De opening van de Nederlandse Dev Days begon met fraaie show door twee dames die zonder vangnet acrobatische toeren aan een touw verrichtten, daarbij vocaal begeleid door een derde. Dat paste goed bij de nieuwe Windows-versie die op audiovisueel gebied nieuwe grote hoogten moet gaan bereiken. Ook het klimmen zonder vangnet leek een aardige metafoor: het in een zo vroeg stadium tonen van de toekomstige Windowsversie is natuurlijk een redelijk groot risico.

Interview

De lange mars naar Longhorn

Het belang van de wet van Moore voor de ontwikkelaar

Longhorn zal volgens Microsoft de belangrijkste nieuwe versie sinds Windows 95 worden. Voor Windows-ontwikkelaars was het dan ook bijzonder interessant een preview te krijgen van hun toekomst. In die toekomst leek vooral weer een belangrijke rol weggelegd voor de client. In het licht van de versterking van de concurrentiepositie van Microsoft is dat misschien redelijk voor de hand liggend. Belangrijkste onderdelen van Longhorn: het nieuw storage subsysteem (codenaam 'WinFS') dat werkt met een in het OS geïntegreerde database en dat een gemakkelijke en een eenduidige wijze van gegevenstoegang voor applicaties biedt, Avalon, een systeem dat de GUI een generatie verder brengt en dat gebruik gaat maken van XAML en SVG (zie kader) en tenslotte Indigo, de communicatie-infrastructuur (nu de verantwoordelijkheid van één team) die alle communicatie van p2p seriële verbindingen tot webservices onder één paraplu brengt. Een aparte vermelding verdient verder nog het Business Solutions Framework (zie kader). Dré de Man sprak met developer-evangelist Nigel Watling over de belangrijkste veranderingen rond Longhorn en de consequenties voor ontwikkelaars.

> 'Geef me VAX/VMS terug, een groen scherm met forms was beter dan dit'

DÉJÀ VU *Als je kijkt naar Longhorn, dan krijg je een beetje een déjà vu of 'back to the future' gevoel: Het object file system Caïro dat al in 1992 (!) aangekondigd werd met als releasejaar 1994, komt er nu eindelijk aan en de rich client komt terug, rijker dan ooit zelfs.*

Watling: 'Het succes van webapplicaties komt voornamelijk neer op gemak van deployment en bereik. Iedereen met een machine op elk willekeurig OS en met een browser, kan via het web naar een applicatie toe gaan. Dat is het grote voordeel van een webapplicatie. Het andere voordeel is, dat je ieder OS, iedere hardware en ieder tool kunt gebruiken om een webapplicatie te schrijven en het is nog simpel ook. Er is echter ook een aantal beperkingen aan webapplicaties verbonden: de user experience van een browser applicatie is vrij simpel, het beheren van de state is complex, bijvoorbeeld. Wij proberen vanuit ons standpunt het beste van beide werelden te bereiken, door een heel erg rijke user experience te bieden en het mensen mogelijk te maken om fantastische applicaties te krijgen die er geweldig uitzien en zeer gemakkelijk te gebruiken zijn. Want veel bedrijven hebben deployment in hun bedrijf meegemaakt, en daarna webapplicaties, en ze krijgen veel kritische reacties van gebruikers die zeggen: "Deze interface is waardeloos, geef me VAX/VMS terug, een groen scherm met forms was beter dan dit."

Hoe werkt dat concreet?

Watling: 'Je klikt op publish vanaf de ontwikkelomgeving en je kunt het naar een server publiceren. Dan kan de user het vanaf de server draaien, of lokaal en het

De openingsshow van de Nederlandse editie van de Microsoft Dev Days, afgelopen januari in Den Haag

vanaf de server installeren, en natuurlijk kun je dit ook met verschillende security beperkingen doen. Je kunt het binnen een sandbox draaien, en je hebt al de voordelen van het draaien via het web plus de toegevoegde flexibiliteit om het te deployen. Het kan dan bijvoorbeeld schrijven naar de HD, of om de registry of wat dan ook te gebruiken.

