Stichting NIOC en de NIOC kennisbank

Stichting NIOC (www.nioc.nl) stelt zich conform zijn statuten tot doel: het realiseren van congressen over informatica onderwijs en voorts al hetgeen met een en ander rechtstreeks of zijdelings verband houdt of daartoe bevorderlijk kan zijn, alles in de ruimste zin des woords.

De stichting NIOC neemt de archivering van de resultaten van de congressen voor zijn rekening. De website www.nioc.nl ontsluit onder "Eerdere congressen" de gearchiveerde websites van eerdere congressen. De vele afzonderlijke congresbijdragen zijn opgenomen in een kennisbank die via dezelfde website onder "NIOC kennisbank" ontsloten wordt.

Op dit moment bevat de NIOC kennisbank alle bijdragen, incl. die van het laatste congres (NIOC2025, gehouden op donderdag 27 maart 2025 jl. en georganiseerd door Hogeschool Windesheim). Bij elkaar zo'n 1500 bijdragen!

We roepen je op, na het lezen van het document dat door jou is gedownload, de auteur(s) feedback te geven. Dit kan door je te registreren als gebruiker van de NIOC kennisbank. Na registratie krijg je bericht hoe in te loggen op de NIOC kennisbank.

Het eerstvolgende NIOC vindt plaats op 18 maart 2027 in Arnhem en wordt georganiseerd door HAN University of Applied Sciences.

Reacties over de NIOC kennisbank en de inhoud daarvan kun je richten aan de beheerder:

R. Smedinga kennisbank@nioc.nl.

Vermeld bij reacties jouw naam en telefoonnummer voor nader contact.

Visualisatie van het Objectgeoriënteerde Paradigma.

Arend Rensink
Faculteit der Informatica, Universiteit Twente
e-mail: rensink@cs.utwente.nl

Samenvatting

Programmeeronderwijs maakt een wezenlijk deel uit van elke informatica-opleiding. Veel opleidingen kiezen ervoor dit onderwijs te schoeien op objectgeoriënteerde leest. Voor studenten die voor het eerst met dit paradigma geconfronteerd worden blijkt het vaak lastig te zijn zich een voorstelling te maken van wat er bij de uitvoering van een programma nu precies gebeurt. Deze bijdrage beschrijft een systematische grafische notatie, aangeduid als *toestandkiekjes*, die erop gericht is om de principes en de werking van objectgeoriënteerde programma's zichtbaar te maken. Niet alleen kunnen toestandskiekjes objecten met hun onderlinge samenhang weergeven, maar ook de constructie van objecten en “lopende methoden” binnen objecten. Deze vorm van visualisatie is vooral bedoeld voor het initiële leertraject, waarin studenten zich de concepten van objecten en hun samenhang en samenwerking eigen moeten maken.

Inleiding

Programmeeronderwijs maakt een wezenlijk deel uit van elke informatica-opleiding. Veel opleidingen kiezen ervoor dit onderwijs te schoeien op objectgeoriënteerde leest. Voor studenten die voor het eerst met dit paradigma geconfronteerd worden blijkt het vaak lastig te zijn zich een voorstelling te maken van wat er bij de uitvoering van een programma nu precies gebeurt. Dit wordt voor een groot deel veroorzaakt door het feit dat zo’n objectgeoriënteerd programma een samenspel is van verschillende klassen en hun instanties: vele studenten kunnen zich geen goed beeld vormen van de manier waarop deze instanties zich ten opzichte van elkaar verhouden en hoe ze door het onderling verdelen van verantwoordelijkheden tesamen tot een correct werkend programma kunnen leiden.

In deze bijdrage beschrijven we een systematische grafische notatie, hier met *toestandskiekjes* aangeduid, die erop gericht is om de principes van de werking van objectgeoriënteerde programma's zichtbaar te maken. Niet alleen kunnen objecten met hun onderlinge samenhang worden weergegeven, maar ook “lopende methoden” binnen die objecten. Toestandskiekjes zijn bedoeld om studenten een gedegen basis te geven voor de visualisatie van het object-georiënteerde paradigma, zoals dit bijvoorbeeld in Java geïmplementeerd is. De notatie is uitgewerkt in het kader van de recente herziening van de cursus “Visueel Programmeren met Java” van de Open Universiteit (2001), en is geïnspireerd door de zgn. ‘geheugenmodellen’ in de daaraan voorafgaande editie.

