Inhoud

Inleiding

If you want to build a ship, don't gather the men to find the wood, prepare the tools or divide up the work and delegate tasks. Instead teach the men the longing for the endless, wide ocean.

Antoine de Saint-Exupéry

"Papa, kunnen hier herten oversteken?", vraagt mijn dochter Sam vanaf de achterbank van de auto. We zijn op vakantie in Frankrijk en rijden door een bos. Zojuist zijn we een verkeersbord gepasseerd dat overstekend wild aangeeft. Ik kijk in mijn spiegel en antwoord "Herten, maar ook andere dieren. Misschien wel wilde zwijnen of vossen." Even is het stil op de achterbank. Sam (7) en Spijk (4) kijken elkaar aan. "Maar dat stond niet op dat bord, pap", zegt Sam dan.
Zie hier de uitdaging van modelleren. Verkeersborden zijn nagenoeg overal gelijk. Driehoekige verkeersborden geven waarschuwingen weer, zoals het bord voor overstekend wild. Ronde verkeersborden geven aanwijzigingen. Denk maar aan het verbod om een straat in te rijden. Een rode rand om een verkeersbord geeft een verbod aan. Een eenvoudige grammatica, waarvan ieder land zijn eigen dialect kent. Maar toch zijn we veilig op vakantie in Frankrijk. We kennen de grammatica.
Het verkeersbord toont alleen een springend hert. Geen wild zwijn. Geen vos. Het verkeersbord *modelleert* overstekend wild. Het is geen exacte representatie van de werkelijkheid, maar een vereenvoudiging ervan. De voordelen zijn duidelijk. Zelfs als je met tachtig kilometer per uur voorbij het bord rijdt, is duidelijk wat er is bedoeld.
Toch heeft Sam gelijk. Ieder model, hoe eenvoudig ook, nodigt uit tot interpretatie. Zelfs een verkeersbord voor overstekend wild. Een hert is geen wild zwijn.

Gezichtspunten

Ook in de systeemontwikkeling zijn de voordelen van modelleren reeds lang bekend. Wie herinnert zich niet het program structure diagram en het data flow diagram. Tientallen modelleertechnieken zijn de revue gepasseerd. Sinds een aantal jaren gelden inmiddels de modelleertechnieken van de Unified Modeling Language, kortweg UML, als gemeengoed. Use cases, activity diagrammen en klassendiagrammen worden gebruikt in projecten van Seattle tot Sydney en van Helsinki tot Kaapstad. Beschouw ze gerust als de verkeersborden van de systeemontwikkeling.

Systeemontwikkelprojecten kennen echter vele gezichtspunten. In projecten worden talrijke rollen onderscheiden. Opdrachtgevers, gebruikers, analisten, architecten, ontwerpers, ontwikkelaars, web designers, databaseontwikkelaars, testers en diverse andere. Ieder beschouwt het project vanuit een eigen gezichtpunt. Deze bonte verzameling verschillende, afwijkende en soms tegenstrijdige gezichtspunten maakt het op tijd en binnen budget opleveren van projecten moeilijk. Opdrachtgevers hebben een beperkt budget, gebruikers herzien voortdurend hun wensen, ontwerpers en ontwikkelaars spreken niet dezelfde taal, en testers worden te laat in projecten betrokken om nog een positieve bijdrage te kunnen leveren.
Veel verschillen in gezichtspunten zijn te overbruggen door te modelleren. Franse verkeersborden wijken niet noemenswaardig af van Nederlandse verkeersborden. Modelleertechnieken helpen ook in systeemontwikkelprojecten de verschillen te overbruggen. Procesdiagrammen en use cases ondersteunen de communicatie over requirements. Activity diagrammen vereenvoudigen de communicatie tussen ontwerpers en testers en sequence diagrammen en klassendiagrammen verbeteren de samenwerking tussen ontwerpers en ontwikkelaars.

Dialecten

Modelleertechnieken ondersteunen communicatie. Net zoals verkeersborden helpen het verkeer in goede banen te leiden. De modelleertechnieken van UML zijn inmiddels onmisbaar voor systeemontwikkelprojecten. Toch helpt het omarmen van UML niet alle problemen uit de wereld. UML is geen *silver bullet*.
Als de interpretatie van overstekend wild al niet eenduidig is, bedenk dan welke verschillen in interpretatie ontstaan bij het modelleren in systeemontwikkelprojecten. En inderdaad. Er zijn geen twee projecten die UML op dezelfde manier gebruiken. Net als verkeersborden per land een eigen dialect van dezelfde grammatica vormen, kent ieder project een eigen dialect van de grammatica van UML. En dat is niet bevorderend voor de communicatie.
Anders dan de grammatica van verkeersborden is de grammatica van UML overdadig complex. Het is vooral deze complexiteit die de dagelijkse toepassing frustreert. Niets is verhelderender dan een team gade te slaan tijdens een eerste workshop waarin use cases worden gemodelleerd. Hoe begin je? Hoe modelleer je use cases eigenlijk? Wanneer leg je relaties aan tussen use cases? En waarom eigenlijk?
En dat terwijl modelleren vereenvoudiging nastreeft. Een rond rood bord met een horizontale witte balk geeft aan dat het niet verstandig is een straat in te rijden. Veel beter dan een groot bord waarop in tekst dezelfde waarschuwing staat. Lees dat maar eens als je met honderd kilometer per uur de afrit van de snelweg oprijdt. Bondigheid is kracht. Belangrijk voor het toepassen van UML is dan ook hoe een diagram wordt opgesteld. Welke modelleertechnieken gebruik je? Wanneer is een diagram af? Welke detailniveau streef je na?

Tenslotte is UML nog steeds onvolledig. Dit geldt ook voor UML 2.0. Voor diverse gezichtspunten biedt de modelleertaal geen soelaas, Voor het modelleren van hiërchische bedrijfsprocessen of user interfaces kent UML geen technieken. Voor het modelleren van gegevens wordt een klassendiagram gebruikt. UML is niet de heilige graal. Het is gewoon een goed hulpmiddel in systeemontwikkelprojecten. Mits pragmatisch toegepast. Dit boek besteedt daarom niet alleen aandacht aan de grammatica van UML, maar vooral aan het praktisch gebruik van UML in alledaagse projecten.

Uit de praktijk

Veel is al geschreven over *wat* UML is. Te weinig is nog geschreven over *hoe* UML kan worden ingezet. De in dit boek voorkomende modelleertechnieken, tips en voorbeelden komen voort uit de alledaagse praktijk. Het boek is een handreiking voor de uitdagingen van systeemontwikkelprojecten.
Dit boek beslaat daarbij bewust een volledig systeemontwikkelproject. Van idee tot applicatie. Het is ontstaan vanuit vijftien jaar ervaring in projecten. Zo is het user interface diagram ontstaan, maar ook het gebruik van een procesdiagram, activity diagrammen om testscenario's en testgevallen te identificeren, en het modelleren van een gegevensdiagram in een klassendiagram. Ook overbrugt dit boek de kloof tussen ontwerp en code aan de hand van een makkelijk toepasbare referentiearchitectuur. Het biedt architecten, ontwerpers, ontwikkelaars en testers de handreikingen die voor het toepassen van van UML nodig zijn, zowel voor eerdere versies van UML als voor UML 2.0.
Bovendien is uit deze ervaring een nauwe samenhang tussen de verschillende modelleertechnieken gedistilleerd. Een procesdiagram is een goed uitgangspunt voor het modelleren van use cases. Er is een sterk verband tussen deze use cases en het user interface diagram. Het klassendiagram voor de applicatie wordt aangevuld uit de uitwerking van user interface diagrammen en sequence diagrammen. UML biedt deze samenwerking maar mondjesmaat. De modelleertaal beschrijft vooral de grammatica van zijn modelleertechnieken, en maar zeer beperkt het gebruik ervan.
Alle voorbeelden in het boek zijn ontleend aan dezelfde eenvoudige case, die uitstekend past in het huidige tijdsbeeld. De online dating service Dare2Date. Dit boek volgt het project dat de webapplicatie voor de dating service realiseert. Tijdens het modelleren van requirements, de user interface, de functionaliteit en zelfs tijdens het ontwikkelen van de code. Het idee voor deze case is ontstaan nadat een vriendin zich bij zo'n service aanmeldde. Binnen enkele maanden vond ze overigens een partner. Inmiddels woont ze samen.
De codevoorbeelden in het boek zijn geschreven in C#, de belangrijkste programmeertaal uit Microsoft's .NET. Ik heb voor deze taal gekozen omdat C# leesbaar is voor zowel Microsoft- als Java-ontwikkelaars. De syntax van C# en Java ontloopt elkaar nauwelijks. Zo kent dit boek een prettige mix tussen theorie en praktijk. Het is pragmatisch, maar met oog voor detail en de sterk wisse-

lende context van systeemontwikkelprojecten. Geen twee projecten zijn immers gelijk. De administratie van een voetbalvereniging is niet hetzelfde als het automatisch factureren van één miljoen verzekerden. Een hert is geen wild zwijn.

Website

Alhoewel dit boek met veel zorg is geschreven en omgeven, is het toch niet uitgesloten dat er foutjes in voorkomen. Uitbreidingen, (code)voorbeelden en eventuele correcties op het boek zijn te vinden op mijn website www.sanderhoogendoorn.com. U bent van harte welkom ook uw reacties op het boek te plaatsen op de website.

