Dimitri Holsteens

Neem de controle over jouw entiteiten in eigen handen

WERKEN MET CUSTOM ENTITIES IN HET ADO.NET ENTITY FRAMEWORK

Met het ADO.NET Entity Framework introduceert Microsoft een eerste versie van een Object-Relational Mapping-tool die geschikt is voor de complexe modellen die enterprise applicaties vaak vereisen. In dit artikel gaat de auteur dieper in op de mogelijkheden van het Entity Framework.

Zoals Eric van Wijk al in .NET Magazine #17 beschreven heeft, berust de werking van het Entity Framework op drie modellen die in XML gedefinieerd worden. Het conceptuele model beschrijft de entiteiten uit het applicatiedomein en hun onderlinge relaties. Het storage-schema definieert het onderliggende databasemodel en de mapping-specificatie beschrijft hoe de eerste twee modellen zich tot elkaar verhouden. Voor een diepgaande beschrijving van de verschillende modellen verwijs ik graag nogmaals naar dat artikel.
Het Entity Framework is niet samen met het .Net Framework 3.5 en Visual Studio 2008 uitgebracht, maar zal waarschijnlijk in het tweede kwartaal van 2008 als add-on beschikbaar worden gemaakt. In december zijn de bèta 3 van het Entity Framework en de CTP 2 versie van de Entity Designer - de design time-omgeving voor Visual Studio 2008 – uitgebracht. De designtool maakt het mogelijk met behulp van een simpele wizard de verschillende modellen vanuit een databaseschema te laten genereren. De drie schema's zitten in dit geval samen in een edmx-file en de designer zorgt voor gegenereerde classes die in overeenstemming zijn met de entiteiten uit het conceptuele model.
Deze rapid application development-techniek is echter voor vele applicaties niet toepasbaar. Applicaties met een al bestaand object-model hebben geen baat bij deze code-generation en niet zelden is een vorm van controle over het domainmodel nodig, die niet bereikt kan worden met de designer generated classes. Gelukkig biedt het Entity Framework verschillende mogelijkheden om het domeinmodel te controleren.

Partial classes en methods

De designer generated classes bevatten alleen de properties die in het conceptuele model beschreven zijn en een static factory-method om objecten te instantiëren. Alle objecten worden echter in de vorm van partial classes gegenereerd. Hierdoor wordt het mogelijk om, zoals we dit al gewoon zijn voor designer generated code, op simpele wijze bijkomende functionaliteit toe te voegen aan de objecten.
Om extra logica toe te voegen aan de setters van de properties kan er gebruikgemaakt worden van de in .NET 3.5 nieuw geïntroduceerde techniek van partial methods. Dit zijn methods waarvan de declaratie zich in één deel van de partial class bevindt, en de implementatie in het andere deel. De gegenereerde classes bevatten voor elke property een declaratie voor een *On<Property>Changing*- en *On<Property>Changed*-method. In de gegenereerde code worden deze methods aangeroepen in de setters van de properties en op deze wijze kunnen we dus ingrijpen in de setters. Codevoorbeeld 1 toont de gegenereerde code voor een property en hoe bijkomende functionaliteit hieraan toegevoegd kan worden via de partial methods.
Hierbij moet wel opgemerkt worden dat het in Entity Framework-versie 1 niet mogelijk zal zijn om een onderscheid te maken tussen 'user code calls' en 'framework calls'. De logica die in de partial methods geïmplementeerd is, zal, zowel bij het ophalen van een object door het framework, als bij het invullen van de properties door de applicatiecode, goed worden uitgevoerd.

Overerven van EntityObject of ComplexObject

De kern van het Entity Framework bestaat uit de Object Services-component. Dit is de laag die instaat voor het ophalen van data in de vorm van CLR-objecten en het terug opslaan van wijzigingen aan deze objecten. Om deze component zijn werk te laten doen, legt het Entity Framework wel bepaalde eisen op aan het object-

```
/** Designer Generated Class **/
public string Title
{
   get
   {
      return this._Title;
   }
   set
   {
      this.OnTitleChanging(value);
      this.ReportPropertyChanging("Title");
      this._Title = StructuralObject.SetValidValue(value, false, 100);
      this.ReportPropertyChanged("Title");
      this.OnTitleChanged();
   }
}
private string _Title;
partial void OnTitleChanging(string value);
partial void OnTitleChanged();

/** Eigen class **/
partial class Course
{
   partial void OnTitleChanging(string value)
   {
      if (null==value || value.Length == 0) throw new
      ApplicationException("Title Required");
   }
}
```

Codevoorbeeld 1.

model. Wanneer we de gegenereerde code bekijken, kunnen we zien dat ze allemaal overerven van EntityObject of ComplexObject. ComplexObject is de base-class voor wat in 'Domain Driven Design'-termen value objects zijn. Hiermee worden objecten bedoeld, die geen eigen identiteit of key hebben. In het Entity Framework worden deze types 'Complex Types' genoemd.
Wanneer we onze eigen classes willen laten integreren met het EF kunnen we ook gebruikmaken van deze overervingtechniek om aan een aantal van de basisvereisten van het Entity Framework te voldoen. Overerven van EntityObject of ComplexObject is echter niet voldoende, het EF verwacht immers ook dat de class, properties en relaties met de nodige codeattributen gemarkeerd worden. Types of members waarop geen attributen zijn aangebracht, blijven onzichtbaar voor het EF, ook al staan ze gedefinieerd in het conceptuele schema. Tabel 1 bevat een overzicht van de verschillende attributen en hun toepassing.

