# 1

# HOOFDSTUK 1

# Inleiding

Dit boek beschrijft alle aspecten van de standaard op het gebied van objectgeoriënteerde systeemanalyse en ontwerp: de Unified Modeling Language (UML). Daarnaast wordt een werkwijze gegeven om op een praktische, doelgerichte manier met de UML-diagrammen om te gaan. Het geheel wordt begeleid door een compleet uitgewerkte case.

Dit boek is bedoeld voor mensen die vanuit een automatiseringsachtergrond bezig zijn met systeemontwikkeling, met andere woorden voor collega's. Zowel mensen met kennis van objecttechnologie, met inbegrip van UML, als mensen die dit gebied, zoals men zegt, ongehinderd door enige kennis betreden, zullen dit boek kunnen waarderen. Doel van dit boek is de lezer tot een effectief en zelfstandig gebruik van UML te leiden.

Zoals gebruikelijk in dit vakgebied zijn veel termen binnen objecttechnologie van oorsprong Engels. Enkel wanneer de overeenstemmende Nederlandse term werkelijk veelvuldig in de Nederlandstalige literatuur wordt gebezigd, is er in dit boek voor gekozen de Nederlandse term te gebruiken. In de overige gevallen is de Engelse term gebruikt, zodat de lezer makkelijk aansluiting kan vinden bij de overvloedige Engelstalige literatuur op dit terrein. Om die aansluiting te garanderen worden ook bij Nederlandse termen de corresponderende Engelse termen aangegeven en zijn deze in de index terug te vinden.

De structuur van dit boek is bepaald door het belang dat wij hechten aan het gebruik van bepaalde diagrammen en door de wijze waarop wij in de praktijk het maken van de diagrammen op elkaar laten volgen. Na een inleidend hoofdstuk over objecttechnologie volgt een overzicht van UML in hoofdstuk 2 ('Systeemontwikkeling met UML'). Vervolgens wordt voor elk UML-diagram eerst een hoofdstuk gewijd aan de concepten die in het diagram worden toegepast en de werkwijze om een dergelijk diagram te produceren. Daarna volgt een hoofdstuk waarin deze theorie wordt toegepast op een case.

De nieuwe case 'Het energiesysteem van Hans en Jacqueline' vormt een doorlopende illustratie van het praktisch gebruik van UML. In deze case wordt ook duidelijk getoond hoe in elke fase wordt voortgebouwd op de resultaten van de voorgaande fasen. Omdat de implementatie van de case geheel onafhankelijk van UML is, wordt ze in dit boek niet verder uitgewerkt. Iedere uitwerking ervan zal onvermijdelijk zijn toegespitst op de gekozen omgeving, programmeertaal, database en de beschikbare klassebibliotheken. Hierdoor zou zo'n uitwerking slechts voor een klein deel van de lezers relevant zijn.

Hoofdstuk 2 is verplichte kost voor eenieder die vanuit welke rol dan ook betrokken is bij softwareprojecten. De hoofdstukken 4 tot en met 9 richten zich op de analyse van software vanuit business- en gebruikersperspectief. Deze hoofdstukken zijn interessant voor IT'ers die samen met de gebruikers en domeinexperts de functionele kant van een applicatie beschrijven. Zij dienen in staat te zijn deze modellen op te stellen. Voor de meer technische architecten en ontwikkelaars zijn deze hoofdstukken van belang omdat de hierin beschreven modellen voor hen het uitgangspunt voor het ontwerp en implementatie vormen.

# 2

# HOOFDSTUK 2

# Systeemontwikkeling met UML

De visuele modelleertaal Unified Modeling Language (UML) is een gezamenlijk product van een groot aantal bedrijven. Het is een standaard die naar aanleiding van een 'request for proposals' van de Object Management Group (OMG) ontwikkeld is door vooraanstaande personen binnen het objectgeoriënteerde vakgebied.

