TOOLS

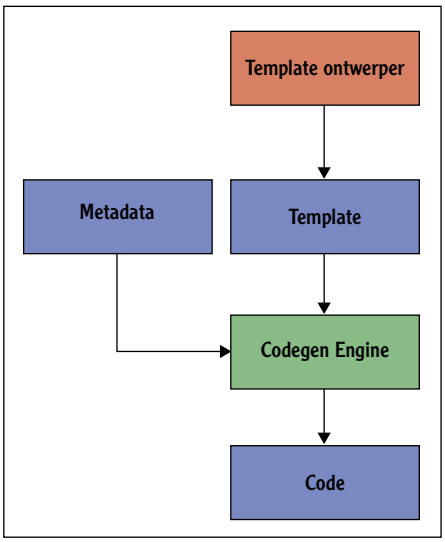
Applicaties worden steeds complexer en omvangrijker. Het is van belang dat de gereedschappen die we gebruiken om applicaties te ontwikkelen ons in staat stellen dit gemakkelijker en sneller te doen. In de meeste applicaties komen grote delen soortgelijke code voor. Codegeneratie kan een goed gereedschap zijn voor het bouwen van deze applicatie-onderdelen. Dit artikel geeft een globaal beeld van codegeneratie, wanneer het kan worden toegepast en wat hier in grote lijnen bij komt kijken.

Codegeneratie in de praktijk

Goed hulpmiddel, maar geen 'silver bullet'

Raymond Binnendijk
is werkzaam als software architect bij Logica in Arnhem
Sieto Hulst
is werkzaam als senior system engineer, eveneens bij Logica in Arnhem
Voor vragen of opmerkingen zijn ze te bereiken op raymond.binnendijk@logica.com en s.hulst@logica.com

Om een globaal beeld te geven van de componenten die een rol spelen bij codegeneratie zijn deze in figuur 1 weergegeven.
Hieronder volgt een beschrijving per component:
Template. Een template is een beschrijving van de structuur van de te genereren applicatie-onderdelen. Variabelen zijn opgenomen op plekken waar deze structuur specifiek voor een applicatie-onderdeel moet worden geconfigureerd.
Metadata. Inhoud van de variabelen voor het configureren van de template(s).
Codegeneratie engine. De engine genereert code op basis van de template(s) en de metadata.
Template ontwerper (optioneel). Via een GUI een model ontwerpen. Een codegenerator kan het model omzetten naar code. Een voorbeeld hiervan is te vinden bij Model Driven Architecture (MDA). Een model beschrijft het systeem in (bijvoorbeeld) UML. Een codegenerator genereert code vanuit het UML-model. Om MDA en codegeneratie te realiseren kun je een bestaande designer gebruiken maar het is ook mogelijk om een eigen designer te schrijven. In Visual Studio is dit mogelijk door een DSL (Domain Specific Language) te gebruiken. De DSL-designer maakt een model dat gebruikt kan worden om code te genereren.



Figuur 1

Toepasbaarheid

Codegeneratie kan een uitstekende methode zijn om op een snelle en doeltreffende manier onderdelen van applicaties te ontwikkelen. Onderdelen die aan verandering onderhevig zijn en eenmalige, zeer specifieke functies binnen een applicatie zijn, zijn minder geschikt om te genereren.
Codegeneratie is uitstekend toepasbaar voor het genereren van een database access layer (DAL). De structuur van een DAL komt vaak overeen met de structuur van de onderliggende database en leent zich er dan ook goed voor om te genereren (de database structuur kan hierbij dienen als metadata). De gewenste functies kunnen per tabel, view en stored procedure worden gedefinieerd (en vervolgens gegenereerd). De DAL bevat gemiddeld ongeveer veertig tot vijftig procent van het totaal aantal regels applicatiecode. Het uitcoderen is relatief saai werk. Codegeneratie komt hier dus uitstekend van pas. Daarnaast

Codegeneratie verkleint de kans op fouten en zorgt voor tijdwinst

verkleint codegeneratie de kans op fouten en kun je gebruikmaken van 'proven' technologie.
Een ander applicatie-onderdeel waarbinnen veel te genereren valt, zijn de data transfer objects (DTO's). DTO's zijn de objecten waarin de data in de business layer van de applicatie verpakt worden. Indien het business domein in grote mate overeenkomt met het database datamodel is de structuur van de DTO's af te leiden uit de metadata van de database. Dit maakt generatie relatief eenvoudig.
Ten slotte is er de grafisch user Interface (GUI) waarbinnen codegeneratie kan worden toegepast. Op basis van metadata (bijvoorbeeld afkomstig van database structuur of domein modellen) kunnen GUI-formulieren en (databound) controls worden gegenereerd.

