Domeinspecifiek code genereren met MetaEdit+

Modelgedreven processen zorgen momenteel voor de nodige veranderingen in de werkwijze van softwareontwikkelaars. Dit artikel beschrijft enkele praktijkervaringen met domeinspecifiek modelleren in het pakket MetaEdit+, waarbij de nadruk ligt op de impact en voor- en nadelen van codegeneratie. MetaEdit+ is in gebruik bij internationale bedrijven als EADS en Nokia, en in Nederland bij bijvoorbeeld ICT Automatisering en KPN.

Angelo Hulshout

Het doel van domeinspecifiek modelleren (DSM) is het verhogen van de productiviteit van softwareontwikkeling. Om dit te bereiken, wordt het handmatig coderen in diverse programmeertalen vervangen door grafische modellering en codegeneratie. Wat DSM anders maakt dan andere Model-Driven Software Development-aanpakken zoals Model-Driven Architecture, is het gebruik van domeinspecifieke grafische modelleertalen. Deze zijn gebaseerd op concepten uit het probleem- in plaats van het oplossingsdomein. Ontwikkelaars modelleren er delen van de probleemruimte mee, waarna ze uit de gecreëerde modellen werkende code genereren. Effectief modelleren ze dus niet de oplossing, maar het probleem. De uiteindelijke software en andere details uit het oplossingsdomein worden zodoende onzichtbaar voor de applicatieontwikkelaar.

Voordat we op deze wijze toepassingen kunnen genereren, moet er aan een aantal randvoorwaarden worden voldaan. Eerst en vooral moet er sprake zijn van vergelijkbare of identieke codeeractiviteiten die zeer regelmatig terugkeren, bijvoorbeeld in softwareproductfamilies. Veelal is dit meteen de basis voor de tweede voorwaarde: de aanwezigheid van een (domeinspecifiek) platform of framework dat de basis vormt voor de resulterende code. Denk hierbij aan het Symbian-besturingssysteem voor Nokia-telefoons of aan het serviceraamwerk dat ICT Automatisering op de BEA-applicatieserver draait, maar dit kan ook een platform voor automatische testcases zijn. Als we hieraan voldoen, kunnen we een domeinspecifieke taal definiëren met een bijbehorende codegenerator. Samen vormen deze de kern van de DSM-ontwikkelaanpak.

De implementatie van DSM heeft niet alleen invloed op de manier van werken, maar ook op de rollen die een ontwikkelorganisatie nodig heeft. Waar we gewend zijn applicatie- en platformontwikkelaars te hebben, of zelfs teams die bestaan uit beide, zal domeinspecifiek modelleren andere rollen introduceren. Zo komt het werk van de applicatieontwikkelaar voortaan (deels) voor rekening van een codegenerator, die we op zijn beurt wel weer moeten bouwen en onderhouden. Boven op die generator ligt een domeinspecifieke taal, die we ook weer moeten ontwerpen en onderhouden. Daarvoor zijn taalontwikkelaars en domeinspecialisten nodig. De laatsten maken ten slotte ook nog de modellen waaruit we de uiteindelijke applicaties genereren.

Platslaan

Dat het introduceren van een aanpak als deze niet vanzelf gaat, ligt voor de hand. In een artikel met de titel 'De methode doet het niet' beschreef informatiekundige Jaap van Rees ooit hoe gebruikers de fout in konden gaan bij het toepassen van de Isac-methode voor de ontwikkeling van informatiesystemen. Voor codegeneratie in het algemeen, maar zeker ook in een DSM-omgeving, geldt eveneens dat de mate van succes afhangt van de wijze waarop mensen werken met de gereedschappen die ze hebben. Gebruikmakend van ervaring die EADS, ICT Automatisering, Nokia en nog enkele andere partijen hebben opgedaan met DSM, zal ik hier schetsen wat er zoal speelt in de praktijk.