Dankwoord

Ruim twee jaar heb ik met veel plezier en energie aan dit boek gewerkt. Ik had dit nooit kunnen volbrengen zonder de onvolprezen steun en het onmetelijke geduld van Babette, Sam, Spijk en nu ook Boet. Iluf.

Het boek is mede tot stand gekomen dankzij de kennis en adviezen van Martijn Blankestijn, Jacques Bloemendal, Dajo Breddels, John van den Broek, Derk Bijmold, Silvio Cacace, Alistair Cockburn, Erwin Derksen, Erwin Doreleijers, Stephan Ferrier, Dennis Gruijs, Peter den Hartog, Kevlin Henney, Johanna Herfkens, Gernand Hierink, Ralph Hofman, Peter Janssen, Gerard Jonker, Huddie Klein, Peter de Kruijff, Rick van der Lans, Stephanie Leyenaar, René Luyben, Robert Martin, Hans Nellestijn, Annelies Nijboer, Rudy Pelsmaekers, Ruud Rademaker, Annemieke "hup, schrijven jij" Rooker, Stephan Segers, Arthur de Snaijer, René Stolvoort, Philippe Tjon-a-Hen, Jaap Verhees, Patrick Verheij, Wouter Vermeulen, Chris Vieveen, Marcel de Vries, Frank Vink, Harry Vroom, Jos Warmer en Aagje Wijnja.

Voor hun enthousiasme en steun wil ik graag bedanken: Albert & Jeanne, André, Angela, Arjen & Liesbeth, Ankie & Jan, Christien, Coen, Deirdre, Dirk, Dré, Edith & Coen, Eric & Annelies, Femke & Joop, Gaspard, George & Hennie, Gerwin, Hans & Petra, Huub, Ina, Jan Willem & Marloes, Jeroen, Johan, Joop & Betsy, Jos, Katja, Lizza, Marc, Marja & Martin, Monique & Bert, Olav, Pascal, Patrick, Paul, Pim & Janine, Raymond, Rita, Rene & Nicole, Sabine, Urso, Werner, Yvonne, iedereen bij de volgende organisaties: Array Publications & Seminars, Capgemini, College van Partners van Ordina, EOS, Google, Hercules 27, IT Works, Microsoft, Ordina, Pearson Education, SDGN, Select Business Solutions, SIGS Datacom, Sparx Systems en alle andere mensen en organisaties waar(mee) ik de afgelopen zestien jaar heb mogen (samen)werken en die ik vergeten ben hier te noemen.

Dit boek is geschreven op locatie in Aarhus, Amsterdam, Barcelona, Brussel, Bunnik, De Meern, Den Bosch, Gent, Garderen, Maastricht, München, Nieuwegein, Nijkerk, Rheden, Rosmalen, Son, Utrecht en Zeist.

De opbouw van het boek

I'm not sure what four nines does, but the ace, I think, is pretty high.

George Clooney (Ocean's eleven)

Dit boek volgt het project dat de webapplicatie voor Dare2Date ontwikkelt. Talrijke facetten van het project passeren de revue. Het modelleren van requirements, het opzetten van testscenario's, de applicatiearchitectuur en het domein van de applicatie, de user interface, de functionaliteit en de database. Deze facetten komen uitgebreid aan de orde in de acht delen waar dit boek uit bestaat. Ik wens u veel leesplezier.

Deel I Introductie

De introductie schetst de basis voor het boek:

- *Dare2Date*. Een gefingeerd krantenartikel over Dare2Date. Het krantenartikel biedt de context van de case.
- *UML*. Er volgt een korte introductie in UML 2.0 en de verschillende modelleertechnieken die de modelleertaal biedt. Kenners van UML 2.0 slaan dit hoofdstuk over. Kenners van eerdere versies van UML raad ik aan dit hoofdstuk wel te lezen.
- *Mythen en misverstanden*. Rond UML heersen diverse mythen en misverstanden. Dit hoofdstuk beschrijft de uitdagingen die projecten tegemoet zien wanneer ze starten met UML.
- *Agile software development*. Veel technieken, tips en trucs in dit boek komen voort uit projecten die zijn uitgevoerd volgens de agile systeemontwikkelmethode Smart. Dit hoofdstuk laat zien waar, wanneer en hoe modelleertechnieken inzetbaar zijn in iteratieve, incrementele en vooral interactieve systeemontwikkelprojecten.

Deel II Requirements

In ieder systeemontwikkelproject komt het beschrijven van de requirements al snel aan bod. Te vaak resulteert dit in te dikke documenten. De modelleertechnieken die in dit deel van het boek zijn beschreven, gelden als katalysator in de communicatie tussen opdrachtgever, gebruikers en het team:

- *Starten met bedrijfsprocessen.* Start uw project met het modelleren van de te ondersteunen bedrijfsprocessen. De resulterende procesdiagrammen geven een project de broodnodige overview.
- *Pragmatische use cases.* Vanuit de procesdiagrammen worden de use case diagrammen en de use cases voor het project opgesteld.
- *Use cases beschrijven.* Het derde en laatste hoofdstuk in dit deel gaat in op het beschrijven van de use cases. Pragmatisch worden zo het doel, de pre- en de postcondities en een stappenplan toegevoegd.

Deel III Modelleren en testen

Testen is een ondergeschoven kindje. Testers worden meestal pas aan het einde van een project geraadpleegd. Ik besteed met opzet vroeg in het boek aandacht aan testen. Het opstellen van de testscenario's en testgevallen wordt daarmee onderdeel van het ontwerp. Zo kunnen testers feedback geven nog voor er een letter code is geschreven:

- *Activity diagrammen.* Bij iedere use case wordt een activity diagram opgesteld. Dit gebeurt op basis van de beschrijving van de use case.
- *Testscenario's en testgevallen.* De activity diagrammen zijn een prettig uitgangspunt voor het identificeren van testscenario's en testgevallen voor de use cases. Dit hoofdstuk beschrijft een bijzonder nuttige techniek om testscenario's op te sporen.

Deel IV Architectuur

Ontwikkelt u onder architectuur? Veel organisaties beamen dit blindelings. Altijd leuk om te vragen wat hiermee wordt bedoeld. In de praktijk kom ik sterk verschillende visies tegen. De grootste gemene deler van deze visies kent in elk geval twee aspecten: domeinarchitectuur en applicatiearchitectuur. Deze maken het ontwerp realiseerbaar:

- *De kloof tussen ontwerp en code.* Er ligt een gapende kloof tussen ontwerp en code. Ontwerpers en ontwikkelaars spreken niet dezelfde taal. Wordt er dan wel gebouwd wat er bedacht wordt? Dit hoofdstuk toont waarom het zo moelijk is ontwerp in code te vertalen.
- *Een referentiearchitectuur.* De kloof is te overbruggen. Ik introduceer hiervoor een eenvoudige referentiearchitectuur. Dit biedt een pragmatische horizon voor het samenbrengen van ontwerpers en ontwikkelaars.
- *Het domein van de applicatie.* Al vroeg in een project wordt het domein van de applicatie onderzocht. Dit hoofdstuk helpt om de hoofdlijnen van dit domein vast te stellen. Het domein groeit uit tot de structuur van de applicatie.

Deel V User interface

Alhoewel er een overvloed is aan literatuur waarin de grafische aspecten en de ergonomie van de user interface wordt behandeld, is er nauwelijks literatuur over het modelleren van de user interface, de functionaliteit van de schermen en de navigatie. Gebruikers schetsen echter graag schermen, vooral tijdens het modelleren van use cases. Uit dergelijke schetsen is een modelleertechniek ontstaan die ik het user interface diagram noem. Het modelleren van dit user interface diagram staat centraal in dit deel van het boek:

- *De user interface modelleren.* Een discussie over het modelleren van user interfaces.
- *Het user interface diagram.* Het user interface diagram is een pragmatische modelleertechniek die nauw aansluit op de use cases.
- *Forms, factories en business classes.* Veel van de functionaliteit van een applicatie is gerelateerd aan de user interface. Ik ga hier in op attributen van forms en klassen, het tonen en wijzigen van gegevens en het identificeren van validaties.

Deel VI Interactie

Alhoewel heel veel teams het liefst omrijden, voert de weg van idee naar applicatie onontkoombaar langs het ontwerpen van functionaliteit. Dit ontwerp kent ruwweg twee aspecten. De structuur van de applicatie is het eerste aspect. De interactie tussen de klassen in een applicatie het tweede:

- *Interactie modelleren.* De verantwoordelijkheden van een use case worden gerealiseerd door samenwerkende, interacterende klassen. Modelleer deze samenwerking in sequence diagrammen.
- *Interactiepatronen.* Herhalende structuren in sequence diagrammen zijn te modelleren in interactiepatronen. Deze patronen vergemakkelijken het modelleren van interactie.

Deel VII Het modelleren van structuur

Het domein van de applicatie vormt het uitgangspunt voor de statische structuur van de applicatie. Deze krijgt vorm in de klassen van de applicatie, de relaties tussen deze klassen en de packages waarin de klassen plaats krijgen:

- *Klassen.* Ruim aandacht voor de klassen uit het domein van de applicatie. Aan de orde komt hoe stereotypen, attributen en methoden van klassen pragmatisch worden gemodelleerd in een klassendiagram.

