Floris Zwarteveen is werkzaam bij Fidit

Data binding revival

HOE EEN LELIJK EENDJE TOCH EEN ZWAAN WERD

Eerlijk zeggen, wie trekt nu al zijn neus op bij het lezen van het begrip data binding? Elke programmeur die het tijdperk meegemaakt heeft van de 2-tier data bound controls heeft waarschijnlijk, net als ik, nog nachtmerries van applicaties die ofwel niet vooruit te branden waren of volstrekt niet onderhoudbaar. Gelukkig is er veel veranderd sinds die tijd.

Programmeurs zijn software-engineers geworden en bij engineering passen design patterns zoals Observer of Model-View-Controller. Maar Microsoft bleef achter in de ondersteuning van al dit moois; tenzij je in C++ kon programmeren, maar dat is maar voor weinigen weggelegd. Met de komst van .NET en zeker met de 2.0-versie van het .NET Framework is hier verandering in gekomen. Zo veel dat zelfs de meest verstokte Java-engineers enthousiasme kunnen opbrengen voor Microsoft-technologie. In dit artikel wil ik laten zien hoe XML en XSD kinderlijk eenvoudig te combineren zijn met datasets via data binding. De oplossing is echter naar een nog hoger niveau te tillen wanneer de moeite genomen wordt om datasets te vervangen door 'echte' domeinclasses.

De DataSet-browser

Al verschillende versies van de Microsoft-gridcontrol heb ik me afgevraagd waarom het toch zo complex is om data eenvoudig van objecten naar een gridcontrol over te zetten en terug. Met behulp van database-binding was het een eitje, maar dat is voor mij nooit een optie geweest. Voor die generatie 2-tier client/server-aanpak heb ik nooit echt warm kunnen lopen en inmiddels is de algemene publieke IT-opinie om deze techniek niet meer te gebruiken; toch? Datasets vormen in .NET een redelijk acceptabele en krachtige interface naar de persistente data, zoals een database of XML-gestructureerde data. In dit geval maak ik gebruik van een simpele XML-file met daarin een deel van mijn iPod-collectie. Een deel hiervan kun je zien in codevoorbeeld 1.

Om optimaal gebruik te maken van de toegevoegde waarde van XML moet je de structuur beschrijven in een XML-schema. Codevoorbeeld 2 laat de bijbehorende XML-schema Definition (XSD) zien als metadefinitie van mijn iPod-collectie.
In een nieuw project maak ik een nieuwe Form met daarop onder andere een gridcontrol en 5 command buttons. Eén om een XML te openen en 4 buttons om door de dataset te browsen; zie alvast afbeelding 2. In het .NET Framework is de binding van data aan controls losgetrokken van de data en wordt afgehandeld door separate objecten. Elke Windows Form heeft hier een BindingContext en één of meer CurrencyManagers voor geïnstantieerd (voor elke datasource één CurrencyManager). De eerste keer dat ik geconfronteerd werd met CurrencyManagers heb ik me lang afgevraagd wat data binding met geld te maken heeft. Echter een vertaling van het Engelse begrip currency is ook: circulatie. En dat is waar deze managers erg goed in zijn: het beheren van de circulatie van data tussen bron en visualisatie. Windows Forms ondersteunen data binding met vrijwel elke control, en dankzij deze binding worden data-bound controls automatisch gesynchroniseerd met de dataobjecten waar ze aan verbonden zijn. In het algemeen geldt: wanneer een dataobject 'gebind' wordt aan een control krijgt de control een CurrencyManager toegewezen. Zie ook afbeelding 1 voor een schematische weergave hiervan.