Het blijft evenwel onder de controle van de sys admin. Die kan dus naar de applicatie kijken en zien wat hij doet. Als de sys admin denkt dat die applicatie geschikt is, kan hij er een signature aan geven, en via een enterprise of group policy door de organisatie verspreiden en het naar een webserver kopiëren. Iedere gebruiker die erop klikt, deployt het naar de hard-disk en draait het lokaal of, mocht er dan later een nieuwere versie zijn, dan kan de update ook geautomatiseerd verlopen: je verhoogt het versienummer, je deployt het opnieuw en dan zal de applicatie checken of er een nieuwe versie is en die vervolgens downloaden.'

Gebruik je daarbij dan de auto-update functie?

Watling: 'Zelfs beter dan dat, stel je voor dat een gebruiker een nieuwe versie gedownload heeft om de een of andere reden terug wil naar de vorige versie, dan staat de eerdere versie nog steeds op de harddisk. Via add remove programs kan hij terugkeren naar de oude. Dat is ideaal wanneer je in een situatie bent waarin je niet online bent, en de oude versie niet kunt downloaden. Het is dus het beste van twee werelden: zeer rijke gebruikerservaring, én gemak van deployment, én controle, en dat alles binnen een zeer goed gedefinieerde security-context die je kunt manipuleren. De sys admin kan precies bepalen wat de applicatie al dan niet mag doen, gebruikmakend van het intrinsieke security-mechanisme van het .NET framework. Dat mechanisme bijvoorbeeld al dan niet toestaan dat een bepaalde applicatie schrijft naar die directory. Longhorn gaat nog een stap verder. Tegen de tijd dat Longhorn er is, zal

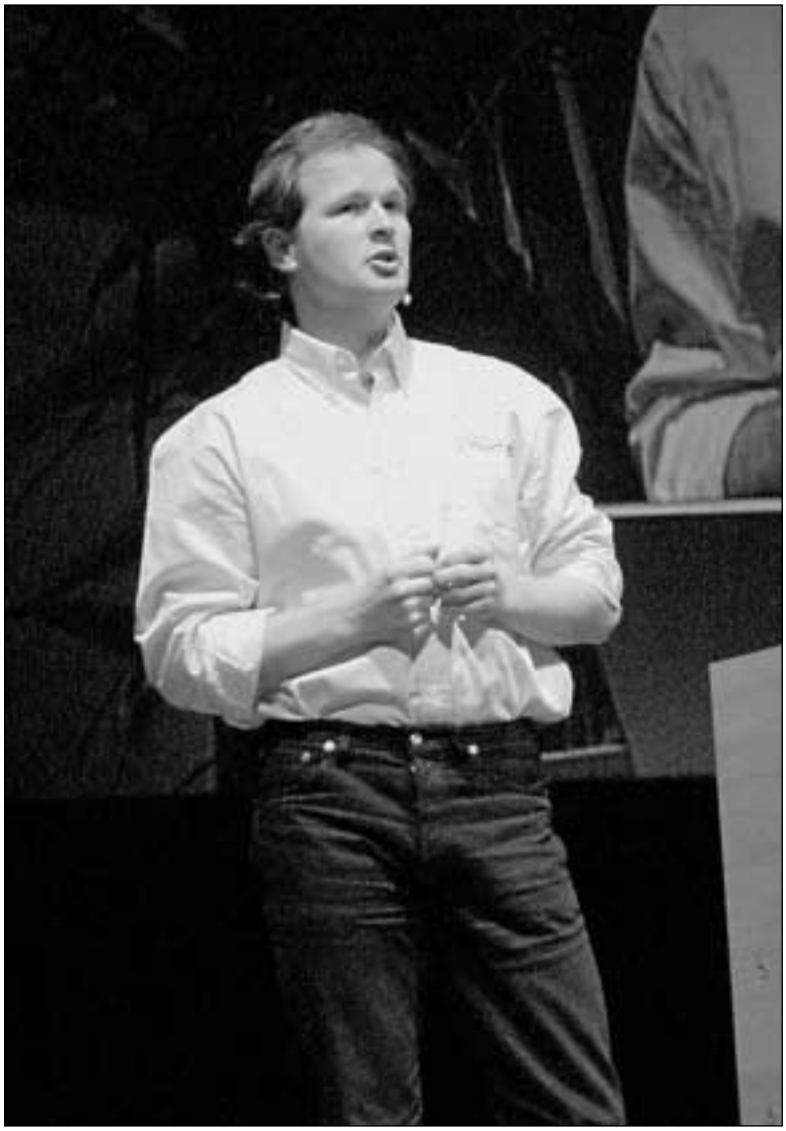
> Nu Moore's law de talen ingehaald heeft is het belangrijker dat we productief zijn