De vorm van visualisatie die toestandskiekjes mogelijk maken is vooral bedoeld voor het *initiële* leertraject, waarin studenten zich de concepten van objecten en hun samenhang en samenwerking eigen moeten maken. Het is ondoenlijk om realistische systemen op deze wijze volledig weer te geven.

Objecten en referenties

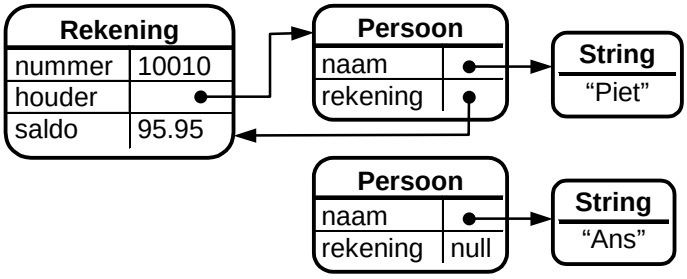
Een toestandskiekje is, zoals de naam al aangeeft, een weergave van de toestand van een (deel van een) Java-programma. Dit kiekje bevat in eerste instantie vooral objecten met hun inhoud — instanties dus van de klassen waaruit het programma bestaat of van bibliotheekklassen die door het programma gebruikt worden. Objecten worden gerepresenteerd door afgeronde rechthoeken

waarin de naam van de klasse is geschreven. De inhoud van een object is gegeven door de verzameling instantievariabelen tesamen met hun actuele waarden; in de graaf worden deze door een 2-koloms *waardentabel* in het object weergegeven, waarin de linkerkolom de naam van de variabele bevat en de rechterkolom de bijbehorende waarde. Waarden van een primitief type worden direct in de tabel geschreven; waarden van een referentietype zijn pijlen naar een (ander of hetzelfde) object in de graaf (of de speciale waarde *null*). Een voorbeeld waarin al deze elementen te vinden zijn is figuur 1, waarin we uitgaan van een Java-klasse Rekening met instantievariabelen

```
int nummer
Persoon houder;
double saldo;
```

en een klasse Persoon met instantievariabelen

```
String naam;
Rekening rekening;
```



Figuur 1: Objecten met waardentabellen en referenties

In figuur 1 is te zien dat de klasse String — en overigens ook de verpakkingsklassen Integer, Character etc. — een uitzonderingspositie innemen: instanties van deze klassen bevatten geen variabelennaam, maar uitsluitend de String-waarde. Dit omdat de naam van de instantievariabele onbekend en bovendien van geen belang is. Merk overigens op dat het onjuist zou zijn de String-waarde direct in de waardentabel (in dit geval van Persoon) te schrijven: String is geen primitief type maar een klasse, en String-waarden zijn dus instanties. (Dit komt bijvoorbeeld tot uiting in het feit dat we String-waarden nooit vergelijken door middel van de operator ==, maar altijd door middel van de methode equals.)

Klassenvariabelen en arrays

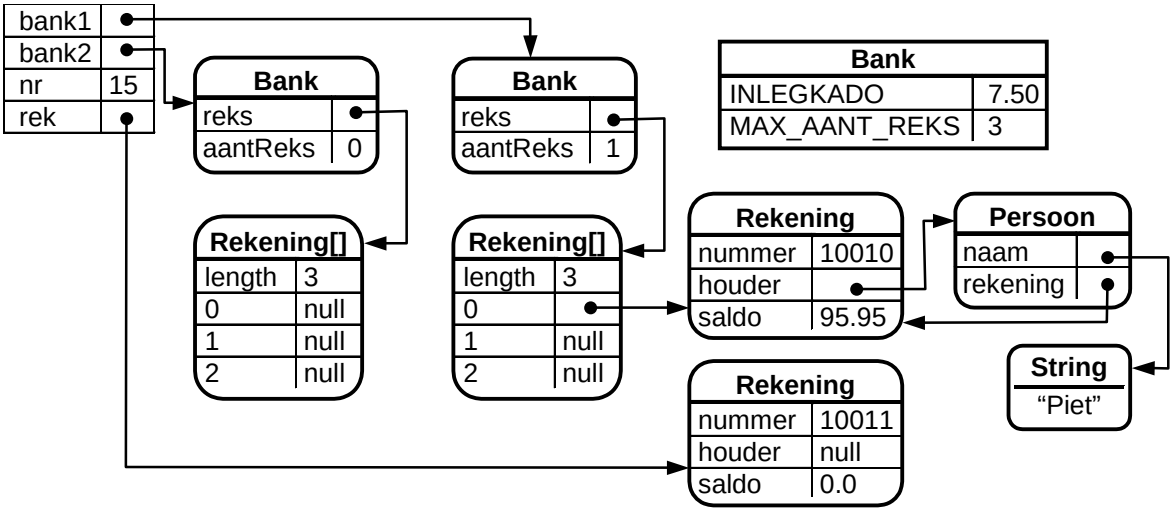
Tot zover weinig nieuws onder de zon: deze voorstelling van zaken is in vele Java-leerboeken terug te vinden; zie bijvoorbeeld Bishop (2001) en Niño en Hosch (2002). Iets interessanter wordt het zodra we de kiekjes uitbreiden met klassenvariabelen (*static variables*) en arrays.