Om de inhoud van het XML-bestand in een gridcontrol te tonen hoef ik alleen een dataset te instantiëren, een XML-file aan deze dataset aan te bieden en de dataset te verbinden ('binden') aan de gridcontrol. De code in codevoorbeeld 3 laat dit zien:
Het voordeel dat ik nu van de CurrencyManagers heb is onder andere positionering binnen de dataset. mCurrencyManager.Position geeft

```
<?xml version="1.0" standalone="yes"?>
<Albums xmlns="urn:xmlns:music-collection:my-ipod"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <Album Genre="Jazz">
      <Artist>Kenny Burrell</Artist>
      <Title>Midnight blue</Title>
      <RecordingDate>1967-04-21</RecordingDate>
   </Album>
   <Album Genre="Pop">
      <Artist>Tom Waits</Artist>
      ...
   </Album>
</Albums>
```

Codevoorbeeld 1. XML-data van mijn iPod-collectie

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema   xmlns:xs="http://www.w3.org/2001/XMLSchema"
             targetNamespace="BrowseableObjects"
             xmlns:alb="BrowseableObjects"
             elementFormDefault="qualified">
   <xs:element name="Albums">
      <xs:complexType>
         <xs:sequence>
            <xs:element  name="Album"
                         type="alb:Album"
                         maxOccurs="unbounded" />
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="Album">
      <xs:sequence>
         <xs:element name="Artist" type="xs:string" />
         <xs:element name="Title" type="xs:string" />
         <xs:element name="RecordingDate" type="xs:date" />
      </xs:sequence>
      <xs:attribute name="Genre" type="xs:string" />
   </xs:complexType>
</xs:schema>
```

Codevoorbeeld 2. XSD iPod-collectie

```
/ show a dialog to pick and choose an XML file
OpenFileDialog dlg = new OpenFileDialog();
dlg.DefaultExt = "XML";
dlg.Filter = "XML Files|*.XML";
dlg.Title = "Open an XML file for browsing";

// if an XML file has been selected bind it to a dataset
if (dlg.ShowDialog(this) == DialogResult.OK)
{
   mCurrentDataset = new DataSet();
   mCurrentDataset.ReadXml(dlg.FileName);

   // just for now assume:
   // we want to see the first table from the dataset
   string tabName = mCurrentDataset.Tables[0].TableName;
   this.dataGrid.DataSource = mCurrentDataset.Tables[tabName];

   // store this reference in a module level private
   // variable for later reuse
   mCurrencyManager = this.BindingContext[this.dataGrid.DataSource];
}
```

Codevoorbeeld 3.

```
/[System.SerializableAttribute()]
[System.Xml.Serialization.XmlTypeAttribute(
   Namespace = "urn:xmlns:music-collection:my-ipod")]
[System.Xml.Serialization.XmlRootAttribute(
   Namespace = "urn:xmlns:music-collection:my-ipod",
   ElementName = "Albums",  IsNullable = false)]
public class Albums : IBrowseableList
{
   private List<Album> albumList = new List<Album>();
   [System.Xml.Serialization.XmlIgnoreAttribute]
   public IList List
   {
      get { return this.albumList as IList; }
   }

