F#: Een functionele taal voor het .NET framework

Voordelen taal en framework matchen goed

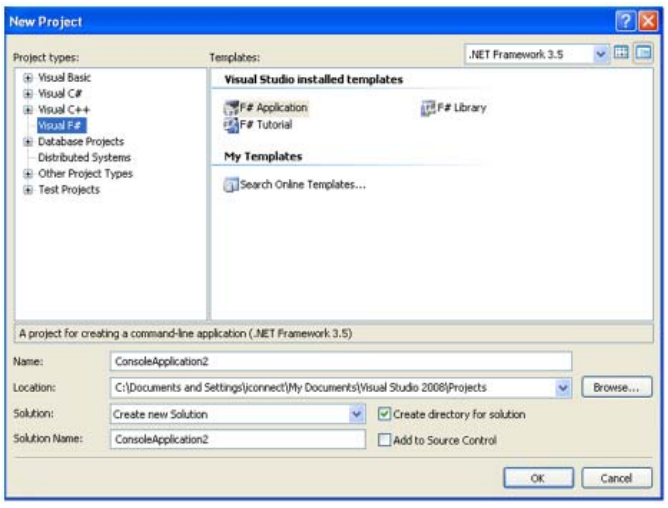
Gerardo de Geest

Functionele programmeertalen bestaan al erg lang en werden tot voor kort bijna alleen maar in de academische wereld gebruikt. Talen als Haskell en ML zijn bekende functionele programmeertalen, maar werken niet samen met het Microsoft .NET platform. Met de introductie van F# is daar verandering in gekomen.

De meeste ontwikkelaars programmeren in een objectgeoriënteerde taal zoals C#. Hierbij maakt de ontwikkelaar klassen, waarvan objecten worden geïnstantiëerd. Deze objecten hebben een bepaalde state waarin het object zich bevindt. Door verandering in state weet het programma welke code uitgevoerd moet worden en naar welke volgende state het moet gaan. Functionele programmeertalen werken heel anders. Er bestaat namelijk geen state in een functionele programmeertaal. Een programma wordt gezien als de evaluatie van een aantal wiskundige functies, waarbij state vermeden wordt.

Wat is F#?

F# is een functionele taal voor het .NET framework. Het combineert de voordelen van functionele programmeertalen met de runtime support, interoperability, libraries en andere voordelen van het .NET framework. F# is afgeleid van de taal OCaml (deze taal heeft geen overeenkomsten met CAML in SharePoint), maar lijkt op sommige punten ook op Haskell en C#.
De programmeertaal F# is ontwikkeld door het Microsoft Research team in Cambridge in Groot-Brittannië. Op dit moment is er een CTP die gedownload kan worden van de MSDN-website. De CTP kan op twee manieren gedownload worden: Als .msi en als .zip. De msi-file is voor gebruikers van Windows en Visual Studio 2008. De zip-file kan door gebruikers van een ander besturingssysteem gebruikt worden, bijvoorbeeld Linux met Mono. Na de installatie van de F# CTP komen er nieuwe projectsoorten beschikbaar in het 'New project' scherm van Visual Studio. De F# applicatie is het project dat gebruikt kan worden om console applicaties te maken en het F# library project om een library te maken. Dit is hetzelfde als bij de C# projecten. Er is echter ook een F# project dat F# tutorial heet. Dit maakt een standaard F# applicatie met wat voorbeeldcode van F#.



FIGUUR 1: NEW PROJECT SCHERM IN VISUAL STUDIO 2008 MET F# CTP

Waarom F#?

