Thema

Nu de economie zich in een dal bevindt herzien veel organisaties hun doelstellingen. Lange termijn-projecten maken plaats voor korte, agile projecten met haalbare doelen en korte ontwikkeltijden. Om de doelen te kunnen halen wordt in nieuwe projecten gestreefd naar lagere kosten en hogere productiviteit. Hoe dat zou kunnen, leest u in een serie van twee artikelen. Hier deel 1.

Pragmatische model driven development

Met smart use cases en domain driven design

Sander Hoogendoorn
is principal technology officer bij Capgemini.

Rody Middelkoop
is senior technology consultant bij Avisi.

Robert de Wolff
is senior technology consultant bij Capgemini.

Dit streven kan enerzijds worden bereikt door outsourcing, anderzijds kunnen standaardisatie, industrialisatie en hergebruik een bijdrage leveren. Een belofte voor de lange termijn op het gebied van stijgende productiviteit en hergebruik is om minder code met de hand te schrijven, maar om meer en meer code te genereren vanuit het ontwerp of model van het project. Codegeneratie van een model kan op veel verschillende manieren plaatsvinden. Het aantal doelen en op te leveren producten is nog veel groter.

Merkgebonden technieken. Traditioneel zijn er tools zoals Oracle Designer en Developer of Cool:GEN die zowel modelleer- als ontwikkelomgeving zijn. Door van een dergelijke omgeving gebruik te maken kan een hoge productiviteit worden bereikt omdat veel aannames kunnen worden gemaakt op het te genereren product. In het algemeen levert deze manier van ontwikkelen nogal proprietary modellen en code op.

DSL's. Domain Specific Language (DSL) zijn tekstuele of visuele weergaven van domeinen, waarvoor een specifieke modelleertaal kan worden ontwikkeld. Hierbij kun je denken aan de definitie van een webservice of zelfs aan een taal voor het definiëren van een hypotheek- of levensverzekeringdomein. DSL's houden een belofte in van hoge productiviteit mits de doelomgeving duidelijk kan worden gedefinieerd in bijvoorbeeld de gebruikte software architectuur en het framework.

UML. De Unified Modeling Language (UML) geeft een aantal technieken voor het modelleren van het gedrag en de structuur van applicaties. Deze standaard technieken, zoals use case of klasse diagrammen worden gebruikt in projecten over de hele wereld in verschillende variaties en 'dialecten'. Code of deliverables genereren met een UML model is goed mogelijk indien de doelomgeving goed is gedefinieerd.

Database. Wanneer de juiste modelleertechnieken ontbreken, benaderen veel tools de database structuur om (delen van) applicaties te genereren. Deze benadering lijkt aantrekkelijk en snel, maar heeft zijn beperkingen. De gegenereerde applicatie imiteert de database en zal zodoende erg op de data zijn gefocussed, in plaats van het werkproces te ondersteunen en automatiseren. Bovendien hebben in deze tijd van service oriëntatie en cloud computing veel applicaties niet eens meer hun eigen database.

Een pragmatische benadering

In projecten die worden uitgevoerd met het Accelerated Delivery Platform, Capgemini's open kennisplatform voor agile softwareontwikkeling, wordt een eenvoudige pragmatische benadering van Model Driven Development gebruikt. Deze benadering bestaat uit een aantal stappen:

Smart use cases opstellen. We modelleren de functionele requirements met gestandaardiseerde smart use cases. Er bestaat een groeiend aantal van ruim dertig standaardtypen van smart use cases, die het leven nog eenvoudiger maken wanneer het aankomt op analyse van de requirements, het opleveren van een schatting voor een project, maar ook testen en codegeneratie.

Creëer het domeinmodel. Daarnaast creëren we het domeinmodel, waarin de entiteiten, value-objects en andere klassen uit het domein van het project zijn opgenomen.

Decoreer het model. Om code te kunnen genereren combineren we de smart use cases met de elementen uit het domeinmodel en de bijbehorende stereotypes. We associëren bijvoorbeeld de smart use case 'Search customer' met de 'customer' klasse en decoreren deze verbinding met het stereotype <<search>>.

Importeer het model. Vervolgens importeren we het model in de codegenerator Tobago MDA om de code en andere deliverables te genereren. Tobago MDA kan teksttemplates op het model loslaten en iedere gewenste tekstgebaseerde deliverable genereren. Dit kan variëren van Word documenten en Excel spreadsheets tot daadwerkelijke code in vele varianten.

