OptimalJ is één van de nieuwe MDA-Java tools. Het kwam samen met een aantal soortgelijke tools aan bod in Software Release Magazine nummer 6/ Java Magazine nr. 1, oktober 2002. In dit artikel gaat het echter niet om het tool op zich, maar om de ideeën erachter, met name over MDA. Wim Bast, chief architect bij Compuware, legt uit hoe OptimalJ zelf op MDA-achtige wijze ontwikkeld werd, bijna als Münchhausen die zichzelf aan zijn haren uit het moeras trok.

Wim Bast: 'Ons gehele ontwikkeltraject is model-gedreven geworden'

Java-code via zelf beschreven en gegenereerde MOF

MDA is iets waar je moeilijk omheen kan, maar wat is het precies? Iedereen heeft er wel een idee over, maar vaak blijken die ideeën niet overeen te komen.

Wim Bast: 'MDA zou ik willen zien zoals het gepresenteerd is door de OMG aan het begin: vooral gericht op voordelen die MDA claimt te bieden en problemen die ermee opgelost zouden moeten worden. Wat heb je nodig om die problemen op te lossen, welke standaards zijn daarvoor beschikbaar en welke standaards zijn in ontwikkeling?

De OMG -een standaardisatie orgaan – is vooral beroemd om twee dingen: Corba, met IDL, en UML. De OMG kent al geruime tijd een zogenaamde OMA (*Object Management Architecture*), die nog steeds erg gericht is op Corba technologie. De bedoeling is dat interfaces geschreven zouden worden om interoperabiliteit te krijgen tussen applicaties, maar ook dat bepaalde generieke services gestandaardiseerd zouden worden, zodat we daarvoor geen eigen oplossingen hoeven te maken. Daarnaast is er UML als generieke modelleringstaal, veel gebruikt in de softwarewereld voor diagrammatisch modelleren van allerlei verschillende zaken. De MDA-filosofie en -architectuur is vooral bedacht om het modelleren beter in relatie te brengen tot applicaties en technologie, en de gap daartussen expliciet op te gaan lossen en om tot standaardisatie te komen. Daarbij heeft men zich een aantal doelen gesteld: portabiliteit, interoperabiliteit, productiviteit bij het bouwen en vooral ook bij het onderhouden van applicaties. Een ander doel is de mogelijkheid om domeinspecifieke modellen te maken, die los staan van de technologie en aan de andere kant technologische modellen te maken die los staan van de domeinspecifieke functionaliteiten. Daaruit vloeide weer de wens voort transformaties te definiëren tussen die twee waardoor je optimaal resultaat kunt krijgen. Dat betekent dus dat je geen concessies doet aan je technologische kracht maar aan de andere kant je in je domeinspecifieke conceptuele modellen niet hoeft te bemoeien met de technologie.'

MODELGEDREVEN Bast: 'De OMG ziet dus in, dat IDL niet de enige technologie is en dat portabiliteit, interoperabiliteit en dergelijke niet alleen door een standaardtechnologie - waardoor alle communicatie concreet moet plaatsvinden – op te lossen is.

Ze ziet dat we meer de oplossingen moeten zoeken in een conceptueel model waarin we onze functionaliteiten onafhankelijk van de gebruikte technologieën beschrijven en waar uit de applicaties en interoperabiliteitsoplossingen in de verschillende technologieën worden gegenereerd. De OMG zegt ook: er zijn een heleboel technologieën, en we moeten al die technologieën aan elkaar verbinden en dat kunnen we beter doen via een modelgedreven aanpak dan door weer een andere technologie er tegenaan te zetten die claimt het allemaal op te lossen. Daarmee omarmen ze ook een bredere gemeenschap, dat is ook heel goed en heel realistisch. Dat is dus eigenlijk, wat we ons moeten voorstellen bij MDA.

