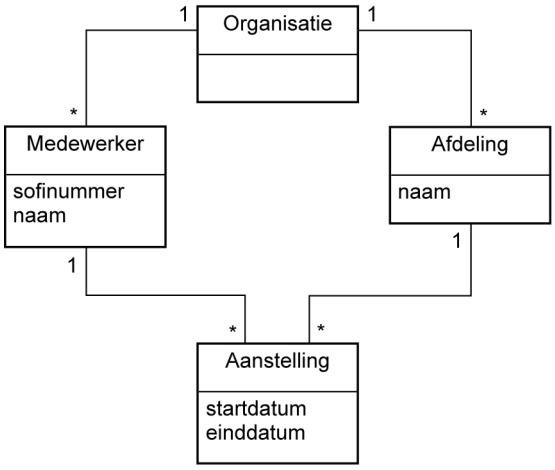
Het model is een momentopname: bij een medewerker wordt de afdeling bijgehouden waarop deze nu werkt. Als een medewerker van afdeling verandert, wordt eenvoudigweg de link naar een andere afdeling gelegd. Het is daarna niet meer mogelijk te achterhalen wat de voorgaande afdeling van een werknemer was.

Als het voor de organisatie van belang is te weten op welke afdeling een medewerker nu werkt maar ook op welke afdelingen de medewerker eerder gewerkt heeft, moeten we ons model wijzigen. Als eerste moeten we de multipliciteit van Medewerker ten opzichte van Afdeling wijzigen. Daarmee zijn we er echter nog niet want we weten dan niet wat de huidige afdeling van de werknemer is. Een eerste oplossing voor dit probleem is het opnemen van een extra associatie, zie figuur 9.25.



FIGUUR 9.25 Aangepast domeinmodel voor een organisatie

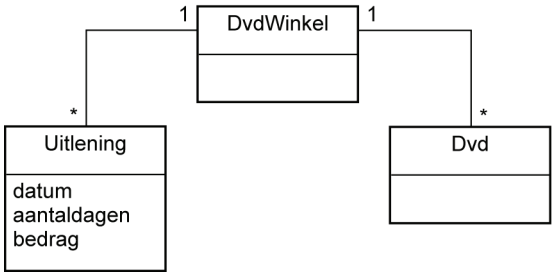
Een tweede oplossing is om een klasse Aanstelling te ontwerpen met een begindatum en einddatum en deze te koppelen aan Medewerker en Afdeling. Figuur 9.26 toont deze oplossing.



FIGUUR 9.26 Alternatief domeinmodel voor een organisatie

Als medewerkers in dit bedrijf altijd op slechts een afdeling werken, dan hoort bij dit model hoort de bedrijfsregel dat aanstellingen van één medewerker elkaar niet kunnen overlappen. De huidige afdeling van een medewerker kunnen we vinden door die aanstelling te nemen waarvan de einddatum nog geen waarde heeft.

OPGAVE 9.7
We kijken in deze opgave naar een domeinmodel voor een deel van de dvd-winkel, zie figuur 9.27.



FIGUUR 9.27 Onvolledig domeinmodel voor de dvd-winkel

De uitlening van dvd's is nog niet volledig gemodelleerd.
a Pas het exemplaarpatroon en/of het verzamelpatroon toe op het model.
b Breid het model uit voor het geval de dvd-winkel alleen de huidige uitleningen opslaat.
c Breid het model uit voor het geval alle uitleningen bewaard moeten worden.

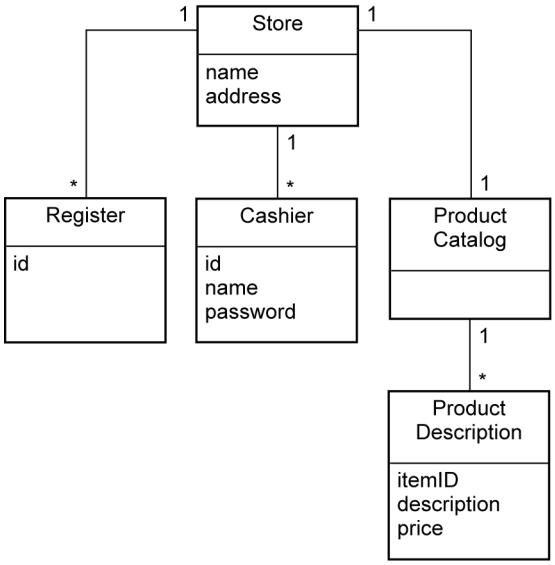
6 Domeinmodel voor de POS-casus opstellen

In deze paragraaf ontwerpen we het domeinmodel voor de POS-casus, gebruik makend van de kennis van de voorgaande paragrafen. Als uitgangspunt gebruiken we de use-casebeschrijvingen en de overige informatie over de POS-casus. We ontwerpen het model op basis van de verdeling van het domein in aanbod, vraag en transactie. We hebben al gezien dat klasse Store als kapstokklasse kan dienen.

6.1 AANBOD

De aanbodkant van de POS-casus betreft het aanbod aan artikelen en de infrastructuur van de winkel: kassa's en medewerkers. Bij de artikelen denken we natuurlijk aan een proces voorraadbeheer maar in de use cases is alleen het verkoopproces beschreven, hoewel er wel aan voorraadbeheer gerefereerd wordt (stap 8 van use case "process sale"). Om het model niet te complex te laten worden, zien we af van het voorraadbeheer, temeer omdat Larman later in zijn boek bij het ontwerp hier ook geen aandacht aan besteedt. Dat betekent dus dat als we het over een artikel hebben, we eigenlijk een artikelsoort bedoelen. Larman gebruikt hiervoor de naam ProductDescription, zie figuur 9.19c.
Voor het beheren van ProductDescription zouden we Store kunnen gebruiken maar Larman gebruikt hiervoor een extra klasse ProductCatalog.
Medewerkers achter de kassa (cashiers) moeten inloggen op het systeem volgens de preconditie van de use case. Daarom moet het systeem gegevens van deze medewerkers bevatten.

Op basis van de voorgaande tekst komen we tot het model van figuur 9.28.



FIGUUR 9.28 Domeinmodel voor POS-casus: aanbod

We plaatsen een aantal opmerkingen bij dit model. Allereerst stellen we vast dat het model een *interpretatie* is van de use-casebeschrijving en additionele informatie.
Uit de use-casebeschrijving is niet duidelijk of het van belang is te weten aan welke kassa een medewerker werkt. We hebben daarom geen associatie gemaakt tussen Register en Cashier.
Attribuutnaam itemID is eigenlijk wat ongelukkig, beter was productID geweest maar Larman gebruikt deze naam ook verder in zijn boek.
Store fungeert als beheerder van Cashier en Register, ProductCatalog als beheerder van ProductDescription. Voor Cashier en Register hadden we , net als voor ProductDescription, ook aparte beheerklassen kunnen nemen.

Klasse zonder attributen

Klasse ProductCatalog is bijzonder in die zin dat de klasse geen attributen bevat. In het algemeen kunnen we stellen dat een klasse zonder attributen 'verdacht' is. Vaak duidt zo'n klasse op een ontwerpfout; de klasse is overbodig. Bij een klasse zonder attributen moeten we ook naar de associaties kijken. Soms wordt het bestaan van zo'n klasse gerechtvaardigd door de associaties die zo'n klasse heeft. Bij klasse ProductCatalog wordt het bestaan gerechtvaardigd door de associatie met ProductDescription: ProductCatalog is de beheerder van ProductDescription.

OPGAVE 9.8
Stel de aanbodkant op van het domeinmodel voor de CRS-casus. Gebruik hiervoor de casusbeschrijving uit de bijlage van hoofdstuk 3, de uitwerking van opgave 6.6 use case Inschrijven voor een semester en de bijlagen van hoofdstuk 6. Beperk u eerst tot het hoofdsuccesscenario. Ga ervan uit dat er geen historie van de inschrijvingen wordt bijgehouden en dat het steeds de inschrijvingen voor één semester betreffen. Uit het antwoord van opgave 9.1 weet u dat Opleiding de kapstokklasse is.

6.2 VRAAG

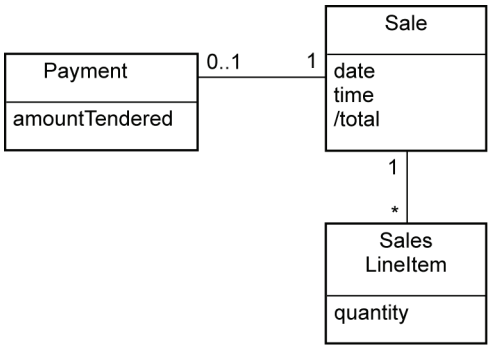
Als we de use case van de POS-casus goed lezen, zien we dat er bij de POS-casus geen gegevens van klanten worden vastgelegd. Dat betekent dat er geen vraagkant is voor dit domeinmodel. Hoewel een klant wel degelijk in de use-casebeschrijving voorkomt als de persoon die de use case Process sale in werking zet (in stap 1 en 10 wordt Customer expliciet genoemd), speelt de klant verder in de beschrijving geen rol . Omdat we het domeinmodel ontwerpen op basis van de use-casebeschrijvingen, nemen we klasse Customer *niet* op in het domeinmodel.