Attribute Type	Omschrijving
EdmSchemaAttribute	Aanduiding op assembly-niveau die aangeeft dat de assembly classes bevat, die overeenkomen met entiteiten zoals gedefinieerd in een conceptueel schema. Wanneer dit attribuut niet aanwezig is, zal EF de assembly negeren.
EdmComplexType	Aanduiding voor een complex object
EdmEntityType	Aanduiding voor een entity object
EdmScalarProperty	Aanduiding voor properties van een primitive type op een entiteit. Hiermee kan aangegeven worden of de property nullable is en of ze al dan niet deel uitmaakt van de entity key.
EdmComplexProperty	Aanduiding voor properties van een complex type op een entiteit
EdmRelationship-NavigationProperty	Aanduiding voor properties waarmee aangegeven wordt dat de property deel uitmaakt van een associatie tussen entiteiten en de rol van de entiteit beschrijft in deze associatie.
EdmRelationShip	Aanduiding op assembly-niveau die een associatie tussen entiteiten beschrijft. Zowel de rol als de multipliciteit worden hierin vastgelegd. Deze associatie moet eveneens in het conceptuele schema gedefinieerd zijn.

Tabel 1. Overzicht van de verschillende EF-attributen

Momenteel is er nog geen designer-ondersteuning voor niet-gegenereerde classes. Voor wie niet de volledige conceptuele, storage- en mapping-schema's manueel wil definiëren, is het wel mogelijk om eenmalig via de designer een edmx-file aan te maken dat gebaseerd is op een databaseschema, vervolgens de code-generation uit te schakelen en manueel de xml aan te passen om de modellen naar de eigen classes te laten wijzen.

De IPOCO-aanpak

Voor scenario's waarbij het niet mogelijk of wenselijk is een inheritance-tree op te zetten met aan de basis een EF-object, heeft het ADO.NET-team de zogenaamde IPOCO-aanpak voorzien. IPOCO staat voor een concept waarbij classes niet langer van een bepaalde base-class dienen over te erven, maar wel dat ze één of meerdere interfaces kunnen of moeten implementeren om door het persistence framework herkend te worden. Daarnaast blijft het nodig om de nodige metadata te verstrekken door middel van de attributen zoals hierboven is beschreven.
Er zijn drie interfaces, een verplichte en twee optionele, die geïmplementeerd kunnen worden om objecten EF-compatibel te maken:

IEntityWithChangeTracker

Deze interface voorziet in een enkele method, de SetChangeTracker-method, en laat het Framework toe een IChangeTracker aan de entiteit te koppelen. IChangeTracker definieert de methods die vanuit de property setters van de properties op de entiteiten aangeroepen moeten worden, wanneer de waarden gewijzigd worden. Change tracking vormt één van de belangrijkste peilers van het EF en zorgt er voor dat steeds alleen de gewijzigde entiteiten en vaak zelfs alleen de gewijzigde properties van die entiteiten naar de database worden weggeschreven. Als het Entity Framework zonder change tracking niet kan functioneren, is de implementatie van deze interface verplicht. Codevoorbeeld 2 toont hoe IEntityWithChangeTracker op eenvoudige wijze geïmplementeerd kan worden. Tijdens het instantiëren van een entiteit zal het Entity Framework het ChangeTracker-object aan de entiteit koppelen. Het enige wat in de code moet gebeuren, naast het bijhouden van de referentie, is expliciet in de setters van properties de EntityMemberChanging-method aanroepen op de ChangeTracker.

IEntityWithRelationships

Deze interface moet alleen geïmplementeerd worden voor entiteiten die relaties met andere entiteiten hebben. Ze voorziet in een read-only-property, waarmee een RelationshipManager-instance beschikbaar wordt gesteld. Deze RelationshipManager wordt door

```
[EdmEntityType(NamespaceName = "IPOCOEntities", Name = "Person")]
public class Person : IEntityWithChangeTracker
{
  private IEntityChangeTracker _changeTracker = null;

  private void reportEntityMemberChanging(string memberName)
  {
    if (null != _changeTracker)
    {
      _changeTracker.EntityMemberChanging(memberName);
    }
  }

  private void reportEntityMemberChanged(string memberName)
  {
    if (null != _changeTracker)
    {
      _changeTracker.EntityMemberChanged(memberName);
    }
  }

  [EdmScalarProperty()]
  public string FirstName
  {
    get
    {
      return _firstName;
    }
    set
    {
      reportEntityMemberChanging("FirstName");
      _firstName = value;
      reportEntityMemberChanged("FirstName");
    }
  }

  void IEntityWithChangeTracker.SetChangeTracker(IEntityChangeTracker
      changeTracker)
  {
    _changeTracker = changeTracker;
  }
}
```

Codevoorbeeld 2.

de Object Services-component gebruikt voor het navigeren over de associaties die in het conceptuele schema zijn gedefinieerd en via attributen zijn aangegeven in de assemblies. De RelationshipManager haalt gerelateerde entiteiten op in de vorm van een generische EntityCollection. Codevoorbeeld 3 laat zien hoe IEntityWithRelationships geïmplementeerd kan worden.