MDA wordt vaak uitgelegd via PSM en PIM, platform specifieke en platform independent modellen. De

bedoeling van een PIM is zo'n model onafhankelijk van een bepaald platform of technologie is. Het PSM moet juist specifiek voor een platform en technologie zijn, waardoor je een volledige oplossing krijgt. Eigenlijk een kwestie van *division of concerns*. Op het moment dat een organisatie gewijzigd wordt, of de structuur van de BTW verandert, dan wil je dat op één plek specificeren ongeacht van de technologieën. Je wilt dat dit soort wijzigingen in alle technologieën die je gebruikt - en iedereen gebruikt veel verschillende technologieën - door laten werken. Als dat werkt, als dat meer geautomatiseerd en ook liefst gestandaardiseerd zou plaatsvinden, dan geeft dat verhoogde productiviteit. Je hoeft namelijk zelf minder na te denken over hoe dat in al die technologieën vertaald moet worden, en er kunnen tools ontwikkeld worden die dat proces automatiseren. Daar gaat het eigenlijk om, simpel gezegd. Het is belangrijk te realiseren dat het hierbij noodzakelijk is een PIM op een aanzienlijk hoger abstractie niveau dan de PSM te definiëren. Anders is het niet mogelijk om de verscheidenheid aan bestaande technologieën te overbruggen. Dat is dus heel wat anders dan bijvoorbeeld een 1-op-1 UML class diagram van de classes in de Java-code. Deze laatste vorm is eigenlijk helemaal geen MDA omdat deze niet in staat is de geclaimde voordelen van de MDA bieden.'

'Het wijzigen van requirements is de main drive: dat is de manier waarop software gemaakt moet worden'

MOF

Naast UML als de facto *standaard modelleringstaal, bestaan binnen de OMG nog veel meer modelleringstalen, bijvoorbeeld CWM, Common Warehouse Metamodel. (CWM is een taal waarin je met allerlei technologieën, bijvoorbeeld relationeel en hiërarchisch, datawarehousing modellen kan beschrijven.) Veel OMG-talen zijn min of meer PSM.*

Bast: 'Al deze modelleringstalen die binnen de OMG gestandaardiseerd zijn, zijn gespecificeerd met gebruikmaking van Meta Object Facility (MOF). Dat is ook een modelleringstaal, met dit verschil dat hij bedoeld is als taal om alle andere talen erin te definiëren, inclusief zichzelf. Die standaard hebben wij zeer expliciet en veelvuldig gebruikt om alle modelleringstalen die in OptimalJ bestaan, ermee te specificeren en vervolgens hieruit de implementaties te genereren. Een aantal aan MOF gelieerde standaards zijn XMI en JMI (Java meta interfaces), dat zijn manieren om tot interoperabiliteit tussen modelgedreven tools te komen. Door toepassingen van deze standaards in OptimalJ hebben we ook de interoperabiliteit op toolniveau kunnen oplossen. Het onderkennen van het extra metaniveau van MOF is heel belangrijk om grip op zo'n development proces te krijgen en te zien op welk metaniveau je je problematiek moet oplossen om zo veel mogelijk profijt te hebben in zoveel mogelijk toepassingen.'

Transformaties

MOF wordt volgens Bast steeds belangrijker voor MDA en Compuware heeft zich er voor ingezet dat deze trent zich ook binnen de OMG ontwikkelde.

Bast: 'Tot voor kort was MOF binnen de OMG meer een technologie die vooral intern gebruikt werd om bepaalde specificatie te kunnen maken, nu wordt het meer en meer een technologie die in zijn algemeenheid gebruikt gaat worden om, MDA-achtige oplossingen te kunnen specificeren. Bij MDA is het heel belangrijk dat je niet alleen de modelleringstalen specificeert, maar ook de transformaties ertussen, eigenlijk tussen de modellen die in die talen zijn gespecificeerd. Daarom is een nieuwe RFP (request for proposal, een startpunt voor een verandering bij de OMG) geopend. Hij heet MOF 2.0 Query/View/Transformations RFP, dat betekent dat in die RFP gevraagd wordt om een taal, waarin je query's, views en transformaties kunt specificeren op talen die in MOF zijn gespecificeerd. Dat is één van meerdere toepassingen om onder andere de gap tussen PIM's en PSM's te kunnen dichten.

Intern, in OptimalJ, hebben wij natuurlijk oplossingen voor het specificeren en executeren van transformaties. Wij hebben dat gerealiseerd in die incrementele zin, dat wil zeggen dat iedere transformatie tussen twee modellen herhaald kan worden uitgevoerd met mogelijke wijzigingen in zowel het input- als het outputmodel. De transformaties kunnen een flink deel van het verschil tussen de abstractieniveaus van de modellen

overbruggen. Daar hebben we onze eigen oplossingen voor gemaakt. We gebruiken de ervaring die we daarin hebben opgedaan om binnen de OMG tot een goede standaard te komen.'