OPGAVE 9.9
Stel de vraagkant op van het domeinmodel voor de CRS-casus en breid het model uit op basis van de uitbreidingen van het scenario.

6.3 TRANSACTIE

De transactie betreft de aankoop van artikelen door de klant. We hebben al gezien dat hiervoor de klassen Sale en SalesLineItem gebruikt moeten worden. Het transactiedeel van het domeinmodel zal meestal de brug zijn tussen de vraagkant en de aanbodkant. Voor de POS-casus is er echter alleen een koppeling van transacties met de aanbodkant.

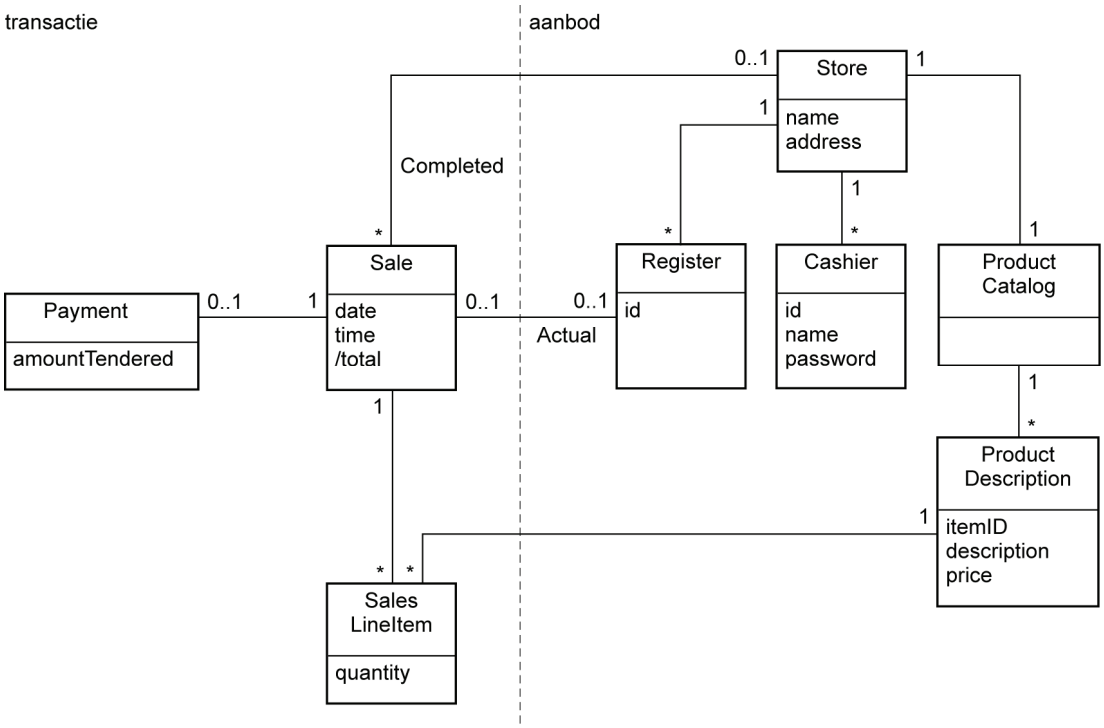
We beginnen met de transactiekant maar verbinden die nog niet met de aanbodkant. Figuur 9.29 bevat het model van de transactie: Sale met bijbehorende SalesLineItems en een Payment. Op het moment dat een nieuwe Sale gestart wordt, zijn er nog geen SalesLineItems en is er nog geen bijbehorende Payment. Vandaar dat de minimale multipliciteit van Sale ten opzicht van die klassen 0 moet zijn.



FIGUUR 9.29 Domeinmodel voor POS-casus: deel van de transactiekant

Bij het koppelen van de transactiekant aan de aanbodkant moeten we een paar problemen oplossen en moeten we een aantal beslissingen nemen.
SalesLineItem bevat informatie over het aantal van een bepaald artikel die tot dezelfde verkoop horen. Het ligt dan voor de hand een associatie tussen SalesLineItem en ProductDescription te maken. Klasse Sale representeert een verkoop. Dat kan een Sale zijn die nu bezig is, waarbij de kassier nu bezig is met die verkoop, maar ook een Sale die al afgerond is.

We moeten dus de huidige Sale en de reeds afgehandelde Sales kunnen onderscheiden. We maken dit onderscheid met behulp van twee associaties. We moeten ons afvragen wie de afgehandelde Sales beheert. We kiezen hiervoor de kapstokklasse Store. Wie beheert de actieve Sale? We kiezen voor Register omdat daar de actuele Sale plaats vindt. Figuur 9.30 toont het complete model.



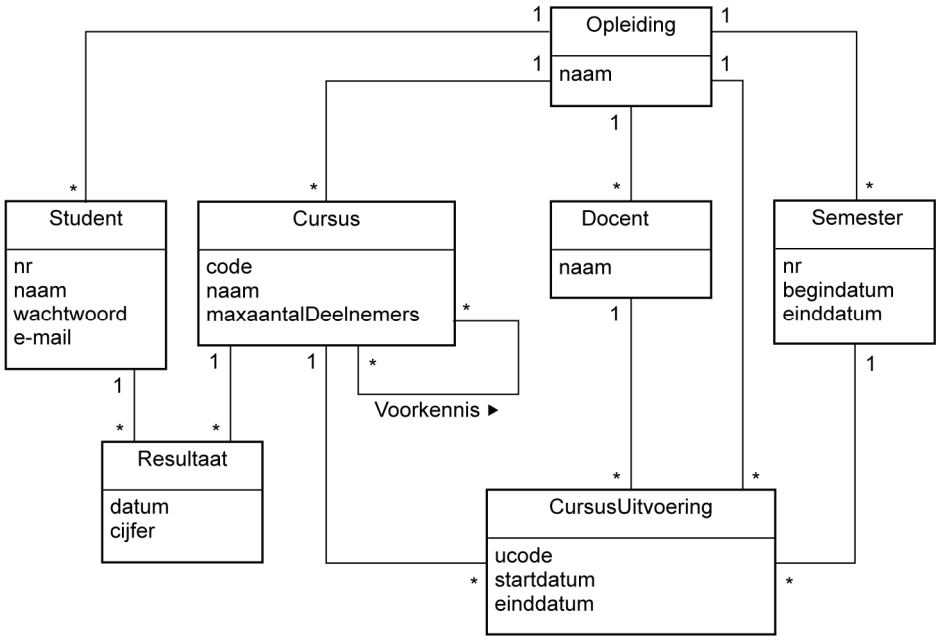
FIGUUR 9.30 Compleet domeinmodel voor de POS-casus

Bij het model van figuur 9.31 hoort een bedrijfsregel voor de berekening van het afleidbare attribuut total.

OPGAVE 9.10

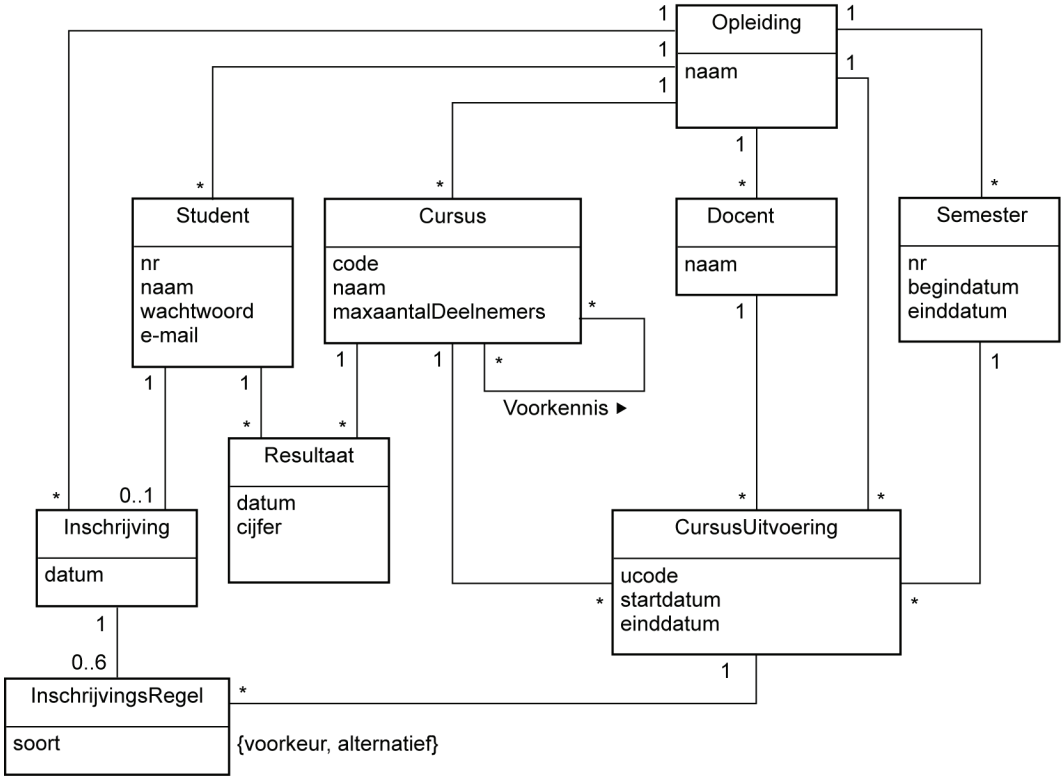
Bij het model ontbreekt nog een bedrijfsregel voor klasse Sale voor de associaties Completed en Actual. Formuleer die regel in natuurlijke taal. Kunt u nu ook aangeven waarom de minimale multipliciteit van de associatie tussen Sale en Store respectievelijk Register 0 moet zijn?