Advertentie

iedereen die 64 MB grafische kaart hebben. Het heeft dus zin om het uit die hardware te halen wat je eruit kunt halen. Je kunt wel praten over thin clients, maar verreweg de meeste mensen gebruiken een pc, er zijn tablet pc's en andere vormfactoren, maar de pc is het werkpaard van de IT. Als dat zo is, kun je beter in plaats van hem te gebruiken als een domme doos gebruik maken van alle kracht die erin steekt, en dat is wat wij willen doen.'

Minder code *Wanneer je Visual Studio Whidbey vergelijkt eerder versies dan zijn veel dingen veel gemakkelijker geworden, zeker in ASP .NET.*

Watling: 'ASP.Net 2.0 is een grote release. Er wordt gesproken over 70% minder code, en er was al niet zo veel code, dus het is een heel grote vooruitgang. We hebben gewoon gekeken naar websites die klanten bouwden en hebben geprobeerd de functionaliteit die mensen schrijven ter beschikking te stellen. Iedereen schrijft een inlogscherm, want al die inlogschermen zijn hetzelfde. Je moet een paar dingen invullen, er staat wat je moet doen als je je password vergeet. Stel het gewoon ter beschikking binnen ASP.NET, zodat je heel snel productief kunt worden, en zorg voor personalisatie en customisation filters, zodat de gebruiker die building block diensten gewoon kan hergebruiken, en controls op de forms kan plaatsen, zodat je websites heel veel sneller draaiende hebt en al de voordelen krijgt van de integratie binnen het platform. Productiviteit is ons grote voordeel, en we moeten gewoon op deze kracht verder bouwen.'



XAML

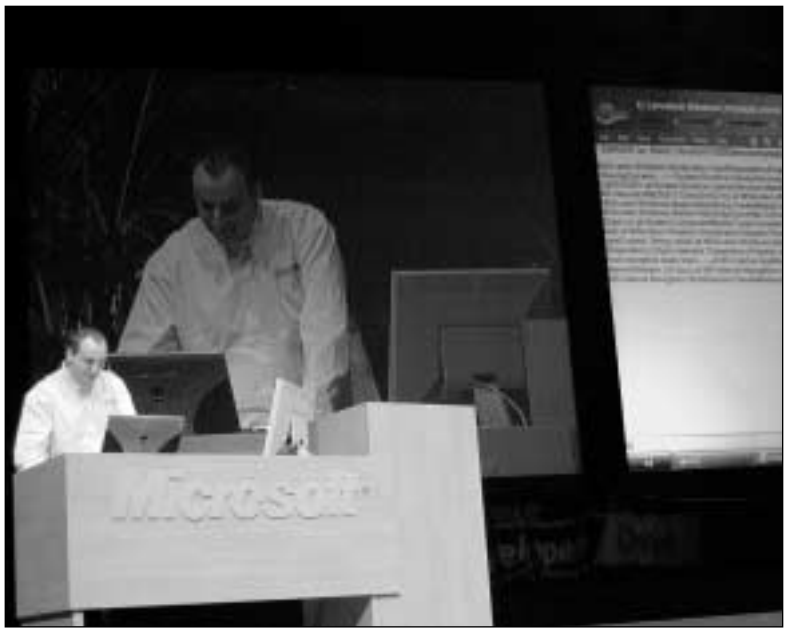
Het nieuwe presentatie subsysteem van Longhorn "Avalon," is in hoge mate gebaseerd op XML, of beter: op een XML-subtaal die Extensible Application Markup Language, genoemd wordt. XAML is trouwens een open standaard. Het schrijven van een Avalon-applicatie begint met het schrijven van de lay-out of UI elementen, iets wat naar verwachting het werk zal worden van gespecialiseerde GUI ontwikkelaars, vergelijkbaar met webdesigners. De functionaliteit van de applicatie wordt in procedurele code geschreven, meestal in code behind files. De compiler genereert code op basis van de XAML daarbij gebruik makend van een equivalent objectmodel. Avalon-gebaseerde applicaties kunnen óf in een desktop window óf in een browser draaien. Het systeem maakt ook gebruik van SVG (zie Software Release Magazine 2002 nr. 3). De voordelen en mogelijkheden van het nieuwe systeem zullen waarschijnlijk pas echt tot hun recht komen met toekomstige displays. Op dit moment bestaan er al displays met een vier keer hogere scherpte dan tot nu gebruikelijk is. De verwachting is dat binnen een paar jaar 300 dpi-modellen de standaard zullen gaan worden.