- Een klassenvariabele wordt weergegeven door de betreffende klasse (waarbinnen de variabele is gedeclareerd) als nieuw soort knoop in de graaf op te nemen; in dit geval echter een gewone en geen afgeronde rechthoek. De variabele zelf bevindt zich weer in een waardentabel binnen de klasse.
- Een array is een instantie (dus een afgeronde rechthoek waarin bovenaan het type is vermeld) met daarin een waardentabel, waarvan de linkerkolom ditmaal geen instantievariabelen maar array-indices bevat, plus het speciale attribuut length, dat het aantal array-elementen aangeeft.

Uitgaand van een klasse Bank met instantievariabelen

```
static int MAX_AANT_REKS = 3;
static double INLEGKADO = 7.50;
Rekening[] reks;
int aantReks;
```

is een mogelijke toestand gegeven in figuur 2.



Figuur 2: Toestandskiekje met klassenvariabelen en arrays

Figuur 2 bevat bovendien een extra waardentabel (links boven) die niet in een object of klasse ingebed is. Dit soort 'losse' tabellen dienen om in voorbeelden eenvoudig te kunnen refereren naar variabelen die in een gegeven context (bijvoorbeeld een klasse of methode) gedeclareerd zijn, maar waarvan het op dat moment niet van belang is die context nader weer te geven. In Figuur 2 is er bijvoorbeeld kennelijk sprake van twee variabelen van het type Bank, een int-variabele en een variabele van het type Rekening, met de in het kiekje vastgelegde waarden.

Figuren 1 en 2 illustreren de volgende aspecten van het objectgeoriënteerde paradigma:

- Het verschil tussen primitieve en referentiewaarden, en het feit dat referentiewaarden pijlen naar objecten zijn, ofwel de speciale waarde null die duidt op de afwezigheid van een referentie.
- Het feit dat objecten zelf geen vaste naam hebben, maar alleen aangeduid kunnen worden door middel van variabelennamen die in een bepaalde context gedeclareerd zijn. (We beschouwen in dit opzicht de UML [3], waarin objecten in veel gevallen van een naam worden voorzien, als gebrekkig.)
- Het feit dat arrays zelf instanties zijn, en een variabele van een array-type dus als waarde null kan hebben — hetgeen heel iets anders is dan een array met 0 elementen of een array-element met de waarde null.
- Het verschil tussen klassenvariabelen en instantievariabelen, en het feit dat de eerste globaal toegankelijk zijn terwijl de laatste altijd in de context van een object gedefinieerd zijn.

