Wim Uyttersprot
is directeur van U2U nv/sa Brussel en is bereikbaar via wim@u2u.be
U2U is Microsoft CTEC, gespecialiseerd in .NET development en MSDN Regional Director

Het .NET Compact Framework op een dienblaadje

Embedded applicaties ontwikkelen met het .NET Compact Framework 1.0 en Visual Studio .NET 2003

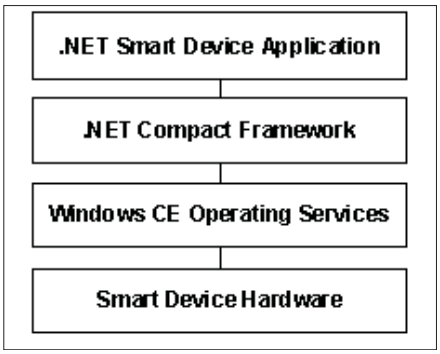
Voor de ontwikkeling van embedded applicaties in de Pocket PC of Windows CE devices kan men een beroep doen op het .NET Compact Framework 1.0. Dit framework is een subset van het standaard .NET Framework en is geïntegreerd in de nieuwe Visual Studio .NET 2003. De developer is dus niet meer verplicht om voor elk soort embedded applicatie gebruik te maken van de Embedded Visual C++ tools. De programmeertalen C# en VB.NET maken het .NET Compact Framework prima toegankelijk.

Het compact framework doorstaat de vergelijking met het standaard (en volledige) .NET Framework. Het is er een subset van, en het biedt op analoge manier een platform van een reeks runtime-componenten. Het belangrijke verschil is dat het onderliggende operating-systeem geen Windows NT is, maar het Windows CE operating-systeem (afbeelding 2). Daarom is het logisch dat de runtime-componenten van het .NET Compact Framework behoorlijk 'compact' moeten zijn.

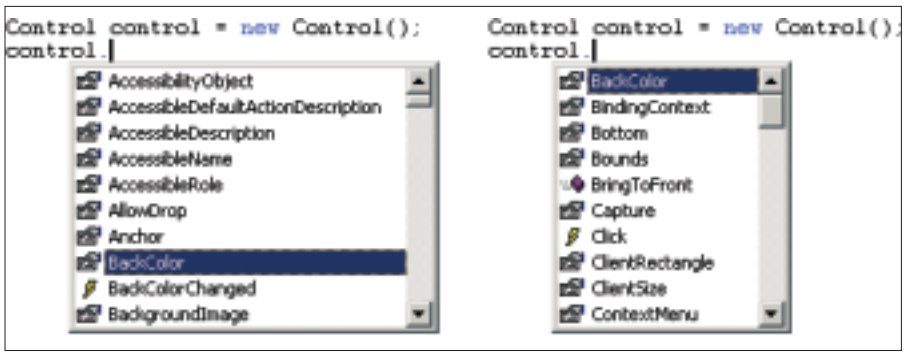
Afbeelding 1. Een toepassingsmogelijkheid in het .NET Compact Framework: Zaanse koeken bestellen via ADO.NET en XML Web services op een Pocket PC

Developers die .NET-applicaties schrijven voor smart devices als Pocket PC's en smart phones (zie afbeelding 1), zullen zich voortaan kunnen beperken tot het schrijven van managed code. Deze managed code zal door toedoen van de Common Language Runtime (CLR) van het .NET Compact Framework just-in-time (JIT) gecompileerd worden volgens de specificaties van de Windows CE Operating Services.

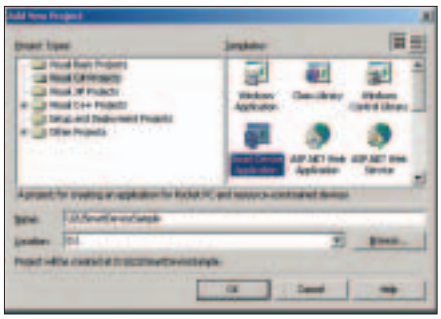
Maar eerst een paar feiten. Het .NET Compact Framework installeert zich bovenop het Windows CE operating-systeem en verbruikt hierbij slechts anderhalve MB aan schijfruimte. Een embedded .NET-applicatie wordt op een Windows CE device geïnstalleerd in Intermediate Language-formaat. Voor het opstarten van zo'n embedded .NET-applicatie, en het laden van de CLR, is er op z'n hoogst slechts 1 MB geheugen nodig. Snel gerekend kunnen we zeggen dat het .NET Compact Framework ongeveer 20 keer zo klein is in bestandsgrootte als zijn grote broer. Hierdoor wordt het geheugenverbruik, eigen aan de .NET-functionaliteit, 5 tot 10 keer verlaagd. Deze optimalisatie heeft natuurlijk gevolgen en die merken we zodra we onze eerste Smart Device-applicatie in