Grote lijnen

Hoe is de ontwikkeling van OptimalJ nu in grote lijnen gegaan?

Bast: 'Nadat wij zaken als requirements en product-strategie redelijk goed in kaart hadden gebracht, hebben we gezegd: "we hebben altijd al een background gehad in een modelgedreven ontwikkeltool, Uniface". We wilden al voordat de term MDA überhaupt bestond een technologisch onafhankelijke modelleringstaal maken, gebaseerd op UML en transformaties naar een aantal technologieën. We hebben vooral gekozen voor J2EE-technologieën en relationele technologieën, maar ook technologieën als COBOL, XML en IDL, omdat je natuurlijk geen applicaties wilt maken die je niet kunt connecteren met de rest van de wereld. We hebben dus geïnventariseerd, welke technologieën we willen adresseren en hoe het logische model eruit moet zien: helemaal gebaseerd op UML, in ieder geval op dat deel van UML wat een duidelijke en heldere semantiek heeft en wat ook vaak gebruikt wordt: het class-model. Dat noemen wij het domain model. Voor de modelleringstalen voor de verschillende technologieën hebben we gebruik gemaakt van onderdelen van de CWM standaard.

Het zal niemand verbazen dat wij alle modelleringstalen in MOF hebben gespecificeerd. We hebben ook een op MOF gebaseerde taal ontwikkeld om de transformaties tussen modelleringstalen te kunnen specificeren. We genereren de repository's voor de modellen en de engines voor de transformaties automatisch in Java. We zijn dus gewoon begonnen met het implementeren van MOF en het implementeren van MOF-transformaties. Toen hebben wij met gebruik van MOF de metamodellen van het domain-model en de specifieke technologieën gemaakt en vervolgens de transformatie-specificaties daartussen. Eigenlijk is het heel simpel.'

Hoogste metaniveau

Bast: 'Waar wij heel bewust voor hebben gekozen als Compuware, is dat wij geen Java compiler of editor zijn gaan maken. We hebben gekozen voor oplossingen die er al gewoon zijn. We hebben ons geconcentreerd op modelgedreven applicatieontwikkeling, dat is zo'n toegevoegde waarde. Daarom hebben we voor de pure Java code development gebruik gemaakt van NetBeans, een open source Java IDE. We hebben dus samengewerkt met het Net-beans team van SUN, die NetBeans heeft gebouwd en als onderlaag levert voor alles wat echt om Java-codering gaat.

Ook MOF zelf hebben we met gebruik van MOF geïmplementeerd. In wezen hebben we een soort boot-

trap uitgevoerd: we hebben eerst dat deel van MOF volgens de standaard met het handje geïmplementeerd, dat voldoende is om MOF zelf ermee te definiëren. Vervolgens hebben we de formele definitie van MOF zelf genomen en dat hebben we gevoed aan ons handmatig geschreven tool. Tenslotte hebben we alles weggegooid wat we handmatig hebben geschreven. En nu zitten we dus in de situatie waarin een heel groot en significant deel van de code die wij in Java hebben gemaakt, modelgedreven is ontwikkeld, inclusief de implementatie van MOF zelf. Ons gehele ontwikkeltraject is modelgedreven geworden, tot en met het hoogste metaniveau toe. Dat geeft gigantische voordelen, het is erg prettig en productief werken. Ik denk ook dat we daarom, samen met het beperken van onze ontwikkeling tot onze toegevoegde waarde, in staat zijn geweest om in een beperkte tijd – circa tweeënhalf jaar - een behoorlijk rijk tool neer te zetten.'

Coarse grain

Hoewel OptimalJ het mogelijk maakt platformspecifieke details te wijzigen en zelfs de transformatie-specificaties te veranderen, is het voldoende om eenvoudige plaatjes van de gewenste functionaliteit te maken.