## 2.1 Leerdoelen

In dit hoofdstuk leren we uit welke onderdelen UML bestaat en hoe die in de systeemontwikkeling gebruikt kunnen worden. We leren ook dat UML op veel manieren gebruikt kan worden, van heel globaal tot heel gedetailleerd, afhankelijk van de doelstelling. Ook zullen we de samenhang tussen de diagrammen zien, zodat we de diagrammen die in de volgende hoofdstukken behandeld worden goed kunnen plaatsen. Dit hoofdstuk is van belang voor iedereen die bij een project waarin UML wordt gebruikt betrokken is. Met het in dit hoofdstuk beschreven overzicht ben je in staat om de rol van jouw werk en dat van anderen goed binnen de totale context te plaatsen.

## 2.2 Het ontstaan van UML

Begin jaren negentig kwamen de eerste objectgeoriënteerde systeemontwikkelingsmethoden in de publiciteit. Daaronder waren de door James Rumbaugh en anderen ontwikkelde methode Object Modeling Technique (OMT) [Rumbaugh91] en de methode van Grady Booch [Booch94]. Deze beide methoden waren zeer populair. In 1995 is James Rumbaugh verbonden geraakt aan Rational, het bedrijf waarbinnen Grady Booch zijn Booch-methode op de markt heeft gebracht. Rumbaugh en Booch zijn samen gaan werken om hun beider methoden te integreren tot één methode, de Unified Method genaamd [Booch95]. In 1996 is ook het bedrijf waarin Ivar Jacobson zijn OOSE-methode [Jacobson92] – ook wel Objectory genoemd – heeft ontwikkeld door Rational overgenomen. Ivar Jacobson is bekend vanwege de use- cases die hij in zijn methode gebruikt. Daarmee was duidelijk dat ook aspecten van de OOSE-methode in deze gezamenlijke methode geïntegreerd zouden gaan worden.

Ongeveer gelijktijdig met dit integratieproces kwam de OMG met een verzoek om voorstellen voor een standaard op het gebied van objectgeoriënteerde modellering. De OMG wilde echter geen gestandaardiseerde methode, dat wil zeggen een werkwijze gecombineerd met eisen aan de resultaten van die werkwijze, ze wilde 'slechts' de resultaten van de werkwijze standaardiseren. Op deze wijze kunnen alle ontwikkelaars hun eigen werkwijze gebruiken, maar zijn hun resultaten uitwisselbaar. Rational had intussen contact gezocht met een aantal andere bedrijven. Gezamenlijk werd de Unified Method omgevormd tot een Unified Modeling Language en als voorstel voor een mogelijke standaard bij de OMG ingediend.

In totaal werden zes voorstellen bij de OMG ingediend. Daarvan was het voorstel van Rational het meest uitgewerkt. De andere voorstellen bevatten echter ook waardevolle zaken. Besloten werd dan ook om de voorstellen tot één Unified Modeling Language te integreren. Een van de zaken die aan UML werden toegevoegd is de Object Constraint Language (OCL) [Warmer03], welke deel uitmaakte van het IBM/ObjecTime voorstel. In 1997 is UML de standaard geworden.

In 2006 is de gereviseerde versie 2.1 van deze standaard [OMG06] uitgebracht. Daarna zijn nog enkele versies uitgebracht waarin kleine wijzigingen zijn verwerkt. In dit boek wordt versie 2.3 beschreven.

## 2.3 Een taal, geen methode

Zoals uit de vorige paragraaf duidelijk is geworden, is UML geen methode, maar een taal. UML standaardiseert geen werkwijze, wel begrippen en diagrammen waarin die begrippen gebruikt kunnen worden. De werkwijze die in dit boek beschreven wordt is een werkwijze die in de praktijk van de auteurs zijn nut heeft bewezen. Omdat de werkwijze geen onderdeel is van de standaard, zal in dit boek de werkwijze strikt gescheiden gehouden worden van aspecten die wel behoren tot de UML-standaard.