Er zijn nog steeds verschillen tussen C# en VB.Net. Ik heb echter het gevoel dat .Net aanvankelijk meer om C# draaide, terwijl nu de ondersteuning van VB sterker lijkt te worden.

Watling: 'Het is waar dat er een zekere evolutie is. Sommige mensen denken dat het framework C# is, en dat is niet waar. Ik denk dat dat misverstand ontstaan is, doordat per definitie de meerderheid van ontwikkelaars binnen Microsoft een C en C++ achtergrond hebben, zodat hun natuurlijke taal C# is. De meerderheid van de ontwikkelaars is Visual Basic-ontwikkelaar. Maar de keuze voor een taal is cultureel bepaald, het is geen technische keuze. Je hebt toegang tot precies dezelfde class-libraries, je hebt vrijwel dezelfde functionaliteit, met een paar verschillen. Je hebt generics in VB en C#, er is OO in allebei, ze hebben meer gemeen dan dat ze verschillen. De verschillen tussen de twee talen komen neer op syntax en voor mij is dat iets wat je gewoon leert. Het belangrijkste met het .NET framework is niet de taal die je kiest maar het leren van het framework, van de base class library's.

De originele intentie bij het ontwikkelen van VB .Net was om absoluut 100% compatibel te zijn met Visual Basic 6. Terwijl het evolueerde werd het steeds moeilijker om aan die compatibiliteit vast te houden. Gewoon omdat die taal al heel lang bestaat, het was de eerste taal die Bill Gates ontwierp. Je hebt al die bagage terwijl C# een splinternieuwe taal is. Die compatibiliteitseis zorgde ervoor dat de ontwikkeling van VB.Net vergeleken met C# veel langzamer ging. Er ontstond intern een groot debat over de vraag of je die compatibiliteit moest opgeven om een echt goede nieuwe taal te krijgen, of om iedereen de mogelijkheid te geven om aan de oude dingen vast te houden. Na overleg met consultants en klanten is de beslissing genomen VB. Net zo krachtig en functioneel als C# te maken, maar met een andere syntax. Ook zijn sommige features anders, vanwege de manier waarop Visual Basic-ontwikkelaars werken. Het klassieke voorbeeld daarvan is edit & continue. C, C++ en C# developers schrijven code, ze bouwen, ze krijgen hun errors, debuggen, en bouwen opnieuw. Visual Basic-ontwikkelaars - of althans een belangrijk deel van hen - werken in edit & continue: ze schrijven code, run, schrijven verder en doen weer run, en gaan daar steeds mee door. Dat is een heel andere manier van werken. Vanwege tijdtekort was edit en continue niet in versie 1 en 1.1 van het framework beschikbaar. Veel mensen waren daar heel erg boos om, omdat ze eraan gewend zijn. Het was echt belangrijk om edit & continue terug in Visual Basic te krijgen omdat een groot deel van onze klanten het eisten. Zo werken ze nu eenmaal. Edit & continue is niet zomaar een taal-feature, het moest in de CLR ingebakken worden en die functionaliteit wordt aan alle talen ter beschikking gesteld. Toch zit het niet in C# en in Whidbey, omdat C# programmeurs zo niet werken. Het is zinvol voor de development van C# zich te concentreren op andere features, die ze wel belangrijk vinden. Het doel is om iedereen dezelfde toegang tot het frame en de class-libraries te geven, maar je tevens te richten op de specifieke community van developers en die zaken aan te bieden waarnaar zij op zoek zijn.