De belangrijkste reden om F# te gebruiken is, omdat het een functionele programmeertaal is die goed integreert met het .NET framework. Het brengt de 'type inference', 'type safety' en 'lazyness' mee van ML naar het .NET framework. De integratie met Visual Studio is hetzelfde als bij C# en alle .NET libraries kunnen gebruikt worden. Hierdoor staat er helemaal niets meer in de weg voor .NET ontwikkelaars om met functionele talen aan de slag te gaan.
Daarnaast biedt F# ook interactive scripting, zoals Python dat bijvoorbeeld ook doet. Dit werkt ook vanuit Visual Studio en is erg handig om even snel wat code uit te proberen.
Tot slot integreert F# ook goed met MATLAB, een applicatie die veel wordt gebruikt voor wiskundige berekeningen en interactieve datavisualisatie. Voor applicaties met complexe wiskundige berekeningen en grafieken is het dus handig om F# te gebruiken.

Hello World

Het 'Hello World'-programma wordt vaak gebruikt als eerste kennismaking met een programmeertaal. Hieronder staat het 'Hello World' programma voor F#. Deze code kan gebruikt worden in het 'F# application' project. De code kan uitgevoerd worden op dezelfde manier als C# code uitgevoerd wordt: door op F5 te drukken.

```
#light
open System
printf "Hello world"
Console.ReadLine()
```

De eerste regel vertelt de F# compiler om de lightweight syntax te gebruiken. Het zorgt ervoor dat de spatie een betekenis krijgt in F#, waardoor keywords als begin, in, end en ook de ; niet meer nodig zijn in de code. Op dit moment is het nog verplicht om boven elke programma '#light' of 'light off' te schrijven, anders verschijnt er een warning. In de volgende versie van F# schrijft de compiler er zelf '#light' boven. De '#light off' optie zal waarschijnlijk alleen gebruikt worden voor code die gegeneerd is of die door OCaml moet worden cross-compiled.

F# is niet een pure functionele programmeertaal, omdat het mogelijk is om de waarde van variabelen te veranderen

De tweede regel opent de System library van F#. Het is de F# versie van using System in C# en doet ook precies hetzelfde. Naast de System library kan iedere andere .NET library gebruikt worden.
De derde regel print 'Hello world' naar de console. Dit kan gedaan worden met printf, de standaard functie van F# om strings te printen. Maar, het kan ook door Console.WriteLine() van de System library aan te roepen, zoals dat in C# wordt gedaan.
De vierde regel zorgt ervoor dat de console blijft staan als het programma uitgevoerd wordt. Deze regel is precies hetzelfde als wanneer deze regel in C# zou zijn geschreven. Dit komt doordat er gebruik gemaakt wordt van de System library van .NET.
Merk op dat de F# wel een warning geeft op deze regel, namelijk: 'This expression should have type unit, but has type string.' Dit betekent dat de F# compiler verwacht dat de functie Console.Readline geen waarde teruggeeft. Echter, de functie geeft een String terug, namelijk wat de gebruiker op de commandline invoert.

De som van kwadraten

Een veelgebruikt voorbeeld bij functionele programmeertalen is de som van kwadraten. Het probleem is als volgt: 'Schrijf een functie met als input een rij getallen en met als output de som van alle kwadraten van de rij input'. Als de input van het programma bijvoorbeeld [1;2;3;4] is, dan is de uitvoer $1^2 + 2^2 + 3^2 + 4^2 = 30$. In C# ziet zo'n methode er als volgt uit:

```
public static int SomVanKwadraten(int[] input){
int total = 0;
  foreach(int i in input){
  total += i*i;
}
return total;
}
```

Aan de hand van dit voorbeeld zullen de basisfunctionaliteiten worden toegelicht. In F# zijn er drie verschillende manieren om ditzelfde programma te schrijven.

De **eerste manier** lijkt heel erg op het C# programma van hierboven. In pure functionele programmeertalen is deze manier niet mogelijk, omdat pure functionele programmeertalen immutable zijn. Dit betekent dat je geen nieuwe waarde kunt toekennen aan een variabele. Echter, binnen F# bestaat het keyword mutable, waardoor het toch mogelijk is om een nieuwe waarde aan een variabele toe te kennen. In de code hieronder krijgt de variabele total een andere waarde.