- *Relaties tussen klassen.* Over weinig aspecten van UML bestaan zoveel misverstanden als over relaties tussen klassen. Dit hoofdstuk beschrijft de relaties tussen klassen en hoe misverstanden zijn te vermijden.
- *Organiseren in packages.* In grotere projecten is het noodzakelijk klassen te groeperen in packages. De afhankelijkheden tussen deze packages krijgen hier nadrukkelijk attentie.

Deel VIII Gegevens

De verhouding tussen klassen en tabellen biedt een interessante uitdaging aan objectgeoriënteerde projecten. UML ontbeert voor het modelleren van gegevens een modelleertechniek. Ik sta in dit deel van het boek stil bij het modelleren van gegevens in UML en bij de vertaling van klassen naar tabellen:

- *Gegevens modelleren in UML.* Alhoewel UML geen gegevendiagram kent, is in een klassendiagram vrij aardig zo'n gegevensdiagram te modelleren. Onder voorwaarden.
- *Van business classes naar tabellen.* Het vertalen van de klasse in de applicatie naar de tabellen in de database is geen sinecure. Dit hoofdstuk beschrijft het vertalen van de klassen en hun associacties naar tabellen en hun relaties in het gegevensdiagram.

Tot slot

Het boek sluit af met enkele overpeinzingen omtrent pragmatisch modelleren.

Deel I Introductie

Dare2Date

- Wat is Dare2Date?
- Welke requirements kent Dare2Date?

UML

- Hoe is UML ontstaan?
- Wat is er nieuw in UML 2.0?
- Welke constructies en modelleertechnieken kent UML?
- Wat is een frame? Wat is een package?

Mythen en misverstanden

- Hoe worden de modelleertechnieken van UML gebruikt?
- Welke modelleertechnieken ontbreken in UML?
- Is UML eenvoudig te leren?
- Is UML een systeemontwikkelmethode?
- Hoe kan een organisatie goed starten met UML?

Agile software development

- Waarom gaat traditionele systeemontwikkeling mis?
- Wat is agile software development?
- Wat kenmerkt agile software development?
- Wat is Smart? Welke fasen kent Smart?
- Welke modelleertechnieken zijn te gebruiken in agile software development?

Dare2Date

There's only one way to have a happy marriage and as soon as I learn what it is I'll get married again.

Clint Eastwood

Onze samenleving kent een groeiend aantal kapitaalkrachtige hoogopgeleide alleenstaande dertigers. Het vinden van een geschikte partner is voor deze singles niet eenvoudig door hun drukbezette agenda en de beperkte gelegenheden om andere singles te leren kennen. Organisaties spelen goed in op deze nieuwe subcultuur, getuige het grote aanbod van tijdschriften, producten en zelfs televisieprogramma's die zich richten op deze snelgroeiende doelgroep. De grootste concentratie van singles is te vinden in de steeds individualistischer wordende grote steden. Het meest zichtbaar is dit door het ontstaan van tal van nieuwe uitgaangsgelegenheden. Er worden zelfs diners en feesten georganiseerd die alleen toegankelijk zijn voor singles.
De single dertiger gebruikt dit aanbod van tijdschriften, producten en uitgaansgelegenheden hoofdzakelijk met één doel: het vinden van een geschikte partner. Dit is echter niet eenvoudig. Door hun ruime levenservaring zijn single dertigers selectief en kritisch. In toenemende mate wordt het internet gebruikt om de prins of prinses op het witte paard te vinden. Singles maken gebruik van chat-programma's en online messaging. De vrijblijvendheid van dergelijke applicaties garandeert echter geen serieuze gesprekspartners. Dit schrikt af.
Dare2Date is een commerciële online dating service, waarmee singles in betrekkelijke anonimiteit contact kunnen leggen met andere alleenstaanden. Een gesprek met initiatiefnemers Frank Schilling en Mark Pond op het zonovergoten terras van een van de grand cafés van Utrecht.

Gat in de markt

"Het is een gat in de markt", legt Frank Schilling uit. "Met onze dating service vinden dertigers heel makkelijk een passende partner. En dat binnen de muren van hun eigen vertrouwde appartement", gaat hij enthousiast verder. "Je hoeft niet naar een drukke kroeg of zo'n degenererend singles-feest. Veel single dertigers zitten hier niet op te wachten. Want wie gaat er mee? De meeste van hun vrienden zijn getrouwd, hebben kinderen en zitten op zaterdagavond liever met een biertje en chips op de bank. De single dertiger heeft daarnaast vaak

een drukke baan en dus maar weinig tijd om een potentiële partner te ontmoeten." Mark Pond knipoogt. "Het is bovendien een kapitaalkrachtige doelgroep."

Geen one-night stand

"Het idee is simpel", legt Pond uit. "Je abonneert je op Dare2Date. Je kunt dit gewoon doen via de Dare2Date website. Je vult je persoonlijke gegevens en de gegevens van je creditcard in. Als deze gegevens correct zijn, krijg je direct je gebruikersnaam en je wachtwoord opgestuurd via e-mail."
De dating service kost de welgestelde single dertiger vervolgens twintig euro per maand. Dat Dare2Date niet gratis is, biedt volgens beide ondernemers de garantie dat alleen mensen die serieus op zoek zijn naar een partner een abonnement nemen. Onze abonnees steken veel energie in het zoeken naar een partner. Het gaat hen niet om een gemiddelde one-night stand. "Een abonnement van twintig euro per maand werpt een wenselijk drempeltje op", zegt Schilling. Zijn zakenpartner Mark Pond gaat verder. "Vervolgens log je in op de portal en je vult je profiel in, inclusief allerlei wensen en voor- en afkeuren. Wat je hobby's zijn, wat je leeftijd is, in welke regio je woont, en natuurlijk ook wat je in een nieuwe partner zoekt. En niet te vergeten je geslacht. We hebben al een overzicht van mogelijke trefwoorden klaargezet. Als voorbeeld. Daarin kun je aankruisen welke er bij jou passen en welke absoluut niet. Denk hierbij bijvoorbeeld aan film, muziek, sporten en natuurlijk ook seksuele voorkeuren. Als je wilt kun je ook nog je foto aanleveren, maar dit is niet verplicht."

De eerste date

Pond stopt zijn verhaal even en neemt een slok van zijn koffie-verkeerd. "De abonnee heeft nu een heleboel opties. Zo kun je de profielen van andere singles bekijken. Natuurlijk kun je hierbij zoeken op voorkeuren, zoals de muziek waar je van houdt, karaktereigenschappen en of iemand bij je in de buurt woont. Je kunt zelfs op opleidingsniveau zoeken. Singles ontmoeten nu eenmaal graag iemand van eenzelfde of hoger opleidingsniveau. Wist je trouwens dat ruim tweederde van onze abonnees vrouw is?"
Schilling neemt het stokje over. "Als je een profiel vindt dat interessant lijkt, dan stuur je een berichtje via Dare2Date. Uiteraard anoniem. Ter bescherming van de privacy van beide singles. Alle abonnees hebben een soort postbus op de website waarin de berichten verschijnen. Ieder bericht bevat een link naar het profiel van de abonnee die het bericht verstuurt." Schilling denkt even na. "Vindt de ontvanger ook de afzender interessant dan stuurt hij of zij een berichtje terug. Misschien wel voor het maken van een eerste date. En dan begint het balletje te rollen."
"Een andere mogelijkheid van Dare2Date", adverteert Mark Pond, "is om je als abonnee up-to-date te laten houden over interessante nieuwe of gewijzigde pro-

fielen. Hiervoor moet je wel VIP zijn. Dit is uiteraard een iets duurder abonnement. Dan ontvang je wel dagelijks een bericht met links naar deze profielen. Uiteraard kun je ook hier je voorkeuren opgeven." Pond glimlacht tevreden. "Een grappige mogelijkheid is dat je kunt opgeven of trefwoorden wel of niet bij je passen. Neem bijvoorbeeld voetbal. Sommigen kunnen niet zonder, anderen hebben er een bloedhekel aan."

Vanuit je luie stoel

Nog even terug naar de doelgroep. "We lokken bezoekers doordat je ook zonder abonnement al kunt zoeken in de profielen op de website", zegt Mark Pond. "Daarbij zie je dan niet de foto's die zijn geplaatst door abonnees en je kunt geen berichten achterlaten. Hiervoor moet je natuurlijk eerst abonnee worden. Je kunt als bezoeker direct doorklikken, lid worden en alsnog een bericht sturen naar een interessant profiel. Abonnee worden is een kwestie van enkele handelingen uitvoeren, niet van drie weken wachten."
"Dit initiatief vult een regelrecht gat in de markt", zegt Frank Schilling. "En meer dan dat. We doen een hoop mensen hiermee een groot plezier. Heel veel singles van nu zien de gang naar een café of club niet zitten. Want wie garandeert dat de mensen die je daar ontmoet dezelfde interesses en voorkeuren hebben? Op Dare2Date filter je die er direct uit. En dat vanuit je luie stoel met een goed glas wijn en een zak chips."

UML

Any intelligent fool can make things bigger, more complex and more violent. It takes a touch of genius and a lot of courage to move in the opposite direction.