Als laatste onderdeel van deze paragraaf behandelen we nog het transactiedeel van de CRS-casus. We gaan uit van het domeinmodel van vraag en aanbod van het antwoord van opgave 9.9, nogmaals getoond in figuur 9.31.



FIGUUR 9.31 Domeinmodel voor de CRS-casus: aanbod en vraag

De inschrijvingen van studenten zorgen voor een koppeling tussen Student en CursusUitvoering. We kunnen het verzamelpatroon toepassen: een Inschrijving met bijbehorende Inschrijfregels. Per inschrijfregel geven we aan of het de voorkeur of een alternatief betreft. De inschrijvingen laten we beheren door Opleiding. Figuur 9.32 toont het resultaat.



FIGUUR 9.32 Domeinmodel voor de CRS-casus

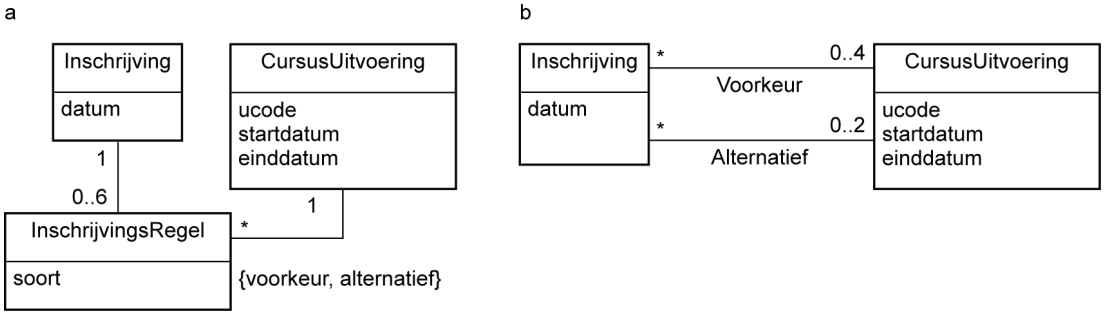
Bij dit model hoort nog een aantal nieuwe bedrijfsregels op basis van de uitbreidingen van de use case. Voordat we deze regels opschrijven, introduceren we eerst een notatie, waarmee de formulering van bedrijfsregels wordt vergemakkelijkt.

Puntnotatie

Om een attribuut van een klasse in een bedrijfsregel wat formeler aan te geven, gebruiken we de notatie <klassenaam>.<attribuutnaam>, bijvoorbeeld Inschrijving.datum. Daarnaast gebruiken we ook de puntnotatie om van de ene klasse naar de andere te kunnen navigeren op basis van het klassendiagram. In dat geval luidt de notatie: <klassenaam>.<klassenaam>. Via deze notatie kunnen we ook meerdere klassen met elkaar verbinden. Met Inschrijving.Inschrijvingsregel.CursusUitvoering worden bijvoorbeeld de bij een Inschrijving horende CursusUitvoeringen aangegeven. Merk op dat we op basis van de multipliciteit van de associatie tussen de klassen soms één instantie bedoelen (maximale multipliciteit 1) of meerdere instanties (maximale multipliciteit >1). Ook kunnen we de twee notaties met elkaar combineren om een attribuut van een met een klasse verbonden andere klasse aan te geven.

Met deze notatie kunnen we de volgende bedrijfsregels formuleren:
1 Inschrijving.InschrijvingsRegel.CursusUitvoering.Cursus is uniek binnen Inschrijving.InschrijvingsRegel, ofwel je mag je maar één keer inschrijven voor een bepaalde cursus.
2 Bij een Inschrijving zijn maximaal vier InschrijvingsRegels met soort = voorkeur
3 Bij een Inschrijving zijn maximaal twee InschrijvingsRegels met soort = alternatief
4 Als geldt dat Student.Inschrijving.InschrijvingsRegel.CursusUitvoering.Cursus een of meer Cursussen heeft als voorkennis dan moet voor die Cursussen gelden: Student.Resultaat.Cursus moet bestaan.

Als laatste deel van deze paragraaf tonen we u een alternatieve modellering voor het inschrijven voor een cursus, zie figuur 9.33.



FIGUUR 9.33 Oorspronkelijke (a) en alternatieve (b) modellering voor inschrijvingen

Bij de modellering van figuur 9.33b zullen meer bedrijfsregels nodig zijn dan voor figuur 9.33a. In het algemeen kunnen we stellen dat een eenvoudiger model vaak gepaard gaat met meer bedrijfsregels. Het ene model is equivalent met het andere model maar niet beter. Onze voorkeur gaat uit naar de modellering van figuur 9.33a.

7 Modelleren is een ambachtelijke activiteit

In de voorgaande paragrafen hebben we u richtlijnen gegeven voor het opstellen van een domeinmodel. We hebben ook duidelijk gemaakt dat een domeinmodel ontwerpen *niet* eenvoudigweg het toepassen van een aantal regels is: hét domeinmodel bestaat niet.
We kunnen wel de richtlijnen tot nu toe samenvatten in de volgende globale aanpak:

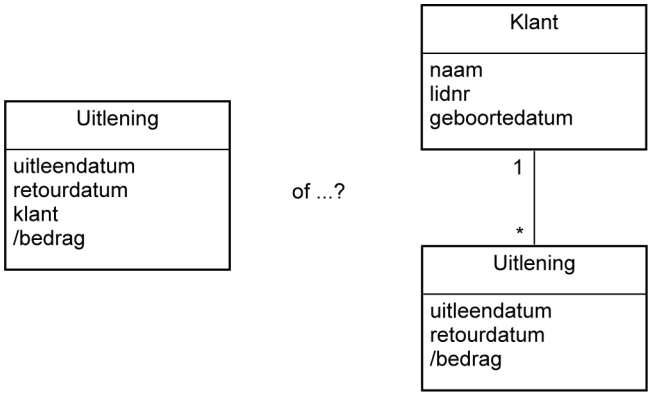
1 Begin met een kapstokklasse die het gehele systeem representeert.
2 Probeer aan de hand van de bedrijfsprocessen een verdeling te maken in aanbod, vraag en transactie.
3 Stel per onderdeel een domeinmodel op, gebruikmakend van de behandelde patronen. Voeg indien nodig bedrijfsregels toe. Het transactiedeel zal zorgen voor de koppeling van het vraagdeel en het aanboddeel.

We hebben bewust niet aangegeven hoe u op het spoor komt van relevante klassen voor het domeinmodel. Wij zijn van mening dat technieken die dat wel doen in de praktijk niet werken. De in de praktijk gangbare methode van tekstanalyse waarbij zelfstandige naamwoorden duiden op klassen en werkwoorden op associaties, is sterk afhankelijk van de tekst. De methode werkt eigenlijk alleen als de tekst al een tekstuele weergave is van het domeinmodel. Wij verwachten dat u intuïtief al goed de klassen kunt herkennen. Het vinden van klassen gaat beter naarmate u meer ervaring hebt met modelleren, goede voorbeelden heeft bestudeerd en kennis hebt van de belangrijkste patronen.

In deze paragraaf stellen we nog enkele problemen aan de orde die u vaak zult tegenkomen bij het modelleren.

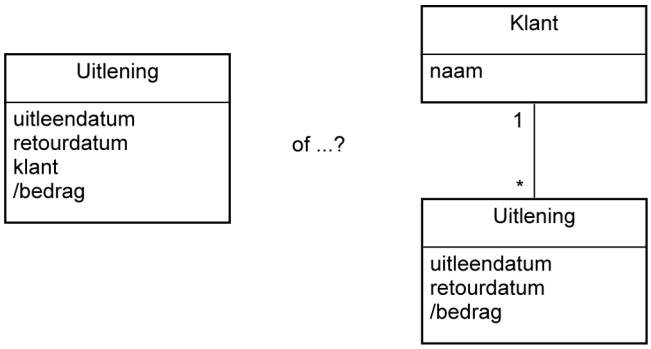
7.1 ATTRIBUUT VERSUS KLASSE

Bij het modelleren moet u soms beslissen of iets als een attribuut of als een zelfstandige klasse gemodelleerd moet worden. Wat moet u dan doen? Stel dat u een uitlening modelleert van de dvd-winkel en zich afvraagt of klant als attribuut of als een klasse gemodelleerd moet worden, zie figuur 9.34.