Genereer code gebaseerd op de architectuur. Om het maximale effect te bereiken baseren we het overgrote deel van de applicatie die we ontwikkelen op eenzelfde pragmatische referentie-architectuur die wordt ondersteund door verschillende frameworks, zowel in Java als in .Net, hoewel andere technologie ook mogelijk is.

Niveaus van use cases

We modelleren het gedrag van de te ontwikkelen software in use cases. We beschrijven de use cases aan de hand van vijf verschillende niveaus van granulariteit (zie figuur 1).

Deze vijf niveaus zijn:

Cloud level. Een samenvatting op hoog abstractieniveau. Zoiets als 'Verkoop boeken online'. Dit niveau is te hoog voor software ontwikkelprojecten, en veel te hoog om code te genereren.

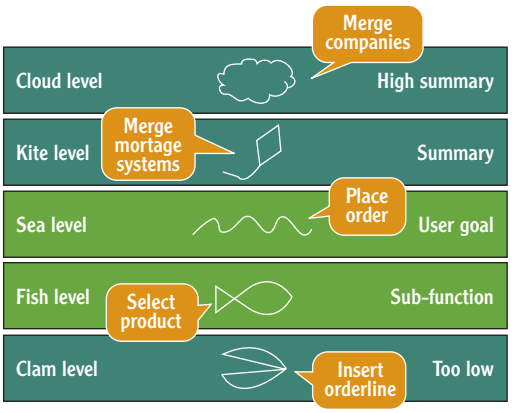
Kite level. De gewone samenvatting. Ook dit niveau is niet interessant voor software ontwikkeling of codegeneratie.

Sea level. Mensen die bekend zijn met het Rational Unified Process (RUP) zouden bekend moeten zijn met dit niveau van use cases, dat ook wel bekend staat als het gebruikersdoel niveau. Een use case op dit niveau beschrijft één elementair bedrijfsproces en realiseert één specifiek gebruikersdoel (b.v. Plaats order).

Fish level. Op dit niveau worden use cases vaak gebruikt en hergebruikt als onderdeel van de sea level use cases. Ze beschrijven processen die nodig zijn om een klant te selecteren of een credit card te valideren bij de uitgever van de kaart. Dit niveau wordt ook wel omschreven als het sub-function niveau en bevat use cases zoals Selecteer product en Betaal bestelling. Deze en de sea level use cases vormen samen een goed begin om de code te kunnen genereren die het gedrag van de uiteindelijke software bepaalt.

Clam level. Hier kom je echt met twee benen op de grond. Use cases op dit niveau zijn eigenlijk geen use cases, maar stappen in een use case op fish level. Een voorbeeld hiervan is het invoeren van nieuwe klanten in de database.

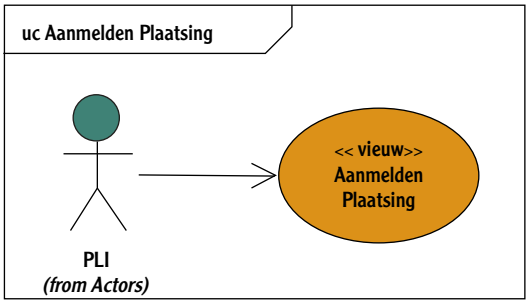
Smart use cases omvatten de use cases op sea en fish level. Samen vormen deze twee niveaus een ideale techniek van use cases met gelijkwaardige granulariteit om de functionele requirements vast te leggen en voor het schatten, plannen, genereren, bouwen en testen van de software.



