De volgende versie van SQL Server – codenaam Yukon, officieuze uitlevering halverwege 2003 – zal met de geïntegreerde .Net clr runtime een belangrijke plaats in de .Net strategie van Microsoft innemen. Voor een database die iets meer dan tien jaar geleden niet eens bestond en nog niet lang een serieus te nemen concurrent van Oracle en DB2 is, een grote stap. Een interview over het ontwikkelen van een database en over de gevolgen van de clr runtime in de database.

.Net en de toekomst van SQL Server

Van page locking naar complexe datatypes

Gert Drapers, in een ver verleden nog auteur voor ons zusterblad Database Magazine, is sinds enige jaren als architect nauw betrokken bij het verder ontwikkelen van SQL Server. De oorspronkelijke engine was een oude Sybase engine, met page locking.

Drapers: 'Toen we begonnen, was het een contract tussen Microsoft, Ashton/Tate en Sybase om een pc-versie op de markt te brengen. Tot en met 6.5 was de main code base Sybase gebaseerd, zelfde page-structuren, zelfde internal structures, processen en locking structures et cetera. Het enige wat we hebben veranderd toen we naar NT gingen, is dat we hebben we de Sybase threading library eruit gegooid hebben en dat we OS threads ervoor gemaakt hebben. Daarna zijn we gaan poetsen en hebben geprobeerd een zo optimaal mogelijke engine te maken voor NT. De Sybase code op zich was geen slechte code, maar het was op geen enkele manier gefactored en dat betekende dat elk stukje code links en rechts van elkaar wist en wist hoe je structures en stages moet benaderen. Het was een wirwar van dependency's en het grootste probleem waar wij tegenaan liepen toen het development team ging groeien, was dat er niet genoeg mensen waren, die alle dependency's wisten. We konden dus met niet meer dan vijf man aan de engine werken want dat leidde meer tot problemen dan tot vooruitgang. Toen is ook vanuit Microsoft de focus komen te liggen op verbetering van de database engine, zodat die betrouwbaarder werd. Een aantal engineers die hun sporen in de database wereld al verdiend hadden, zeiden, oké, dan gaan we het in principe opnieuw bouwen. We hebben niet voor de strategie gekozen: gooi alles maar weg, we beginnen met een nieuw source control systeem, en begin maar met main en ga maar bouwen. We hebben een intern revolutie traject ingezet waarin we onszelf als eis stelden, dat we elke dag een draaiende engine hadden.'

Wereldlicht Drapers: 'We wilden een aantal dingen veranderen: we wilden van page level locking naar row level locking, we wilden van 2k pages naar grotere pages, dat werden uiteindelijk 8k pages. Van 2k naar 8k was niet echt een probleem, zoek en vervang 2k naar 8k en het was redelijk snel aan de gang. Maar locking was waanzinnig, want undo-behaviour in Sybase - omdat het page locking was - was gewoon het herinneren van het before image van die page. Als je undo zegt, zet je gewoon die oude page image terug. Met row level locking kan dat niet meer. Het betekent dat je logging verandert, dat betekent dat je back up restore gaat veranderen, dus de locking change was de meest intensieve verandering. Toen hebben we ook drie weken geen draaiend product gehad. Maar goed, zo zijn we van de van de 6.x code base naar 7.0 gekomen, aan het einde van 7 is de inschatting van de meeste engineers dat er tien à twaalf procent van de oude code nog steeds aanwezig was. Met SQL Server 2000 zijn er alleen nog wat main structures over van de Sybase code, en er werken nu zo'n 350 engineers aan de engine.'