Albert Einstein

De Unified Modeling Language (UML) is een algemene visuele modelleertaal die is bedoeld om de verschillende producten in een systeemontwikkelproject te specificeren, te ontwikkelen en te documenteren. In de praktijk richt het gebruik van de modelleertaal zich vooral op analyse en ontwerp. In beginsel is UML een combinatie van diverse modelleertechnieken die afkomstig zijn uit verschillende objectgeoriënteerde systeemontwikkelmethoden, inmiddels aangevuld met heden ten dage geldende best practices. UML is bedoeld om in projecten gebruikt te worden, onafhankelijk van systeemontwikkelmethoden, faseringen of domeinen. Ik heb de modelleertechnieken van UML gebruikt zien worden door financiële organisaties, in de industrie, door overheden en voor het ontwerpen van embedded software.
De specificatie van UML kent technieken voor het vastleggen van zowel het gedrag als de statische structuur van applicaties. In UML worden applicaties ontworpen als een verzameling modelelementen. De statische structuur van de applicatie beschrijft deze elementen. Het gedrag van de applicatie wordt vastgelegd in de samenwerking van deze modelelementen, meestal uitgezet in de tijd. Lekker abstract toch?

De prehistorie

Alhoewel objectoriëntatie met de introductie van Java, UML en recenter Microsoft's .NET weer in het middelpunt van de belangstelling staat, werd al in 1967 de eerste objectgeoriënteerde programeertaal ontwikkeld, Simula-67. En hoewel deze programmeertaal nooit breed is toegepast, vonden de concepten van Simula-67 gretig aftrek toen aan het begin van de jaren tachtig een nieuwe generatie objectgeoriënteerde programmeertalen ontstond, met name Smalltalk en C++. Pas met de introductie van deze programmeertalen, die al snel algemeen beschikbaar waren, kwam de belangstelling voor het paradigma goed op gang.
Al spoedig realiseerde men dat de bestaande, traditionele modelleertechnieken niet toereikend waren voor het ontwikkelen van applicaties volgens het nieuwe paradigma. Tijdens mijn studie ondervond ik ook de uitdagingen van het

modelleren van klassen en objecten in program structure diagrams en data flow diagrams. Eind jaren tachtig ontstond er een hele generatie nieuwe methoden, zoals die van Shlaer/Mellor, Coad/Yourdon, Booch en OMT van Rumbaugh. Ieder van deze methoden kende zijn eigen concepten, notaties en processen. Een ietwat vreemde eend in de bijt was de methode Objectory van Ivar Jacobson, waarin een vernieuwend systeemontwikkelproces op basis van use cases centraal stond.
Ondanks het feit dat ieder van deze methoden hetzelfde vakgebied bestreek, ontstond er een ware explosie aan concepten en notaties, vaak met een grote overlap. Na een reeks Hoekse en Kabeljauwse twisten leidde deze explosie tot het inzicht dat een algemene collectie concepten en notaties geen overbodige luxe was, wilde het vakgebied verder rijpen. Een eerste aanzet tot het unificeren van de verschillende methoden werd gegeven toen James Rumbaugh zich in 1994 aansloot bij Rational, en aldaar met Grady Booch hun methoden combineerde (in de Unified Method 0.8). Toen ook Ivar Jacobson zich aansloot bij Rational, werd de basis gelegd voor de Unified Modeling Language.

De geboorte van UML

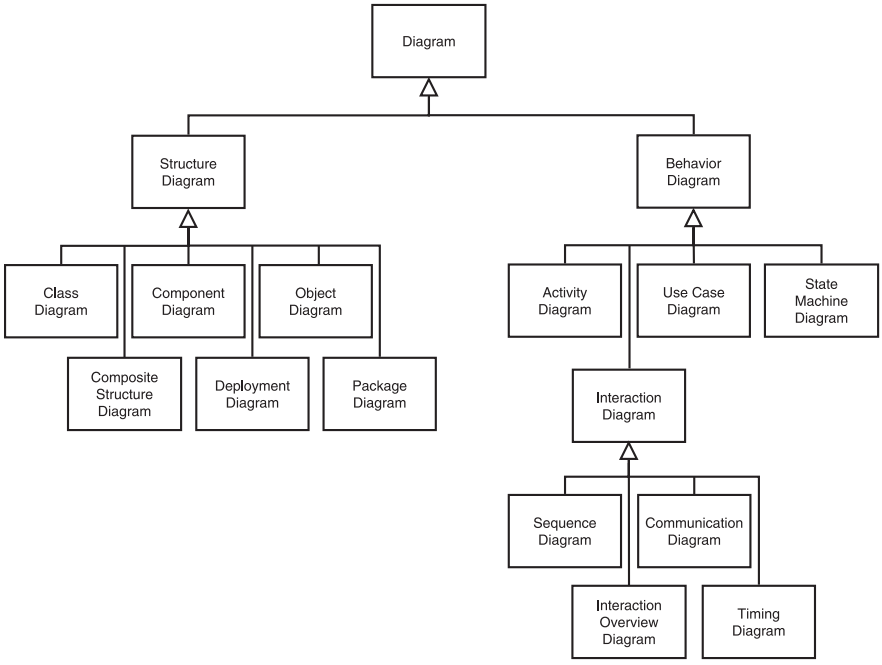
De Object Management Group (OMG) is een internationale organisatie die standaarden beheert, ontwikkelt en promoot op het gebied van objectoriëntatie, waaronder CORBA, en UML. De OMG streeft met deze standaarden een raamwerk na voor de ontwikkeling van objectgeoriënteerde applicaties. De Object Management Group richt zich zo op (soms wat te) ambitieuze doelen als herbruikbaarheid, overdraagbaarheid en de interoperabiliteit van software. De organisatie telt enkele honderden leden, waaronder vooral leveranciers van ontwerp- en ontwikkelgereedschappen en eindgebruikers van de standaarden.
In 1996 vaardigt de OMG een request for proposal uit voor een standaard voor objectgeoriënteerde modellering. Uiteraard leverden de methodologisten van Rational één van de voorstellen, naast de voorstellen van andere leden. Dit voorstel was door Rational al gedoopt als UML 1.0. In september 1997 werd bij de OMG een eindvoorstel voor een algemene objectgeoriënteerde modelleertaal ingediend. Dit voorstel voor UML werd in november 1997 unaniem goedgekeurd door de leden van de OMG. Alhoewel het definitieve voorstel door Rational inmiddels UML 1.1 werd genoemd, koos de OMG toch voor de naam UML 1.0.
De nieuwe modelleertaal is door de OMG bedoeld als een general all-purpose modelleertaal voor ontwerpers en ontwikkelaars in systeemontwikkelprojecten. De OMG stelt nadrukkelijk dat de modelleertaal onafhankelijk is van leveranciers en gebruikt kan worden in om het even welk systeemontwikkelproces. De modelleertaal floreert vooral in iteratieve systeemontwikkelprocessen, waarbij de verschillende modelleertechnieken incrementeel leiden tot visuele representaties van moderne applicaties.

UML 2.0

Sinds het verschijnen van UML werkten de leden van de OMG voortdurend aan de verdere ontwikkeling van de modelleertaal. Dit heeft in de loop der jaren geleid tot een aantal nieuwe versies. In 1998 verscheen UML 1.2, in 1999 UML 1.3. Vervolgens zag UML 1.4 het licht in 2001 en UML 1.5 in 2002. Ieder van deze nieuwe versies vertoonde vooral wijzigingen in de diepte, zoals het opnemen van returns in sequence diagrammen of de package visibility uit Java. UML 1.3 bevat vooral wijzigingen in het use case diagram, UML 1.4 kent profielen, en UML 1.5 voegt action semantics toe aan het toch al brede palet van de modelleertaal.
Al in 2000 startte de OMG met het opstellen van vier requests for proposals voor een ruimere revisie van haar modelleertaal. Voor ieder van deze requests for proposal werd een aantal voorstellen ingediend door de leden van de OMG. Om een heel lang verhaal kort te maken, na lange discussies stabiliseren deze voorstellen uiteindelijk in 2003. Inmiddels ligt er voor ieder van de requests for proposal één goedgekeurd voorstel. Tezamen vormen deze goedgekeurde voorstellen de specificaties van UML 2.0.
UML 2.0 betekent een grote verandering ten opzichte van de vorige versies van de modelleertaal. Veel van deze veranderingen raken vooral het metamodel van UML en vallen buiten het bereik van dit boek. Opvallend aan UML 2.0 is de toevoeging van een aantal modelleertechnieken. Het object diagram en het package diagram konden ook in de vorige versies van UML worden gemodelleerd, zonder als zodanig benoemd te zijn. Daarnaast kent UML 2.0 interaction overview diagrammen, composite structure diagrammen en timing diagrammen. Van de bestaande diagrammen onderging het sequence diagram de grootste wijzigingen door de toevoeging van interaction en combined fragments. Het activity diagram tenslotte, is inmiddels op een andere leest geschoeid, waardoor het nog meer mogelijkheden kent.

De modelleertechnieken van UML

UML 2.0 kent dertien modelleertechnieken. Deze zijn te verdelen in drie categoriën. *Structure diagrams* belichten de structuur van een applicatie. *Behavior diagrams* tonen het dynamische gedrag van een applicatie. Binnen deze behavior diagrams belichten de *interaction diagrams* de samenwerking tussen modelelementen die het gedrag waarmaken. Ieder van de modelleertechnieken belicht de applicatie vanuit een ander gezichtspunt. De taxonomie van modelleertechnieken in UML is weergegeven in afbeelding 1.



afbeelding 1 – Een taxonomie van UML modelleertechnieken

UML kent onderstaande structure diagrams:

- *Klassendiagram.* De structuur van de applicatie wordt gemodelleerd als klassen in een klassendiagram.
- *Composite structure.* Een composite structure beschrijft de decompositie van een klasse.
- *Component diagram.* Het component diagram modelleert afhankelijkheden tussen de componenten in een applicatie.
- *Deployment diagram.* Deze modelleertechniek beschrijft de fysieke configuratie van een applicatie.
- *Object diagram.* Het object diagram toont de relaties tussen objecten op een gegeven moment in de tijd.
- *Package diagram.* Packages structureren de klassen in de applicatie in grotere gehelen. Het package diagram modelleert afhankelijkheden tussen packages.