   [System.Xml.Serialization.XmlElementAttribute("Album")]
   public List<Album> AlbumList
   {
      get { return this.albumList; }
   }
}
```

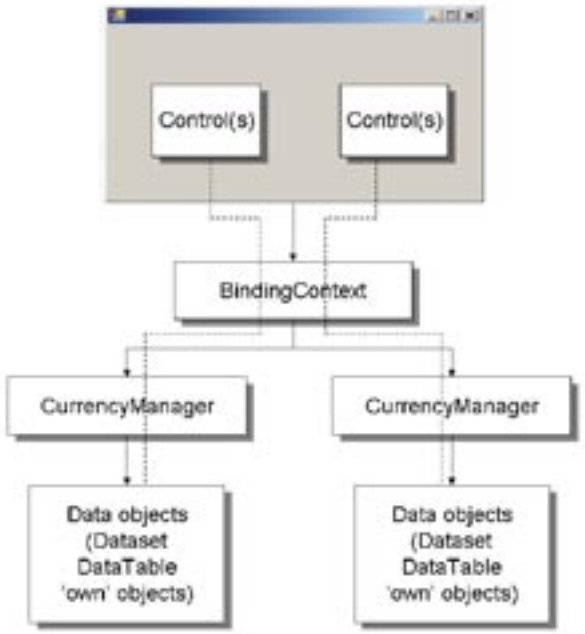
Codevoorbeeld 4. De Albums-class

de huidige positie in de verzameling. De First, Previous, Next en Last command buttons zijn dan ook eenvoudig te implementeren door de waarde van deze Position te variëren. Overigens hoef ik niet bang te zijn voor exceptions als ik een incorrecte waarde aanbiedt aan Position, de CurrencyManager controleert en reguleert zodat er alleen maar geldige waardes toegekend kunnen worden.

Een greep uit de functionaliteit die de CurrencyManager nog meer biedt:

- mCurrencyManager.AddNew om elementen toe te voegen
- mCurrencyManager.EndCurrentEdit om wijzigingen vanaf de user interface door te zetten naar de achterliggende dataset
- mCurrencyManager.CurrentChanged waar een delegate aan gehangen kan worden als het huidige element verandert
- mCurrencyManager.PositionChanged wanneer de huidige positie verandert; hetgeen heel handig kan zijn om master-detailrelaties te modelleren.

Het resultaat van de simpele browser is het window zoals getoond in afbeelding 2.



Afbeelding 1. Binding Context en Currency Managers

Objects als vervanging van datasets

Met het lezen van de vorige paragraaf heb ik de rechtgeaarde en dogmatische OO-engineer al horen denken. Waarom datasets? Waarom geen echte objecten? En inderdaad, dat vroeg ik me ook af toen ik aan het programmeren was. Dat moet toch beter kunnen. En gelukkig, het kan ook beter, veel beter zelfs, want ik ben ondanks de verbetering nog steeds geen groot voorstander van het gebruik van datasets:

- Te veel overhead vanwege het databasegeoriënteerde karakter van datasets.
- XML-serialization vanuit en naar een dataset is zeker niet het snelste mechanisme.
- Dataset is een native .NET-type en daarom niet te gebruiken buiten een Windows-omgeving.
- De zogenaamde 'impedance mismatch' tussen OO en tabellen vereist dat ik structuren toch altijd plat moet slaan naar tabelgestructureerde data.

Kortom het is tijd om de browser te herzien zodat deze overweg kan met getypeerde (domein) objecten.

Domeinclasses

Ik wil met mijn 'class-oplossing' natuurlijk geen achteruitgang ten opzichte van de dataset-oplossing. Dat betekent de volgende eisen:

- Ik moet een willekeurige XML-file aan kunnen bieden.
- Als de XML-file een verwijzing bevat naar een objectpresentatie moet de gridcontrol aan deze objectrepresentatie gebonden worden. Als de objectrepresentatie er niet is, kun je terugvallen op de eerder beschreven datasetmethode.

Het class-model behorend bij het XML-schema uit codevoorbeeld 2 is opgenomen in afbeelding 3.
Om het generieke gedrag netjes te modelleren is er voor gekozen een interface te definiëren die elke 'browseable' class moet implementeren. Dit geeft de mogelijkheid om in de toekomst eenvoudig nieuwe browseable classes toe te voegen. Vooralsnog is het alleen nodig om een property-list te implementeren met een verwijzing naar de lijst van elementen waardoor gebladerd kan worden. Om deze type-safe te maken is er gebruikgemaakt van generics. Een nadeel van het aanbieden van een (generieke) List op een interface is dat iedereen die de List gebruikt, ook zelf kan beslissen welke typen elementen er in gezet worden. In dit geval wil ik afdwingen dat er alleen elementen van het type Album aan de List toegevoegd worden; generics kunnen dit uitstekend reguleren. In het