Ffiguur 1

Smart use cases modelleren

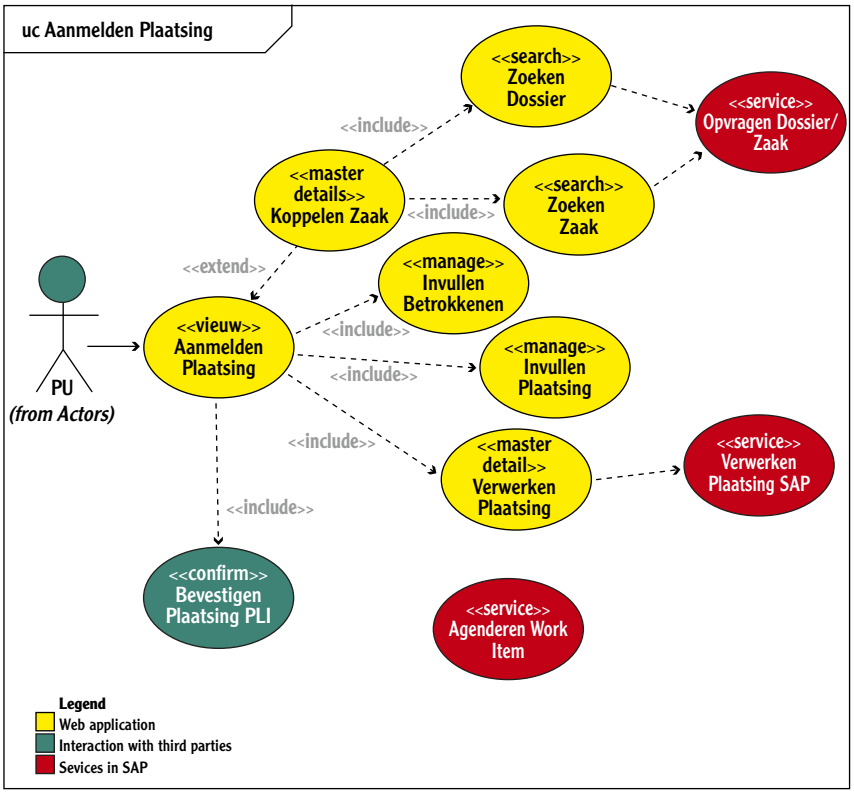
In eerste instantie worden use cases gemodelleerd op user goal level. Iedere use case op dit niveau beschrijft een enkel elementair bedrijfsproces. In de use case wordt het gewenste scenario beschreven (ook wel het 'happy day scenario' genoemd) en alle afwijkingen van dit scenario worden beschreven als alternative flows.



Ffiguur 2

Door voor deze werkwijze te kiezen wordt zeer beperkt gebruik gemaakt van de UML use case diagram modelleer techniek. De eisen worden vastgelegd in tekst en voornamelijk beschreven in Word documenten. Een goed voorbeeld van dit type is de use case Wijzigen adres, die werd opgesteld door een grote internationale financiële instelling. De use case omvatte 65 pagina's tekst, 12 schermen en tientallen alternatieve flows. Aangezien user goal level use cases sterk van elkaar verschilt qua scope, complexiteit en omvang, is het moeilijk, zo niet ondoenlijk, om code te genereren van use cases op het niveau van het gebruikersdoel.

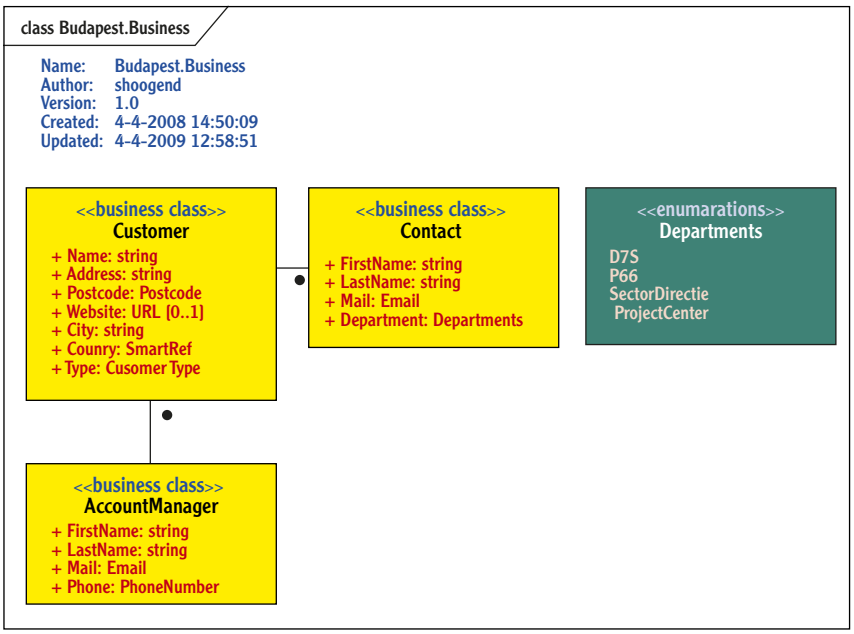
Zodra de use cases op het niveau van het gebruikersdoel helder zijn (of beter nog: als de elementaire bedrijfsprocessen voor het project duidelijk zijn), is het echter ook goed mogelijk om use cases op fish level aan het use case diagram toe te voegen. We