Analyse

Onafhankelijk van de plek van toepassen, moet een analyse leiden naar de juiste templates en metadata vanuit het probleemdomein. Voor het genereren van DALs en DTOs of GUI forms voor data georiënteerde applicaties is de analyse relatief eenvoudig (sterke overeenkomst met de structuur van de onderliggende database). Maar wanneer je vaker gebruik maakt van codegeneratie, zul je merken dat binnen meer onderdelen van applicaties een zich herhalende structuur herkenbaar is.
Enkele voordelen van codegeneratie op een rijtje:

- Veel herhaaldelijk (saai) werk wordt uit handen genomen
- Kans op fouten neemt af
- Mogelijkheid om gebruik te maken van 'proven' technologie
- Tijdwinst
- Gelijksoortige functionaliteit wordt op dezelfde manier uitgecodeerd (nette consequente herkenbare structuur)
- Standaard coding structuur en werkwijze wordt (deels) afgedwongen
- Mogelijkheid om relatief snel structurele wijzigingen door te voeren

Bij complexere applicatiestructuren moet worden afgewogen of de voordelen opwegen tegen de nadelen van de overhead van het moeten ontwikkelen en onderhouden van de templates, metadata en de codegeneratie engine (er zijn ook standaard generatoren, daarover straks meer). Wat meespeelt is of de codegeneratie alleen voor de huidige applicatie of ook voor andere applicaties in andere projecten toegepast kan worden. Een codegenerator zou een belangrijk onderdeel kunnen uitmaken van een framework dat van project naar project kan worden meegenomen en continue verbeterd. Codegeneratie kan de algemene structuur voor elke nieuwe applicatie neerzetten en waar mogelijk een aantal applicatieonderdelen uitgenereren. Verdere uitleg hiervan valt buiten de scope van dit artikel.
Alternatieven voor codegeneratie kunnen zijn: overerving en/of reflection en/of handmatig uitcoderen. De bijbehorende ontwikkeltijd, performance en onderhoudbaarheids voor- en nadelen bepalen per situatie wat de beste keuze is.

Codegeneratie in .NET

Er zijn verschillende manieren van codegeneratie. Iedere manier heeft zijn voor- en nadelen. We onderscheiden de volgende vier typen:

- Brute force
- CodeDom
- Bestaande generatoren (bijvoorbeeld: T4/XSLT)
- Bestaande generator in combinatie met kant-en-klare templates.
- Voorbeelden zijn: .NetTiers in combinatie met CodeSmith of NHibernate. Ook Software Factories maken in grote mate gebruik van codegeneratie.

Brute force

Brute force codegeneratie is de meest basale manier van codegeneratie. De ontwikkelaar ontwerpt en bouwt een eigen codegenerator inclusief de bijbehorende template(s). Aan de hand van één deze templates schrijft de generator direct code in een bestand. Deze methode is snel en volledig toegesneden op het gewenste resultaat, maar het betreft hier nieuwbouw van een codegenerator, de generator moet dus helemaal zelf worden ontwikkeld en getest.

CodeDom

CodeDom staat voor Code Document Object Model en is een onderdeel van het .Net Framework. CodeDom is een manier om een stuk code te beschrijven door middel van een boomstructuur. Er bestaat een root element waaronder CodeDom namespaces worden gedefinieerd. Iedere namespace kan dan een lijst van type declaraties bevatten. Op deze manier wordt de structuur van de code die gegenereerd moet worden, opgezet. Het genereren via CodeDom is de meest complexe optie van de vier methoden. Dit komt doordat het opzetten van de boomstructuur in niets lijkt op de standaardmanier van ontwikkelen. Wanneer deze boomstructuur is opgezet, kan de code snel en eenvoudig worden gegenereerd. Normaal gesproken zal bij codegeneratie niet snel voor CodeDom gekozen worden. Er zijn op het moment eenvoudigere en snellere methoden beschikbaar die hetzelfde resultaat opleveren.

T4

De term T4 staat voor Text Template Transformation Toolkit. Deze toolkit wordt door Microsoft gebruikt om binnen Visual Studio onder andere code te genereren bij het aanmaken van nieuwe projecten,

classes, forms en dergelijke. De toolkit werd tot voor kort apart geleverd maar is inmiddels geïntegreerd in Visual Studio 2008.
Om met T4 code te genereren ontwikkelt de gebruiker een template met de extensie .tt. Visual Studio 2008 herkent de template en genereert automatisch de code als de template wordt opgeslagen en maakt hiervoor automatisch een nieuw bestand aan.

