Ronald Kroon

Codegeneratie met DSL

MODELLEREN VOOR BETERE CODE

Met de nieuwe DSL-tools voor Visual Studio 2005 kun je zelf designers bouwen voor Visual Studio. Met deze designers kun je code laten genereren om het ontwikkelproces te versnellen. Bovendien wordt de leesbaarheid en de onderhoudbaarheid van de code verbeterd door het werken met modellen. In dit artikel zal ik de mogelijkheden van Domain Specific Languages (DSL) in Visual Studio 2005 laten zien.

Een DSL is een taal die geschikt is voor een beperkt domein, in tegenstelling tot generieke talen als C#, Java of UML. Voorbeelden van DSL's zijn: SQL, spreadsheet-formules en de Windows Form Designer. Deze talen zijn niet geschikt om een complete applicatie mee te bouwen of een compleet ontwerp te maken. Door hun specialisatie zijn die talen echter wel krachtiger binnen hun domein. Met een paar klikken bouw je bijvoorbeeld een user-interface als onderdeel van een applicatie. Met de DSL-tools voor Visual Studio 2005 kun je (als DSL-ontwikkelaar) nu zelf een designer bouwen voor jouw eigen DSL. Met een dergelijke designer kun je (als DSL-gebruiker) vervolgens modellen maken in de gespecificeerde taal (DSL). Op basis van die modellen kan de designer vervolgens code genereren om het ontwikkelproces te versnellen. Zie ook tabel 1.

Referentieapplicatie

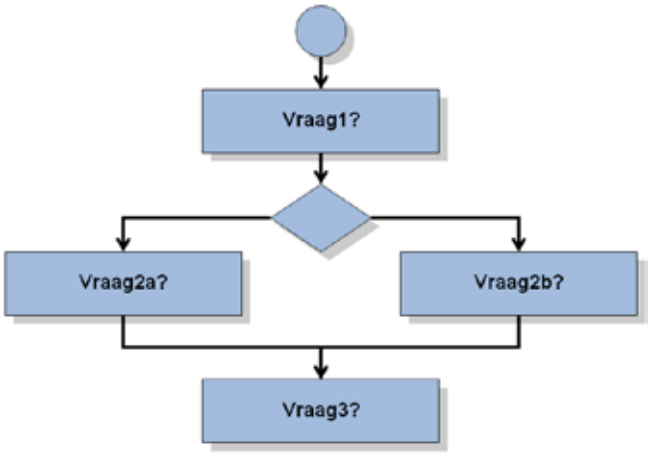
Voordat je begint met het bouwen van een designer, moet je goed nadenken over wat je wilt ondersteunen. Bouw een designer voor dingen die je vaker gebruikt, zoals een class-diagram, een workflow of een data-access layer. Zorg dat de designer specifiek blijft voor een beperkt domein, anders ben je meer tijd kwijt aan het bouwen van de designer dan je ooit terugverdient met gegenereerde code. Het is erg handig om vooraf een referentieapplicatie te bouwen. Deze dient als voorbeeld voor hoe de gegenereerde code er uit komt te zien. Daarnaast gebruik je die applicatie ook om een framework op te zetten die de gegenereerde code ondersteunt. De variabele delen komen in aanmerking voor codegeneratie, de niet-variabele delen worden in een framework ondergebracht. Voor de variabele delen die weinig worden gebruikt of moeilijk in een model te vatten zijn, bouw je ruimte in voor handmatige uitbreidingsmogelijkheden (bijvoorbeeld door partial classes te genereren).