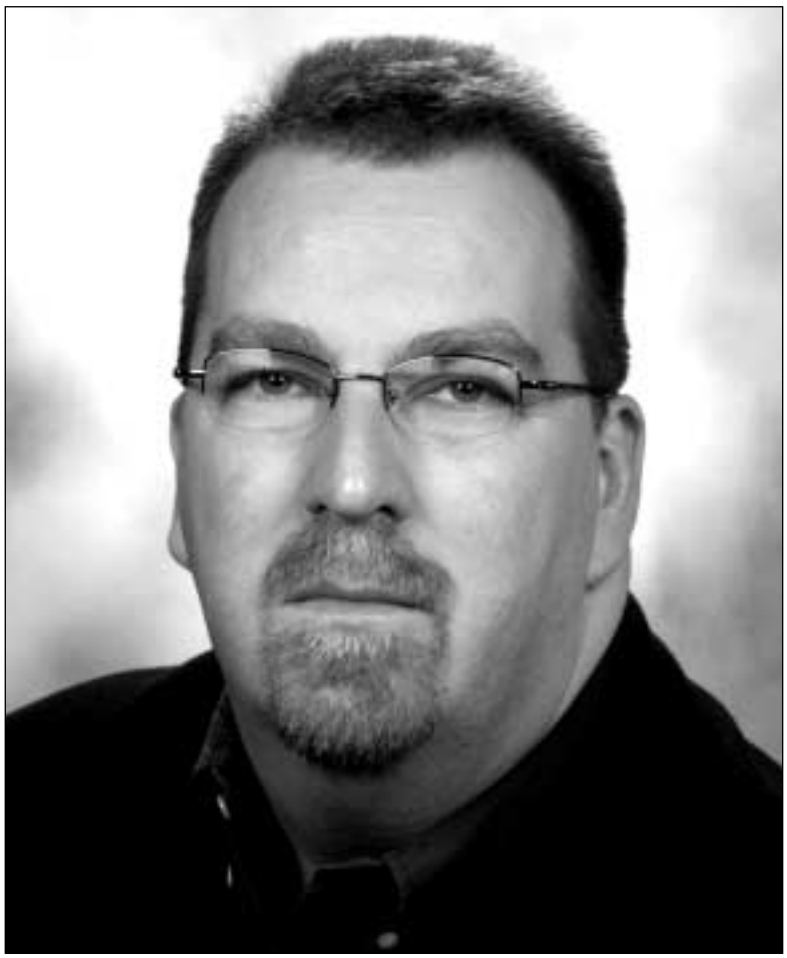
Nu gaat die engine weer veranderen om de clr (common language runtime, de .Net runtime) op te kunnen nemen, of is daar toen al op vooruit gelopen?

Drapers: 'In zekere zin wel, omdat we in 7.0 alles al opgedeeld hebben in aparte eenheden. De grote schei-

ding is de onderkant/bovenkant, storage engine/relational engine geweest. In de storage engine hebben we ook onze eigen subdivisies. Het originele design van de storage engine ging zelfs zover, dat de storage en de relational engine los te koppelen waren. Op dat moment – in 1996 - was onze doelstelling dat we de storage engine in het OS konden stoppen. De interface tussen storage en relational engine is nog een *ole db* style interface. Op dat moment was dat met het idee dat mensen daarop zelf konden pluggen en ISV's het als een datastore konden gebruiken. Dat heeft het wereldlicht nooit gezien maar die *factoring* die laat ons nu toe om dat soort wijzigingen te maken. Want de enige groep die echt het meeste betrokken is bij de hele clr implementatie is het team wat languages interpreteert, de *language and proces engine*. Dat is dezelfde groep als waar *stored procedures* en functies zitten. Logisch gezien hoort het ook in die groep terug en dat is de groep waar de code ook hoort. Er is een kleine uitval naar de *storage engine* voor wat betreft we moeten nu een index kunnen bouwen op basis van deze informatie maar die informatie krijgen we nog steeds van dezelfde provider in de relational engine.

> 'We konden dus met niet meer dan vijf man aan de engine werken'

Op *storage engine* niveau weten we niet dat dit een clr-object is. Hetzelfde geldt voor XML-data types. De enige entiteit die iets weet van XML data types, is de indexer. De refactoring die we in 7.0 gedaan hebben, heeft dit mogelijk gemaakt, in 6.x zou dat niet mogelijk zijn geweest. Dan waren we ook nooit zo ver gekomen als we nu waren met SQL Server. Het grote probleem van 6.5 en de vorige releases was: op zich kon het wel nog grotere databases aan, de query performance kon het redelijk aan, maar je kon je database niet onderhouden. Als ik een terabyte op 6.5 zou draaien, kon ik geen back up doen die in een redelijk tijdsbestek is te restoren. We gingen puur onderuit op manageability en met het operationeel houden van de database.'