.NetTiers in combinatie met CodeSmith

Er zijn kant en klare oplossingen voor codegeneratie op de markt. Een voorbeeld hiervan is .NetTiers in combinatie met CodeSmith. De CodeSmith codegeneratie engine kan overweg met NetTiers templates. De templates beschrijven de structuur van een aantal veelvoorkomende applicatie-onderdelen. In de templates zijn diverse best-practices/design-patterns verwerkt. Een bestaande (MS SQL) applicatie-database levert de metadata voor de generatie.
Aan de hand van deze database worden de DAL, de DTO's en eventueel een beheer GUI gegenereerd.

Codegeneratie voorbeeld (T4)

Het volgende voorbeeld van codegeneratie met behulp van T4 is bedoeld om te laten zien hoe codegeneratie werkt. In codevoorbeeld 1 is een T4 template te zien. De template beschrijft een functie die gegevens uit een database ophaalt en schrijft deze vervolgens naar de console.

```
<#@ template language="C#" #>

//Deze code is gegenereerd met behulp van T4.

using System;
using System.Data.SqlClient;

public class <#= this.ClassName #>
{
   public static void RunExample()
   {
         SqlConnection sqlConnection = new
SqlConnection("<#= this.ConnectionString #>");
         sqlConnection.Open();

     SqlCommand sqlCommand = new SqlCommand("<#= this.
Query #>", sqlConnection);
          SqlDataReader sqlDataReader = sqlCommand.
ExecuteReader();

         while (sqlDataReader.Read())
         {
                  Console.WriteLine("Name:  " +
(string)sqlDataReader["Name"] + " Salary: " + Salary.
ToString());
         }

         sqlConnection.Close();
   }
}

<#+
string ClassName = "T4Example";
string ConnectionString = "Server=localhost;Database
=T4Example;Uid=T4User;Pwd=T4Password;";
string Query = "Select [Name], Salary From Employee
Where Salary > 2000";
#>
```

Codevoorbeeld 1

```
//Deze code is gegenereerd met behulp van T4.
using System;
using System.Data.SqlClient;

public class T4Example
{
   public static void RunExample()
   {
         SqlConnection sqlConnection = new SqlConne
ction("Server=localhost;Database=T4Example;Uid=T4Use
r;Pwd=T4Password;");
         sqlConnection.Open();

       SqlCommand sqlCommand = new SqlCommand("Select
[Name], Salary From Employee Where Salary > 2000",
sqlConnection);
          SqlDataReader sqlDataReader = sqlCommand.
ExecuteReader();

         while (sqlDataReader.Read())
         {
                  Console.WriteLine("Name:  " +
(string)sqlDataReader["Name"] + " Salary: " + Salary.
ToString());
         }

         sqlConnection.Close();
   }
}
```

Codevoorbeeld 2

T4 templates stellen je in staat om op eenvoudige wijze code te genereren. Er kan zowel in VB als C# code worden gegenereerd. Hieronder is gekozen voor een C# oplossing.

Als we naar de template kijken, zien we dat deze een taal specificeert d.m.v. het 'language' attribuut. Deze taal specificeert de taal van de template, niet die van de uiteindelijke code. Wanneer we niet aangegeven wat de taal van de resulterende code is, wordt aangenomen dat deze overeenkomt met de taal van de template. Vervolgens zien we dat de template bestaat uit regels code die eruit zien als normale regels code. Deze code wordt één op één overgenomen in de gegenereerde code.

De code die tussen de tekens <#= en #> staat, is code die wordt vervangen door de waarde die het tussenliggende statement oplevert. In dit geval wordt verwezen naar waarden van variabelen die gedefinieerd staan tussen de <#+ en #>.

Wanneer er code wordt gegenereerd uit de bovenstaande template resulteert dit in code voorbeeld 2. In de resulterende code is duidelijk te zien dat de plekken waar verwijzingen naar variabelen stonden in de template zijn vervangen door de waarde van deze variabelen.

Hulpmiddel

Codegeneratie kan een uitstekend hulpmiddel zijn bij ontwikkeltrajecten. Het is echter geen 'silver bullet'. Niet iedere situatie leent zich voor codegeneratie. Een ontwikkelaar/architect dient zorgvuldig de afweging te maken of codegeneratie lonend is. «

Referenties

Code Generation in Microsoft .Net, Kathleen Dollard
Codegeneratie met DSL, Ronald Kroon
www.codegeneration.net
www.codesmithtools.com
www.nettiers.com
www.hibernate.org