Lopende methoden en constructoren

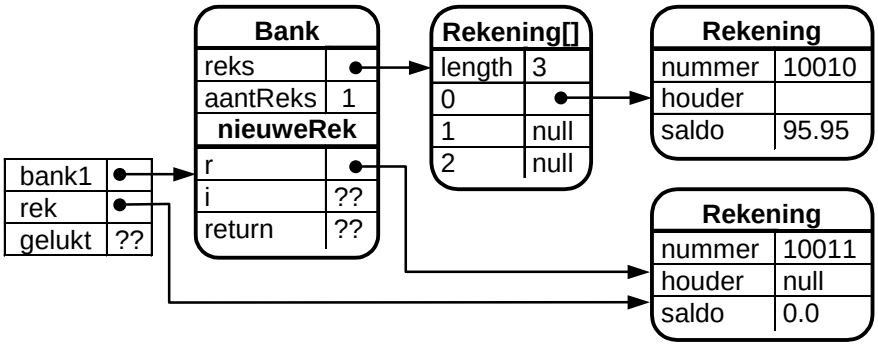
De kracht van toestandskiekjes is echter vooral gelegen in het feit dat ook de formele parameters en lokale variabelen van lopende methoden er probleemloos in kunnen worden opgenomen. Zo'n lopende methode wordt weergegeven binnen het uitvoerende object, door middel van een nieuwe waardentabel met daarboven de naam van de methode. Heeft de klasse Rekening bijvoorbeeld methoden om het rekeningnummer op te vragen en het saldo bij te stellen:

```
int getNummer() {
    return nummer;
}
void setSaldo(double saldo) {
    this.saldo = saldo;
}
```

en Bank een methode om een rekening toe te voegen:

```
boolean nieuweRek(Rekening r) {
    for (int i = 0; i < aantReks; i++)
        if (reks[i].getNummer() == r.getNummer())
        return false;
    if (aantalRek < MAX_AANT_REKS) {
        r.setSaldo(INLEGKADO);
        reks[aantReks] = r;
        aantReks++;
        return true;
    }
    else return false;
}
```

dan is Figuur 3 een voorbeeld van een toestand die (uitgaand van Figuur 2) bereikt wordt bij een aanroep gelukt = bank1.nieuweRek(rek), voordat de for-lus in nieuweRek is begonnen:



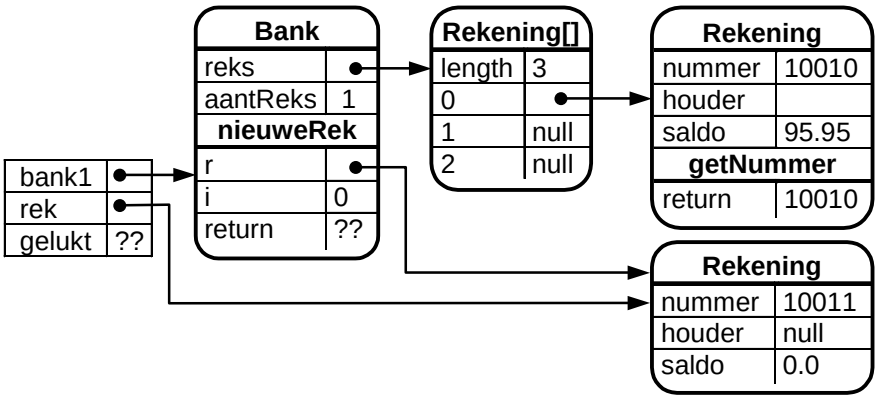
Figuur 3: Toestand aan het begin van de uitvoering van gelukt = bank1.nieuweRek(rek)

Aan het Bank-object is hier geïllustreerd hoe een lopende methode wordt weergegeven. De formele parameter r heeft een kopie van de waarde van de actuele parameter rek gekregen; omdat het om een referentiewaarde gaat is zo'n kopie een pijl naar hetzelfde object. De lokale variabele i is al wel in de tabel van nieuweRek opgenomen maar heeft nog geen geldige waarde (in Java moeten lokale variabelen expliciet geïnitialiseerd worden); dit is weergegeven door de aanduiding '??'. Bovendien bevat de tabel van nieuweRek een "pseudo-variabele" return; deze wordt gebruikt om de terugkeerwaarde in aan te geven.

