Core

Gert Drapers is bij Microsoft Principal Software Architect voor Visual Studio Team Edition for Database Professionals. Daarvóór was hij onder meer software architect voor SQL Server en Indigo. Ter gelegenheid van de DevDays verliet hij Redmond even om zijn geboorteland op te zoeken. Sander Hoogendoorn, Wouter Goedvriend en Dré de Man spraken met hem over ADO.Net 3.0, OR-mapping, het belang van databases en de toekomst van applicaties én over zijn bug jail.

De bug jail van Gert Drapers

Dré de Man
Tekst en foto's

Met ADO.Net lijkt het erop alsof Microsoft nu echt object/relational mapping voor iedereen beschikbaar maakt. N-Hibernate deed dat daarvóór al. Wat is nu het verschil?
Drapers: 'Als er iemand sceptisch is over O/R-modellen dan ben ik het wel. Ik heb te veel SQL Server-implementaties moeten doen waarbij het probleem daarvandaan kwam. Ik denk dat het in eerste instantie lood om oud ijzer is.
Uiteindelijk denk ik dat wij een betere integratie voor elkaar kunnen krijgen doordat het beter gaat integreren met alle tools. Dat is uiteindelijk waar een deel van de winst zit en waarom het interessant is voor bedrijfsapplicaties om dat soort technieken te gebruiken. Mijn grote angst is meer: we geven de klant meer keuzes, vroeger had je een dataset en een datareader en een connectie en een command object en dat was het, en nu worden daar meer abstractielagen bovenop gebouwd, en hoe meer abstractielagen je er bovenop bouwt, hoe meer keuzes je kunt maken, en keuzes betekent fouten. Het voordeel ervan is dus dat we er daardoor voor zorgen dat de gebruikers de juiste richting op blijven lopen en zichzelf niet aan een grote steen gaan stoten.'

En tools?
Drapers: 'Tools hebben tot op zekere hoogte voordelen maar uiteindelijk zal iemand toch moeten gaan nadenken, je moet voor je gaat beginnen vragen kunnen beantwoorden als wil ik strongly typed werken of niet, wil ik strongly typed queries kunnen bouwen of wil ik mijn queries nog steeds als een string kunnen opbouwen, wil ik een row set terug hebben of wil ik een collectie van classes terughebben, moet ik het gaan serializen om het vervolgens naar een webservice te kunnen gooien? Dat soort vragen zul je nog steeds van tevoren moeten beantwoorden en dat is moeilijk op te vangen in een tool tenzij je een checklist gaat automatiseren.'

Hoe goed is de SQL die door Microsoft gegenereerd wordt, versus de SQL die een goede ontwikkelaar kan schrijven?
Drapers: 'Wanneer we generiek statement genereren voor een update profiteren we van het feit dat we een veel betere beschrijving hebben van het entiteitmodel, zodat we veel slimmer statements kunnen schrijven. Vanuit dat oogpunt is het een grote verbetering. Als jij een automatische tracking aanzet voor changes en die wijzigingen weg laat schrijven, dan wordt daar een generiek update statement geschreven op basis van de entiteit zoals die bij ons bekend is. Kan dat efficiënter? Ja, dat kan efficiënter. Je zou dat in een stored procedure kunnen gooien die slimmer omgaat met vergelijkingen om te kijken wanneer iets gewijzigd is of niet, de complexiteit van het

Drapers: “Kan het efficiënter? Ja, dat kan efficiënter”

merendeel van de statements zit in de complexiteit van de where clause. Het zit niet zozeer in het insert of het update statement, maar in die gigantische where clauses waar je én moet kijken of je informatie anders is én moet compenseren voor null waarden. Daar krijg je een redelijk complexe where clause door die niet in alle gevallen performant is.’