Daarnaast kent UML de volgende behavior diagrams:

- *Activity diagram.* Het activity diagram modelleert processen.
- *Use case diagram.* Use cases worden gebruikt voor het communiceren over en vastleggen van de requirements van de applicatie.

- *State machine diagram.* Het state machine diagram beschrijft de levenscyclus van een object (een instantie van een klasse).

UML omvat de volgende interaction diagrams:

- *Sequence diagram en communication diagram.* Er is een tweetal modelleertechnieken dat de samenwerking tussen klassen weergeeft. Dit zijn het sequence diagram en het communication diagram. Het communication diagram heette in eerdere versies van UML nog collaboration diagram.
- *Interaction overview.* Een interaction overview is een instantie van een activity diagram, maar modelleert interactie.
- *Timing diagram.* De interactie tussen objecten wordt in de tijd gemodelleerd in een timing diagram.

Naast deze modelleertechnieken kent UML constructies voor de organisatie en groepering van modelelementen. De meest gebruikte is de *package*. Een package bundelt een verzameling modelelementen onder een noemer. Daarnaast kent UML 2.0 het *frame*. Een frame wordt gebruikt om een (deel van een) diagram af te kaderen. Tenslotte kent de modelleertaal constructies die de modelleertaal uitbreidbaar maken. UML 2.0 onderscheidt allerlei constraints en keywords, waaronder stereotypen. Leveranciers van ontwerp- en ontwikkelgereedschappen maken ongebreideld gebruik van deze constructies om bijvoorbeeld het genereren van code mogelijk te maken. Een veelal vergeten onderdeel van UML is de Object Constraint Language (OCL). Met OCL zijn pre- en postcondities, bedrijfsregels en zelfs queries te beschrijven.

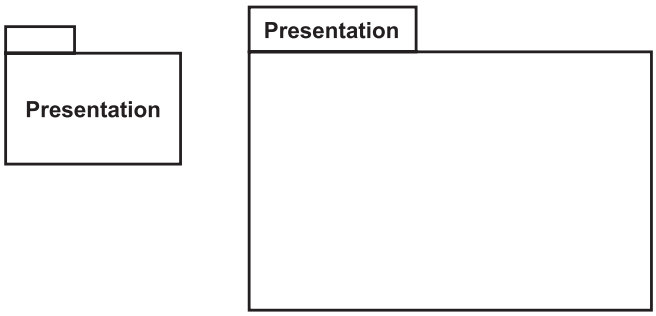
Keywords en stereotypen

UML definieert talrijke begrippen die gebruikt worden in de diverse modelleertechnieken. Deze begrippen, zoals **package**, **«interface»**, **sd**, **ref**, **{abstract}** of **«include»**, worden in de modelleertaal *keywords* genoemd. In modelleeromgevingen worden keywords nogal eens vervangen door grafische symbolen of iconen.
De meeste keywords in UML zijn stereotypen. Stereotypen worden gebruikt voor het classificeren van gelijksoortige modelelementen. Bijvoorbeeld voor typen relaties tussen use cases (zoals **«include»** en **«extend»**), voor type klassen in een architectuur (zoals **«business class»** en **«task»**), voor rollen in interactiepatronen (zoals **«data class»**), of voor typen depedencies tussen modelelementen (zoals **«call»** en **«derive»**). In sommige modelleeromgevingen worden stereotypen zelfs gebruikt om aan te geven hoe code wordt gegenereerd. UML gebruikt stereotypen bovendien voor het definïeren van profielen, voor bijvoorbeeld EJB, .NET en CORBA.
Modelelementen hebben maximaal één stereotype. Het is niet verplicht om een modelelement te voorzien van een stereotype. Een stereotype wordt weergege-

ven tussen *guillemets* (« en »). Indien u typografisch minder goed bent uitgerust, kunt u wellicht << en >> gebruiken.

Packages

Een package is een modelelement dat in UML wordt gebruikt voor het groeperen van modelelementen. Zo kan een package bijvoorbeeld use cases, klassen of zelfs andere packages bevatten. Iedere package heeft een naam. Een package wordt weergegeven als een een rechthoek, met in de linkerbovenhoek een tweede kleinere rechthoek. Deze laatste heet de *tab*. Een package kent twee representaties. In de eerste bevat de package de naam, in de tweede de tab. Beide representaties zijn weergegeven in afbeelding 2. De eerste van deze representaties komt het meest voor in package diagrammen. De tweede representatie toont in het algemeen de inhoud van de package in de rechthoek.

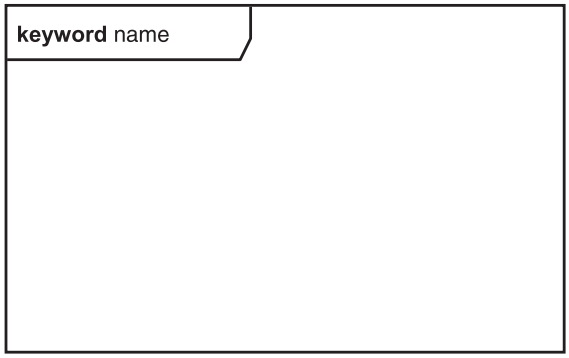


afbeelding 2 – Packages

In de verreweg de meeste gevallen worden packages gebruikt voor het groeperen van bij elkaar horende klassen. Iedere klasse behoort in UML tot een package. Een package geldt als eigenaar van de klassen (en andere modelelementen) die de package bevat. Een package representeert in UML een *namespace* (niet te verwarren met een namespace in .NET). Dit betekent dat de klassen die eigendom zijn van een package een unieke naam binnen deze package moeten hebben. Het is wel mogelijk om verschillende klassen met eenzelfde naam in verschillende packages te hebben.

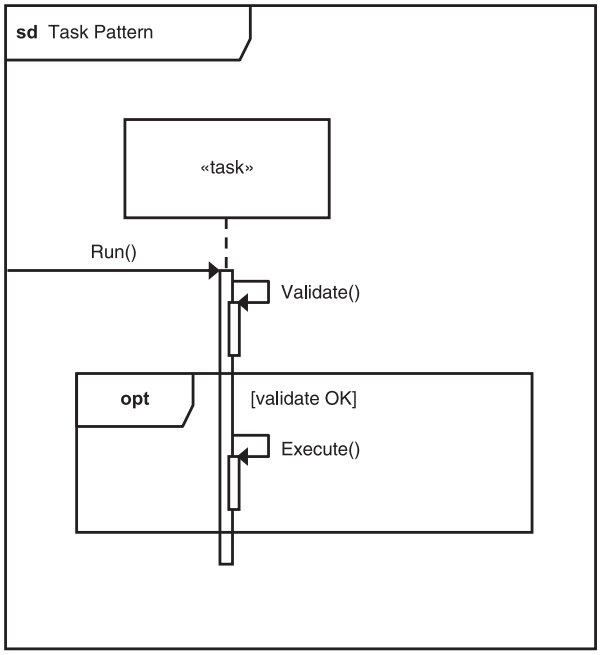
Frames

UML 2.0 introduceert een verder betekenisloos modelelement dat helpt om vooral interactiediagrammen beter te structureren. Dit modelelement heet een *frame*. Een frame is weergegeven als een rechthoek, met links bovenin een klein pentagon. In afbeelding 3 is zo'n frame afgebeeld. Het kleine pentagon is de *heading* van het frame, en bevat een keyword, de naam van het frame en eventueel parameters.



