Anko Duizer is trainer/coach bij Class-A

Transacties worden eenvoudig!

NIEUW TRANSACTIEPROGRAMMEERMODEL VEREENVOUDIGT HET SCHRIJVEN VAN GEDISTRIBUEERDE CODE

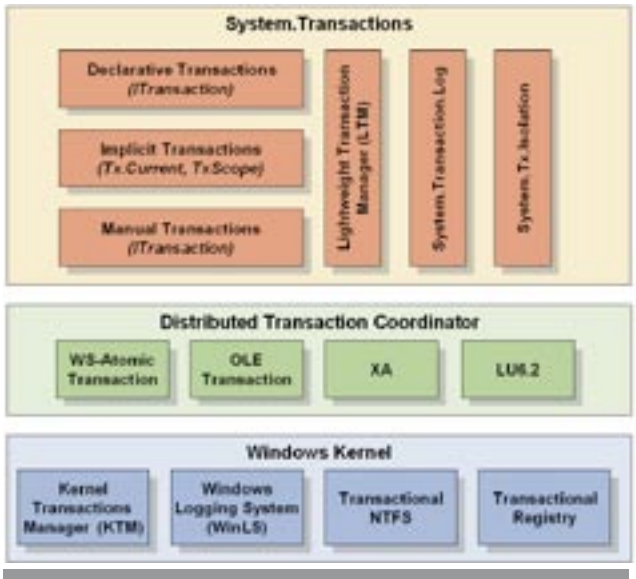
Met de komst van Visual Studio 2005 krijgen we een nieuw transactiemodel: System.Transactions. Dit is een eenduidig en eenvoudig programmeermodel dat diverse functionaliteiten biedt die het leven van een .NET-ontwikkelaar eenvoudiger maken. Wat zijn de mogelijkheden van dit nieuwe programmeermodel? Maak kennis met System.Transactions als transactieprogrammeermodel [1].

Transacties zorgen voor eenvoud in de code. Een belangrijk aspect van transacties is het 'alles of niets'-principe. Het geheel van acties is succesvol, of alles wordt teruggedraaid. Zeker in de wereld van gedistribueerde componenten zorgt dit voor structuur en eenvoud. In .NET zijn er momenteel twee mogelijkheden: System.EnterpriseServices of een 'provider' die System.Data.IDbTransaction implementeert. De huidige transactiemodellen hebben nadelen. Bij het eerste is het noodzakelijk te erven van ServicedComponent. Hierdoor is COM+-registratie noodzakelijk. Er bestaan mogelijkheden voor 'Services without components' waardoor er niet per se COM+-registratie noodzakelijk is. Bij het tweede transactiemodel is er uitsluitend support voor lokale transacties in de desbetreffende resource manager. Bovendien wijken de modellen sterk van elkaar af. Niet alleen de implementatie maar de programmeermodellen zijn sterk afwijkend.

Wat is System.Transactions?

System.Transactions is een nieuwe framework library die helpt om gedistribueerde, transactiegebaseerde applicaties te ontwikkelen. In de lijst staan de belangrijkste features:

- een eenvoudig te gebruiken programmeermodel
- een lichtgewicht transactiemanager
- automatische promotie van transacties



Afbeelding 1. Overzicht van System.Transactions

In afbeelding 1 staan schematisch de onderdelen van System.Transactions. Met de komst van Visual Studio 2005 worden de meeste blokken ingevuld. De volgende blokken ontbreken in zeker in Visual Studio 2005: Declarative Transactions, Kernel Transaction Manager, System.Tx.Isolation en System.Tx.Log.

In de afbeelding is te zien dat de verschillende programmeermodellen worden ondersteund op basis van dezelfde onderliggende technologie. Het model met de declaratieve attributen vertoont veel overeenkomst met het Enterprise Services-model zoals dat vandaag beschikbaar is. De andere twee zijn nieuw en zijn onderwerp van dit artikel. Zowel de impliciete als handmatige transacties hebben een plaats in het gedistribueerde applicatielandschap. Ze hebben onderliggend dezelfde technologie en interfaces. Het handmatige model biedt echter de meeste flexibiliteit. Het impliciete model heeft de eenvoud als belangrijk voordeel. System.Transac-

```
using System;
using System.Data.SqlClient;
using System.Transactions;

namespace ClassA {

   public class ImplicitTransaction {

      public static void Main() {
        using(TransactionScope _transactionscope1 =
                new TransactionScope()) {
          using(SqlConnection _sqlconnection1 =
                  new SqlConnection(ConnectionString)) {
            _sqlconnection1.Open();

            // do your work here...

            }
          _transactionscope1.Consistent = true;
        }
      }
   }
}
```

Codevoorbeeld 1.
Raamwerk voor impliciete transacties

(1) De codevoorbeelden in dit artikel werken op Windows XP Service Pack 2 en Windows 2003. Zorg er voor dat 'network access' aanstaat voor de Distributed Transaction Coordinator (DTC).