Ffiguur 3

hebben een aantal duidelijke richtlijnen opgesteld om te bepalen wanneer deze extra use cases zinvol zijn. In de richtlijnen is bijvoorbeeld opgenomen dat er niet meer dan één formulier per use case wordt behandeld, hoe met complexe berekeningen moet worden omgegaan, hoe de services in servicegeorienteerde projecten worden behandeld, of hoe je met ETL in BI-projecten omgaat. Het volgende is een goed voorbeeld van een smart use case diagram. Let op: dit model is geen functionele decompositie; de use case op het niveau van het gebruikersdoel heeft zijn eigen taken en verantwoordelijkheden.

Ffiguur 4



In deze werkwijze worden alle elementaire bedrijfsprocessen uitgewerkt in zo'n diagram met alle mogelijke actoren, een use case op gebruikersdoel-niveau en een aantal begeleidende use cases op het sub-function niveau. De verzameling van deze use cases (op beide niveaus) worden smart use cases genoemd. Nogmaals: van ieder van deze use cases wordt een beschrijving gemaakt. Deze zijn echter veel eenvoudiger en bevatten slechts weinig (of geen) alternatieve stromen. We modelleren deze diagrammen overigens niet alleen in normale web of Windows scenario's, maar ook in service georiënteerde en zelfs cloud scenario's, waarbij ook het leveren van de services in smart use cases is uitgedrukt.

Smart use cases stereotypen

Een belangrijke stap naar het genereren van code van use cases is het toevoegen van stereotypen aan de smart use cass. Zo'n stereotype beschrijft standaardgedrag. Deze extreme standaardisatie van de requirements stroomlijnt de analyse en maakt het mogelijk om templates per stereotype te maken, die het gewenste gedrag implementeren in bijvoorbeeld Java, C# of andere talen, zoals PHP of Delphi. We tellen inmiddels ruim dertig van deze smart use case stereotypen, zoals «manage», «master detail», «select» of «search», zoals in het afgebeelde diagram, maar ook «collect» in BI-projecten en verschillende stereotypen voor services in servicegeoriënteerde projecten.

Maak het domeinmodel

De volgende stap naar het genereren van code en andere deliverables is het maken van een domeinmodel voor het project. Naast de smart use cases, die het gedrag vastleggen, voorziet het domeinmodel in een structureel overzicht van het systeem. Simpel gezegd bevat het domeinmodel alle entiteiten, types, regels, validaties en services van het bedrijfsdomein, uitgedrukt in de terminologie van de klant. Bovendien beschrijft het model de relaties tussen de verschillende types.

In deze pragmatische benadering volgen en werken we verder op de principes van Eric Evans in zijn domain driven design paradigma. Zoals je kunt verwachten wordt het domeinmodel uitgedrukt in UML klassendiagrammen. Eerst wordt vorm gegeven aan de entiteiten in het domeinmodel (zoals Customer, Order, Subscription, of Course) en hun relaties. Vervolgens worden de eigenschappen en de types gemodelleerd.

We onderscheiden vijf verschillende categorieën met typen properties van entiteiten, die toepasbaar zijn in verschillende situaties:

Basic types. Doorgaans neigen ontwikkelaars ernaar om property types te modelleren in basic types, zoals string. integer, boolean of date. In veel gevallen weten we echter veel meer over deze typen dan

dat het een string of een integer is, zoals bijvoorbeeld in het geval van Postcode of Isbn. In dat geval is het beter om een value object te gebruiken.

Value objects. Een value object heeft geen identiteit, maar geeft een waarde weer. Value objects zijn zeer bruikbaar om property types te modelleren, die kunnen worden gevalideerd, zoals Bsn, Isbn of Email.

Enumeraties. We modelleren property types als een enumeratie als de property alleen een gelimiteerd, vast aantal van verschillende waarden kan hebben. Denk hierbij aan: ContractType, Gender of Level.

Smart references. Wij modelleren properties als smart reference als de property uitsluitend een beperkt, maar mogelijk veranderend aantal verschillende waarden kan hebben. Dit geldt bijvoorbeeld voor properties zoals Department, Prefix of Country (als geen andere informatie van landen nodig is).

Associations. Als het property-type zelf ook een entiteit of domeinobject is, modelleren we die relaties liever als een association.

