.NET column

De blikken die ik toegeworpen krijg op een verjaardag wanneer ik me iets te 'geek' gedraag zijn goed te vergelijken met de reactie van vakgenoten wanneer ik begin te praten over functionele talen. In deze column zal ik proberen begrip te kweken en zelfs suggesties geven voor het toepassen van de functionele talen.

Functioneel

Wat is een functionele taal? Een functionele taal is een taal waarin de functies geen enkel side effect hebben. Het enige dat een functie doet, is op basis van zijn parameters een uitkomst retourneren. Dit is geen waterdichte definitie. Mijn beschrijving van functionele talen is bedoeld om gevoel te krijgen voor wat een functionele taal is. De oorsprong en de toepassing van de functionele talen ligt historisch vooral in de wiskunde. Dit heeft tot gevolg gehad dat de meeste boeken en artikelen lastig leesbaar zijn voor niet-wiskundigen. In een functionele taal wordt beschreven wat er moet gebeuren, niet hoe. In plaats van gedetailleerde instructies voor de aannemer beschrijven we het huis en laten het bouwen aan hem over. Voor programmeurs die imperatieve talen zoals C, C++, Java, Visual Basic en C# gewend zijn is het even wennen om de uitvoering aan de compiler over te laten.

Wat zijn dan de voordelen? Omdat er geen side effects zijn is het testen van de code veel makkelijker dan bij een imperatieve taal. Het enige dat gecontroleerd hoeft te worden is of bij een gegeven input, de output van de functie klopt. Per definitie verandert er verder niets in het systeem. Hierdoor is de kwaliteit van het testen ook makkelijker te garanderen; het maakt niet uit in welke volgorde functies worden aangeroepen. Een ander belangrijk voordeel dat ook op deze eigenschap rust is het optimaliseren van de executie van de code. Twee functies die in het programma na elkaar worden aangeroepen en waarbij de tweede functie geen gebruik maakt van het resultaat van de eerste functie, kunnen zonder meer parallel worden uitgevoerd. Sterker nog: een functie hoeft pas uitgevoerd te worden als het resultaat nodig is.

Nu we het aantal processors in computers zien toenemen, is het maken van multi-threaded programma's een steeds belangrijker middel om de performance van een applicatie te verhogen. Het ontwerpen van parallelle processen is echter niet eenvoudig; race conditions en deadlocks liggen op de loer. Deze gevaren zijn niet aanwezig in de wereld van de functionele talen. Ik vermoed dat het implementeren van delen van applicaties in functionele talen daarom een lucratieve en veilige weg zou kunnen zijn om de nieuwe hardware optimaal te benutten.

Voorbeelden van dergelijke toepassingen in OLTP-applicaties zijn het controleren van business rules en het genereren van code. Bij het controleren van business rules wordt er geen state veranderd, ook het parallel uitvoeren zou geen probleem hoeven zijn. Bovendien is dit het terrein waar het Unit testen van de applicatie cruciaal is voor de kwaliteit van de applicatie. In veel ontwikkelstraten is het genereren van code heel gewoon aan het worden. Door de generator in een functionele taal te bouwen is de kwaliteit van de generator, en daarom ook van de gegenereerde code, beter te garanderen. Bovendien is een veel gehoord argument tegen functionele talen, de relatieve traagheid, hier veel minder van belang.

Voor het .NET-platform zijn al lange tijd ondermeer Haskell, Scheme, L# en F# beschikbaar. Als straks Visual Studio 2008 uitkomt zullen C# en Visual Basic ook meer kenmerken krijgen van functionele talen in de vorm van LINQ. Het wordt tijd om weer iets nieuws te leren.

Erno de Weerd
Info Support. Over deze column kan verder gediscussieerd worden op http://blogs.infosupport.com/ernow