Gert Drapers

De clr runtime in de database

De komst van de clr runtime in de database brengt nogal wat veranderingen met zich mee. Welke zijn dat zo al en wat zijn de consequenties voor dba's en ontwikkelaars?

Drapers: 'Ik denk dat twee aspecten heel erg belangrijk zijn. Ten eerst vanuit een hostingperspectief. Wanneer je zegt: ik heb nu zo'n runtime en die kan ik binnen een bestaand proces gaan opnemen en integreren. Op zich heel mooi, maar dat stelt een groot aantal eisen. Aan de ene kant aan de omgeving die de hosting omgeving neerlegt en aan de andere kant ook aan de runtime.

Op dit moment zijn er een viertal dingen die je kunt doen met de clr: je kunt procedures, functies, triggers en user-defined datatypes schrijven. Waarschijnlijk is de laatste de gevaarlijkste. Want een procedure, ach, mensen doen vandaag de dag al rare dingen. SQL is een waanzinnig mooie taal zolang het set-oriented is. Zodra je een string uit elkaar moet puzzelen, of mathematische calculaties moet doen dan wordt het al erg vervelend, soms onmogelijk en mensen gaan dan een uitweg zoeken. We hebben alle primitieven in de clr die zeggen van: ah, u gaat een uitstapje maken naar het OS? U heeft daar geen rechten toe. Dat kunnen we detecteren. *Code-access security* zal ervoor zorgen dat je dat soort dingen kunt blokkeren, er zijn bepaalde *namespaces* die gewoon niet zinvol zijn in de database. *Winforms* is een voorbeeld: het is niet echt zinvol om een procedure te schrijven die winforms aanroept. We zijn heel duidelijk gaan kijken: hoe kunnen we een omgeving creëren voor de programmeur zodat hij zich-

zelf niet in zijn voeten schiet maar toch de kracht heeft van .Net en hoe kan een administrator daar mee omgaan.'

Want die krijgt ineens hele andere dingen op zijn bord.

Drapers: 'Daar zit de grootste leercurve, dat een DBA te maken krijgt met een programmeer paradigma, waar hij niet aan gewend is. Dat wordt een uitdaging voor de dba's. Ook, en met name, hoe gaat hij dat echt in de hand houden zodat hij die code die in zijn database draait ook echt vertrouwt, want hij wordt erop aangesproken. De twee mechanismen die we daarvoor gebruiken zijn is dat code, zodra die daarvoor een referentie heeft in de database, ook daadwerkelijk leeft in de database, jij bouwt dus een stuk clr-code, een assembly en die assembly heeft de functionaliteit die je hebt en die wil je in SQL Server gaan brengen, dan wordt die assembly ook daadwerkelijk in de database opgeslagen en optioneel kunnen we ook de sourcecode erbij opslaan. Dat geeft in ieder geval het beheervoordeel dat wanneer jij je database back-upt, dan wordt dat ding ook geback-upt, het wordt ook meegedeployed. Wanneer je jouw database naar een andere site gaat deployen zit al die informatie er automatisch al bij. Het is de administrator of iemand in de administrator rol - of dat daadwerkelijk ook de administrator is, is een andere vraag – die bepaalt welke types assembly's gebruikt mogen worden. Default is dat je alleen in safe-mode assembly's geladen kunnen worden, dat betekent dat je geen OS-uitstapjes kunt maken, we laten het *sowieso* niet toe dat je threads creëert, dat je processen creëert of andere gerelateerde functie-calls, puur om onszelf als SQL-engine te beschermen tegen óf malicious code, óf in het algemeen tegen onbenullige procedures die geschreven zijn.