IEntityWithKey

De implementatie van deze interface is optioneel, maar biedt wel enkele belangrijke performanceverbeteringen wanneer ze geïmplementeerd wordt. Vooral in scenario's waarin de entiteiten complexe relaties hebben of scenario's waarbij entiteiten losgekoppeld en later weer aangekoppeld worden aan de objectcontext. Dit laatste is een veelvoorkomend scenario voor omgevingen, waarbij entiteiten via services aan verbruikers aangeboden worden. Vaak wordt een entiteit in dit geval losgekoppeld van de context, doorgegeven aan de verbruiker, in een volgende operatie opnieuw aangeboden door de verbruiker en opnieuw aan een EF-context gebonden. Wanneer de interface niet geïmplementeerd wordt, zal het Entity Framework telkens opnieuw een entity key bepalen uit de metadata-files, hetgeen een zware performance-impact kan hebben, zeker wanneer de modellen een aanzienlijke omvang beginnen te krijgen. Desondanks stelt de implementatie van deze interface zeer weinig voor. Zoals in codevoorbeeld 4 wordt getoond, volstaat het om in een enkele property en een field van type EntityKey te voorzien. Het framework zal zelf de EntityKey opbouwen aan de hand van de metadata en attributen en deze via de property op de entiteit zelf bijhouden.

Op weg naar Persistence Ignorance?

Al bij de eerste CTP's heeft het designteam vanuit de community opmerkingen gekregen over de eisen die opgelegd werden aan de entity-classes. Hetgeen niet verwonderlijk is aangezien Persistence Ignorance een belangrijke eigenschap is van domeinobjecten in Domain Driven Design. De introductie van de IPOCO-aanpak mag dan ook als een eerste stap in de richting van verdere loskoppeling van entiteiten en het EF gezien worden. In toekomstige versies van het EF zal getracht worden de implementatie van alle interfaces en de markering met attributen optioneel te maken, ondanks dat dit mogelijk een negatieve impact heeft op de performance van het EF. Op die manier zullen gebruikers zelf de trade-off kunnen maken tussen het Persistence Ignorant houden van hun domeinmodel en de performance van het systeem. Zover zijn we echter nog niet. Voor EF-versie 1.0 zullen de mogelijkheden voor custom-objecten beperkt blijven tot hetgeen hierboven beschreven is.

```
[EdmEntityType(NamespaceName = "IPOCOEntities", Name = "Person")]
public class Person : IEntityWithRelationships
{
  private RelationshipManager _relationShipManager = null;

  [EdmRelationshipNavigationProperty("IPOCOEntities",
    "CourseInstructors", "Course")]
  public EntityCollection<Course> Courses
  {
    get
    {
      return ((IEntityWithRelationships)(this)).RelationshipManager.
        GetRelatedCollection<Course>("CourseInstructors", "Course");
    }
  }

  #region IEntityWithRelationships Members
  RelationshipManager IEntityWithRelationships.RelationshipManager
  {
    get
    {
      if (null == _relationShipManager)
        { _relationShipManager =
          RelationshipManager.Create((IEntityWithRelationships)this);
        }
      return _relationShipManager;
    }
  }
  #endregion
}
```

Codevoorbeeld 3.

```
[
EdmEntityType(NamespaceName = "IPOCOEntities", Name = "Person")]
public class Person : IEntityWithKey
{
  private int _personId = 0;
  private EntityKey _key = null;

  [EdmScalarProperty(EntityKeyProperty = true, IsNullable = false)]
  public int PersonID
  {
    get
    {
      return _personId;
    }
    set
    {
      reportEntityMemberChanging("PersonID");
      _personId = value;
      reportEntityMemberChanged("PersonID");
    }
  }
  #region IEntityWithKey Members

  System.Data.EntityKey IEntityWithKey.EntityKey
  {
    get
    {
      return _key;
    }
    set
    {
      reportEntityMemberChanging("-EntityKey-");
      _key = value;
      reportEntityMemberChanged("-EntityKey-");
    }
  }
  #endregion
}
```

Codevoorbeeld 4.

Dimitri Holsteens is werkzaam als softwarearchitect bij Compuware België (www.compuware.be), waar hij zich bezighoudt met het ontwerpen en omwikkelen van distributed applications. Zijn e-mail is dimitri.holsteens@compuware.com

Referenties

ADO.NET Team Blog: http://blogs.msdn.com/adonet/default.aspx
Entity Framework Bèta 3: http://msdn2.microsoft.com/nl-be/data/aa937695(en-us).aspx
Danny Simmons(developer ADO.NET Team) Blog: http://blogs.msdn.com/dsimmons/default.aspx