Het moge duidelijk zijn dat ontwikkelen op basis van codegeneratie uit modellen pas echt effectief is als de codegenerator ontwikkelaars ook daadwerkelijk werk uit handen neemt. Ook moet het niet zo zijn dat de tijd die we besparen volledig opgaat aan modelleerwerk, taaldefinitie en het schrijven van de generatoren. Hier ligt direct een rol voor zowel de domeinexpert als de softwareontwikkelaar. Dit omdat er maar één manier is om het gewenste effect te bewerkstelligen in een DSM-omgeving: door voldoende afstand te creëren tussen modelleertaal en code, gecombineerd met een krachtige codegenerator. Deze afstand noemen we ook wel het abstractieniveau van de modelleertaal, of de afstand tussen het probleemdomein (het domein van de 'gebruiker') en het oplossingsdomein (het domein van de ontwikkelaar).

Een eerste poging om een DSM-taal te maken, resulteert vaak in de verkeerde modellen. Dit komt helder naar voren in de telefoniecasus waarover ik vorig jaar heb verteld op de Code Generation-conferentie. Centraal in services als doorschakeldiensten of helpdesks staat de herkomst en bestemming van een oproep, traditioneel geïdentificeerd door het welbekende tele-

DSM in schadeverzekeringen

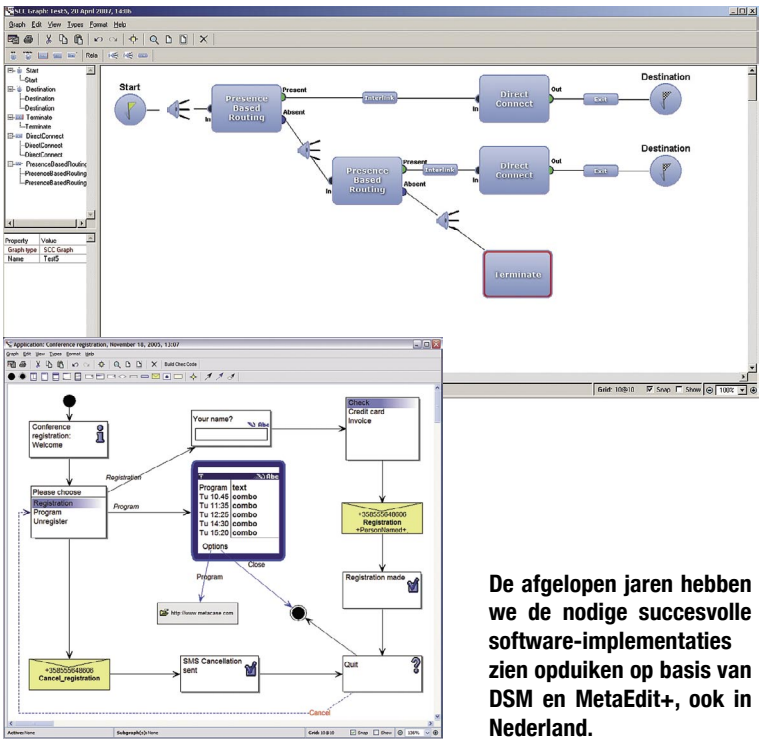
Een Duitse verzekeringsmaatschappij is aan de slag gegaan met DSM en heeft een model gemaakt van een schadeverzekering voor de auto. Met een druk op de knop is hieruit code te genereren voor een website waarop klanten de polis kunnen afsluiten. De afstand tussen het verzekeringsproduct (probleemdomein) en de gegenereerde website is dermate groot, dat de bouwer van het model geen enkele kennis van softwareontwikkeling (oplossingsdomein) hoeft te hebben. De DSM-taal en de codegenerator maken alle details onzichtbaar waar softwareontwikkelaars doorgaans mee te maken krijgen.

foonnummer. Bij telecomingenieurs is dit zo ingeburgerd dat ze alles ophangen aan dit nummer en de digits waaruit het bestaat. Historisch en technisch zijn beide echter onderdeel van het oplossingsdomein. Bij Voip bellen we bijvoorbeeld met iets dat lijkt op een e-mailadres. Een modelleertaal die werkt met digits is daarom niet zo zinvol. De uiteindelijk ontwikkelde taal praat op een veel hoger abstractieniveau over de herkomst en bestemming van een oproep, zonder het concept 'adres'. De implementatie is verborgen.