Het voordeel is dat op het moment dat die code in SQL is, met uitzondering van udt's (*user defined datatypes- red.*) dat functies en procedures en triggers eruitzien als een gewone TSQL procedure of functies, dus we kunnen gewoon zowel aggregate functions als table functions in die functies aan. Degene die SQL programmeert ziet niet dat een procedure is die in C# geschreven is of dat het een TSQL procedure is. Dat maakt het toegankelijker voor dat soort mensen. Maar er is een heel duidelijk punt, er is een leercurve voor dba's, wat moet ik ermee gaan doen, hoe kan ik beoordelen of die code daadwerkelijk wel te vertrouwen is. We geven ze daarvoor een aantal hulpmiddelen, dat wanneer ze de defaults volgen ze alleen safe environments draaien, dat is heel duidelijk het uitgangspunt, niet de andere kant wat we vroeger wel als uitgangspunt zagen.'

User defined datatypes

User defined datatypes is iets waar menig programmeur zich sterk toe aangetrokken zal voelen. Aan de andere kant roept het ook wel een tovenaarsleerling gevoel op.

Drapers: 'Het is een heel interessant traject voor programmeurs. Je krijgt automatisch de vraag, oké, ik begrijp dat ik nieuwe datatypes kan maken als een point datatype, of een nieuwe date/time, want wij hebben geen time-veld, en er zijn mensen die zich daar groen en geel aan ergeren, dus je zou nu aparte date en time types kunnen maken.

Dan komen er automatisch een aantal vragen naar boven: wat gebeurt er met indexen, wat met selectivity, hoe kan een *where clause* in mijn T-SQL begrijpen wat die operator betekent voor dat datatype. Dat zijn allemaal dingen, de implementaties van procedures en functies en triggers, dat is allemaal vrij triviale code. Maar voor een user defined datatype komen allemaal dingen om de hoek kijken als: hoe ga je met *nullability* om, de meeste datatypes in een programmeertaal vinden *null* een vervelende iets om mee te werken, maar goed, we hebben een *three state logic*. Alle implementa-

> 'We hebben onszelf als eis gesteld dat we elke dag een draaiende engine hadden'

ties over hoe ga ik statistiek informatie teruggeven aan de query-processor, over *uniqueness*, hoe kunnen we die informatie gebruiken binnen een index, hoe kan ik een index bouwen op basis van een *userdefined* datatype. Dat betekent dat de implementatie van *user defined datatypes* gegeven het framework of een implementatie door middel van een set van interfaces en een *base class*, die je de mogelijkheid geeft om al dat soort dingen te beïnvloeden en op die manier zijn we in staat om *user defined datatypes* te schrijven die goed overweg kunnen met de *query processor* en met indexen en met *where-clauses* en met de operators die je gaat implementeren. Echter, een aantal van die interfaces zijn optioneel. De kans bestaat dus, dat mensen de korte weg nemen en zeggen: we gaan een *user defined datatype* maken, we doen de rest maar niet, we hebben het nu niet nodig. Later komen we in een productieomgeving, indexen worden ineens wel belangrijk, en we komen erachter dat er geen performance is.

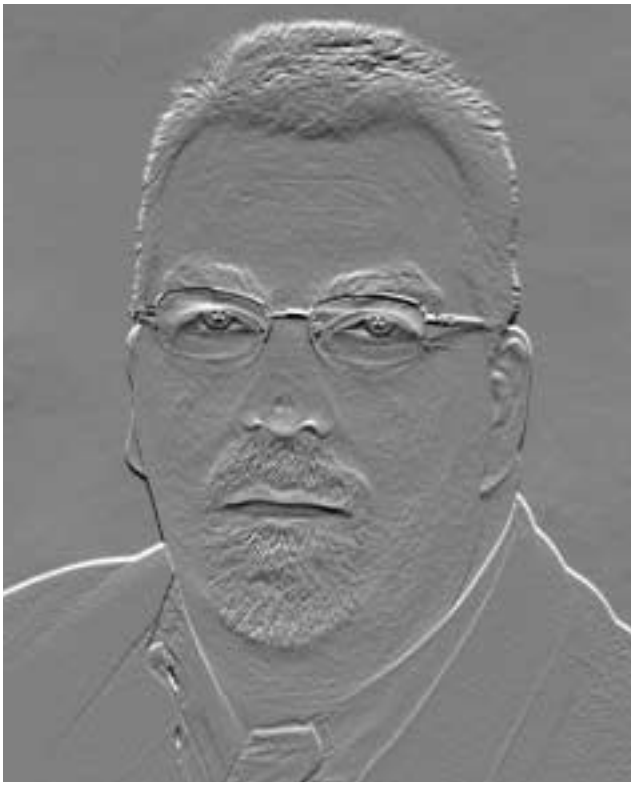
> ‘Er werken nu aan de engine zo’n 350 engineers’