> Waarom zou een middle-tier ontwikkelaar zich zorgen hoeven te maken over SQL-statements?



Lester Madden liet bij het demonstreren van Avalon zien, dat zelfs dyslectici XAML-code kunnen schrijven

RIJKER EN COMPLEXER *In het streven de ontwikkelaars tegemoet te komen zit iets tegenstrijdigs: aan de ene kant worden dingen steeds gemakkelijker en eenvoudiger, terwijl aan de andere kant het abstractieniveau toeneemt.*

Watling: 'Naarmate de functionaliteit van de programma´s die we schrijven complexer wordt, wordt het zaak die complexiteit de baas te worden. Iedereen is het erover eens dat webservices zeer zinvol zijn. Wil je echter webservices doen met transacties en security en reliable messaging, is het heel erg moeilijk. De mensen die daarin slagen zijn echte techies, ze moeten weten wat er heen en weer gestuurd wordt en moeten echt goed zijn in het daarmee werken. Ze moeten alles zelf doen en code typen totdat hun handen bloeden. Maar op het moment dat Indigo af is, zullen specificaties als WS-security en WS-transactions klaar zijn, en natuurlijk reliable messaging. Je hoef er dan niet meer over na te denken, je denkt ook niet meer over de specificatie van DCOM. Anders ben je een triest figuur, en dat zal net zo gelden voor webservices security in de toekomst. Het is de *law of leaky abstractions*: je kunt op een bepaald abstractieniveau werken, maar omdat het een abstractie is, zijn er altijd punten waar dat niet opgaat. Dan moet je terugvallen op het niveau daaronder om iets voor elkaar te krijgen. Dus het is continue strijd, de applicaties worden rijker en complexer, en dan komen de tools en de frameworks er achteraan om het weer simpeler te maken. Het Microsoft business framework maakt gebruik van object spaces, om de developer te ontkoppelen van de implementatie van SQL. Weer een poging om het gemakkelijker te maken voor de ontwikkelaar.

Maar het is een belangrijk punt: hoe meer de IT-industrie rijpt, des te abstracter wordt het. Je kiest niet meer voor Assembler, in de zin van "dat is mijn favoriete programmeertaal". Dat is ook niet een erg productieve programmeertaal, hoewel je heel dicht op de hardware zit. Daarom gingen mensen over op C, nu gebruiken mensen C niet meer, want we gaan alweer naar een ander abs-

tractieniveau, C#, dat misschien zes, zeven jaar geleden als te langzaam zou zijn gezien. Nu *Moore´s law* de talen ingehaald heeft is het belangrijker dat we productief zijn, en dat de hardware dat kan oplossen. Een jaar of vijf van concentreren op de problemen van OO-programmeren, talen en zaken als garbage collection, en focussen op hoe dat soort dingen werken, heeft ons de mogelijkheid gegeven de performance daar te krijgen waar we hem nodig hebben. De programmeur kan Assembler achter zich laten, net als C++ , en kan een simpeler abstractere programmeertaal gebruiken met een rijke set van classes, die hem in staat stelt productiever te zijn en met minder complexiteit om te gaan.'

DATABASE IN MEMORY *We spraken net over Moore´s law en de gevolgen daarvan. Persoonlijk denk ik dat het grote geheugengebruik dat 64 bit mogelijk zal maken een nog groter effect zal hebben. Ik voorzie daardoor grote veranderingen: je kunt dan hele databases in memory plaatsen, waardoor je heel veel meer met je code doen: abstraheren van SQL, complexe datatypes gebruiken. In feite is het relationele model dan niet meer van belang.*