codevoorbeeld 4 wordt getoond hoe de uiteindelijke Albums-class er uit ziet. In de Albums-class moet ik er voor zorgen dat ik zowel de IBrowseable-interface implementeer én dat ik de XML-serializer nog voor me kan laten werken. Vandaar de, op het eerste gezicht, vreemde constructie met de twee verschillende properties die feitelijk hetzelfde doen.

Dynamisch een class-naam bepalen

Ik wil in mijn browserapplicatie geen referenties naar de classes opnemen die ik ga browsen. Als ik namelijk in de toekomst nieuwe classes wil browsen, wil ik dit kunnen zonder dat ik de browser opnieuw compileer; dat hoef ik in het geval van de datasets toch ook niet. Voordat ik een XML-file kan bekijken, moet ik twee dingen weten:

- De naam van de class (namespace en class-name).
- De naam van de assembly waarbinnen de class-implementatie gevonden kan worden.

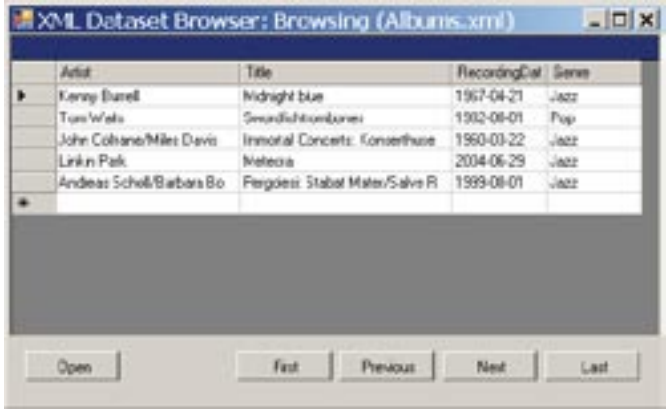
Om het eerste probleem op te lossen ben ik er van uitgegaan dat de naam van de class identiek is aan de naam in het XML-schema. Deze keuze is mede ingegeven omdat ik later het XML-schema wil gebruiken om de code voor de class-files te genereren. (Hierover in een later artikel meer.) In de XML-file heb ik de padspecificatie van het bijbehorende XML-schema in de attributes van het root-element opgenomen. In het XML-schema geeft het targetNamespace-attribuut de namespace van de class, en het name-attribuut van het eerste element de naam van de gewenste class. In dit geval: 'BrowseableObjects' en 'Albums'. Codevoorbeeld 5 laat zien hoe de uiteindelijke class-name bepaald wordt.
Dan ben ik ten slotte één stap verwijderd van het uiteindelijke type van de class. Ik weet nu alleen de class-name als string. Aan de constructor van een XmlSerializer moet ik echter het class-type meegeven. Hier is een eenvoudige 'truc' voor als je de naam van de assembly weet; zie codevoorbeeld 6.

De objectbrowser

De Albums-class, of ik moet eigenlijk zeggen een instantie van de Albums-class is nu eenvoudig 'bladerbaar'. De code wijkt eigenlijk weinig af van de code zoals we die kennen van de datasetvariant; zie codevoorbeeld 1. In codevoorbeeld 7 is de belangrijkste code van het initialiseren van de gridcontrol getoond; alleen de exception handling is weggelaten.

Het voordeel

Om het voordeel van deze aanpak volledig te benutten voeg ik een referentie toe naar de assembly met de definitie van het Albums-class-type. Als ik gebruik wil maken van echte businessobjecten immers, moet ik wel weten hoe ze er uit zien. Als ik dan bijvoorbeeld een nieuwe rij wil toevoegen, hoef ik geen databasegeoriënteerde code te schrijven; ik kan de mij meer vertrouwde Album-objecten manipuleren. Businesslogica is daardoor beter herbruikbaar geworden, de software meer gelaagd en de lagen hebben een betere scheiding van verantwoordelijkheden. In de bijgeleverde code is een voorbeeld opgenomen van een tooltip. Wanneer de rij verandert wordt de tooltiptekst aangepast. Op basis van de huidige positie van de CurrencyManager kan het huidige Album-object eenvoudig bepaald worden met:



Afbeelding 2. Bladeren door XML-data