Ieder van deze categorieën property types kunnen goed worden gebruikt in de codegeneratie. Enumeraties en smart references kunnen bijvoorbeeld worden gebruikt om drop-down lijsten te vullen, en associations kunnen natuurlijk worden gebruikt om master-detail gedrag of om configuratiebestanden voor object-relationship mapping te genereren.

Scenario's voor codegeneratie

In wezen zijn er drie scenario's voor codegeneratie die UML modellen toepassen.

- **Modeling tool codegeneratie.** In het eerste scenario kent de UML modelleeromgeving, zoals bijvoorbeeld Enterprise Architect, functionaliteit om code te genereren. Aan de ene kant maakt deze benadering het mogelijk om dicht bij het originele model te blijven, maar aan de andere kant verplicht het je om steeds die specifieke modelleeromgeving te gebruiken.
- **Development tool codegeneratie.** In het tweede scenario kent de ontwikkelomgeving functionaliteit om het UML model vorm te geven en hier code uit te genereren. Je kunt soms ook een plug-in voor de ontwikkelomgeving gebruiken, die je hiermee helpt. In dit scenario zit je echter vast aan een specifieke ontwikkelomgeving en bijbehorend platform.
- **Intermediaire codegeneratie.** Het derde scenario gebruikt een intermediaire codegenerator dat het UML model importeert (via het standaard exportformaat XMI) en direct de code uitspuwt. In dit geval – hoewel je min of meer kiest voor een specifieke codegenerator – heb je de keuzevrijheid wat betreft modelleer- en ontwikkelomgeving.

Werken met Tobago MDA

Omdat we in veel projecten werken (Java, .Net, SAP, BI of SharePoint) waar de klant beslist over de te gebruiken modelleer- of ontwikkelomgeving, werkt het derde scenario tamelijk goed. In de praktijk neigen we ernaar om Enterprise Architect of een van de Rational modelleergereedschappen te gebruiken in combinatie met .Net en Visual Studio of Java en het Eclipse platform. Hoewel andere codegeneratoren ook kunnen worden gebruikt, hebben wij gekozen voor Tobago MDA. Dit pragmatische gereedschap is ontwikkeld door het core team van Capgemini's agile Accelerated Delivery Platform en is vrij beschikbaar voor iedereen die zich registreert op de wiki van het platform (www.acceleratteddeliveryplatform.com).

Tobago MDA voert, net als andere intermediaire model driven development tools, een aantal acties uit:

- **Define project.** Wanneer je een nieuw project start moet je eerst de settings definiëren.
- **Define patterns.** Een patroon is een set van tekstgebaseerde bestanden die samen kunnen worden gebruikt om code of andere deliverables te genereren. Zo'n patroon bestaat uit een patroondefinitie, de hoofd template en mogelijk een kleine directorystructuur met de benodigde aanvullende templates. Ieder patroon kan worden gebruikt om één type deliverable te genereren, bijvoorbeeld een domeinobject in .Net, een domeinobject in Java, een Excel spreadsheet met een smart use case schatting of een webpagina gebaseerd op een enkele use case. Het spreekt voor zich dat de verzameling patterns die beschikbaar zijn voor Tobago MDA snel groeit vanuit de projecten waarop het wordt toegepast. Op het ogenblik bevat deze collectie meer dan 100 verschillende patronen die zijn gericht op verschillende programmeertalen en frameworks en zelfs Word documenten voor het testen en Excel spreadsheets voor het maken van schattingen omvatten.
- **Import model.** De volgende stap is het exporteren van het model uit de UML modelleeromgeving, met gebruikmaking van het standaard XMI formaat, en dit importeren in Tobago MDA.
- **Generate deliverable.** Last but not least bindt de codegenerator een set van patronen aan een (set van) modelelement(en). Om te definiëren welke patronen werken op welke modelelementen worden stereotypes toegevoegd aan het model. Als een domeinobject bijvoorbeeld het stereotype «domain object» krijgt, gelden er een (groot) aantal patronen, inclusief .Net en Java domeinklassen, een domeinobject unit test, een table gateway, zowel Hibernate als nHibernate configuratiebestanden en zelfs een SharePoint web part. «

In het volgende nummer gaan de auteurs verder in op onder meer het veranderen van het model zonder code kwijt te raken en het gebruik van patterns in Tobago MDA.

Er zijn nu al meer dan 100 patterns voor Tobago MDA en de verzameling groeit snel

Referenties:

www.acceleratteddeliveryplatform.com