Op dit moment zijn we nog in debat over de vraag of we het nu wel of niet verplicht moeten stellen. De doelstelling is een beetje dat we tijdens de eerste bèta van ons product de losse implementatie doen. Dan gaan evalueren wat er gebeurt. Wat krijgen we aan feedback terug van een selecte groep klanten die daarmee gaan werken. Daarna gaan we gewoon kijken: moeten we dit nu wel of niet gaan doen?’

OBJECTDATABASE PURISTEN Drapers: ‘Er is een categorie datatypes die wij graag geschreven zien worden. Dan is er ook een andere groep, object-database puristen noem ik het maar, die zeggen: “oh, dan kan ik het datatype ‘klant’ maken, het type heeft een naam en een adres etc. en die ga ik in de database stoppen”. Op dit moment is dat niet onze directe doelstelling maar we zijn er wel heel druk mee aan het testen want we weten dat klanten dit gaan doen, en we kunnen het ze ook niet verbieden, want het is gewoon een *structured datatype* wat ze aan het bouwen zijn. Het voordeel voor de klant is dat als hij een select van de database doet, en hij gebruikt .Net, dat hij niet een row terug krijgt, maar echt een instantiëring van zijn klasse. Daar zit een stukje interessante techniek achter waar we het niet veel over hebben, dat is: je definieert een datatype in de database, maakt niet uit wat het is, en je bent een client, en jij doet een select van die table waar die column in zit, hoe ziet dat eruit? Jij hebt dat datatype niet, die klasse ken jij niet, hoe gaat dat werken?

Dat is eigenlijk hetzelfde principe als bij een webservice. We gaan daar een aantal verschillende opties bieden. We hebben natuurlijk te maken met data access technologie die vandaag de dag al bestaat. Maar je kunt ook een volgende stap maken, afhankelijk van *security* en *trust relationships*. Je kunt vragen, oké, als jij dat datatype niet hebt, stuur dat datatype dan voordat je de result set hebt naar mij toe in dezelfde string. Dan kom je in het downloadable gebeuren, met dezelfde problemen als internet security. Daar zullen dezelfde *code access rules* gevolgd moeten worden, want de hele doelstelling is om daar een programmeermodel te creëren wat *middle tier* hetzelfde is als de database, als object client. Dan kun je vervolgens een cast doen en zeggen: dit was een adres, dus cast het maar naar een adres en nu kan ik er wat meer interessants mee doen. Of je kunt zeggen, oké, ik heb een proxy-implementatie, zoals mijn webservice, en ik zeg, ik ga hem gebruiken en dan wist je van tevoren al dat je die informatie ging krijgen.’

DOWNLOADABLE Drapers: ‘Die drie niveaus geven je in .Net in ieder geval de vrijheid om de verschillende applicaties die daarvan gebruik gaan maken te kunnen bouwen en het ook voor een gebruiker interessant te maken en te zeggen: oké, je krijgt nu een object. Anders zit hij nog steeds met hetzelfde paradigma als hij vandaag al heeft: kolommetjes in een *row*. En jij moet zeg-