## 2.4 Modeling Maturity Levels

Waarom modelleren we eigenlijk? In de praktijk blijkt dat modellen op meerdere manieren voor verschillende doeleinden gebruikt worden. Om hier orde in te scheppen definiëren we verschillende niveaus van modelleren, Modeling Maturity Levels (MMLs) genoemd. Deze niveaus zijn nuttig om vast te stellen waar iemand zich bevindt en naar welk niveau iemand streeft. We onderscheiden de volgende zes niveaus.

### *Modeling Maturity Level 0: Geen specificatie*

Op niveau nul bevindt de specificatie van de te bouwen software zich geheel in het hoofd van de ontwikkelaar(s). Omdat er niets wordt vastgelegd ontstaan er snel conflicten tussen de gebruiker(s) en ontwikkelaar(s) over wat nu precies de bedoeling is. De programmeur neemt alle beslissingen. Als de oorspronkelijke programmeur vertrekt, gaat alle kennis over het deel van het systeem dat door hem is gebouwd verloren. Feitelijk is op dit niveau geen professionele softwareontwikkeling mogelijk.

### *Modeling Maturity Level 1: Tekstuele specificaties*

De specificatie wordt vastgelegd in één of meer documenten, volledig in natuurlijke taal. De specificatie zorgt ervoor dat de ontwikkelaar en de gebruiker de afspraken expliciet maken. Omdat natuurlijke taal per definitie ambigu is, zal bij een dergelijke specificatie toch veel onduidelijk blijven. De ontwikkelaar zal nog steeds veel beslissingen moeten nemen. Een tweede probleem is het feit dat de specificatie na het wijzigen van de software moeilijk up-to-date te houden is.

### *Modeling Maturity Level 2: Tekst met diagrammen*

De specificatie op dit niveau wordt vastgelegd in één of meer tekstuele documenten, zoals op Modeling Maturity Level 1. Dit wordt echter aangevuld met enkele diagrammen op hoog niveau, vaak om de algemene architectuur van het systeem te verduidelijken. Het voordeel van het gebruik van diagrammen op dit niveau is dat ze helpen om de tekst beter te structureren en beter te kunnen begrijpen. De diagrammen worden hier voornamelijk gebruikt als communicatiemiddel. Alle nadelen van het eerste niveau gelden hier nog steeds.

### *Modeling Maturity Level 3: Modellen met tekst*

Op niveau drie wordt de specificatie vastgelegd in een aantal modellen. Met model wordt hier bedoeld: een diagram of tekst geschreven in een niet-natuurlijke taal waarvan de betekenis helder en eenduidig is. UML is een voorbeeld van zo'n taal. Andere voorbeelden zijn: Z [Wordsworth92], SDL [Ellsberger97], maar zelfs programmeertalen kunnen binnen deze definitie als specificatietaal worden beschouwd. De broncode is een specifieke, niet ambigue specificatie van het systeem, zij het dat broncode meestal minder geschikt is om mensen begrip te geven van de werking van het systeem.

Bijbehorende tekstdocumenten beschrijven de details en de motivatie achter de modellen. Op dit niveau zijn de modellen een afspiegeling van het systeem. De modellen worden, meestal met de hand, omgeschreven naar code. Omdat de modellen een preciezere specificatie vormen dan de tekst op niveau twee zal de programmeur veel minder onduidelijkheden tegenkomen en hoeft hij derhalve minder eigen beslissingen te nemen.

### *Modeling Maturity Level 4: Exacte modellen*

Op niveau vier vormen de modellen het belangrijkste onderdeel van de specificatie van een systeem. Op dit niveau hebben de modellen een dergelijke samenhang dat men ook zou kunnen spreken van één enkel model dat bestaat uit een aantal verschillende onderdelen. Zo bestaat een UML-specificatie uit verschillende diagrammen. Elk diagram kan beschouwd worden als een apart model, maar als het onderlinge verband tussen de diagrammen groot genoeg is vormt het geheel één model.