DSL-project

Genoeg theorie, voorbeelden zijn altijd leuker en duidelijker. We gaan een designer maken voor vragenformulieren. Met die designer moet je vragen kunnen modelleren en een flow voor de volgorde kunnen aangeven. Afhankelijk van de antwoorden moeten andere vragen gesteld worden; zie afbeelding 1. Als je de Visual Studio 2005 SDK Version 4.0 (februari 2007) [1] hebt geïnstalleerd, kun je via New Project →Other Project Types →Extensibility kiezen voor een Domain-Specific Language Designer. Vervolgens krijg je een wizard voor wat algemene instellingen. Als template heb ik de MinimalLanguage gekozen en als extensie .form. Hiermee krijg je een solution met een Dsl-project en een DslPackage-project. Het Dsl-project bevat onder andere het bestand DslDefinition.dsl. Dit bestand bevat een schema met de taalelementen van de DSL in de eerste kolom (Classes en Relationships), en de diagramelementen in de tweede kolom; zie afbeelding 3. De diagramelementen zijn een representatie van de taalelementen. Je ziet dan ook een verbindingslijn tussen taal- en diagramelementen die bij elkaar horen. In de MinimalLanguage bestaat een model (ExampleModel) uit ExampleElements die onderling verbonden kunnen worden via de relatie ExampleElementReferencesTargets. De ExampleElements worden gerepresenteerd door ExampleShapes, en de relaties tussen ExampleElements worden gerepresenteerd door ExampleConnectors. Als je deze solution start, wordt een tweede instantie van Visual Studio geopend (zie ook kader 1). Hierin zie je dat er nog een project is gecreëerd door de wizard: het Debugging-project. In deze versie van Visual Studio is de designer voor de MinimalLanguage beschikbaar, inclusief een voorbeeld van een model (Sample.form). Een model wordt opgeslagen in twee xml-bestanden. Het eerste bestand krijgt de extensie die we zelf hebben gekozen: .form. Hierin staat de essentie van het model, de classes en relaties. Het tweede bestand heeft de extensie

Rol	DSL-ontwikkelaar	DSL-gebruiker	Website-gebruiker
Middel	DSL-tools	Designer voor VS	Website
Doel	Designer voor VS	Website (bijvoorbeeld)	Browsen

Tabel 1. Wie doet wat met DSL's?



Afbeelding 1. Voorbeeldmodel voor een vragenformulier

.form.diagram. Daarin staat de opmaak van ons model. Overigens krijg je natuurlijk het model te zien (en niet de xml) als je dubbelklikt op het .form-bestand.

DSL-definitie

We gaan nu de MinimalLanguage DSL-definitie ombouwen tot de DSL-definitie voor onze eigen designer. Uit afbeelding 1 kunnen we in ieder geval de volgende classes afleiden: StartPoint (rondje), Question (blokje) en Decision (ruitje). Daarnaast is blijkbaar een Flow (pijltje) nodig om deze classes te verbinden. In de DSL- definitie is een relatie altijd een verbinding tussen precies twee (soorten) classes. In ons geval moet het pijltje echter getekend kunnen worden tussen verschillende en dezelfde (soorten) classes. Bijvoorbeeld van StartPoint naar Question, van Question naar Question, van Question naar Decision en omgekeerd. We zouden hiervoor verschillende relaties kunnen creëren, maar het is handiger om één base-class voor de drie classes te introduceren. Dan kunnen we één relatie definiëren die willekeurige relaties tussen StartPoints, Questions en Decisions mogelijk maakt. Deze base-class noemen we Action. Dit wordt een abstracte base-class voor StartPoint, Question en Decision; zie afbeelding 3. Deze abstracte class krijgt geen schermrepresentatie. Tussen Actions definiëren we de relatie Flow. Bij de drie concrete classes maken we ook een schermrepresentatie in de vorm van diagramelementen. Het StartPoint wordt gerepresenteerd door de StartShape. De Decision wordt gerepresenteerd door de DecisionShape. En de Question wordt gerepresenteerd door de QuestionCompartmentShape. Een compartment-shape geeft je de mogelijkheid binnen de shape nog één of meer lijstjes van subelementen te tonen. In het geval van de Question wordt dat een lijstje met Options. Deze Options worden dan ook als embedded relationship van een Question opgenomen. Voor de relatie Flow definiëren we een FlowConnector die is ingesteld om als pijltje op het scherm getoond te worden. Nadat het diagram in afbeelding 3 gemaakt is, moeten er nog toolbox-items toegevoegd worden. Dit is een van de dingen die in de DSL Explorer toegevoegd kunnen worden; zie afbeelding 4. Als je DSL-definitie klaar is, genereer je de bijbehorende code met het knopje Transform All Templates in de Solution Explorer; zie afbeelding 5. In feite zijn de DSL-tools zelf ook een voorbeeld van een designer waarmee je modellen kunt maken en code kunt genereren. In het model leg je de taal- en diagramelementen vast. De gegenereerde code resulteert in jouw add-in. Nu ben je klaar om modellen te kunnen tekenen; start de debugger.