Afbeelding 2. Expliciet transactiemodel

tions steunt op geen enkele wijze meer op COM(+) interop. Er is een native library voor zowel de Lightweight Transaction Manager (LTM) als de Distributed Transaction Coordinator (DTC).

Impliciete transacties

Het impliciete transactiemodel is nieuw en niet te vergelijken met de huidige beschikbare modellen. Hoe het model er uitziet en werkt is het beste uit te leggen aan de hand van codevoorbeeld 1.

Het gebruik van de *using*-constructie springt het meest in het oog. Dit is dan ook meteen het meest cruciale onderdeel van het impliciete programmeermodel. Het is onmiddellijk duidelijk wat de *scope* van de transactie is door het gebruik van deze programmeerconstructie; alles binnen de *using*-constructie. Alleen om deze reden is dat al een schitterend model en is het gebruik hiervan aan te raden. De achterliggende reden is dat deze constructie ervoor zorgt dat er wordt 'Disposed'. Transacties zorgen voor locks op de database. Wanneer we vertrouwen op de Garbage Collector om objecten op te ruimen en resources vrij te geven, kan dit wel eens tot een vervelende situatie leiden. Door het gebruik van de *using*-constructie weten we zeker dat een *Commit* of *Rollback* wordt aangeroepen. Zodra de *using*-scope eindigt wordt namelijk *TransactionScope.Dispose* aangeroepen, wat zorgt voor een goede afhandeling van de transactie.

Binnen de *using* wordt een nieuwe transactiescope aangemaakt: *new TransactionScope()*. Deze class kent een *constructor* met veertien 'overloads'. Voor dit artikel gaat het te ver om ze alle veertien te beschrijven. Belangrijk om te weten is dat dit het punt is waar invloed kan worden uitgeoefend op de transactie. Wanneer er niets wordt meegegeven is de default constructor: *TransactionScope (TransactionOption.Required)*. Iedereen die bekend is met het huidige declaratieve programmeermodel zal deze optie herkennen. Het betekent dat alles binnen de *scope* altijd in een transactie draait. Wanneer het nodig is wordt een nieuwe transactie gestart. Ook het openen van de connectie naar de SQL Server database gebeurt via een *using*-constructie. Zodra de methode *Open()* wordt aangeroepen, wordt de connectie onderdeel van de transactie (enlistment). Het statement *_transactionscope1.Consistent = true* geeft het signaal af dat een *commit* moet worden uitgevoerd zodra de *TransactionScope.Dispose* wordt aangeroepen. Zodra de transactie uit de *scope* komt, controleert de transactiecoördinator de *consistent*-property en besluit op basis daarvan tot een *commit* of *rollback*.

Wat gebeurt er wanneer binnen de *scope* een fout (exception) optreedt? Wanneer de exceptie niet in de code wordt afgehandeld, wordt allereerst de Dispose van de *SqlConnection* aangeroepen. Daarna is de Dispose van de *TransactionScope* aan de beurt. Deze kijkt naar de property *consistent*, die standaard op *false* staat. De standaard waarde van de property *consistent* zorgt voor een robuust programmeermodel. Wanneer een fout optreedt in

```
using System;
using System.Data.SqlClient;
using System.Transactions;
using System.Transactions.Ltm;

namespace ClassA {

  public class ExplicitTransaction {

    public static void Main() {
      ICommittableTransaction _icommittabletransaction1 =
      Transaction.Create();
      using(SqlConnection _sqlconnection1 =
      new SqlConnection(ConnectionString)) {
        _sqlconnection1.Open();
        _sqlconnection1.EnlistTransaction((ITransaction)_
        icommittabletransaction1);

        // do your work here...

      }
      _icommittabletransaction1.Commit();
    }
  }
}
```

Codevoorbeeld 2.
Expliciete transacties

de code staat de waarde namelijk op *false*, waardoor de transactie wordt teruggedraaid. De property moet dan ook op het laatste moment in de *using*-scope op *true* worden gezet. Dit heeft tot gevolg dat de transactie wordt teruggedraaid. Het impliciete transactieprogrammeermodel is eenvoudig en robuust. De verwachting is dat dit het meest toegepaste model gaat worden binnen System.Transactions.