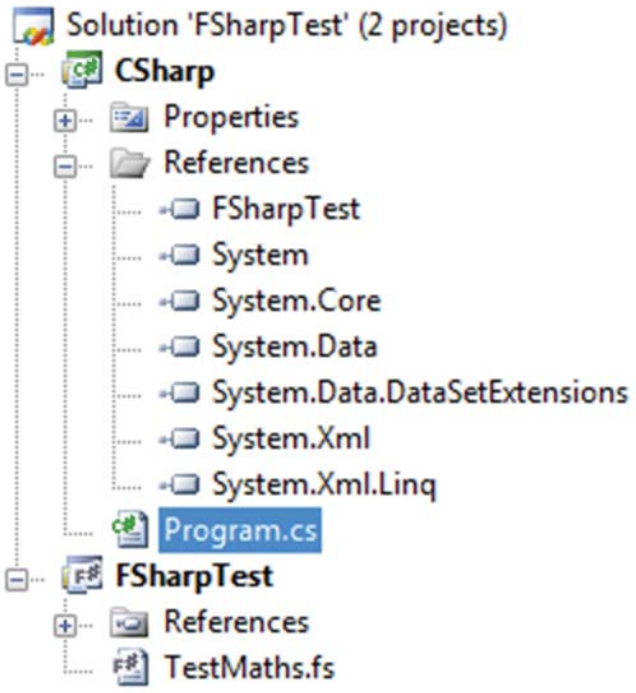
```
let sqr x = x * x
let SomVanKwadraten input =
    let mutable total = 0
    for i in input do
        total <- total + sqr i
    total
```

In de eerste regel wordt een hulpfunctie gedefinieerd. Met het 'let' keyword worden functies gedefinieerd in F#. De functie heet sqr en heeft een parameter x. Merk op dat het type van x niet wordt gedefinieerd. De F# compiler herleidt dit zelf uit de code. Dit principe wordt ook wel 'type inference' genoemd en komt vaak voor bij functionele talen. Verder beschikt Visual Basic .NET 9.0 ook over type inference.
De tweede regel definieert de functie SomVanKwadraten met als variabele input. Hierbij is ook het type van input niet gespecifeerd. De compiler leidt aan de hand van de volgende regels af dat het type van input een lijst van getallen is.
De derde regel begint met een spatie. Omdat de code in leightweight mode is geschreven, betekent dit dat de volgende regels de body van SomVanKwadraten definiëren. Deze regel maakt de variabele total en zet deze op 0. Merk op dat het keyword mutable wordt gebruikt, zodat de variabele van waarde veranderd kan worden.
In de vierde regel staat het begin van een for –loop, waarvan in regel 5 de body wordt gedefinieerd. De for-loop loopt door de lijst van alle getallen, kwadrateert ieder getal en verhoogt total met het kwadraat. Tot slot wordt in regel 6 total geretourneerd. Merk op dat het keyword 'return' niet nodig is.

De **tweede manier** om de som van kwadraten uit te rekenen, is door gebruik te maken van recursie. In de code hieronder wordt dit getoond.

```
let sqr x = x * x
let rec SomVanKwadraten input =
    match input with
    | [] -> 0
    | h::t -> sqr h + SomVanKwadraten t
```

De eerste regel definieert weer de functie sqr en de tweede regel definieert weer de functie SomVanKwadraten. Merk op dat in de tweede regel het keword 'rec' gebruikt wordt. Dit geeft aan dat deze functie door zichzelf aangeroepen kan worden.
De derde regel introduceert het keyword 'match'. Dit is vergelijkbaar met het switch statement in C#. De parameter input wordt in regel 4 vergeleken met de lege lijst. Als er geen getallen in input zitten, dan is de som van de kwadraten 0. Als er wel getallen inzitten, dan wordt in regel 5 het eerste getal van de rest gescheiden met de '::' operator. Bijvoorbeeld als de input [1;2;3] is, dan krijgt h de waarde 1 en t de waarde [2;3]. Vervolgens wordt 1 gekwadrateerd en opgeteld bij de som van de kwadraten van 2 en 3.