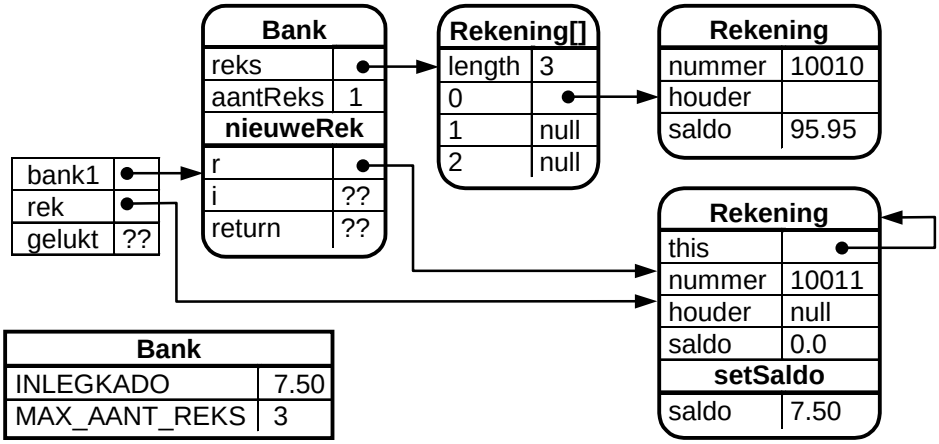
Merk op dat de instantievariabele houder van het bovenste Rekening-object in Figuur 3 leeg is gelaten: er staat een pijl noch null. Hiermee wordt aangegeven dat de waarde van deze variabele voor de doel van het voorbeeld oninteressant is en buiten beschouwing wordt gelaten.

We kunnen nu de werking van nieuweRek simuleren door de toestand na elke programmastap in een nieuw kiekje weer te geven. Een paar voorbeelden volgen.

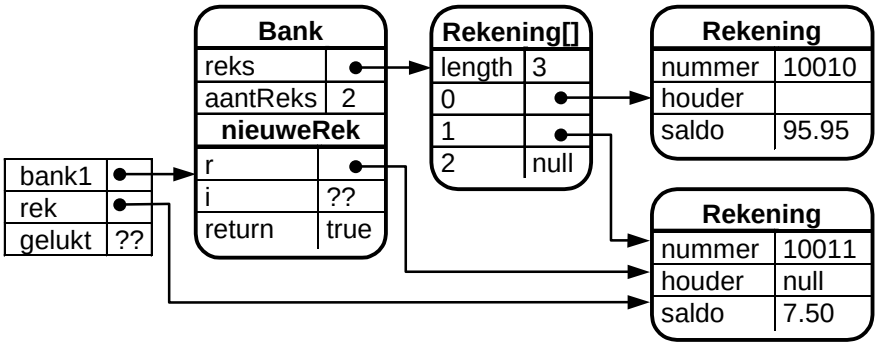
- Bij de test in de for-lus of het nummer van de nieuwe rekening nog niet bestaat wordt de methode getNummer op reks[0] aangeroepen (figuur 4).
- Nadat de nieuwe rekening de "unieke nummer"-test heeft doorstaan wordt het standaard inlegkado er als beginsaldo op gestort, door een aanroep van setSaldo (figuur 5). Voor de duidelijkheid hebben we in dit kiekje aan de betreffende Rekening-instantie de pseudo-variabele this toegevoegd, die in de code van setSaldo wordt gebruikt om verschil te maken tussen de lokale variabele saldo en de instantievariabele van dezelfde naam.
- Vervolgens worden de array reks en de teller aantReks aangepast (figuur 6). De terugkeerwaarde wordt op true gezet.



Figuur 4: Toestand bij het testen op overeenkomstige rekeningnummers



Figuur 5: Toestand bij het toewijzen van het INLEGKADO aan de nieuwe Rekening



Figuur 6: Toestand vlak voor terugkeer van nieuweRek

Uiteindelijk is de methode nieuweRek klaar en levert de waarde true op, die aan de variabele gelukt wordt toegewezen; de waardentabel van nieuweRek verdwijnt uit de Bank-instantie.

Lopende constructoren worden in principe net zo weergegeven als lopende methoden, met dien verstande dat in dat geval het uitvoerende object (waarbinnen de constructor met zijn waardentabel wordt geplaatst) voorafgaand aan de uitvoering van de constructor eerst gecreëerd wordt. Ook klassenmethoden kunnen op soortgelijke manier worden weergegeven: de uitvoerder is dan niet een instantie, maar de klasse zelf, weergegeven door een gewone rechthoek.

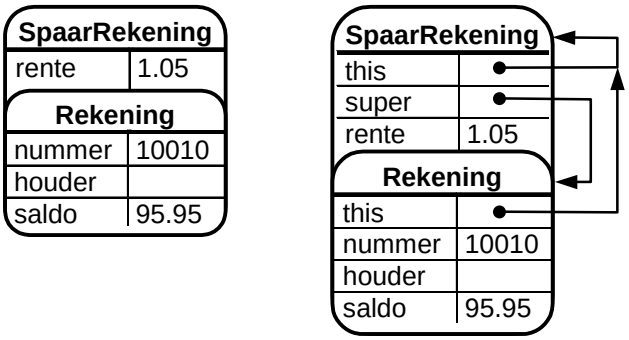
Bovenstaande kiekjes illustreren de volgende aspecten van het objectgeoriënteerde paradigma:

- Het feit dat lokale variabelen, inclusief formele parameters, in hun initialisatie en levensduur verschillen van instantievariabelen, maar overigens op dezelfde manier gebruikt worden.