Watling: 'Dat is zeker waar. Wanneer je grote hoeveelheden geheugen tot je beschikking hebt, kun je dingen doen die daarvoor onmogelijk waren - interessant dat je daarover begint. Jim Gray, de man die SQL uitvond, heeft daarover interessante dingen gezegd. Hij is een Microsoft *distinguished engineer* en praat over Moore's Law en datastorage en ja, hij praat over de mogelijkheden die je hebt wanneer je de hele database in memory hebt. Dan wordt het een keuze wanneer je de data persist, en de database is alleen maar een back-up, omdat je alleen maar met memory te maken hebt de hele tijd. Het moment van wegschrijven naar de schijf is ook een keuze. Wat mijn voorspelling is en wat Jim Gray zei, en mij stimuleerde, was dat je in de eerste plaats één grote database in memory zult hebben, dan heb je een strategie voor hoe vaak je de cache naar de disk schrijft, en dan heb je nieuwe dingen zoals *always add databases*, waarbij je in plaats van in memory een record te vervangen, iets toevoegt. In plaats van updates doe je dus steeds inserts, en wanneer je het dan naar disk persist, heb je misschien andere threads, die de inserts daartegen oplossen. Het is allemaal afhankelijk van de keuzes die je maakt over concurrency en lifespan van data, er is een groot potentieel om dat te doen. Het is een opwindend gebied waar we naartoe gaan.'

Nigel Watling, developer-evangelist bij Microsoft sprak dit jaar op de Nederlandse Dev Days

ABSTRACTIE *Bij Object Spaces zie je dat de ontwikkelaar verre van SQL wordt gehouden. Misschien minder revolutionair, maar toch een interessante ontwikkeling?*

Watling: 'Op dit moment heb je ontwikkelaars die business logica bouwen en daarnaast heb je meestal de data-access jongens, die zich erop concentreren wat de meest performante manier is om data van en naar de database te krijgen. Je bent dan erg beperkt door relationele grenzen, je gebruikt T-SQL, daar heb je dus je mensen voor. Eén van de belangrijkste dingen in de ontwikkelingen met betrekking tot computers op dit moment is dat we drie verschillende manieren hebben om naar data te kijken. We switchen tussen verschillende formaten en modellen en omdat verschillende modellen verschillende betekenissen en toepassingen hebben, moet je transformeren. XML is een goed formaat om te transporteren en perfect voor webservices. Programmeurs zijn

Microsoft business framework

Het framework zal bestaan uit twee lagen, aangevuld met een derde die uit Business Applications zal bestaan, hetzij uit Microsoft's eigen stal, hetzij afkomstig van een ISV. De basislaag common services zal integratie, workflow en security functionaliteit behelzen. Het doel van deze functionaliteit zal zijn het samenbinden van de processen die meestal binnen business application software draaien. De tweede laag bestaat uit gemeenschappelijke componenten en processen die door deze software gebruikt worden, inclusief object entiteiten als customer-accountdefinities. Het doel ervan is loosely coupled objecten te gebruiken die via webservices en XML communiceren. Op die manier zouden uiteindelijk componenten op iedere laag uitwisselbaar moeten worden.

Javux en Longhorn

Longhorn (zou die naam duiden op een poging om de concurrentie op de hoorns te nemen?) is volgens Microsoft de belangrijkste nieuwe versie sinds Windows 95. Dat op zich zou al een reden kunnen zijn om in dit stadium dingen te laten zien. Duidelijk is, dat Microsoft niet bang is dat de concurrentie van deze informatie zal profiteren. Het Microsoft platform is ook te veelomvattend en het R&D budget van de reus uit Redmond te groot om in dit stadium de concurrentie te moeten vrezen (de beslissingen van de Chinese en sociaal-democratische IT-machthebbers en politici echter des te meer).
Longhorn is ook de eerste versie waarbij het Windowsplatform tot zijn recht komt als een platform waaraan één architectuur ten grondslag ligt, zo u wilt die van chief architect Bill Gates. Dat is misschien ook wel het belangrijkste verschil wanneer je het Longhorn-platform met Javux (i.e. Linus/Java) wilt vergelijken. De architectuur van het Windows platform oefent ook aantrekkingskracht uit op IT-toptalent. Zo heeft een paar weken geleden nog Borland's chief scientist Blake Stone te kennen gegeven dat hem een aanbod is gedaan wat hij niet kon weigeren.
Microsoft is traditioneel sterk in marketing, De Longhorn–preview moet zeker in dat licht gezien worden. Dat neemt niet weg dat Microsoft een bijzonder interessant platform aan het opbouwen is. Ik ben zeer benieuwd wat de concurrentie daar tegenover zal stellen.

vertrouwd met objecten en ze bieden een erg sterke en rijke syntax voor middle-tier applicaties. Tenslotte houdt het opslaan van data zich bezig met SQL, een set gebaseerde taal die abstraheert en data opslaat. Mensen slaan sinds tijden data op in relationele databases. Veel research houdt zich met de vraag bezig hoe deze mappings (*van het ene model in het andere, red.*) gebeuren. Wanneer *Moore's law* onze vriend is, waarom zou dan een middle-tier ontwikkelaar zich zorgen hoeven te maken over SQL-statements? Waarom kan hij zich niet gewoon met objecten bezig houden?