FIGUUR 9.34 Klant als attribuut en klant als klasse

Uit de figuur is het duidelijk dat klant hier als klasse gemodelleerd moet worden omdat klant niet een getal of een stukje tekst is maar een apart concept uit de werkelijkheid. We hebben dat aangegeven door lidnr, naam en geboortedatum bij Klant op te nemen.
Wat nu als we van een klant alleen de naam willen opslaan, zoals in figuur 9.35?



FIGUUR 9.35 Klant als attribuut en klant als klasse

Technisch gezien zijn de twee modellen nu min of meer equivalent maar we geven de voorkeur aan de modellering rechts omdat we nu ook onze klanten kunnen beheren. Het concept klant uit de werkelijkheid is meer dan een naam, hoewel we van de klant alleen de naam opslaan.

Regel

Algemeen kunnen we stellen dat je een aparte klasse gebruikt als het om aparte concepten gaat. Ook in geval van twijfel kiezen we voor een aparte klasse.

7.2 WANNEER MODELLEREN WE EEN ASSOCIATIE?

Een associatie is een verband tussen twee klassen of beter gezegd tussen instanties van twee klassen. Nu bestaan er in de werkelijkheid vele verbanden tussen concepten. Dat zou betekenen dat we vele associaties tussen klassen nodig hebben om de werkelijkheid te modelleren. Bedenk echter dat we niet de gehele werkelijkheid modelleren maar de werkelijkheid zoals we die interpreteren voor ons systeem. We gebruiken alleen associaties als het voor het systeem van belang is deze vast te leggen.
Bij de dvd-winkel zoeken klanten eerst dvd's voordat ze deze naar de balie brengen om ze te kunnen lenen. Omdat het zoeken naar dvd's niet voor ons systeem van belang is, leggen we de associatie tussen Klant en DvdExemplaar niet vast in ons domeinmodel. Bij een uitlening is het natuurlijk wel van belang aan welke klant bepaalde dvd's worden uitgeleend. Daarom modelleren we wel een associatie tussen Uitlening en Klant.
Het modelleren van "overbodige" associaties is niet echt fout maar het domeinmodel wordt daardoor nodeloos complex.

7.3 BON, OVERZICHT, RAPPORT

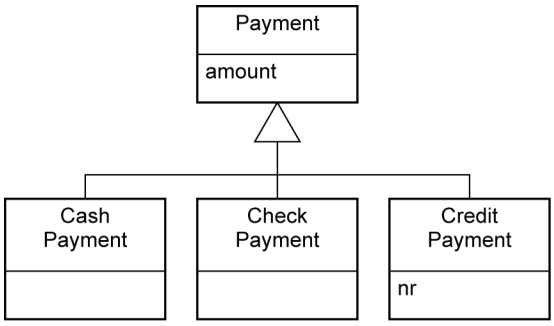
In use-casebeschrijvingen komen vaak overzichten met gegevens voor die afgedrukt moeten worden, bijvoorbeeld de bon bij de POS-casus. In het algemeen modelleren we zaken als bon, overzicht of rapport niet. De reden is dat ze niet veel meer zijn dan een weergave van gegevens die in het systeem aanwezig zijn. Wel zullen we dan bij de eisen opnemen dat het systeem de mogelijkheid moet hebben overzichten, rapporten of bonnen te produceren.

8 Verfijning van het domeinmodel

In deze paragraaf bespreken we de geavanceerdere modelleerconcepten generalisatie en specialisatie. Met behulp van deze concepten kunnen we een hiërarchie van klassen modelleren. Zo'n hiërarchie maakt gebruik van overerving waardoor duplicatie van attributen en associaties (en dus later van code) vermeden kan worden. Verder besteden we aandacht aan abstracte klassen.

8.1 GENERALISATIE IN ALGEMENE ZIN

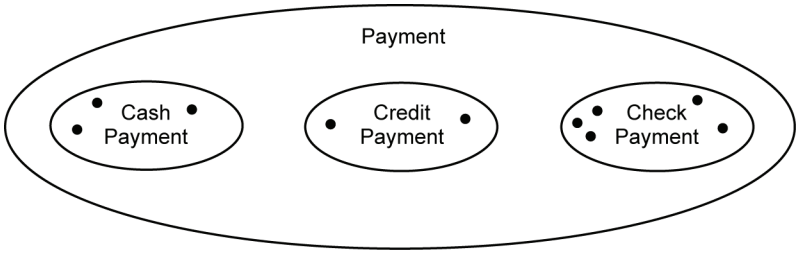
Bij de POS-casus hebben we een klasse Payment gemodelleerd. Die klasse betrof contante betalingen. In latere iteraties blijkt dat naast contante betalingen ook de mogelijkheid geboden moet worden met een cheque of met een credit card te betalen. We onderscheiden zo de klassen CashPayment, CreditPayment en CheckPayment. Deze klassen hebben gemeenschappelijke kenmerken, bijvoorbeeld het bedrag, maar ook verschillende zoals het nummer van de credit card. We modelleren deze klassen op de manier van figuur 9.36. Hierin is klasse Payment de superklasse en zijn de andere klassen subklassen. De superklasse is een meer algemene klasse, de subklassen zijn specifieker.



FIGUUR 9.36 Hiërarchie van Payment

Een relatie tussen een meer algemene klasse (de superklasse) en een specifiekere klasse (de subklasse) noemen we een generalisatie in algemene zin.

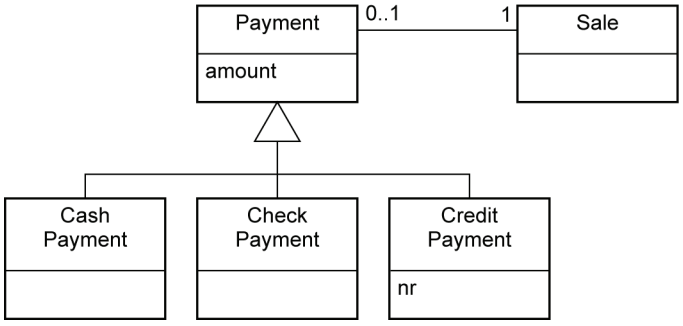
Superklassen en subklassen zijn gerelateerd aan elkaar op basis van (wiskundige) verzamelingen. Verzamelingen kunnen we weergeven met een Venn-diagram. Figuur 9.37 toont het Venn-diagram van de Payment-hiërarchie. De instanties van de klassen beelden we af als elementen van de overeenkomstige verzameling.



FIGUUR 9.37 Venn-diagram van Payment

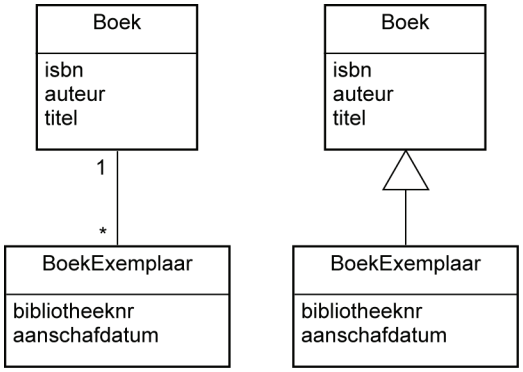
Aan de hand van figuur 9.37 zien we dat verzameling Payment uit 10 elementen bestaat (de zwarte stippen). Drie elementen behoren tot de verzameling CashPayment, twee tot CreditPayment en vijf tot CheckPayment. Merk op dat er in dit geval geen elementen van Payment zijn die niet tot een van de deelverzamelingen horen. We zien ook dat ieder element maar tot één deelverzameling behoort. We duiden deze situatie aan met de terminologie: *totaal* en *exclusief*. Als een element in meerdere deelverzamelingen kan voorkomen spreken we van *overlappend* in plaats van exclusief. Als de omhullende verzameling elementen bevat die niet in een deelverzameling voorkomen, spreken we in plaats van totaal over *partieel*. Met deze begrippen kunnen we zo vier soorten hiërarchieën karakteriseren: totaal en exclusief, totaal en overlappend, partieel en exclusief, en tenslotte partieel en overlappend.

De elementen van een deelverzameling zijn ook element van de omhullende verzameling. Op basis van figuur 9.37 kunnen we dan bijvoorbeeld zeggen: Een CashPayment *is* een Payment. Omdat een CashPayment een Payment is, erft CashPayment ook alle eigenschappen (dat wil zeggen attributen en associaties) van Payment.
De modellering van de associatie tussen de Payment-hierarchie en Sale ziet er daarom uit zoals figuur in 9.38. Alle subklassen van Payment moeten een associatie hebben met een Sale. Omdat de subklassen alle associaties erven van hun superklassen, kan deze gemeenschappelijke associatie aan Sale toegekend worden.



FIGUUR 9.38 Payment-hiërarchie met associatie

OPGAVE 9.11
Bij het exemplaarpatroon hebben we een correcte modellering gezien voor een boek en bijbehorende boekexemplaren. Is een modellering met een subklasse zoals in figuur 9.39 ook een goede modellering?