- Het feit dat een methode (of constructor) geacht wordt *lokaal* binnen een object te werken, en dat de ene methode gedurende executie een andere kan aanroepen.
- De rol van de terugkeerwaarde van een methode.

Overerving

Een belangrijk aspect dat in bovenstaande voorbeelden nog *niet* aan bod is gekomen is *overerving*. Dit vergt een uitbreiding van de notatie, echter geen wezenlijke: de door overerving aangebrachte abstractielagen in een object kunnen we eenvoudig weergeven door er voor de superklasse een eigen waardentabel in op te nemen. Als we bijvoorbeeld een klasse SpaarRekening definiëren die erft uit Rekening en daaraan een attribuut rente toevoegt (een gebroken getal), dan kan een object van SpaarRekening weergegeven worden door het kiekje in figuur 7. Dit figuur bevat twee representaties van hetzelfde object. In de rechter representatie hebben we ook pseudo-variabelen this en super opgenomen: this wijst altijd naar het gehele object, super naar het deel van het object dat de superklasse representeert.



Figuur 7: Instantie van de subklasse SpaarRekening; rechts met pseudo-variabelen this en super

Conclusie

De hier gepresenteerde toestandskiekjes kunnen in het initiële programmeeronderwijs een bijdrage leveren aan een beter begrip van het objectgeoriënteerde paradigma, doordat ze studenten de mogelijkheid bieden zich een beeld te vormen van wat er (op een zeker niveau van abstractie) gebeurt bij de verwerking van een objectgeoriënteerd programma. Toestandskiekjes zijn ontwikkeld voor de programmeertaal Java in het kader van de herziening van de cursus “Visueel Programmeren met Java” van de Open Universiteit (zie [4]), maar kunnen ook voor andere programmeertalen gebruikt worden.

Het zij opgemerkt dat er geen grafische standaardnotatie bestaat die het hier nagestreefde doel dient. In het bijzonder voorziet de standaard “diagramtaal” UML (zie [3]) niet in een diagramsoort waarmee een dergelijke visualisatie mogelijk is. Weliswaar kunnen in klassen-diagrammen eventueel objecten worden opgenomen, maar hun onderlinge samenhang kan niet worden aangegeven en hun betekenis in het kader van zo’n diagram is verre van duidelijk. Anderzijds biedt UML de mogelijkheid toestands*overgangen* te modelleren (door middel van zgn. *state diagrams*); echter, de weergave van de toestanden zelf, die in onze toestandskiekjes centraal staat, blijft vrijwel geheel buiten beschouwing.

Voor wat betreft de representatie van Java claimen we dat de weergave in toestandskiekjes *correct* is — ze vormen een grafische representatie van een formeel model voor de werking van de Java virtuele machine — en bovendien *volledig* — alle aspecten van Java-programma’s

kunnen erin worden uitgedrukt, inclusief klassenattributen (*static variables*), formele parameters en lokale variabelen van constructoren en methoden, arrays, en overerving.

De praktijk zal moeten uitwijzen of de visualisatiemogelijkheden die toestandskiekjes bieden in de praktijk inderdaad leiden tot een helderder begrip van het objectgeoriënteerde paradigma. De cursusevaluatie van de Open Universiteit zal hierover uitsluitsel kunnen geven.

Referenties

J. Bishop (2001), *Java gently*, Addison Wesley, derde editie.

J. Niño en F.A. Hosch (2002), *An Introduction to Programming and Object-Oriented Design using Java*, John Wiley & Sons.

Open Universiteit (2001), *Visueel Programmeren met Java*, cursus T.25.1.3.1, Open Universiteit Nederland, (derde, gewijzigde druk).

J. Rumbaugh, I. Jacobson en G. Booch (1998), *The Unified Modeling Language Reference Manual*, Addison-Wesley Pub Co.