Vergelijkbare voorbeelden uit andere projecten laten duidelijk zien dat we bij de taalontwikkeling de input nodig hebben van domeinexperts, die vanuit het probleemdomein redeneren. Anders ontstaat een grafische programmeertaal, waarbij codegeneratie hetzelfde is als het platslaan van het model in tekst die de compiler begrijpt. Dat levert geen tijdwinst op en zadelt de domeinexpert op met softwareontwikkelvraagstukken waar hij niets van wil weten.

Drempel

Het andere element van de DSM-combinatie, een krachtige codegenerator, maakt het mogelijk om met de juiste taaldefinitie echt aan productiviteit te winnen. Hier is het zaak om het gereedschap zo veel mogelijk te laten genereren met zo min mogelijk additionele informatie van de gebruiker. Een generator zoals die in oude Case-tools en de eerste generatie UML-pakketten zat, produceerde raamwerken waaraan we vervolgens met de hand (al dan niet in het modelleerpakket zelf) de ontbrekende code moesten toevoegen. DSM is gebaseerd op het principe dat we alle code kunnen negeren, mits het probleemdomein voldoende is afgebakend en gedefinieerd.



De afgelopen jaren hebben we de nodige succesvolle software-implementaties zien opduiken op basis van DSM en MetaEdit+, ook in Nederland.

In het MetaEdit+-gereedschap van het Finse Metacase werkt dit principe door in de taal van de codegenerator. Het daar afbakenen van een domein resulteert in een heleboel restricties, bijvoorbeeld ten aanzien van het aantal voorkomens van een element in één model. Door toepassing van deze principes hoeft de codegenerator verschillende controles op de correctheid van het model niet meer te implementeren.

Een van de belangrijkste drempels voor softwareontwikkelaars om MetaEdit+ te gebruiken, was lange tijd de onmogelijkheid om waardes en toestanden bij te houden. Een codegenerator geschreven in de MetaEdit+-specifieke taal Merl gaat element voor element door een model heen en produceert de code die bij de gepasseerde elementen hoort. In principe hoeft het gereedschap daarbij niets te onthouden over voorgaande stappen, al houdt het zelf wel een stack bij voor het geval dit toch een keer nodig is. Het hiermee leren omgaan bleek in de praktijk soms een lastig proces. Zo is het voorgekomen dat een ontwikkelaar de modellen exporteerde naar XML, om daar vervolgens een zelfgeschreven codegenerator in Java op los te laten, zodat hij wel met variabelen kon werken.

Inmiddels heeft Metacase deze behoefte onderkend en de drempel verlaagd door Merl uit te breiden met de mogelijkheid om variabelen te gebruiken. Een goede vervolgstap zou zijn om de operatoren in de taal iets minder cryptisch te maken, zodat ervaren ontwikkelaars zich er eerder in thuis voelen. Eenmaal thuis in Merl bouwen ze echter in no time een codegenerator. Belangrijk hierbij is dat dat bijna altijd gebeurt op basis van de handgeschreven code van een eerder geïmplementeerde oplossing. Het schrijven van een generator is dan eigenlijk niet meer dan het parametriseren van die code.

De afgelopen jaren hebben we de nodige succesvolle software-implementaties zien opduiken op basis van DSM en MetaEdit+, ook in Nederland. De aanpak en de tooling zijn nog steeds volop in ontwikkeling, maar nu ook Microsoft (DSL Tools) en UML-moeder Object Management Group (werkgroep Domain-Specific Languages) ermee aan de slag zijn gegaan, voltrekt die ontwikkeling zich alleen maar sneller. De technische hobbels worden snel lager en kleiner in getal. Een volgende uitdaging is de introductie van de werkwijze die bij modelgedreven ontwikkeling hoort.

Angelo Hulshout is senior consultant bij ICT NoviQ. Hij is onder meer betrokken bij de introductie van de DSM-aanpak bij klanten, in nauwe samenwerking met Metacase.

Redactie Nieke Roos