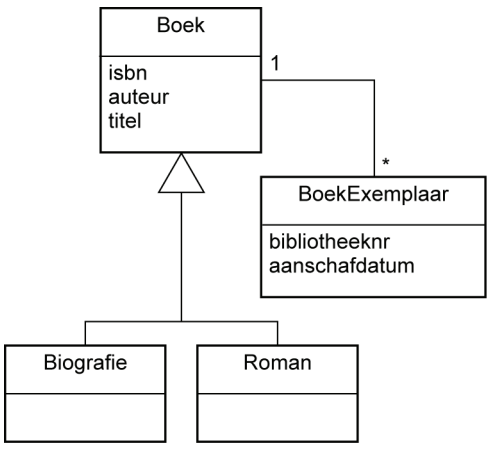


FIGUUR 9.39 BoekExemplaar subklasse van Boek?

OPGAVE 9.12
Iedere hiërarchie die partieel is, kan worden getransformeerd in een hiërarchie die totaal is. Toon dit aan.

8.2 WANNEER MODELLEREN WE EEN GENERALISATIE?

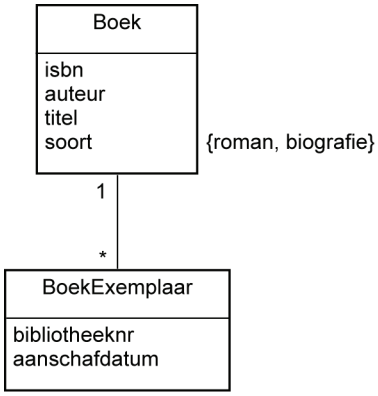
Als we in een verzameling een of meerdere deelverzamelingen kunnen onderscheiden, is het correct een hiërarchie van klassen te modelleren. Boekexemplaren vormen geen subklasse van boek maar we zouden wel verschillende soorten boeken kunnen onderscheiden en zo subklassen kunnen vormen: roman, biografie enzovoort, zie figuur 9.40.



FIGUUR 9.40 Boek-hiërarchie met associatie

De modellering van figuur 9.40 is correct maar we moeten ons afvragen of deze modellering zinvol is. Bibliotheken hanteren een veel uitgebreidere classificatie van boeken dan we gemodelleerd hebben. We zouden dus nog veel meer subklassen moeten onderscheiden. Het domeinmodel zou zo heel groot worden. Als we naar de subklassen kijken zien we dat

ze alleen verschillen in de naam van de klasse; ze hebben geen eigen attributen en ook geen eigen associaties. Een equivalente maar veel eenvoudigere modellering voor dit geval is een attribuut soort op te nemen in klasse Boek ter vervanging van de hiërarchie, zie figuur 9.41.



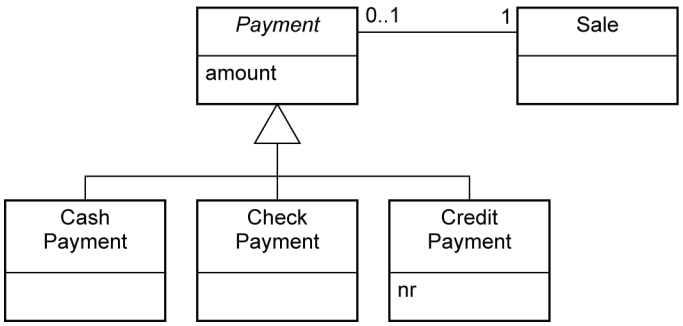
FIGUUR 9.41 Beter model voor het model van figuur 9.40

Richtlijn: Het gebruik van een subklassenhiërarchie is alleen zinvol als een of meer subklassen eigen attributen of associaties kennen. Daarnaast moet de kwalificatie van de overeenkomende verzamelingen exclusief zijn.

We kunnen op twee manieren tot een hiërarchie komen. Als we meerdere klassen hebben gemodelleerd die gemeenschappelijke attributen en/of associaties bezitten, kunnen we deze in een aparte superklasse modelleren. Dit noemen we generalisatie in engere zin. Als we vanuit een algemene klasse meer specifieke subklassen ontwerpen spreken we van specialisatie.

8.3 ABSTRACTE KLASSEN

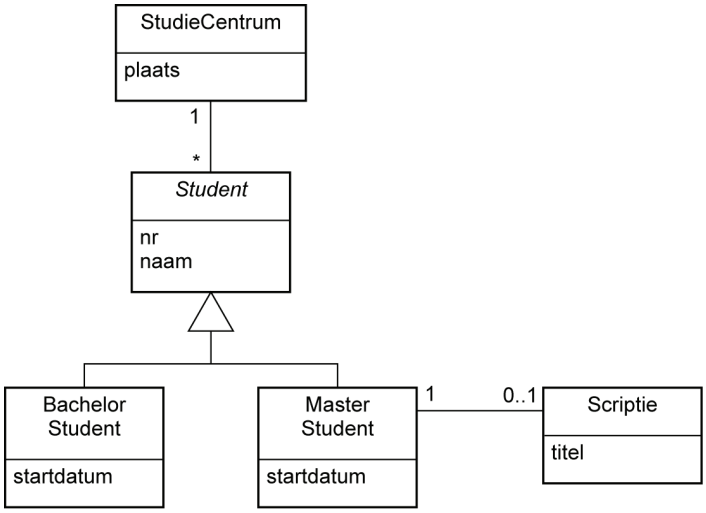
Als we een hiërarchie hebben die totaal en exclusief is, dan biedt het voordelen de superklasse als *abstract* te definiëren. Van een abstracte klasse kunnen *geen* instanties bestaan. In geval van de Payment-hiërarchie kunnen we klasse Payment als abstract beschouwen. We geven een abstracte klasse in een klassendiagram met een cursief geschreven naam weer, zoals in figuur 9.42.



FIGUUR 9.42 Payment als abstracte klasse

8.4 ONEIGENLIJK GEBRUIK VAN GENERALISATIE

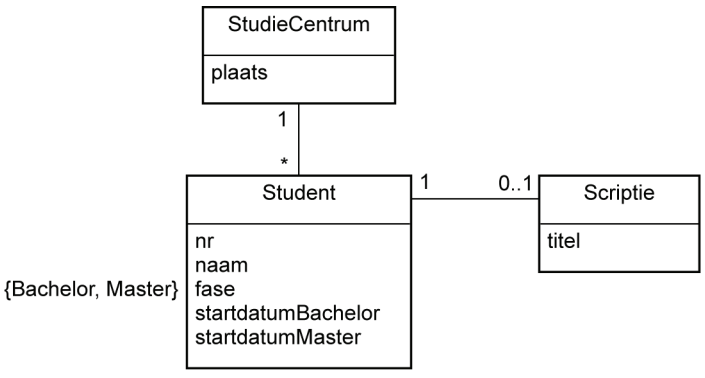
Studenten van de Open Universiteit Nederland (OU) kunnen we onderverdelen in bachelor-studenten en master-studenten. Er is hier sprake van een totale en exclusieve hiërarchie dus een voor de hand liggende modellering zou het domeinmodel van figuur 9.43 zijn.



FIGUUR 9.43 Domeinmodel voor studenten van de OU

OPGAVE 9.13
Is het niet beter om Student van een attribuut startdatum te voorzien? Het attribuut startdatum van BachelorStudent en MasterStudent is dan overbodig.

Op het eerste gezicht lijkt het model volledig correct. BachelorStudent en MasterStudent hebben eigen attributen en associaties en gemeenschappelijke attributen zijn opgenomen bij de klasse Student.
Het probleem zit hem in het feit dat bachelor en master toestanden zijn waarin een student kan verkeren. Op een zeker moment zal een student van de bachelorfase overgaan naar de masterfase. Technisch zal dat betekenen dat een BachelorStudent-instantie van de student moet worden verwijderd en vervangen door een nieuwe MasterStudent-instantie. Daarbij moeten de algemene studentgegevens gekopieerd worden.
Een betere modellering ziet u in figuur 9.44.



FIGUUR 9.44 Beter domeinmodel voor studenten van de OU

Als we te maken hebben met verschillende toestanden van een object, modelleren we die toestanden dus niet met een hiërarchie maar met een attribuut met als waardenverzameling de verschillende toestanden.

OPGAVE 9.14

Als u het domeinmodel van figuur 944 vergelijkt met dat van figuur 9.43, ziet u dat bij het model van figuur 9.44 nog een aantal bedrijfsregels nodig zijn. Formuleer deze bedrijfsregels in natuurlijke taal. Ga hierbij uit van een student die zijn gehele opleiding bij de Open Universiteit Nederland doet.