afbeelding 3 – Frame

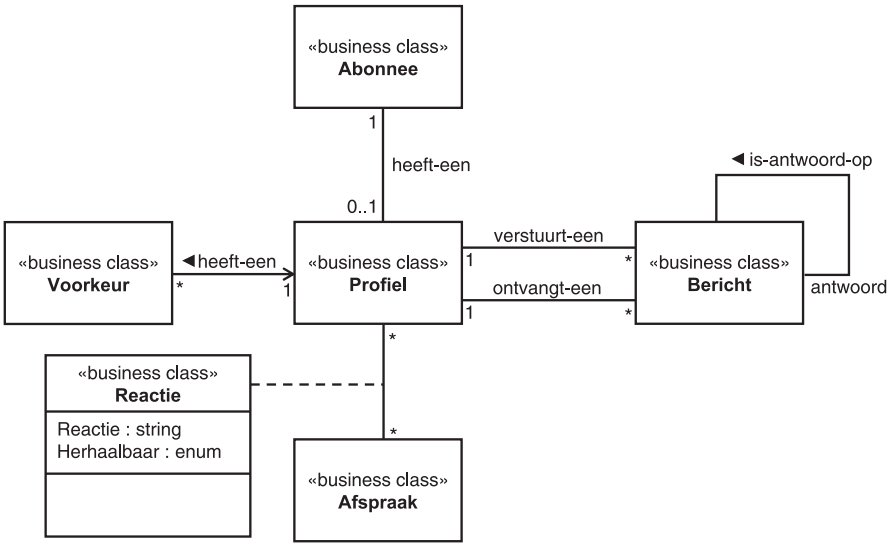
Ieder diagram in UML is omvat door een frame. Hierbij representeert **keyword** het type van het diagram, en **name** de naam van het diagram. In de meeste gevallen wordt het echter weggelaten. Slechts in diagrammen waar er communicatie is tussen de modelelementen in het frame en hun omgeving is het zinvol het frame te modelleren, zoals het initiëren van de interactie in een sequence diagram.
De belangrijkste toepassing van frames ligt in het sequence diagram. In vorige versies van UML was deze modelleertechniek onvoldoende uitgerust om conditionele handelingen en herhalingen te modelleren. In UML 2.0 wordt het frame gebruikt om fragmenten in interactie te modelleren. De heading van het frame bevat nu de operator van het fragment. Een sequence diagram met frame is gemodelleerd in afbeelding 4.
Dit sequence diagram modelleert een interactiepatroon. Het is weergegeven in een frame omdat het interacteert met de buitenwereld. Dit is gevisualiseerd doordat de message **Run()** start op het frame. Het frame met keyword **opt** is een gecombineerd fragment en geeft aan dat de interactie in het fragment alleen wordt uitgevoerd als aan de conditie **validate OK** is voldaan.
Het is ook mogelijk in een diagram te refereren aan een ander diagram. Dit kan door het tweede diagram op te nemen als frame in het eerste. Als keyword krijgt dit frame nu **ref**, gevolgd door de naam van het opgenomen diagram. In verschillende modelleertechnieken, zoals het sequence diagram, is het nu mogelijk messages te versturen naar dit frame. De interactie loopt alleen dan door als het tweede diagram ook gestart wordt vanaf zijn eigen frame, zoals het diagram in afbeelding 4.



afbeelding 4 – Sequence diagram met frames

Klassendiagram

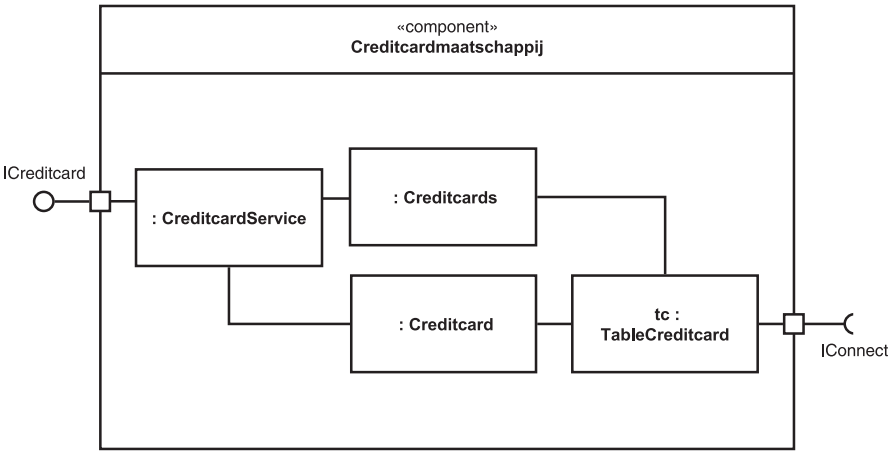
Van oudsher is het klassendiagram de modelleertechniek om de statische structuur van de applicatie vast te leggen. Het is de modelleertechniek die het meest verbonden is met het objectgeoriënteerde paradigma. Lang voor het ontstaan van UML kende het klassendiagram al een groot aantal notatiewijzen. De statische structuur van een applicatie bestaat hoofdzakelijk uit klassen en interfaces. Het klassendiagram modelleert ook de relaties tussen deze modelelementen.
Een klasse representeert de gezamenlijke kenmerken en het gemeenschappelijke gedrag van een verzameling objecten. Een klasse is afgebeeld als een rechthoek. In afbeelding 5 is een klassendiagram voor Dare2Date weergegeven.
Een klasse is meestal verdeeld in drie compartimenten, zoals hier **Reactie**. Het bovenste compartiment bevat de naam van de klasse, en eventueel een stereotype. De kenmerken, of attributen, van een klasse worden in het middelste compartiment geplaatst. Van een attribuut kan in het klassendiagram een aantal eigenschappen zijn gemodelleerd, zoals het type van het attribuut. Het derde compartiment van een klasse beschrijft het gedrag van de klasse in operaties. Ook van deze operaties zijn verschillende eigenschappen vast te leggen. Denk aan de in- en uitvoerparameters en de retourwaarde van de operatie. Tenslotte modelleert het klassendiagram de relaties tussen de verschillende klassen en interfaces. UML is ruim voorzien in dergelijke relaties die bijvoorbeeld afhankelijkheden en aggregaties representeren.



afbeelding 5 – Klassendiagram

Composite structure diagram

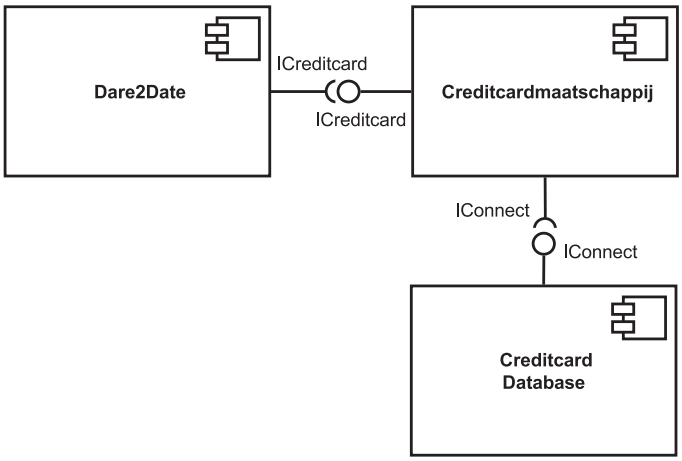
Het composite structure diagram maakt het mogelijk de interne structuur van een klasse of een component te modelleren. Deze klasse of component wordt wel *composite structure* genoemd. Het diagram toont de samenwerkende *parts* (onderdelen) waaruit de composite structure is opgebouwd. Een part is een collectie van nul, een of meer instanties van een klasse in composite structure. Het composite structure diagram toont ook de relaties die tussen deze parts bestaan. Deze worden gemodelleerd als *connectors*.
De composite structure communiceert met de buitenwereld via interfaces. Deze worden in het composite structure diagram zichtbaar gemaakt en eventueel gegroepeerd in de *ports* van de klasse of component. Een port kan zowel de benodigde als de geleverde interfaces representeren.
In afbeelding 6 is de bedrijfscomponent **Creditcardmaatschappij** gemodelleerd als een composite structure. Deze component heeft een viertal parts, te weten **Creditcardservice**, **Creditcards**, **Creditcard** en **TableCreditcard**. De samenwerking tussen deze parts is gemodelleerd middels connectors.
De bedrijfscomponent in afbeelding 6 heeft twee ports. Deze zijn weergegeven als kleine ruiten op het frame van de component. **Creditcardmaatschappij** levert interface **ICreditcard**, maar heeft om te functioneren interface **IConnect** nodig. Deze laatste interface plugt **Creditcardmaatschappij** in de database in.



afbeelding 6 – Composite structure diagram

Component diagram

UML beschouwt een component als een vervangbaar deel van de applicatie. Zo'n component realiseert een of meer interfaces. De realisatie van deze interfaces is door de component ingekapseld. In een composite structure diagram is te modelleren hoe een component deze interfaces realiseert. Een component diagram modelleert de afhankelijkheden tussen verschillende componenten. Daarbij ligt de nadruk zowel op de interfaces die een component levert, als op de interfaces die een component nodig heeft om te functioneren.
Een component diagram laat zien hoe de interfaces van verschillende componenten op elkaar aansluiten. Een component is weergegeven als een klasse, maar met een icoon rechtsboven. Dit icoon is eventueel te vervangen door het stereotype **«component»**. Geleverde interfaces zijn weergegeven als een bol, benodigde interfaces als een halve cirkel. Deze notatie wordt *ball-and-socket* genoemd.
In afbeelding 7 is opnieuw de bedrijfscomponent **Creditcardmaatschappij** gemodelleerd. Dit component diagram toont de afhankelijkheid van deze component van de achterliggende database. Het diagram toont ook de applicatie als component en als afnemer van interface **ICreditcard** van **Creditcardmaatschappij**.
Het component diagram kent zijn beperkingen. Het modelleren van afhankelijkheden is beperkt en biedt weinig informatie. Om te identificeren welke klassen binnen een component afhankelijk zijn van welke interfaces van andere componenten is eventueel een composite structure diagram voor de componenten te modelleren.



afbeelding 7 – Component diagram

Deployment diagram

Het deployment diagram is een beperkte modelleertechniek waarmee de configuratie van de applicatie op run-time wordt aangegeven. Het deployment diagram van de meeste applicaties is dermate eenvoudig dat het nauwelijks de moeite loont om het op te stellen. Pas in complexe netwerktopografieën komt deze modelleertechniek goed tot zijn recht.
Een prachtig deployment diagram trof ik aan als voorbeeld in de systeemontwikkelmethode van een internationale bank. Het bestond uit twee kubussen met een lijn ertussen. In de ene kubus stond het woord “client” genoteerd, in de andere het woord “server”.