FIGUUR 2: EEN SOLUTION MET EEN F# EN EEN C# PROJECT

De **derde manier** laat zien dat een programma geschreven in een functionele programmeertaal een evaluatie is van wiskundige functies.

```
let SomVanKwadraten input =
    input
    |> Seq.map (fun x -> x * x)
    |> Seq.sum
```

De eerste regel is hetzelfde als bij de eerste manier en definieert weer de functie SomVanKwadraten. Regel 2 neemt de input en met de |> operator kunnen er functies op de input worden uitgevoerd. Zo wordt in regel 3 de map functie uitgevoerd op de input. De map functie zorgt ervoor dat een functie wordt uitgevoerd op elk element in de rij. Dus als input [1;2;3] is, dan is het resultaat van regel 3 [1;4;9]. Merk op dat in dit voorbeeld niet de functie sqr is gedefinieerd, maar dat de functie rechtstreeks wordt meegegeven als argument aan map. Het was ook mogelijk geweest om de sqr functie mee te geven.
In de vierde regel wordt het resultaat van de derde regel opgeteld. In het geval van [1;4;9], is het resultaat dus 14.

F# en C# combineren in dezelfde solution

De kracht van F# is dat het heel gemakkelijk gebruikt kan worden in combinatie met .NET. Het is zelfs mogelijk om een deel van een programma te schrijven in F# en een ander deel van hetzelfde programma in C#, zoals is te zien in Figuur 2.
Het project FSharpTest bevat het F# bestand TestMaths.fs. In dit bestand zit de voorbeeldfunctie SomVanKwadraten. Het CSharp project is een C# console application project. Merk op dat dit project een referentie heeft naar het FsharpTest project. In de main methode van Program.cs staat de volgende code:

```
int som = TestMaths.SomVanKwadraten(new int[] { 1, 2, 3, 4 });
Console.WriteLine(som);
Console.ReadLine();
```

De F# methode SomVanKwadraten kan dus aangeroepen worden alsof het een statische methode is van de klasse TestMaths. Vanuit deze code is ook helemaal niet te zien dat de functie SomVanKwadraten is geïmplenteerd met F#. De parameter input van de functie is namelijk een array van integers.
De koppeling met C# is niet altijd zo eenvoudig als in dit voorbeeld. Dit komt doordat er in F# datatypes bestaan die in C# niet beschikbaar zijn. Een voorbeeld hiervan is de BigInteger klasse die in F# beschikbaar is en waarin veel grotere getallen dan een Int64 opgeslagen kunnen worden. In F# wordt een BigInteger overigens op dezelfde manier behandeld als een gewone integer en hoeft de ontwikkelaar hier dus geen aandacht aan te besteden.

Conclusie

F# is een functionele programmeertaal die werkt op het .NET platform. Het is beschikbaar als CTP vanaf de MSDN website en integreert met Visual Studio 2008. Functioneel programmeren is een heel ander concept dan object-georienteerd programmeren en dus zal het voor de meeste ontwikkelaars even wennen zijn om de concepten onder de knie te krijgen. F# is waarschijnlijk voornamelijk nuttig voor wiskundige applicaties. Doordat een F# project makkelijk samen met een C# project onderdeel kan uitmaken van dezelfde solution, is het mogelijk om delen van de applicatie in F# te ontwikkelen en andere delen in C#. Dit zorgt ervoor dat de ontwikkelaar het beste van beide talen kan gebruiken.
Toch is F# niet een pure functionele programmeertaal, omdat het bijvoorbeeld mogelijk is om de waarde van variabelen te veranderen. Hierdoor wordt het ook mogelijk om programma's te schrijven die enigszins op C# lijken.

Links

http://msdn.microsoft.com/en-us/fsharp/default.aspx
http://www.mono-project.com

Gerardo de Geest, heeft technische informatica gestudeerd aan de TU Delft en is nu werkzaam als consultant. In zijn vrije tijd experimenteert hij graag met nieuwe technologien op IT gebied. Zijn weblog is te vinden op: http://www.de-geest.com/blogs/gerardo/default.aspx