ZELFTOETS

In dit hoofdstuk is terloops aandacht geschonken aan het domeinmodel voor de dvd-winkel. Stel nu een volledig domeinmodel op. Ga ervan uit dat de uitleningen in het systeem bewaard moeten blijven.
Gebruik de zelftoets van hoofdstuk 6 en de uitwerking ervan als casusbeschrijving. Het is niet nodig de betalingen te modelleren. De leeftijdgrenzen zijn: geen, 6, 12 en 16. Maak gebruik van een opsplitsing in aanbod, vraag en transactie en pas patronen toe. Een globaal model heeft u al gemodelleerd in opgave 3.
Formuleer een aantal bedrijfsregels die u niet in het model kon opnemen. Gebruik in uw formulering daar waar nodig de puntnotatie.

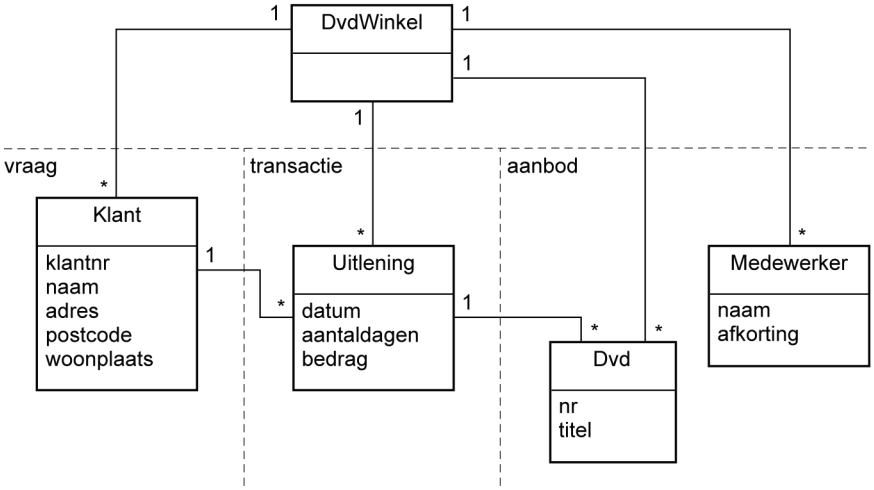
TERUGKOPPELING

1 Uitwerking van de opgaven

9.1 Bij de dvd-winkel is er geen geschikte klasse die de rol van kapstokklasse kan spelen. We voegen een extra klasse toe: DvdWinkel. Bij de CRS-casus is de domeinklasse Opleiding een goede kandidaat om als kapstokklasse te fungeren

9.2 Bij de dvd-winkel kunnen we de dvd's en de gegevens van de winkel en medewerkers tot de aanbodkant rekenen. De vraagkant betreft de klanten, de transactie het vastleggen van een uitlening van dvd's aan klanten.
Bij de CRS-casus vormen de cursussen en de bijbehorende docenten de aanbodkant. Studenten vormen de vraagkant en de inschrijvingen voor de verschillende cursussen vormen de transacties.

9.3 Een mogelijke uitwerking ziet u in figuur 9.45. Als u meer of minder attributen heeft gemodelleerd of andere multipliciteiten heeft aangegeven, is dat nog niet van belang. Het gaat hier om het herkennen van de belangrijkste klassen en deze in te delen naar aanbod, vraag en transactie.

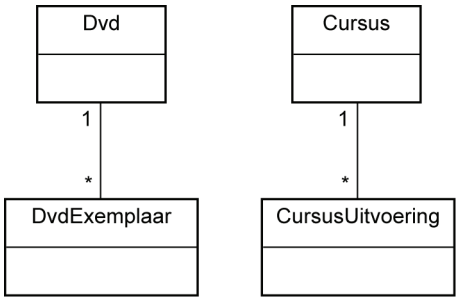


FIGUUR 9.45 Eerste domeinmodel voor dvd-winkel

9.4 Bij de CRS-casus gaat het onder andere om een inschrijving van een student voor meerdere cursussen. We zouden klasse Inschrijving en klasse Inschrijvingsregel kunnen modelleren. Inschrijving kunnen we dan aan Student koppelen, Inschrijvingsregel met Cursus.

Bij de dvd-winkel gaat het onder andere om een uitlening van meerdere dvd's. We zouden hier klasse Uitlening en Uitleenregel kunnen modelleren. Uitlening kan dan de datum van uitlening bevatten en gekoppeld worden met Klant. Een Uitleenregel kan de overeengekomen retourdatum voor een specifieke dvd bevatten en is verbonden met (de geleende) dvd.

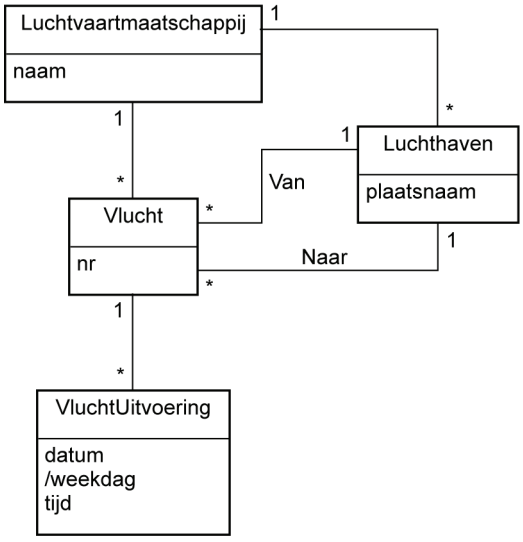
9.5 Bij de dvd-winkel onderscheiden we Dvd en DvdExemplaar en bij de CRS-casus Cursus en Cursusuitvoering, zie figuur 9.46.



FIGUUR 9.46 Exemplaarpatroon bij de dvd-winkel en CRS-casus

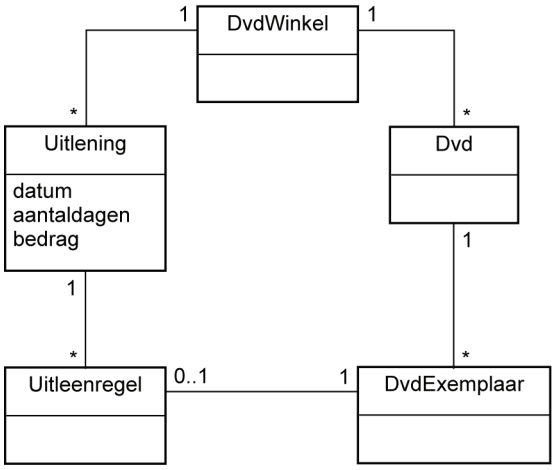
Dvd bevat algemene kenmerken voor alle exemplaren zoals titel, jaar van uitgifte enzovoorts. DvdExemplaar bevat gegevens die specifiek zijn voor een exemplaar zoals de streepjescode.
Cursus bevat algemene kenmerken van een cursus zoals titel, soort, studiebelasting. CursusUitvoering bevat gegevens van een cursus die gegeven wordt zoals startdatum, tijd, docent enzovoorts.

9.6 Ook hier hebben we te maken met een exemplaarpatroon: Vlucht en Vluchtuitvoering. Figuur 9.47 toont het domeinmodel.



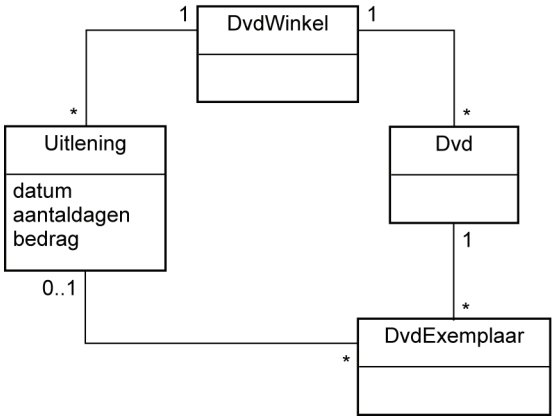
FIGUUR 9.47 Domeinmodel voor luchtvaartmaatschappij

9.7 a Bij uitlening passen we het verzamelpatroon toe: een Uitlening heeft meerdere Uitleenregels. Bij Dvd passen we het exemplaarpatroon toe. Het resultaat ziet u in figuur 9.48.



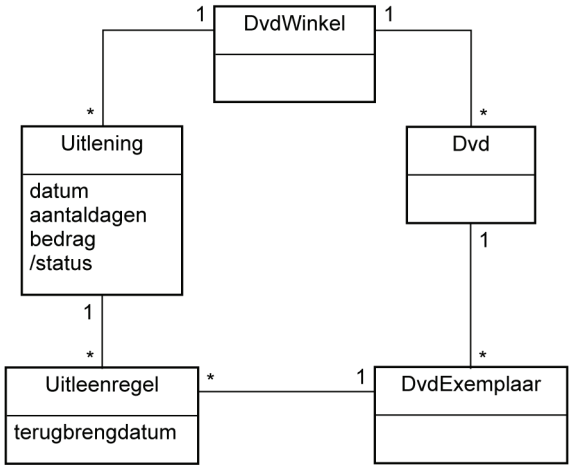
FIGUUR 9.48 Domeinmodel voor de dvd-winkel

b Een uitleenregel betreft een DvdExemplaar. Als we geen historie bijhouden, kan een DvdExemplaar al dan niet uitgeleend zijn en heeft dus multipliciteit 0..1 met betrekking tot Uitleenregel, zie figuur 9.48 Vergelijk dit model met het model van figuur 9.22. We kunnen Uitleenregel hier weglaten en DvdExemplaar met Uitlening verbinden met multipliciteit *, zie figuur 9.49.