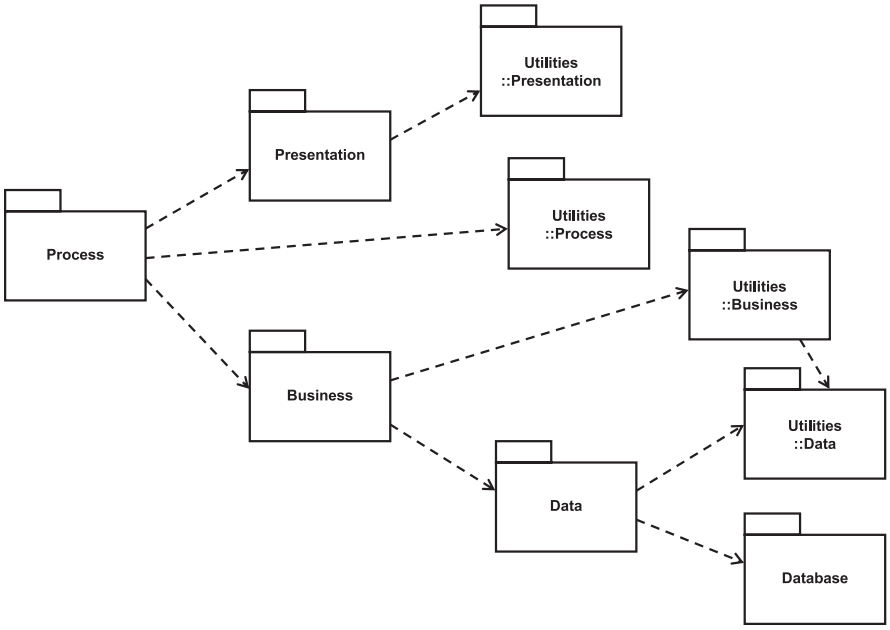
Object diagram

Het object diagram is een afgeleide van het klassendiagram. Met deze modelleertechniek worden relaties tussen objecten weergegeven op een bepaald moment in de tijd. Alle attributen krijgen de waarde die ze op dit specifieke moment hebben. Het object diagram kan het best worden gezien als een momentopname uit het geheugen van een draaiende applicatie. Alhoewel er vast zinvolle toepassingen voor zijn, heb ik nog nooit een object diagram gebruikt in projecten.

Package diagram

Het package diagram bestond in eerdere versies van UML officieel niet. Een package is een constructie in UML om gelijksoortige modelelementen te verzamelen, zoals een groep use cases of bij elkaar horende klassen. In de eerdere ver-

sies van UML was modelleren met packages dan ook mogelijk in een use case diagram of klassendiagram. In UML 2.0 is het package diagram uitgegroeid tot een zelfstandige modelleertechniek. Een package diagram kan afhankelijkheden tussen packages weergeven, maar ook dat een klasse in een package afhankelijk is van andere packages of van een klasse in een andere package.



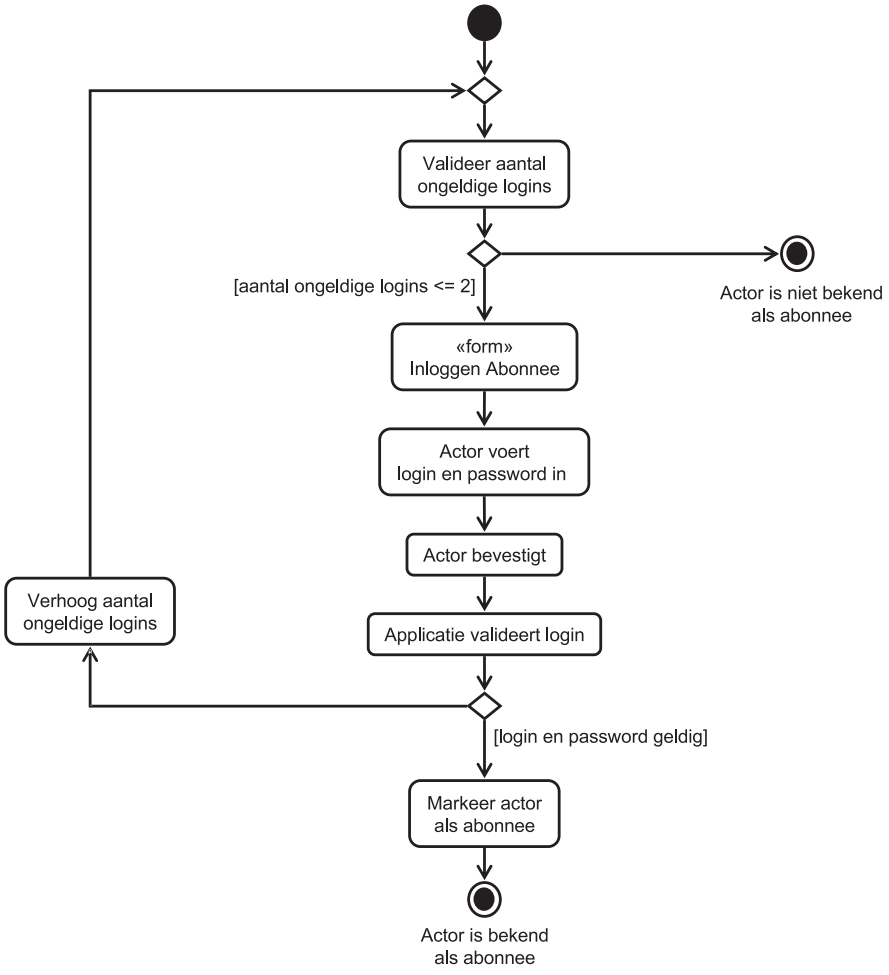
afbeelding 8 – Package diagram

Het package diagram is een populaire modelleertechniek om de structuur van een applicatie op een hoger niveau te modelleren dan met individuele klassen. Zo modelleert het package diagram in afbeelding 8 packages als lagen in de applicatie.

Activity diagram

Er zijn diverse modelleertechnieken om processen te modelleren, zoals process hierarchy diagrams, process thread diagrams, petrinetten, data flow diagrams en flow charts. Ook UML kent zo'n modelleertechniek. Dit is het activity diagram. Voor UML 2.0 is het activity diagram onder de motorkap ingrijpend gewijzigd. Aan de buitenkant van het diagram is hier niet veel van te merken.
Het activity diagram wordt op diverse manieren gebruikt. Zo wordt het gebruikt om bedrijfsprocessen te modelleren. Soms wordt het activity diagram gebruikt voor een nadere uitwerking van complexe functionaliteit waarbij beslismomenten een belangrijke rol spelen, zoals de aanvraag van een creditcard. Het activity diagram specificeert hier de functionaliteit. In de meeste

gevallen wordt de modelleertechniek gebruikt om use cases nader toe te lichten.
Een activity diagram, zoals in afbeelding 9, beschrijft activiteiten in diverse typen *activity nodes*, die worden weergegeven als rechthoeken met afgeronde hoeken. De volgorde waarin deze activity nodes worden uitgevoerd, is aangegeven middels transities die van de ene activity node naar de volgende wijzen. Deze volgorde wordt wel de *control flow* genoemd.



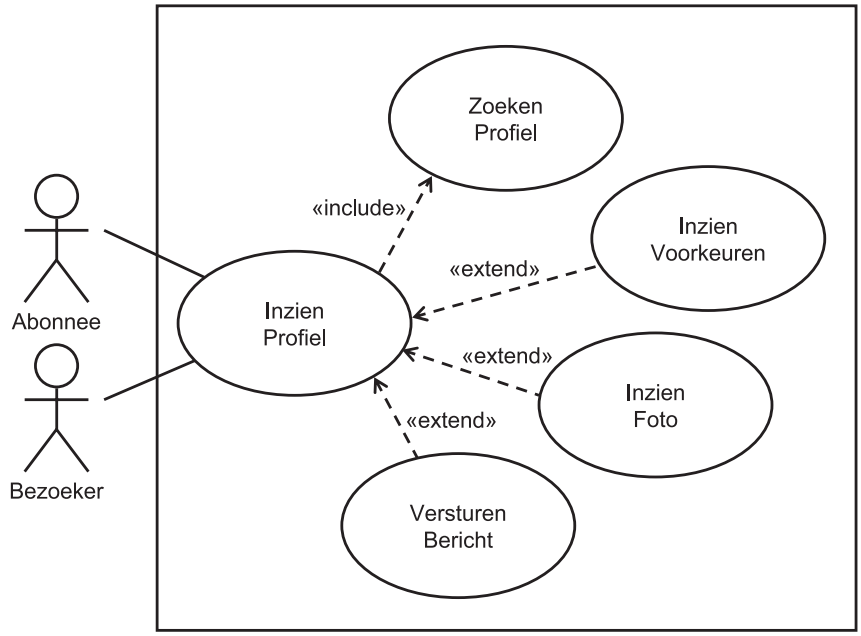
afbeelding 9 – Activity diagram

Het activity diagram in afbeelding 9 modelleert use case **Inloggen Abonnee** uit Dare2Date. Van sommige activity nodes, zoals **Valideer aantal ongeldige logins**, is de uitkomst bepalend voor het verdere verloop van het diagram. Dan wordt er een *decision node* (een beslismoment) toegevoegd, weergegeven als een ruit. Welke activity node na het beëindigen van **Valideer aantal ongeldige log-**

ins start, is afhankelijk van de te toetsen voorwaarden. Deze voorwaarden worden aangegeven middels een *guard*. In afbeelding 9 kan een abonnee nooit meer dan drie pogingen ondernemen om in te loggen. De guard **[aantal ongeldige logins <= 2]** voorkomt dat de abonnee een vierde poging onderneemt.