gen wat het is en het enige wat wij jou vertellen is dat het een string of een integer is, maar de hele semantiek valt weg. Het andere voordeel ervan is, dat die klasse die dat datatype implementeert meer *methods* kan bevatten dan SQL Server gebruikt. Het voordeel daarvan is weer, dat waneer dat datatype of als proxy, of downloadable op die client komt, dat die gewoon kan zeggen wanneer het een order line zou zijn, doe daar iets interessants mee, bedenk een method die interessant is voor data objects, en die komt gewoon mee. Object-oriented database mensen vinden dat een waanzinnig goed idee. Wij zijn er nog een beetje voorzichtig mee, en de reden daarvoor is puur vanuit een database-achtergrond geredeneerd: technisch gezien werkt het, het werkt vandaag al, maar is het voor de database het beste om te doen, vanuit de database engine gedacht. Puur over het feit van query, performance, selectivity van indexen, dat is nog een traject waar een heleboel mensen van zeggen: dat is waanzinnig, want wat oo-databases niet hebben opgelost is het probleem, hoe krijg ik dat als object nu ook op mijn client. Daarop gingen zij allemaal nat want de meeste oo-databases die de markt ooit heeft gezien, die waren echt oo in de database maar het programmeermodel was óf volledig *proprietary* of nog steeds geen object op de client. Dat zou nu dus heel goed kunnen, maar dat is zo'n beetje waar we nog in de afwachtende houding zijn. Ja, het kan en het werkt maar we zijn er heel voorzichtig mee en het is niet zoals we het promoten op dit moment.'

Het gevaar ervan lijkt me dat je op deze manier heel gemakkelijk iets kunt programmeren wat qua performance voor de database verschrikkelijk is.

Drapers: 'Het performance traject hebben we redelijk goed in de vingers, wat betreft de basis clr data-types. We weten hoe je een integer of een float moet indexen, dus ten aanzien van de basis datatypes hebben we dat goed in de vingers en kunnen we ook datatypes schrijven die daar goed mee om kunnen gaan. Maar als je inderdaad complexe datatypes gaat bouwen dan wordt dat een heel ander verhaal. Ik zie dat als iets wat later allemaal een stuk beter zal worden, metadata van de clr gaan ons daar een heel eind mee op weg helpen. Maar dat is ook een leertraject wat wij in moeten, want dat hebben mensen nooit gekund. Ik denk dat de eerste focus van mensen ook is: waarom zou ik dit moeten gebruiken?

Eén van de andere dingen die het clr-team moet wijzigen: SQL maakt gebruik van *fibres*, ik noem het doe-het-zelf *threads*, het is in principe een *lightweight thread* die je zelf scheduled. SQL Server kan daar erg goed mee omgaan, en haalt daar ook een heleboel performancewinst uit, voor de clr is dat nieuw want die gebruikt op dit moment alleen *operating system threads*. Dat is ook een wijziging waarin zij mee moeten, ten aanzien van: hé, je zult wel in een scheduler moeten kunnen draaien, zoals jouw host zegt dat dat moet werken. Hetzelfde geldt ten aanzien van memory: SQL Server controleert *memory grants* van elke *working thread*. Die heeft gewoon een grote pool aan memory. Het enige waar hij naar luistert is het OS dat zegt; er zijn andere mensen die memory nodig hebben, dan gaat hij memory opgeven. Maar we willen dat alles dat in ons proces draait, memory vraagt aan de memory manager in SQL Server. Op die manier kun je beter partitionering en pinning doen. Met name op een 16 en 32 way machine wil jij dat je memory grant ook gepind worden aan bepaalde processoren zodat het over dezelfde lus gaat en dezelfde L1 en L2 cache uitkomt. In dat soort trucjes moet de clr kunnen meespelen als je een echt optimale omgeving wilt creëren. Dat zijn dus dingen waar ze druk mee bezig zijn voor de volgende versie.'

> 'We zijn er wel heel druk mee aan het testen want we weten dat klanten dit gaan doen, en we kunnen het ze ook niet verbieden'

JAVA-INTEROP

Technisch gezien zit .Net en de toekomstige SQL server versie heel goed in elkaar, hebben een zeer groot potentieel. Aan de andere kant zie je dat mensen om heel andere rede-

nen vaak voor andere producten kiezen: omdat ze zich niet willen committeren aan het Windows platform.