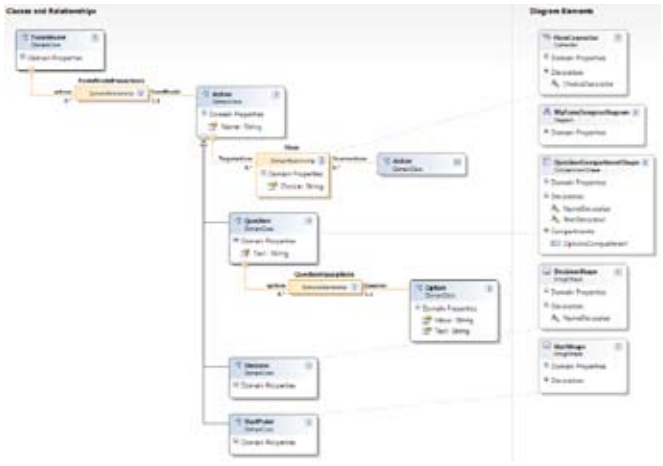
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Globalization;
using Microsoft.VisualStudio.Modeling.Validation;
using Microsoft.VisualStudio.Modeling;

namespace Ordina.MyFormDesigner
{
  /// <summary>
  /// Partial class for the validation methods for FormModel
  /// </summary>
  [ValidationState(ValidationState.Enabled)]
  public partial class FormModel
  {
    /// <summary>
    /// Ensure there is exactly one StartPoint.
    /// </summary>
    [ValidationMethod()]
    private void ValidateOneStartPoint(ValidationContext context)
    {
      int count = 0;
      List<StartPoint> startPoints = new List<StartPoint>();
      foreach (Action action in this.actions)
      {
        StartPoint startPoint = action as StartPoint;
        if (startPoint != null)
        {
          count++;
          startPoints.Add(startPoint);
        }
      }
      if (count == 0)
      {
        string error = "The form should have a start point";
        context.LogError(error, "No start point", this);
      }
      else if (count > 1)
      {
        string error = "The form may only have one start point";
        context.LogError(error, "Too many start points",
          startPoints.ToArray());
      }
    }
  }
}
```

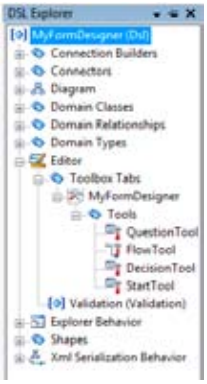
Codevoorbeeld 1. Validatie voor precies één StartPoint

Afbeelding 2. De DSL van de MinimalLanguage-template

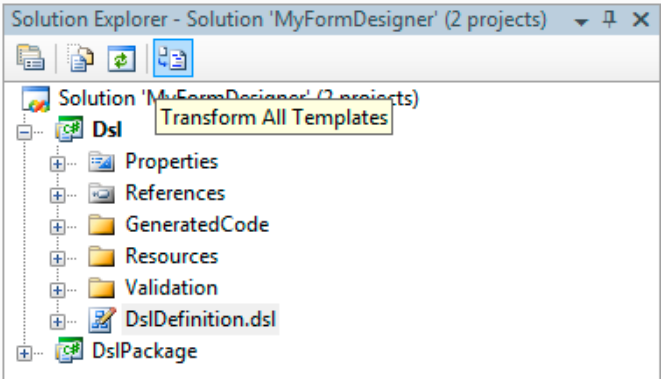
Voor het debuggen van add-ins voor Visual Studio, zoals DSL-designers, maak je gebruik van de Experimental Hive. Dat is een aparte versie van de registry-settings voor Visual Studio. Hierin staan alle normale instellingen plus de instellingen voor de add-in die je aan het maken bent. Dit zorgt er voor dat je de designer kunt testen zonder deze te installeren. En jouw normale versie van Visual Studio blijft gewoon werken, mocht de experimental hive corrupt raken. Je start Visual Studio in de Experimental Hive met de command-line arguments '/ rootsuffix Exp'.



Afbeelding 3. De DSL voor vragenformulieren



Afbeelding 4. De DSL Explorer



Afbeelding 5. Knop rechtsboven genereert de code voor de designer