Use case diagram

Zonder de andere modelleertechnieken van UML onrecht aan te doen, gelden use cases, en ook het use case diagram als het schaap met vijf poten. Te pas en te onpas worden documenten als use cases bestempeld. Geen project meer zonder use cases. Het use case diagram in UML is een eenvoudige modelleertechniek, bedoeld voor het verzamelen van de requirements van een applicatie. Een use case diagram beschrijft activiteiten die de gebruikers van de applicatie ondernemen. Een gebruiker wordt een actor genoemd en een activiteit een use case. Het use case diagram beschrijft de relaties tussen actoren en use cases en tussen use cases onderling. In afbeelding 10 is zo'n use case diagram weergegeven.



afbeelding 10 – Use case diagram

De applicatie wordt hierbij als een black box beschouwd, gemodelleerd als een rechthoek. Actoren staat buiten deze applicatie en zijn gemodelleerd als draadpoppetjes. Iedere actor representeert de rol van een gebruiker van de applicatie. In afbeelding 10 bijvoorbeeld **Abonnee**. Actoren zijn echter niet per definitie menselijk, maar kunnen bijvoorbeeld ook andere applicaties weergeven waarmee wordt samengewerkt.

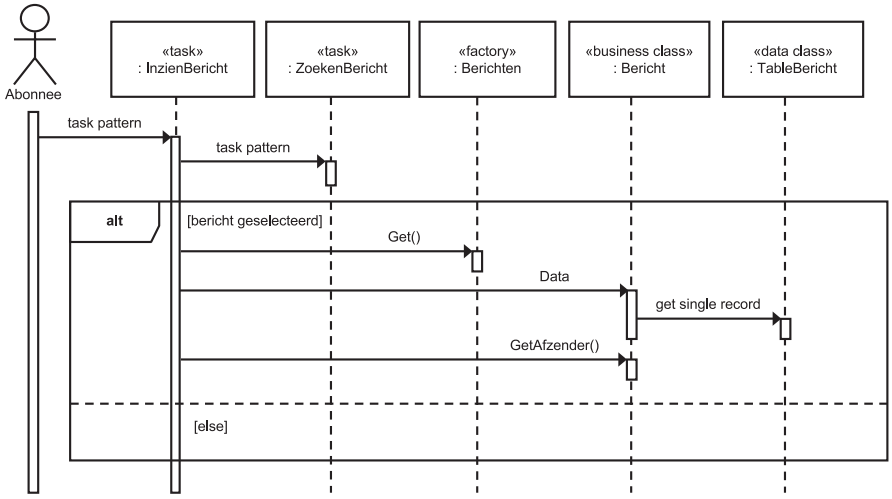
Ieder use case diagram kent een of meerdere use cases. Een use case beschrijft een manier van het gebruik van de applicatie door een actor. Een use case wordt weergegeven als een ovaal met daarin de naam van de use case. Actoren voeren use cases uit. **Abonnee** voert **Inzien Profiel** uit. Tenslotte modelleert het use case diagram de onderlinge relaties van de use cases. De relatie **«include»** modelleert hergebruik tussen use cases. Het is niet onwaarschijnlijk dat use case **Zoeken Profiel** in meer use case diagrammen voorkomt. De relatie **«extend»** beschrijft uitbreidingen op een use case. Deze uitbreidingen worden niet in alle gevallen uitgevoerd.
Het is gebruikelijk dat bij use cases een aantal additionele eigenschappen wordt beschreven. Welke eigenschappen dit zijn varieert nogal in de literatuur. De belangrijkste is het doel. Waarom voert een actor een use case eigenlijk uit? Vaak worden ook de precondities beschreven waaronder een use case mag worden uitgevoerd en de postcondities die de use case waarmaakt. Eventueel wordt een korte beschrijving opgenomen. In de regel hanteren projecten een sjabloon voor het beschrijven van deze eigenschappen.

State machine diagram

Een state machine diagram geeft de levenscyclus van een individuele klasse uit de applicatie weer. Deze modelleertechnieken heeft alleen zin als deze cyclus diverse statussen en statusovergangen kent. Een nieuwe hypotheek wordt bij een verzekeraar door diverse personen en soms applicaties geautoriseerd. Hiervoor is een state machine diagram op te stellen. Weinig klassen in applicaties hebben een voldoende intrigerende levenscyclus om een state machine diagram te verantwoorden. Reden waarom deze modelleertechniek zelden van stal wordt gehaald, alhoewel deze in uitzonderlijke gevallen buitengewoon nuttig is.

Sequence diagram

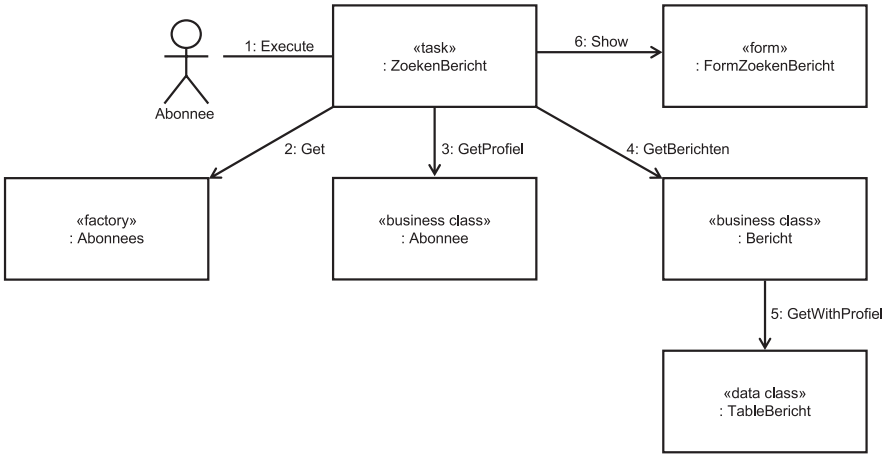
In een sequence diagram wordt de samenwerking tussen *participanten* in de tijd gemodelleerd, zoals in afbeelding 11. In verreweg de meeste gevallen betreft zo'n participant een object of klasse, maar sinds UML 2.0 hoeft dit niet per se. De participanten in een sequence diagram zijn horizontaal als rechthoeken gemodelleerd. Een *lifeline* representeert een participant en is hier gemodelleerd als een stippellijn bij de participanten. De tijd loopt in elk geval van boven naar beneden.
De samenwerking tussen de participanten is gemodelleerd als het versturen van messages. Deze zijn gerepresenteerd als pijlen tussen de betrokken lifelines. Zo verstuurt **InzienBericht** een message **Get()** aan **Berichten**. Het versturen van een message impliceert hier het aanroepen van een methode. Doordat een sequence diagram de tijd van boven naar beneden modelleert, is het betrekkelijk eenvoudig te interpreteren.



afbeelding 11 – Sequence diagram

Communication diagram

Ook een communication diagram geeft de samenwerking weer tussen participanten. Ook hier is het versturen van messages weergegeven als pijlen. Waar het sequence diagram deze modelleert in de tijd, houdt het communication diagram vast aan de relaties tussen de participanten. Het communication diagram is vaak een afspiegeling van een klassendiagram. Zij het dat nu alleen klassen als participant zijn weergegeven die een rol spelen in de samenwerking. Om toch tijd te kunnen modelleren, worden de achtereenvolgende messages in het communication diagram genummerd, zoals is weergegeven in afbeelding 12.



afbeelding 12 – Communication diagram

De beide interactiediagrammen bieden een transformatie van de requirements naar de samenwerking van de klassen van een applicatie. Beide technieken gelden als katalysator bij het identificeren van methoden bij klassen. Omdat beide technieken dezelfde samenwerking modelleren zijn ze uitwisselbaar. In de projecten die ik heb meegemaakt geven ontwerpers steevast de voorkeur aan het sequence diagram, vooral vanwege het prettiger modelleren van tijd. Naast deze bekendere modelleertechnieken kent UML 2.0 ook nog het interaction overview diagram en het timing diagram, die ook interactie modelleren, maar in de praktijk (nog) weinig populariteit genieten.

Interaction overview diagram

Een interaction overview diagram is een gespecialiseerd activity diagram. In een interaction overview diagram komen evenwel interacties voor, die eerder gemodelleerd zijn in een sequence diagram. Van deze interacties wordt bij voorkeur alleen een frame getoond, en liever niet alle participanten, hun lifelines en de messages die worden verstuurd. In een interaction overview diagram is zo de interactie op hoog niveau te modelleren.
Het interaction overview diagram is nieuw in UML 2.0. De combinatie van een activity diagram en een aantal kleine sequence diagrammen ziet er koddig uit. Vergelijk het interaction overview diagram maar met een slaapbank. Een slaapbank combineert de eigenschappen van een bank en een bed. Maar een slaapbank slaapt niet prettig omdat er geen volwaardig matras onder ligt, en hij zit niet lekker omdat een bank andere kussens heeft dan een bed.

Timing diagram

Als de timing in een interactie de belangrijkste reden is om een interactiediagram op te stellen, dan is een timing diagram te modelleren. Een timing diagram modelleert voornamelijk de veranderingen in de status van één individueel object over tijd. Het timing diagram is nieuw in UML 2.0. Een boeiende vraag is hoe populair het timing diagram gaat worden. Ook een sequence diagram en een communication diagram modelleren immers de interactie in de tijd, zij het niet voor individuele objecten.