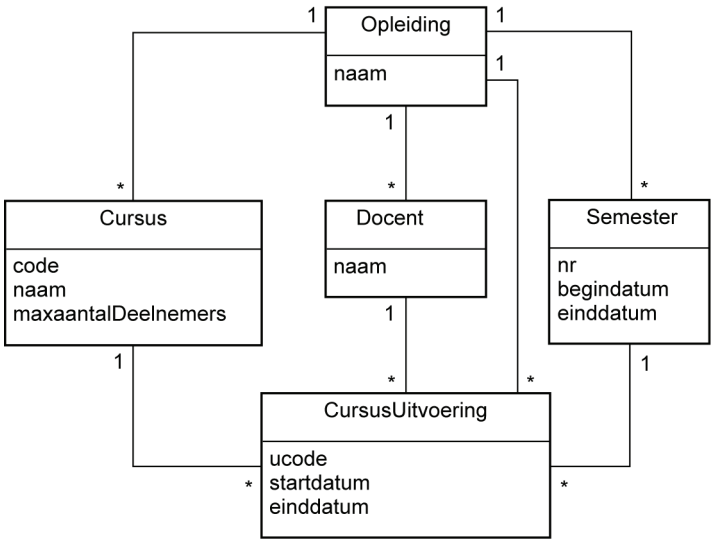
FIGUUR 9.49 Aangepast domeinmodel voor de dvd-winkel

c We gaan weer uit van figuur 9.48. Een DvdExemplaar kan nu in meerdere uitleenregels voorkomen (maar niet op één moment). De multipliciteit verandert van 0..1 naar *. Bij Uitleenregel kunnen we nu een attribuut terugbrengdatum opnemen. Bij Uitlening kunnen we een afleidbaar attribuut status opnemen om na te kunnen gaan of alle dvd's van een uitlening weer teruggebracht zijn, zie figuur 9.50.



FIGUUR 9.50 Aangepast domeinmodel voor de dvd-winkel

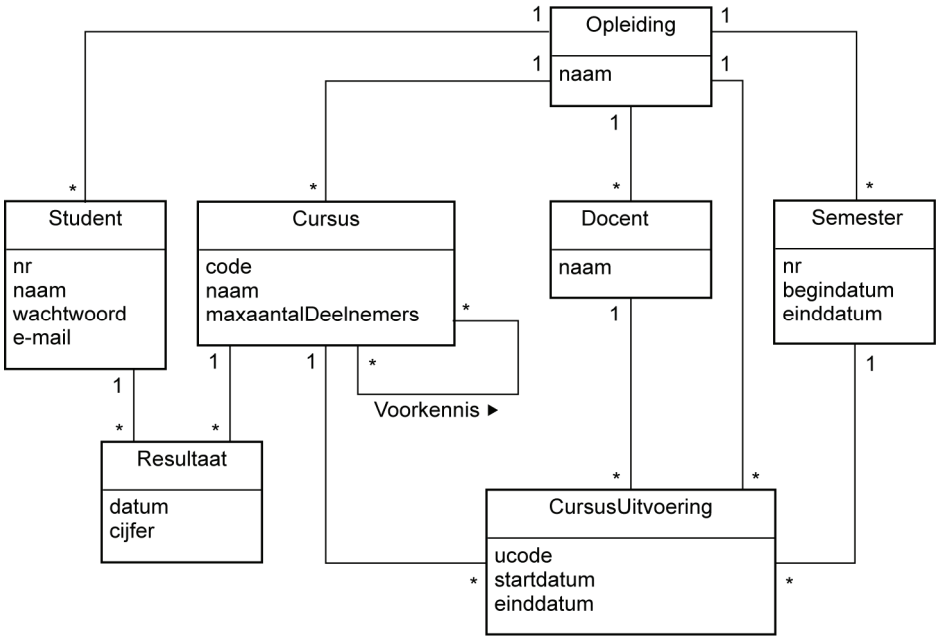
9.8 Bij het aanbod gaat het om de cursussen die in een bepaald semester aangeboden worden. We hebben al gezien dat het exemplaarpatroon hier van toepassing is: Cursus met bijbehorende klasse CursusUitvoering. Bij iedere CursusUitvoering is de docent bekend. We komen op basis van de use case en deze overwegingen tot het model van figuur 9.51.



FIGUUR 9.51 Domeinmodel voor de CRS-casus: aanbod, eerste versie

Voor het toewijzen van docenten aan cursusuitvoeringen is er een ander bedrijfsproces aan de inschrijving van studenten vooraf gegaan. In het kader van dat proces was het nodig ook een associatie tussen Cursus en Docent te modelleren met naam "Kan gegeven worden door". Als we alleen naar het inschrijven van studenten kijken, is die associatie niet nodig.

9.9 De student vormt de vraagkant van het domeinmodel; studenten worden beheerd door Opleiding. Bij het bekijken van de uitbreidingen van de use case moeten we ons afvragen of het domeinmodel voldoende informatie bevat om de vermelde tests uit te kunnen voeren. In een van de uitbreidingen wordt nagegaan of de student zich inschrijft voor een cursus die al behaald is. We moeten dus ook vastleggen welke cursussen de studenten reeds gehaald hebben. Het ligt voor de hand om niet alleen vast te leggen dat een student een cursus heeft gehaald, maar ook wanneer, en met welk resultaat; daarom is een associatie tussen Student en Cursus niet voldoende, en introduceren we een klasse Resultaat. Cursussen kunnen bovendien voorkenniseisen hebben: een student moet een bepaalde cursus gehaald hebben om aan een andere te kunnen deelnemen.
De voorkenniseisen kunnen we modelleren met behulp van een reflexieve associatie. Het domeinmodel (aanbod en vraagkant) wordt dan zoals figuur 9.52 toont.

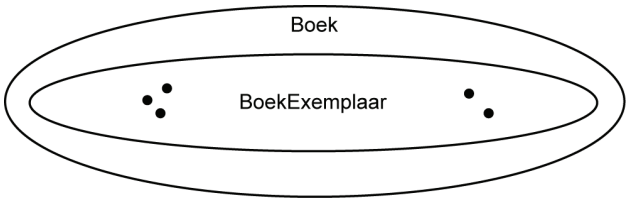


FIGUUR 9.52 Domeinmodel voor de CRS-casus: aanbod (en vraag), tweede versie

9.10 De associaties van Sale met Store en Register moeten elkaar uitsluiten, dat wil zeggen een Sale-object is of verbonden met een Register (dan is de Sale de actuele Sale) of met Store. In dat laatste geval is de Sale afgehandeld. We kunnen de bedrijfsregel aldus formuleren:

Een Sale-object behoort op ieder moment of tot Register of tot Store. We hebben te maken met een exclusive-or. Soms wordt in het klassendiagram dit wel aangegeven met een stippellijn tussen de associaties en bij de stippellijn wordt {xor} gezet. Wij zullen van deze notatie verder geen gebruik maken maar steeds dit soort constraints in tekst weergeven.
Aanvankelijk hoort een Sale bij een Register en nog niet bij Store dus de associatie met Store moet minimale multipliciteit 0 hebben. Als de Sale is afgerond hoort Sale niet meer tot Register en dit betekent dat de minimale multipliciteit van Sale ten opzichte van Register ook 0 moet zijn.

9.11 Nee, de modellering met een subklasse is foutief. Als we een Venndiagram maken op basis van figuur 9.39, krijgen we figuur 9.53. In de figuur hebben we drie exemplaren proberen weer te geven van een bepaald boek en nog eens twee exemplaren van een ander boek.



FIGUUR 9.53 Venn-diagram voor Boek en BoekExemplaar

Als we figuur 9.53 vergelijken met figuur 9.37 zien we al dat er iets niet goed is: In figuur 9.37 zijn er meerdere subklassen. We kunnen de elementen bij een bepaalde subklasse indelen op basis van een kenmerk (contant, check, credit card). In figuur 9.53 is daarvan geen sprake: Alle elementen behoren tot de enige subklasse BoekExemplaar.
Er geldt dat bij ieder element de algemene boekgegevens als isbn, auteur en titel opgenomen moeten worden.

