“Metadata zijn de hoekstenen van het .NET Framework”

INTERVIEW MET .NET-SPECIALIST JEFFREY RICHTER

Het .NET-initiatief mag dan de volgende stap zijn in de logische reeks die begon in de MS-DOS prehistorie, voor Jeffrey Richter is de uitleg van de ins en outs van .Net-programmering duidelijk de laatste stap in een logische opeenvolging die begon met de introductie van Windows 3.0 in 1990.

“In sommige opzichten is .NET een nieuw programmeerparadigma,” zegt Jeffrey Richter, “en in sommige opzichten ook weer niet. Ten eerste is er de verwerkings-engine, of runtime engine, en die dingen zijn er al sinds de jaren ’70, of zelfs de jaren ’60. Visual Basic heeft een runtime gehad, en de C runtime is ook al een flink aantal jaren aanwezig. De kern van het .NET Framework is een runtime engine waarnaar wordt verwezen als de Common Language Runtime [CLR]. Een van de ‘coole’ zaken met de CLR is dat hij personen ondersteunt bij het schrijven van programma’s in een aantal verschillende programmeertalen. Er zijn zo’n twintig programmeertalen die op het platform kunnen worden gebruikt. Wanneer je normaal gesproken start met het ontwerpen van een softwareproject, begin je met te denken over de features, de eigenschappen en wat belangrijk is voor het project. Is performance belangrijk, of is het belangrijker het project snel af te hebben? En wat te denken van ‘rapid development’?”

Richter vervolgt: “Gebaseerd op deze beslissingen koos je voor een programmeertaal, omdat verschillende programmeertalen ook verschillende eigenschappen boden. C of C++ lagen erg dicht op het ‘bare metal’, waardoor je grote controle had over de performance. Maar het was lastig een ‘rapid application’ te bouwen met C of C++, omdat de programmeur zich met elk klein detail moest bemoeien. Aan de andere kant bood bijvoorbeeld Visual Basic 6.0 de mogelijkheid snel applicaties te bouwen, maar was je verder verwijderd van het ‘metaal’. Dus had je deze hele kleine ‘sandbox’ waarin je niet gemakkelijk threads kon creëren of netwerkcommunicatie kon toepassen. De CLR lost dit probleem op. Het is een rijkelijk gevuld ontwikkelplatform dat een vracht aan eigenschappen biedt, en die eigenschappen zijn beschikbaar onafhankelijk van de programmeertaal die je gebruikt. Ik denk, om eerlijk te zijn, dat veel mensen er geen rekening mee houden dat ze een deel van de applicatie in één taal kunnen schrijven, en een ander deel in een andere taal. En alles past naadloos in elkaar.”

Jeffrey Richter

Jeffrey Richter is een van de medeoprichters van Wintellect (www.wintellect.com), een training-, debugging en consultancyonderneming die zich heeft toegelegd bedrijven te ondersteunen om sneller betere software te bouwen. Hij is de auteur van verscheidene Windows programmeerboeken, waaronder Programming Applications for Microsoft Windows en Programming Server-Side Applications for Microsoft Windows 2000, en de bestseller Applied Microsoft .NET Framework Programming (Januari 2002) . Jeffrey is tevens een redacteur van MSDN magazine, waar hij de .NET-column schrijft. Hij is medewerker van het Microsoft .NET Framework-team sinds oktober 1999.

Common language runtime

De CLR biedt verscheidene eigenschappen waardoor hij beschikbaar is voor alle programmeertalen. Richter: “Ten eerste waarborgt de CLR dat code typesafe is en dat hij alleen een ‘veilige’ verwerking uitvoert. Dit betekent dat de code van de ontwikkelaar niet een verwerking kan uitvoeren die het geheugen zou kunnen corrumperen en waarborgt dat de ‘state’ van een object nooit corrupt kan raken. Als de code van de programmeur probeert iets te doen wat ‘niet netjes’ is, vangt de CLR het probleem op en krijgt de ontwikkelaar een ‘exception alerting’ betreffende het probleem. Dit helpt de ontwikkelaar meer bugvrije code te schrijven. Ten tweede biedt de CLR ook Code Access Security

waardoor je een fijnmazige bescherming krijgt. Het is bijvoorbeeld mogelijk om schrijfmogelijkheid (write access) te bieden voor een specifieke subdirectory van de schijf, terwijl je leestoegang (read access) zonder enige restrictie biedt. Als derde levert de CLR rijke threading-mogelijkheden aan alle applicaties, geschreven in een willekeurige programmeertaal, waardoor ontwikkelaars high-performance en schaalbare applicaties kunnen bouwen. Ten vierde heb je met de CLR een 'garbage collection' omgeving bij alle talen. Via garbage collection kan een applicatie niet het geheugen corrumperen en de verantwoordelijkheid te weten wanneer een object weer moet worden vrijgemaakt is de taak van de garbage collection. Hierdoor hoeft de ontwikkelaar zich geen zorgen meer te maken over geheugenbeheer en kan hij of zij zich concentreren op de applicatielogica. Ten slotte gebruikt de CLR overal en altijd 'exception handing'. Dankzij exception handling kunnen ontwikkelaars hun cleanup-methoden en error recovery code scheiden van de logica van hun 'main' method. Hierdoor wordt het schrijven, lezen en onderhouden van code veel gemakkelijker. Bovendien kan de ontwikkelaar hierdoor niet meer een fout, die door een method wordt geretourneerd, over het hoofd zien waardoor de applicatie ongecontroleerd kan 'runnen'. De ontwikkelaar moet de exception vatten en het probleem oplossen."