De modellen op dit niveau zijn precies genoeg om een directe link te hebben met de code. Tekstdocumenten worden nog steeds gebruikt, maar vormen nu uitleg bij de modellen en beschrijven de motivatie achter gemaakte keuzes. Op dit niveau hoeven programmeurs maar weinig eigen beslissingen te nemen. Omdat automatische generatie van belangrijke delen van de code mogelijk is, kunnen de modellen en de code up-to-date gehouden worden. Om dezelfde reden is het op dit niveau mogelijk om sterk iteratief te werken.

### *Modeling Maturity Level 5: Alleen modellen*

Op niveau vijf zijn de modellen volledig en precies genoeg om alle details te beschrijven. De code kan geheel uit de modellen gegenereerd worden. Net zoals Assembler vandaag gebruikt wordt, kan de code in principe geheel onzichtbaar blijven voor de ontwikkelaar. Op dit niveau is de modelleertaal feitelijk een hoger niveau programmeertaal geworden. Vandaag de dag is dit niveau helaas nog onbereikbaar, met uitzondering van bijzondere, beperkte domeinspecifieke omgevingen.

Het ambitieniveau van dit boek is om UML te kunnen gebruiken op niveaus drie en vier. Op de niveaus nul en één worden geen modellen gebruikt. De diagrammen op niveau twee hebben een hoog 'plaatjes'-gehalte. Het zijn slechts schetsen zonder een duidelijke en precieze betekenis. Niveau vijf is op dit moment nog niet haalbaar. Er

zijn (nog) geen algemeen bruikbare specificatietalen waarin alle delen van een softwaresysteem in voldoende detail beschreven kunnen worden.

## 2.5 De diagrammen en hun samenhang

UML biedt een aantal diagrammen die gezamenlijk de specificatie van het softwaresysteem vormen.

- Het use-case-diagram toont hoe het systeem kan worden gebruikt door externe entiteiten zoals menselijke gebruikers.
- Het klassediagram (Eng. class diagram) toont de statische structuur van het softwaresysteem weergegeven als klassen en hun relaties.
- Het objectdiagram toont de statische structuur van het softwaresysteem weergegeven als objecten en hun relaties.
- Het sequence-diagram toont de volgorde in tijd van de boodschappen die in het systeem verstuurd en ontvangen worden.
- Het communicatiediagram toont hoe de objecten samenwerken om een doel te bereiken.
- Het toestandsdiagram toont de toestanden waarin een object zich kan bevinden gedurende zijn levensloop.
- Het activiteitsdiagram toont de activiteiten die door een deel van het systeem worden uitgevoerd, inclusief eventueel parallellisme.
- Activiteitsdiagrammen kunnen ook gebruikt worden voor het beschrijven van bedrijfsprocessen en de workflow.
- Het componentdiagram toont de verdeling van het gehele systeem in componenten en de relaties tussen die componenten.
- Het deployment-diagram toont hoe de softwarecomponenten in een bepaalde systeemconfiguratie worden gebruikt.

Het use-case-diagram wordt ook wel een requirements-diagram genoemd. Het klassediagram en het objectdiagram vormen samen de statische diagrammen. Het sequence-, communicatie-, toestands- en activiteitsdiagram worden ook wel de dynamische diagrammen genoemd. Het component- en deployment-diagram zijn de implementatiediagrammen.

Naast de diagrammen biedt de UML-standaard een taal waarin beperkingen, condities en regels die gelden in de diagrammen weergegeven kunnen worden. Deze taal heet de Object Constraint Language (OCL). Hoofdstuk 6 gaat in op het gebruik van deze taal.

Figuur 2-1 geeft een overzicht van de samenhang van de verschillende diagrammen. Een pijl van het ene diagram naar het andere geeft aan dat het tweede diagram informatie uit het eerste diagram nodig heeft. Een gestippelde pijl betekent dat het eerste diagram invloed heeft op het tweede, zonder dat delen van het tweede diagram direct uit het eerste voortkomen. In het midden van de figuur zie je een dikke pijl. Het klassediagram (in verschillende stadia) dat in het traject langs die pijl wordt ontwikkeld, is het centrale diagram in de methode.