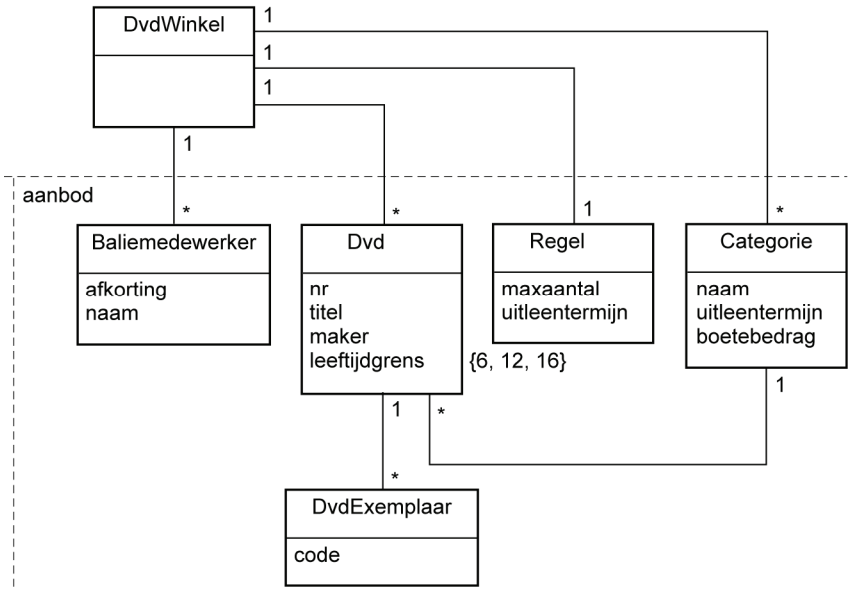
9.12 We kunnen een hiërarchie totaal maken door een extra subklasse te maken en daar de elementen in te plaatsen die nog niet tot een andere subklasse behoorden. Zo behoort ieder element van de superklasse altijd tot een subklasse, hetgeen de voorwaarde is voor het predicaat totaal.

9.13 Nee, want we hebben hier te maken met twee homoniemen namelijk startdatum van de bacheloropleiding en startdatum van de masteropleiding.

9.14 1 Als Student.fase = Bachelor dan moet gelden dat Student.Scriptie niet bestaat
2 Als Student.fase = Bachelor dan moet Student.startdatumBachelor een waarde hebben en moet Student.startdatumMaster geen waarde hebben.
3 Als Student.fase = Master dan moeten zowel Student.startdatumBachelor als Student.startdatumMaster een waarde hebben

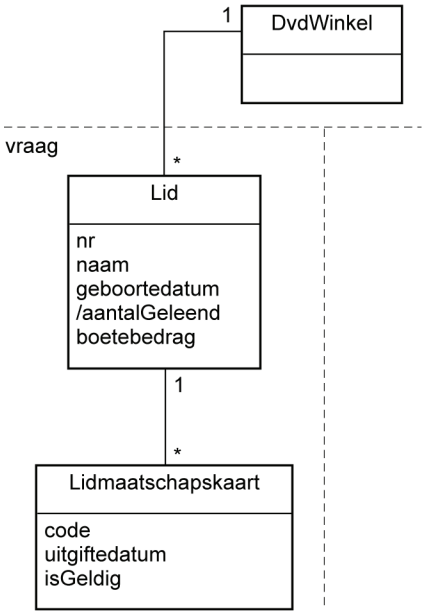
2 Uitwerking van de zelftoets

We beginnen met de aanbodkant. Het aanbod bestaat uit de dvd's en bijbehorende categorieën, de baliemedewerkers en gegevens over uitleentermijnen, boetes en bedragen. Bij de dvd's kunnen we het exemplaarpatroon toepassen. Figuur 9.54 toont het domeinmodel voor de aanbodkant.



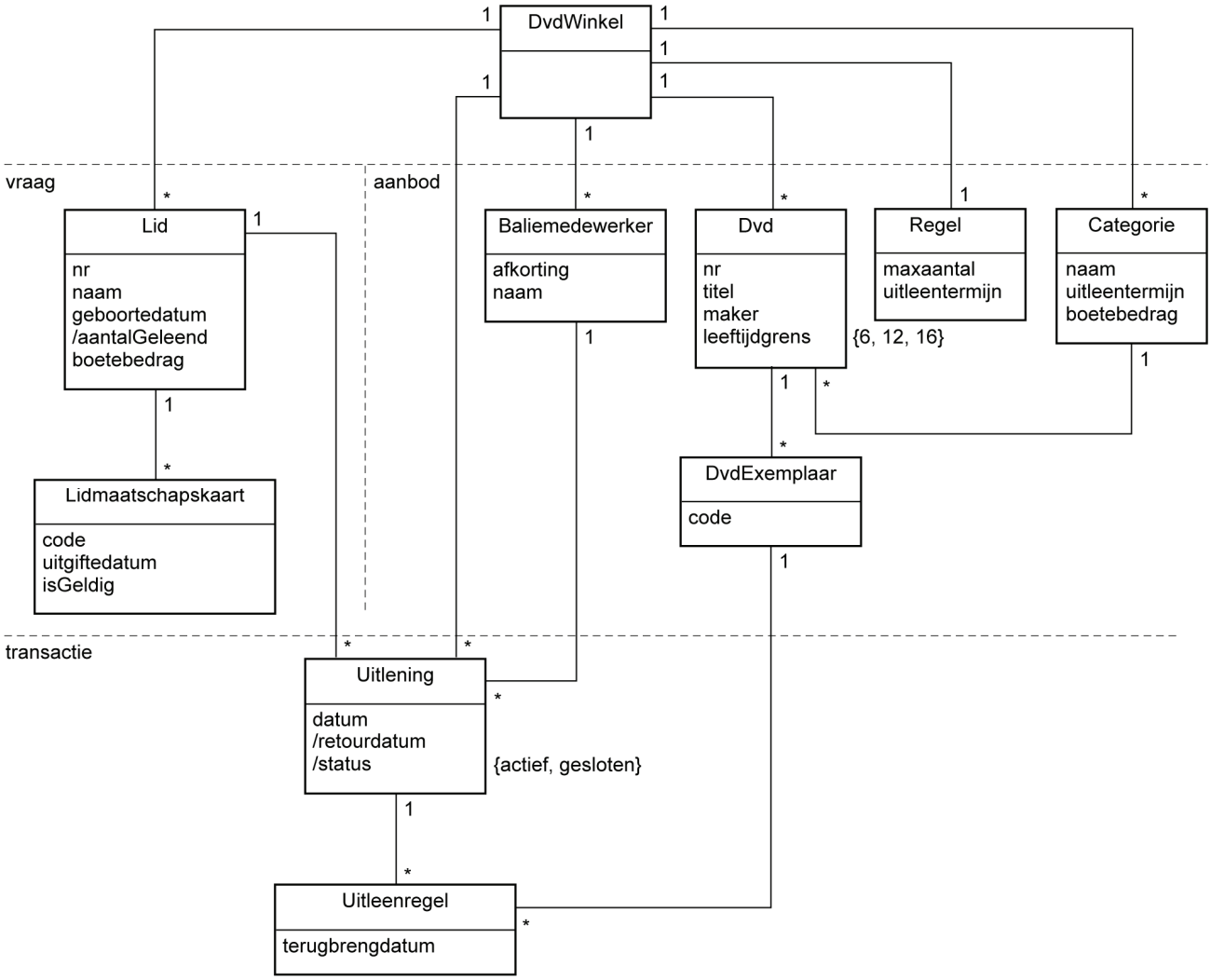
FIGUUR 9.54 Domeinmodel voor de aanbodkant van de dvd-winkel

We hebben een klasse Regel toegevoegd om daarnaar straks te kunnen refereren bij de bedrijfsregels.
De vraagkant bestaat uit de klanten met hun lidmaatschapskaart. Om na te kunnen gaan of een lid een bepaalde dvd mag lenen moeten we ook de geboortedatum bij een lid opnemen. Voor het controleren van een uitlening hebben we naast het boetebedrag een afleidbaar attribuut aantalGeleend opgenomen, zie figuur 9.55.



FIGUUR 9.55 Domeinmodel voor de vraagkant van de dvd-winkel

De transactiekant bestaat uit de uitleningen. Hier kunnen we het verzamelpatroon toepassen: een Uitlening met Uitleenregels. Omdat de uitleningen bewaard moeten blijven kunnen we in Uitleenregel registreren of een dvd is teruggebracht. Als alle dvd's van een uitlening zijn teruggebracht kan deze als afgesloten beschouwd worden. Het is dan handig een status bij Uitlening op te nemen met als mogelijke waarden {actief, gesloten}. De uitlening wordt verbonden met de klant en de baliemedewerker, de Uitleenregel met DvdExemplaar.
Op basis van de huidige datum en de uitleentermijn kan de retourdatum bepaald worden. Ook de status van een Uitlening is afleidbaar. Het volledige model ziet u in figuur 9.54.



FIGUUR 9.56 Domeinmodel voor de dvd-winkel

Bedrijfsregels:
1 Voor iedere Uitlening moet gelden:
Aantal jaren van (Huidige datum-Uitlening.Lid.geboortedatum)>=
Uitlening.Uitleenregel.DvdExemplaar.Dvd.leeftijdgrens
2 Uitlening.retourdatum = Huidige datum + DvdWinkel.Regel.uitleentermijn
3 Uitlening.status = gesloten als voor iedere bijbehorende Uitlening.Uitleenregel geldt dat terugbrengdatum een waarde heeft, anders geldt Uitlening.status = actief
4 Lid.aantalGeleend = aantal Uitleenregels zonder waarde voor terugbrengdatum van Uitlening met status = actief