Metadata

Metadata, legt Richter uit, zijn de hoekstenen en de belangrijkste onderdelen van het .NET Framework. Metadata zijn een set van tabellen die alle compilers moeten voortbrengen bij de compilering van broncode. Er zijn twee belangrijke soorten metadata-tabellen: definitietabellen en referentietabellen. De definitietabellen beschrijven de types en leden (members) die in de broncode zijn gedefinieerd. Deze informatie wordt door de CLR geïnterpreteerd en het zijn deze metadata die de CLR vertellen over de types van de ontwikkelaar waardoor ze een deel van het 'systeem' kunnen worden. De types kunnen zo dus door elke programmeertaal worden gebruikt. Richter: "In een bepaald opzicht zijn metadata de grootste gemene deler van alle programmeertalen."

Metadata worden gebruikt om een groot aantal technologieën mogelijk te maken. "De FCL's serialisatie engines bijvoorbeeld gebruiken metadata om te weten hoe ze typevelden moeten serialiseren in een binaire of XML-stream. Visual Studio's IntelliSense gebruikt metadata om de ontwikkelaar suggesties te bieden bij het redigeren (editen) van hun code. De garbage collector gebruikt metadata om te weten wanneer velden in een object naar andere objecten verwijzen, waardoor de GC weet welke objecten wel en welke niet hun geheugen kunnen terugeisen. Compilers kunnen dankzij metadata naar types verwijzen die zijn gedefinieerd in een andere programmeertaal zonder extra hulpinformatie zoals 'header files'. En ASP.NET XML webservices gebruiken metadata om te bepalen hoe ze een type-method als een webservice-method kunnen tonen (expose) zonder enige inspanning van de programmeur. ASP.NET gebruikt de metadata ook om automatisch de Web Service Description Language (WSDL) te construeren voor webservice-methoden.

Performance

Hoe zit het met de performance van het .NET Framework? Richter: "Als je vandaag de dag een applicatie zou bouwen in unmanaged C++ en in een taal als C#, dan zul je merken dat de unmanaged C++ applicatie waarschijnlijk beter zal performen. Maar het hangt in werkelijkheid af van de applicatie die je aan het bouwen bent. Je moet rekening houden met de vracht aan eigenschappen die je níet krijgt met unmanaged C++, zoals runtime type safety, code access security en garbage collection. Er is uiteraard iets performanceverlies door deze technologieën. Ik denk echter dat op de lange termijn .NET Framework-applicaties feitelijk beter zullen performen dan unmanaged applicaties. Laat me een reden geven waarom dit het geval is. Wanneer je een unmanaged C++ applicatie compileert, vertel je de compiler om alleen maar 386 CPU-instructies te gebruiken. Op deze manier zal de applicaties niet op alle mogelijke CPU's draaien. Hiertegenover staat dat de CLR de code 'at runtime' compileert in CPU-instructies. Dit betekent dat als de CLR op een Pentium 4 machine draait, hij Pentium 4 CPU-instructies zal gebruiken.

Bovendien, als de CLR ziet dat de machine slechts één processor heeft, dan kan hij alle code verwijderen die alleen nodig is voor multiprocessormachines. En, als de CLR ziet dat een bepaalde 'if'-statement nooit waar of geldig (true) zal zijn op de host-machine, dan kan de CLR de 'if'-test en de code volledig verwijderen bij de compilering naar CPU-instructies. Ook waarborgt de CLR dat de typevelden correct zijn gelijnd op de juiste byte-boundary's waardoor de performance op sommige CPU-architecturen drastisch verbetert. De CLR zou in theorie de code-path van een method tijdens de verwerking kunnen analyseren en de code herschikken of hercompileren terwijl de applicatie draait. Als de CLR bijvoorbeeld ziet dat een test altijd onwaar (false) is, zou hij de code kunnen herschikken om branch predictions te verbeteren. Alles overziend kun je dus rustig stellen dat de CLR de kern vormt van het .NET Framework."

.NET Framework

Richter vervolgt: "Het Microsoft .NET Framework verhoogt de programmeerabstracties die aan de softwareontwikkelaar worden geboden. Dat wil zeggen, het .NET Framework zorgt er voor dat de netwerkcommunicatie en het zenden/ontvangen van data kinderspel wordt. Maar het .NET Framework is zo veel meer dan alleen maar 'netwerken'. Zijn objectgeoriënteerd ontwerp vereenvoudigt het programmeren en het biedt faciliteiten om makkelijk met databases te werken, het persisteren van data, het manipuleren van XML, enzovoort. Het verbetert tevens vele als zo ervaren ongemakken zoals de DLL-hel, versioning, uninstall, beveiliging en robuustheid. Ten slotte, niet alleen kunnen developers nu XML Webservices ontwikkelen, het .NET Framework ondersteunt hen nu ook bij het bouwen van applicaties en websites."