Drapers: 'Wat je nu ziet gebeuren – en dat verrast ons ook min of meer – is dat het geen of-vraag gaat worden, klanten hebben gewoon èn èn. Of ze willen of niet, het bestaat gewoon: legacy, mergers, acquisities, verschillende eilandjes binnen een bedrijf. Divisies zijn autonoom. Ze hebben een eigen budget en gaan gewoon verschillende kanten op. Er is geen bedrijf van enige omvang dat dit probleem niet heeft. Ik ben de laatste drie maanden heel druk bezig geweest met allerlei vormen van Java en .Net interop(erabiliteit, DdM). Men wil heel graag dat die dingen met elkaar kunnen samenwerken. Die divisie vindt Java wel heel mooi en die divisie niet. Hoe kunnen we die nou laten samenwerken? Er is hetzelfde probleem met mainframes. Hoe lang proberen we al niet mainframes te vervangen? Als iemand eerlijk is, dan zegt hij: als je goed kijkt neemt de hoeveelheid mainframes weer toe, want mainframes hebben een aantal zeer goede eigenschappen.

> 'Als je inderdaad complexe datatypes gaat bouwen dan wordt dat een heel ander verhaal'

Toen zijn we daar eens naar gaan kijken, we waren met IBM aan het praten en zeiden dat we graag op apiniveau wilden praten. Dat was de eerste discussie die ik had, over transacties, in november 2001. We hebben al een overeenkomst dat de twee partijen samen mogen werken. Voordat iemand van de technische hoek met een technische man van IBM praat is er een legertje advocaten dat eerst gaat bepalen waar we het wel en niet over mogen hebben. Als die klaar zijn dan mogen wij in een kamertje met elkaar praten. Meestal kennen we die mannen toch al, maar goed... De database wereld is heel klein, de hoeveelheid database mensen binnen IBM is ook niet heel erg groot meer, en er zitten inmiddels een heleboel IBM database mensen bij ons, dus ook op die manier kennen we elkaar al.'

Eigen agenda's Drapers: 'Uiteindelijk mochten we dan met elkaar gaan praten. Het eerste wat we kozen was: hoe kunnen we een Websphere interop tot stand brengen met com+? Dat kan, maar vandaag de dag moet je c-code schrijven, dat zouden we gemakkelijker kunnen maken. Maar de mogelijkheden blijven beperkt. Het grootste bezwaar was dat het toch applicatie programmering vereist. Het andere probleem is: dit gaat het cross-platform probleem niet oplossen. Dus wij kwamen tot de conclusie, dat we dat transactional rpc style interop moeten kunnen creëren, willen we dit ooit op kunnen lossen – wat er overigens nooit van gekomen is. Uiteindelijk zijn we met een aantal voorstellen gekomen op basis van XML-messages zodat we acid based transacties konden doen en business transacties. De volgende webservices enhancements versie zal dan ook een infrastructuur versie bevatten, zoals we nu van security en routing en attachment hebben voor transacties, in ieder geval voor acid transacties.'

'Wat het werken met IBM en BEA betreft, moet ik zeggen: natuurlijk heeft elk bedrijf zijn eigen agenda erachter zitten maar toch is er afgesproken, dat we dit gaan doen. Want iedereen is er eindelijk achter dat we het alleen op het protocolniveau kunnen gaan redden. XML en webservices blijken genoeg fundamenten te bieden om een reële implementatie te kunnen doen waar klanten ook iets aan hebben. Ik denk dat iedereen zich dat voldoende realiseert. We gaan toch meer een message based paradigma in. Naar mijn idee is het hype stadium echt wel voorbij maar voordat het mainstream zal zijn, moeten we nog zeker nog anderhalf, twee jaar wachten. Onlangs hebben we nog gediscussieerd over de vraag of dat hele standaardisatie proces niet te vertragend is. Nu ja, standaardisatie vertraagt altijd, maar het is ook een heel groot goed.'

Dré de Man