```
int pos = mCurrencyManager.Position;
Album album = mCurrentBrowseableList.List[pos] as Album;
```

```
private string GetClassName(string xmlFileName)
{
  // this function returns the full class name
  // of the XML schema
  // it could be greatly improved with xpath queries ;-)

  // first get the exact location of the XSD file
  XPathDocument xpXMLDoc = new XPathDocument(xmlFileName);
  XPathNavigator navXML = xpXMLDoc.CreateNavigator();

  // the root element contains the comment
  navXML.MoveToRoot();

  // assume the first element contains the schema location
  navXML.MoveToFirstChild();
  string xsdPath = navXML.GetAttribute("schemaLocation",
    "http://www.w3.org/2001/XMLSchema-instance");

  // now open the XML schema and determine the classname
  XPathDocument xpXSDDoc = new XPathDocument(xsdPath);
  XPathNavigator navXSD = xpXSDDoc.CreateNavigator();

  // the root element contains the comment
  navXSD.MoveToRoot();

  // the first child is the xs:schema
  navXSD.MoveToFirstChild();

  // the first child of the xs:schema is the xs:element
  navXSD.MoveToFirstChild();

  // this element contains the desired name
  string fullClassName = "BrowseableObjects." +
    navXSD.GetAttribute("name", "");

  return fullClassName;
}
```

Codevoorbeeld 5. Determining class-name

```
Assembly a =
  Assembly.LoadFrom(@"C:\My Assemblies\BrowseableObjects.dll");
Type classType = a.GetType(className);
```

Codevoorbeeld 6.

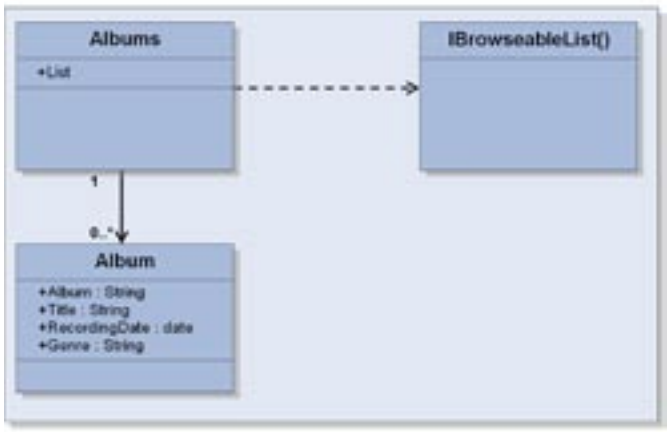
```
mCurrentClassType = GetClassType(GetClassName(fileName));
TextReader reader = new StreamReader(fileName);
XmlSerializer serializer = new XmlSerializer(mCurrentClassType);

mCurrentBrowseableList =
  (IBrowseableList) serializer.Deserialize(reader);
reader.Close();

//currency manager used for cursoring through the array in objects
currencyManager = (CurrencyManager)
  dataGrid.BindingContext[mCurrentBrowseableList.List];

dataGrid.DataSource = mCurrentBrowseableList.List;
```

Codevoorbeeld 7. Het initialiseren van de objectbrowser



Afbeelding 3. Class-model

In de verdere code kan ik nu met een Album-object werken zoals ik dat binnen mijn domein gedefinieerd heb.

Wat verder

In dit artikel heb ik laten zien dat data binding met .NET Framework 2.0, in combinatie met XML en XML-schema's en de CurrencyManager, een krachtig raamwerk kan bieden om data van visualisatie te scheiden zonder datasets. In een paar voorbeelden heb ik de essentie van de technieken hiervoor proberen duidelijk te maken. De volledige code kan gedownload worden van http://www.microsoft.nl/netmagazine13. Het zal duidelijk zijn dat er nog veel meer mogelijkheden in het verschiet liggen. De al eerder genoemde codegeneratie op basis van een XML-schema wil ik graag in een volgend artikel behandelen. Het huidige project is overigens al voor een groot deel gegenereerd, maar dit artikel zou te groot worden als alles in één keer behandeld zou worden.

Floris Zwarteveen is werkzaam bij Fidit (http://www.fidit.nl) waar hij zijn passie voor softwareontwikkeling in de praktijk brengt via programmering, training en advisering, voornamelijk op het gebied van .Net. Je kunt hem bereiken op florisz@fidit.nl

Referenties

Matthew MacDonald and Bill Hamilton; ADO.NET in a nutshell, O'Reilly.

Daniel Cazzulino; Code Generation in the .NET Framework using XML Schema; mei 2004; msdn.microsoft.com

Dare Obasanjo; XML Serialization in the .NET Framework; januari 2003;

Roadmap for Windows Forms data binding; http://support.microsoftcom/?id=313482