Dat is geen nieuw idee. Eén van de belangrijkste kritiekpunten op .NET tot nu is dat wij niet zoals J2EE een objectrelationeel framework hebben, die dingen kun je al doen in J2EE. Maar wij hebben het voordeel dat we van hun fouten leren en van wat iedereen vóór ons gedaan heeft. Weliswaar helpt Moore's law ons bij een object-relationeel framework, maar dat betekent niet dat je de performance kunt verwaarlozen. Met *Object Spaces* werken we er heel hard aan om de performance al standaard heel goed te hebben en ten tweede om de ontwikkelaars toe te staan deze te tunen en hun eigen serialization mechanismen te implementeren om er zeker van te zijn dat de performance *up to scratch* is. Als wij een framework aanleveren dat de mapping verzorgt tussen de objecten en hoe zij uiteindelijk opgeslagen worden, dan kan de ontwikkelaar gewoon zeggen: sla deze objecten op. Die abstractie is een groot voordeel.'

WHIDBEY *Terug naar het steeds abstracter maken van het schrijven van code. Tot nu toe heeft Microsoft nog niet de volgende stap gezet, namelijk naar een MDA-achtige manier van werken. Aan de andere kant doen geruchten de ronde dat Microsoft UML-experts heeft ingehuurd, en die zijn natuurlijk bezig iets voor het framework te maken. Zelfs Gates heeft zich daarover – zij het in zeer bedekte termen – uitgelaten. Wat zullen we daarvan uiteindelijk te zien krijgen in de Whidbey?*

Watling: 'Laat ik voorop stellen dat ik geen expert ben op dat gebied. Het lijkt er zeker op dat Microsoft een aantal van die grote modeldriven en UML-mensen aantrekt. Maar ik verwacht niet dat Microsoft UML-experts aantrekt, alleen om opnieuw te implementeren wat al door anderen is gedaan. Een deel van het Dynamic Systems Initiative waarover ik het had gaat over mapping. De ontwikkelaar moet zich tot op zekere hoogte bewust zijn van de fysieke deployment waar de applicatie naar toe gaat. We proberen regels te definiëren zodat de programmeur geen fout kan maken die pas tijdens het deployen ontdekt wordt. Het is een gebied dat veel aandacht van Microsoft krijgt, om deployment heel simpel te maken. Met modeling zal, denk ik, iets gaan gebeuren. Mijn perceptie is dat UML in zeer sterke mate ontworpen is met het OO-concept in het achterhoofd. Ik ga graag in debat met UML-experts door te stellen dat je UML kunt uitbreiden tot heel veel andere nieuwe dingen, maar dat UML niet noodzakelijkerwijs de perfecte taal is voor de wereld waarin we leven, een gedistribueerde application omgeving, een service georiënteerde architectuur. Misschien is UML niet de beste manier om dingen te doen, hoewel veel mensen tijd investeren om het te leren. Mijn persoonlijke gok is dus – en het is een gok – dat Microsoft werkt aan een taal die service georiënteerde architecturen modelleert. Op deze verschillende niveaus, dus waar UML echt zin heeft voor ontwikkelaars, zijn er misschien andere domeinspecifieke talen die meer geschikt zijn, voor de hardware of het besturingssysteem. Het gaat niet alleen om applicaties, want daarvoor heb je UML. Het is gecompliceerder en rijker dan dat en ik verwacht dat we op dat gebied aan het werk zijn met al die experts om wat UML in het verleden gedaan heeft te verbeteren en op veel verschillende lagen te werken in plaats van alleen op de applicatielaag. Maar dat is mijn persoonlijke gevoel.'

Tekst en fotografie: Dré de Man