Wat is het idee achter transaction promotion?
Drapers: ‘De reden dat we een aparte transactie api gebouwd hebben, is dat transacties op zich niet exclusief zijn voor databases. Wat we wilden bewerkstelligen, is dat je in .Net je eigen resource manager kunt bouwen, die transactief is. Het idee erachter is dat als je nu ander zaken hebt die in je transactie mee moeten doen we die allemaal kunnen enlisten bij dezelfde tranction manager. Vandaar dat we het transactiemodel eist gewijzigd hebben in 2.0. Je hebt een lokale transaction manager die echt draait in memory. Je kan de transactiemanager van de resource manager gebruiken, in de meeste gevallen een database, Oracle, IBM, SQL Sever, dat zijn uiteindelijk resource managers. Ze hebben echter ook hun eigen transaction manager want elke database heeft zijn eigen transaction manager, er is geen database die altijd met een externe transactiemanager werkt. Vervolgens kijkt de lokale transactie naar de externe transactiemanager. Dat wilden we in een api kunnen neerleggen en met een automatisch escalatiepad. Want wat we hadden uitgevonden van com+, of MTS of wat daarvoor lag: als je dat soort technieken aan gebruikers geeft dan begrijpen ze daar geen jota van. Als je kijkt naar MTS of com+ dan hebben 80% van de mensen die een transactie object gebruiken maar één resource manager. Het gevolg daarvan is dat je 20% van je performance weggooit, aan communicatie met de DTC-transactiemanager want dat is volkomen onnodig. Vandaar dat we dat transaction promotion hebben bedacht waarbij je dus zegt: ik heb een lokale transactie , die wordt gepromoot naar een databasetransactie, die wordt gepromoot naar een gedistribueerde transactie. Alleen niet op het moment dat de transactie begint, maar wanneer de resource aan je transactie wordt toegevoegd. Dat is de fundamentele wijziging. Alleen SQL Server 2005 kan dat op dit moment aan, ik weet dat onze vrienden bij Oracle er ook mee bezig zijn, want de meeste databases kunnen op het als hun werk enlisted is in een transactie de ownership van de transactie niet meer wijzigen. Op een gegeven zeggen ze: ik heb werk in de transactie zitten en ik weet wat de roots van de transactie zijn maar ik kan de roots van de transactie niet wijzigen, wat uiteindelijk is wat je dan wil. Want wanneer je zegt: er komt nu een resource manager bij, maakt het qua kosten helemaal niet uit of je nu de tweede transactiemanager die nu lokaal is of remote, want het toevoegen van een gedistribueerde transactie is toch wel duur. Maar het is ook een uitzondering, in de meeste gevallen. Dus dat is het model dat we gekozen hebben. Wat we daarmee gedaan hebben, is dat – en dat was in 2.0 al het geval – een sytem.transactie een ADO. Net transactie is. Die twee dingen zijn compatibel. Je kunt transacties beginnen in sytem.transactions en die vervolgens gebruiken en dat is equivalent aan het gebruik van een databasetransactie. Wat niet zo goed naar voren is gekomen, is dat dat een best practice zou moeten zijn voor gebruikers. Als je transacties programmeert dan wil je daarom altijd system.transactions gebruiken en niet meer de databasetransacties. Als je dan straks ándere transacties krijgt, zoals in Vista het filesysteem, de registry, al die objecten die transacted zijn, dan kun je die ook gaan gebruiken binnen je transactie. Het voorbereidend werk wat we voor Vista al gedaan hadden, is dat met de kernel transaction manager die nu in Vista zit, je filesysteem en je registry ook transacted objects zijn. Als je dus kijkt naar setup in Vista, dan is dat een heel verschil met XP. Setup in Vista, daar hoef je alleen maar te zeggen begin transaction, dan zet hij een begin transaction op het filesysteem en de registry. Gaat er iets fout tijdens die setup dan worden alle filesysteemwijzigingen en registrywijzigingen automatisch ongedaan gemaakt. Dat scheelt uiteindelijk een heleboel geld met betrekking tot het opruimen van machines. Dat soort dingen willen mensen ook in hun applicatie.’

Je kunt dus een atomic transaction tussen een webservice en een database doen.
Drapers: ‘De vraag is of je een atomic transaction wilt tussen een webservice

“De complexiteit van het merendeel van de statements zit in de complexiteit van de where clause”

Een feature team heeft een cumulatieve bug score die lager ligt dan de som van de individuele bug scores

en een database. Ik zou het persoonlijk niet op prijs stellen denk ik. Maar het kan wel. Tussen twee services kan ik me voorstellen dat er bepaalde dingen zijn die je per se onder een transactie zou willen uitvoeren en dat kan. Maar dat gaat nooit over het internet. Dat wil je niet. Ons motto in het transactieteam was: een gedistribueerde transactie gebruik je alleen als je niet anders kan, als al je andere opties niet werken dan kan je als laatste redmiddel nog een gedistribueerde transactie doen, het anti-availability middel noemden we het. Dat is wat een heleboel klanten niet begrijpen. Ik praat met een heleboel klanten die massieve investeringen doen in clustertechnologieën, en in high availability-technologie en vervolgens gedistribueerde transacties tussen cluster nodes gaan zitten doen. Dan ga je ze uitleggen: je begrijpt toch wel dat als er één cluster node niet meer is dat je gedistribueerde transactie absoluut niet meer gaat werken en dat je systeem daarom per definitie niet meer beschikbaar is. Dan kan de node wel op zijn, dan kun je claimen dat je node 99.999% uptime heeft, maar je applicatie heeft geen uptime. Daar verkijken heel veel mensen zich op.'