```
<
// This file contains the start point of the form.

namespace Ordina.TestForm.Actions
{
  public static class StartPoint
  {
    public static IAction GetFirstAction()
    {
      return new ReadyQuestion();
    }
  }
}
```

Codevoorbeeld 3. Gegenereerde code bij de template voor StartPoint

```
#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.Mode
lingTextTransformation" debug="true"#>
<#@ output extension=".cs" #>
<#@ MyFormDesigner processor="MyFormDesignerDirectiveProcessor"
requires="fileName='Sample.form'" #>
// This file contains the start point of the form.

namespace Ordina.TestForm.Actions
{
<#
  foreach (Action action in this.FormModel.actions)
  {
    StartPoint startPoint = action as StartPoint;
    if (startPoint != null)
    {
#>
  public static class StartPoint
  {
<#
      if (startPoint.Targetactions.Count > 0)
      {
        Action firstAction = startPoint.Targetactions[0];
        string actionType = firstAction.GetType().Name;
#>
    public static IAction GetFirstAction()
    {
      return new <#= firstAction.Name #><#= actionType #>();
    }
<#
      }
#>
  }
<#
    }
  }
#>
}
```

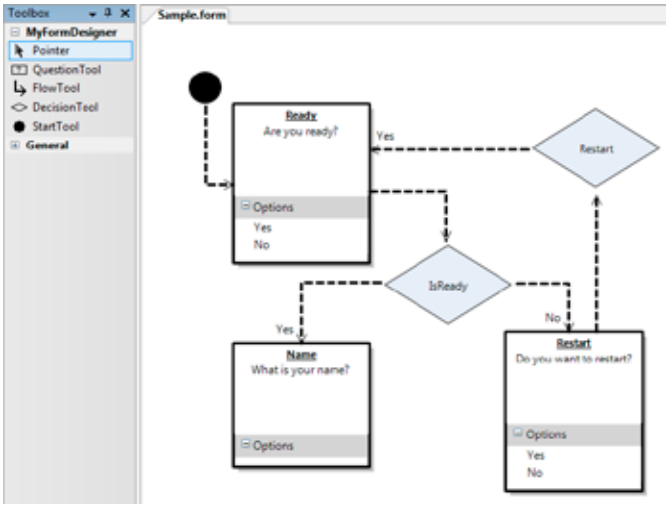
Codevoorbeeld 2. Template voor de class StartPoint

Validatie

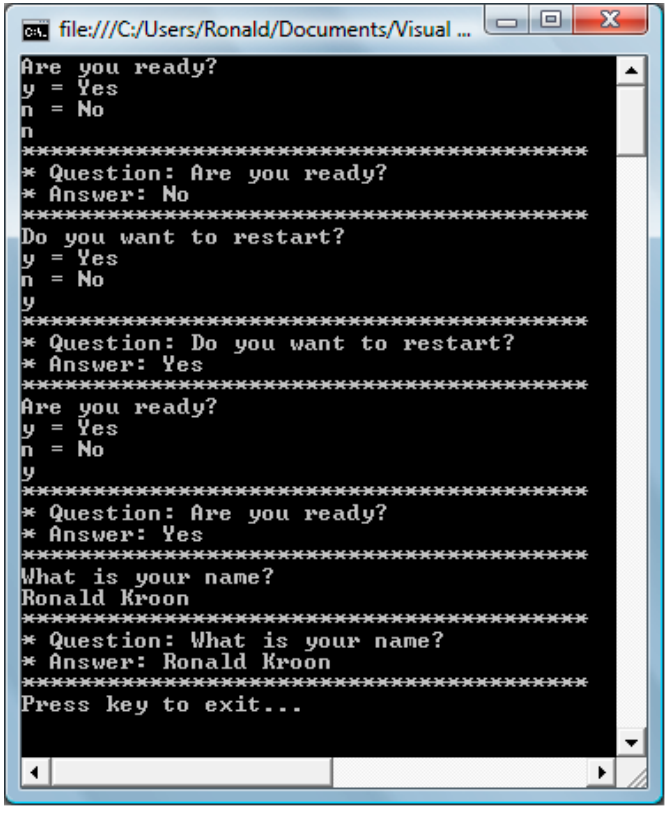
In afbeelding 6 staat een voorbeeld van een model dat we nu kunnen tekenen met onze designer. Het startsymbool gebruiken we om aan te geven waar het formulier moet beginnen. Uiteraard mag er slechts één startsymbool in het model voorkomen, maar dat wordt nu nog niet afgedwongen. We moeten dus nog een validatieregel toevoegen. Dit doen we door de gegenereerde partial class FormModel uit te breiden. Hiervoor voegen we het bestand FormModel.cs toe. Daarin komt de 'andere helft' van de partial class FormModel met daarin onze validatieregel; zie codevoorbeeld 1. Deze regel controleert of er precies één startsymbool is, en als dat niet zo is, krijg je een foutmelding in de Task List. De methode krijgt het attribuut ValidationMethod en in de DSL Explorer geven we aan wanneer er gevalideerd moet worden (bij openen, bij opslaan, enzovoort). Er zijn natuurlijk nog wel wat validatieregels te bedenken die een goed gebruik van de designer bevorderen.