Bast: 'Je kunt gewoon een diagram met de klassen klant en order maken, alsof alle technologische problemen niet bestaan. Aan de andere kant zorgt OptimalJ ervoor dat er automatisch met gebruik van bewezen J2EE patronen technologiespecifieke modellen voor J2EE worden gemodelleerd, en daarna ook gecodeerd. OptimalJ ondersteunt die coarse grain patronen (design practices en implementatie practices die noodzakelijk zijn om goed performende J2EE applicaties te maken). Bij gedistribueerde applicaties werkt het gewoon niet als je heel frequent heel weinig informatie uitwisselt. Het is daarom noodzakelijk om de componenten coarse grain te realiseren, waarbij er met lage frequentie grotere hoeveelheden informatie worden uitgewisseld. Dat

OptimalJ dit automatisch voor je regelt is voor veel J2EE-developers heel erg aantrekkelijk. De interfaces en structuren die uitgewisseld worden zijn nu eenmaal niet zo triviaal als in een of andere niet-gedistribueerde omgeving, dat moet je weten en daar moet je mee kunnen omgaan. Veel van die expertise ligt vast, waardoor je heel snel werkende applicaties kunt maken, dus niet alleen een setje interfaces of zo. Tegelijkertijd geven we de mogelijkheid om op al die niveaus tot op codeniveau ook je aanpassingen te kunnen maken. Dat is iets wat Java-developers altijd zullen blijven willen en dat is ook iets wat we ondersteunen, waarbij we al die lagen toch nog consistent met elkaar houden. Daar hebben we heel bewust voor gekozen. Iedereen wil eigenlijk altijd controle hebben op al die niveaus en dat moet ook mogelijk zijn, maar tegelijkertijd wil je die toegevoegde waarde van het automatische genereren van de modellen en de code. Dat was een requirement voordat we überhaupt begonnen met bouwen en implementeren. En het was ook het requirement dat met name de manier heeft gedreven, waarop we onze transformatie-engines hebben geïmplementeerd.'

Testen en XP

Testen is sterk geïntegreerd en geautomatiseerd in de productie van OptimalJ zelf. De schrijver van de test en de programmeur zijn echter verschillende personen.

Bast: 'In XP schrijf je eerst je test en ga je daarna programmeren. Wanneer je dat doet, kom je er al snel achter dat het weinig toegevoegde waarde heeft. Modelgedreven testen waarin je meer afstand neemt van de technologie en meer de functionaliteit specificeert, heeft meer toegevoegde waarde omdat je veel meer de implementatie test. Daar zijn we behoorlijk ver in gevorderd.

Iedere keer als er iets ingecheckt wordt, dan wordt het ook getest. Er vinden vier of vijf builds per dag plaats, inclusief tests. In iedere build worden een aantal taken gebouwd, waarin een bepaald requirement of een bepaald probleem opgelost is. Deze taken zijn gerelateerd aan een projectplanning en een releaseplanning. Door het integreren van automatische testen en de builds zijn de code en de modellen eigenlijk continu stabiel. Dat werkt heel prettig, zo zou een software factory moeten werken. Niet: jongens we gaan allemaal even onder water en een jaar later gaan we proberen alles aan elkaar te integreren. Dan blijkt bij een test ineens wat je echt had moeten doen.'

Main drive

Misschien een beetje gechargeerd gesteld, maar dat gebeurt toch niet zelden.

Bast: 'Ja, ik proef als jouw conclusie dat de IT-wereld hardleers is, oude wijn in nieuwe zakken. Ik denk dat het vooral gaat bij projecten waarbij je product een

lange levenscyclus moet hebben. Heel vaak worden er producten ontwikkeld met de illusie dat het na drie maanden af moet zijn en dan klaar is. Wat er in werkelijkheid gebeurt is, dat men op dat moment begint aan een product wat jaren mee moet. Het moet continu gewijzigd worden omdat de eisen die eraan gesteld worden voortdurend veranderen. Het is zelfs al zo, dat de eisen na drie maanden alweer veranderd zijn. Dat is iets wat je eigenlijk continu als uitgangspunt moet nemen bij de inrichting van een softwarefabriek. Dat is de manier waarop software gemaakt moet worden: het wijzigen van requirements is de main drive. Het is niet zo dat op een dag je design af is en dat je dan gaat bouwen. Nee, het is een continu wijzigen van je requirements wat je moet oplossen. Als je dat tot uitgangspunt neemt van al je technologieën, tools en de manier waarop je dat inricht, dan zal je product veel langer levend blijven dan wanneer je het aanpakt vanuit de illusie: "Eens is het af". OptimalJ is bedoeld voor het inrichten van een dergelijke softwarefabriek en zo gaan wij zelf ook te werk bij het ontwikkelen van OptimalJ.'

Tekst en fotografie: Dré de Man