Wat doet ADM nu precies?
Drapers: 'Er zit een aantal heel interessante dingen in. Stel, je gebruikt een O/R-mappingtechnologie, maar je hebt je eigen class, een cache of een hash list of een dictionary, iets wat wel een relatie heeft met het entiteitsmodel maar geïnstantieerd wordt vanuit de applicatie. Het komt dus niet uit de database. In ADM kun je zeggen: dit is een data class en die verhoudt zich op deze manier tot de rest van het entiteitsmodel. Wat je daarmee krijgt is dat je een lokale resource class, een linked list of een dictionary of een hash-tabel in je ER-model op kunt nemen zodat het voor de applicatieprogrammeur één model wordt. Dat is uiteindelijk toch de reden waarom je dat soort dingen wil gaan gebruiken, omdat het voor die applicatieprogrammeur vele malen eenvoudiger wordt zijn applicatie te bouwen.'

Werken jullie bij het ontwikkelen van je eigen software nu agile?
Drapers: 'Ja, al werd dat in het begin niet door iedereen ondersteund, met name niet bij het management. Wij doen development sprints van vier weken, en twee weken integration sprints, dus in totaal zes weken. Bij de integratie wordt ook software van het team in India geïntegreerd. Development is wel verantwoordelijk voor het schrijven van zijn eigen testomgeving. 70% code coverage is verplicht anders komt de check in niet eens door de gate heen. We hebben dus een gated check in proces. We doen dan wel geen continue integratie maar we doen wel een test bij de check in. Afhankelijk van de resultaten van de test wordt geaccepteerd of afgewezen. Dan wordt hij geïntegreerd in de feature branch en die wordt uiteindelijk geïntegreerd in de business units branche. We hebben ook een concept dat heet bug jail. Iemand mag alleen maar zoveel bugs hebben anders moet hij stoppen met het feature werk. Dat is een puntensysteem: priority 0 moet hij meteen stoppen, als hij maar een priority 0 bug heeft is de rest geblokkeerd, priority 1 mag hij er twee van hebben, bij twee drie enzovoorts. Dat telt dan bij elkaar op en vormt een bug score. Een feature team heeft een cumulatieve bug score die lager ligt dan de som van de individuele bug scores. Als jij dus twee slechte collega's hebt kan het zijn dat het featureteam gestopt wordt. Het grote probleem dat wij bij Microsoft hadden was: je begint te ontwikkelen, een heleboel goede ideeën en vervolgens zijn de features er wel maar de kwaliteit is er niet. Wij hebben dat omgedraaid: je mag alleen aan je feature werken als je maar een beperkt aantal openstaande problemen hebt.
Omdat we daar meteen mee begonnen zijn, hebben we geen gigantische bug debts. Als je een bestaand product hebt en dan gaat wijzigingen dan zit je al snel aan 300 openstaande bugs of zo. Wij zijn daar in 2001 mee begonnen. Het grote voordeel is dat je bij de testorganisatie kunt laten zien: onze unit tests hebben zoveel procent van de probleem al gevangen, voordat het bij jullie komt. Het voordeel voor hun was dat ze zich met bepaalde scenariogebaseerde testen konden bezighouden. De unit tests worden automatisch met elke built en elke check in gedraaid en dan hebben we nog een set van testmachines die alle platform-combinaties doen, dat kun je alleen als je het automatiseert. We hebben daarbij trouwens veel plezier van de accessibility api's.'

Van welk deel van de nieuwe technologie zullen we nu het eerst echt profijt hebben?
Drapers: 'Ik heb geen idee van de adaptatiegraad in Nederland van webservices, maar iedere businessapplicatie heeft ergens een databaselaag zitten. Als je puur kijkt naar het aantal mensen waar dit toepasbaar voor is, dan is dat vele malen groter dan WCF, of WPF. Vanuit een businessapplicatie-oogpunt ligt ADO.net vooraan in de rij. Ik denk dat WPF daarna komt en de belangrijkst reden is dat op een dusdanige manier je applicatie development verandert. Ik denk dat over vijf jaar dat de enige manier zal zijn waarop nog Windows apps gebouwd worden, op basis van een WPF-achtige opzet waarbij lay-out apart gedefinieerd wordt van alle actions. De reden waarom dat heel snel zal gaan is met name dingen als Silverlight waar dat soort techniek ook onder zit, die gaan dat gewoon waanzinnig accelereren. Dat is een heel belangrijke kapstok waar heel veel aan opgehangen wordt en als communicatie-infrastructuur is WCF, maar is er is maar een beperkte groep mensen die dat soort werk doen. Hoeveel mensen bouwen webservices? Het zal toch een specialistische tak van sport blijven naar mijn idee. In ieder geval niet zo mainstream als een databasetool en algemene businessapplicaties.