Codegeneratie

Op basis van ons model gaan we nu code genereren. Hiervoor schrijven we een zogenaamde Text Template. Dit bestand krijgt de extensie .tt. In een dergelijk template-bestand kun je informatie uit het model gebruiken en dat combineren met vaste teksten. Er zijn ook alternatieve manieren voor codegeneratie, zoals de System.CodeDom-namespace of een xsl-transformatie op de xml van het model, maar de Text Template is verreweg de gemakkelijkste manier. Zie de template voor de StartPoint-class ter illustratie; zie codevoorbeeld 2. Voor de duidelijkheid heb ik hierin de code voor het benaderen van het model rood gemaakt en de vaste teksten blauw. Zodra je de template opslaat of als je alle templates transformeert wordt de bijbehorende code gegenereerd; zie codevoorbeeld 3. Zo maken we ook nog één template voor alle Question-classes en



Afbeelding 6. Voorbeeldmodel van een vragenformulier



Afbeelding 7. Console application met vragen volgens het model

één voor alle Decision-classes. Voor één template wordt altijd één bestand gegenereerd, maar in ons geval komen er dus verscheidene classes in dat ene bestand. Ter ondersteuning van de gegeneerde classes heb ik ook nog enkele zelfgeschreven classes toegevoegd (het ondersteunende framework), zoals een IAction-interface en een base-class voor alle Questions.

Resultaat

Om het formulier als applicatie te starten, heb ik van het Debugging-project een Console Application gemaakt en enkele regels code toegevoegd voor het starten van de applicatie. En zie het resultaat in afbeelding 7. Deze solution is ook te downloaden van GotDotNet[2]. Met deze oplossing kunnen we heel eenvoudig (via het model) het vragenformulier aanpassen en uitbreiden. Ook nieuwe formulieren zijn gemakkelijk te maken.

Ronald Kroon is consultant ICT bij Ordina (www.ordina.nl). Hij heeft de DSL-technologie toegepast voor de Microsoft-ontwikkelstraat van Ordina (www.ordinasoftwarefactory.nl). Hij is bereikbaar via ronald.kroon@ordina.nl.

Referenties

1) http://www.microsoft.com/downloads/details.aspx?FamilyID=51a5c65b-c020-4e08-8ac0-3eb9c06996f4&DisplayLang=en

2) http://www.gotdotnet.com/Community/UserSamples/Details.aspx?SampleGuid=4ca5f3f7-6f3b-459e-bf2b-7d9fa3c4bfea