Expliciete transacties

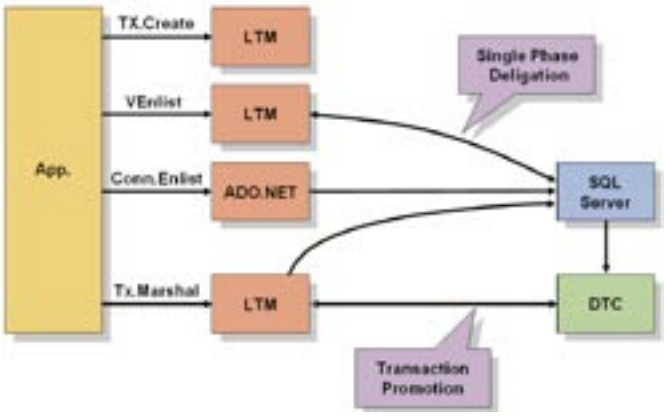
Een tweede model is het expliciete ofwel handmatige model. In dit model bestaat maximale controle over de transactie. In de basis wordt door de ontwikkelaar zelf in de code handmatig geregeld dat bijvoorbeeld een connectie onderdeel wordt van een transactie. Hiervoor zijn interfaces en classes beschikbaar. Deze zijn schematisch weergegeven in afbeelding 2.

Het expliciete programmeermodel biedt maximale flexibiliteit. De prijs die hiervoor moet worden betaald, is de foutgevoeligheid. Het is aan de ontwikkelaar om expliciet een *commit* of *rollback* uit te voeren. De controle ligt dus volledig bij de ontwikkelaar. Bovendien is het niet eenvoudig om robuuste transactiecode te ontwerpen!

In codevoorbeeld 2 staat een (onvolledig) stuk C#-code waar een handmatige transactie wordt gecreëerd. Het eerste interessante statement is *Transaction.Create()*. Deze aanroep geeft een *ICommittableTransaction*-interfaceverwijzing terug. In afbeelding 2 is duidelijk dat dit de enige interface met een *commit* is. Het basisidee is dat diegene die de transactie aanmaakt de enige is die tot een *commit* kan beslissen. Anderen kunnen onderdeel worden van de transactie, maar de controle ligt bij de creator. Andere partijen die onderdeel willen worden van de transactie krijgen een *ITransaction*-verwijzing door. Op het moment dat in de code een connectie wordt geopend naar SQL Server, wordt deze niet automatisch onderdeel van de transactie. Hiervoor moet expliciet een enlist worden uitgevoerd.

Promotie van transacties

In het nieuwe transactieframework is een zogenaamde LTM (Lightweight Transaction Manager). Dit is een volwaardige 'in-memory'



Afbeelding 3. Promotie van transacties

transactiemanager, een zeer goedkope en optimale transactiemanager. Lightweight-transacties kunnen volledig automatisch worden gepromoot naar een volwaardige DTC-transactie. Dit gaat volkomen transparant. Het transactieframework heeft altijd de voorkeur voor een lightweight-transactie. Standaard wordt bij een nieuwe transactie dan ook altijd een lightweight-transactie aangemaakt. Indien nodig wordt deze dan gepromoot. Dit staat weergegeven in afbeelding 3.

Code die momenteel gebruik maakt van Enterpise transactions heeft ook voordeel van de LTM. De onderliggende code gaat onder water namelijk gebruik maken van System.Transactions. De architectuur en het ontwerp van een applicatie bepalen of er echt voordeel gehaald kan worden door het gebruik van de LTM. Er is een speciale relatie tussen System.Transactions en SQL Server 2005. SQL Server 2005 heeft begrip van lightweight-transacties en kan deze verwerken. Wanneer een applicatie gebruikmaakt van SQL Server 2005 en transacties, kan de ontwikkelaar het lightweight-model volledig benutten. Dit leidt tot een veel betere performance van transacties in combinatie met SQL Server 2005. Zodra een andere resource manager wordt gebruikt, zoals Oracle of SQL Server 2000, wordt onmiddellijk gepromoot naar een DTC-transactie, ook voor de eventueel bestaande SQL Server 2005-acties binnen de transactiescope. Uiteindelijk is het de bedoeling dat promotie volkomen transparant is voor de ontwikkelaar. Het moet niet uitmaken of een transactie lokaal draait of via de DTC.

Transactieconfiguratie

Er zijn diverse parameters in het transactieframework die beïnvloed kunnen worden. Op die manier kan het transactiemechanisme volledig worden aangepast aan de situatie. Een voorbeeld hiervan is te zien in codevoorbeeld 3.

In codevoorbeeld 3 wordt een object *TransactionOptions* in leven geroepen. Met dit object kunnen diverse transactieparameters worden beïnvloed. In codevoorbeeld 3 wordt het *IsolationLevel* gezet. Dit is een belangrijke parameter die de schaalbaarheid en performance van een systeem sterk beïnvloedt. In het huidige transactiemodel kan het *IsolationLevel* niet worden gezet per transactiescope. Dit kan alleen gebeuren op het niveau van de DTC of per SQL-statement. Hier zijn dus nieuwe mogelijkheden. Er zijn vele andere parameters die op een vergelijkbare manier kunnen worden veranderd. In codevoorbeeld 3 is te zien dat de gezette waarde van de huidige transactie weer eenvoudig gelezen kunnen worden. Er is volledige transparantie.

Toekomstvast programmeermodel

De komst van System.Transactions in Visual Studio 2005 vereenvoudigt het programmeren van transacties. Er is een uniform model met diverse toepassingen. Het impliciete model is vermoedelijk het model dat het meest gebruikt gaat worden. Voor het

```
using System;
using System.Data.SqlClient;
using System.Transactions;

namespace ClassA {

  public class IsolationLevel {

    public static void Main() {
      TransactionOptions _transactionoptions1 =
      new TransactionOptions();
      _transactionoptions1.IsolationLevel =
      IsolationLevel.ReadCommitted;
      using(TransactionScope _transactionscope1 = new
           TransactionScope(TransactionScopeOption.Required,
           _transactionoptions1)) {
        ITransaction _currenttx = Transaction.Current;
        Console.WriteLine(_currenttx.Identifier);
        Console.WriteLine(_currenttx.ToString());
        Console.WriteLine(_currenttx.IsolationLevel);
        Console.WriteLine(_currenttx.Status);

        using(SqlConnection _sqlconnection1 = new
             SqlConnection(ConnectionString)) {
          _sqlconnection1.Open();

          // do your work here...

          }
        _transactionscope1.Consistent = true;
      }
    }
  }
}
```

Codevoorbeeld 3.
Veranderen van het IsolationLevel

handmatig of expliciete model is zeker een plaats wanneer maximale controle is gewenst. Het programmeermodel zoals dat is neergelegd in de diverse classes en interfaces ziet er toekomstvast uit. Het is uitgebreid, maar ook eenvoudig genoeg om nieuwe transactiemonitoren of -mechanismen te overleven. Dit moet robuuste code gaan opleveren. Voor welk model ook wordt gekozen, er is altijd profijt van het lightweigt-transactiemechanisme. Dit zorgt voor een veel betere performance! Tel daarbij op dat er geen COM(+)-objecten meer noodzakelijk zijn voor .NET-transacties en er is geen enkele reden meer om niet snel aan de slag te gaan met System.Transacties in .NET-applicaties.

Anko Duizer is werkzaam als trainer/coach bij Class-A (www.class-a.nl). Daarvoor heeft hij vijf jaar bij Microsoft gewerkt als consultant. Onder zijn klantenkring bevinden zich voornamelijk Top100-bedrijven in Nederland. Sinds begin 2001 is hij bezig met .NET. Speciale interesse heeft Anko voor de architectuur en het ontwerp van gedistribueerde databaseapplicaties en servicegeoriënteerde systemen. Zijn e-mailadres is anko.duizer@class-a.nl

Referenties

http://longhorn.msdn.microsoft.com/
http://www.winfx247.com/247reference/System/Transactions/System.Transactions.aspx
http://pluralsight.com/wiki/default.aspx/Don.TransactionAsks