Afbeelding 2. Het .NET Compact Framework installeert zich bovenop het Windows CE operating-systeem



Afbeelding 3. Het .NET Compact Framework is compact. Neem de Control-class als voorbeeld: in de standaardversie van het .NET Framework telt deze class 316 publieke members (linker voorbeeld), in de compacte versie telt de Control-class slechts 82 publieke members (rechter voorbeeld)



Afbeelding 4. De template voor een .NET Compact Framework-applicatie

Visual Studio .NET aanmaken. In het compact framework zijn namelijk tal van helper-functies en helper-property's niet zo belangrijk. We illustreren dit met een voorbeeld: voor het aanmaken van een embedded Windows-applicatie hebben we slechts 28 van de 35 desktop controls tot onze beschikking. Maken we gebruik van de welbekende Control class, dan moeten we het stellen met 27 van de 76 properties, 38 van de 182 methodes en 17 van de 58 events (zie afbeelding 3).

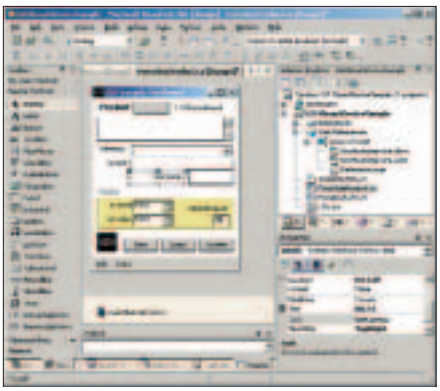
```
using System;
using System.Drawing;
using System.Collections;
using System.Windows.Forms;
using System.Data;

namespace U2USamples
{
    public class FormProducts : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button buttonGetProducts;

        public FormProducts()
        {
            InitializeComponent();
  }

        private void InitializeComponent()
        {
            this.buttonGetProducts = new System.Windows.Forms.Button();
            this.buttonGetProducts.Location = new System.Drawing.Point(1, 1);
            this.buttonGetProducts.Size = new System.Drawing.Size(102, 24);
            this.buttonGetProducts.Text = "Get Products";
            this.buttonGetProducts.Click += new
                  System.EventHandler(this.buttonGetProducts_Click);
            ...
         }

private void buttonGetProducts_Click(object sender, System.EventArgs e)
  {
            ...
  }
    }
}
```

Afbeelding 5. Het event handling-mechanisme in het .NET Compact Framework verschilt niet van dat van het standaard .NET Framework

Visual Studio .NET 2003

Visual Studio .NET 2003 maakt webservices uitermate toegankelijk voor een embedded applicatie. In een Visual Studio .NET-aanpak, waar een webservice-project werd geïmplementeerd, is het eenvoudig om een project van het type 'Smart Device Application' toe te voegen als client van de webservice. Het volstaat om als project template te kiezen voor het type 'Smart Device Application' (zie afbeelding 4). Visual Studio .NET zet dan een project op met daarin een lege Windows Form, precies op maat van een Pocket PC respectievelijk van een Windows CE-device. In de Solution Explorer zien we in de lijst met referenties, die verwijzen naar de gebruikte assembly's, op het eerste gezicht geen verschil, totdat we de properties van de referenties opvragen. We zien dat de referenties verwijzen naar de assembly's van het compact framework. Zo verwijst bijvoorbeeld de System.Windows.Forms-referentie naar een path gelijkaardig aan 'C:\Program Files\Microsoft Visual Studio .NET 2003\CompactFrameworkSDK\v1.0.5000\Windows CE\System.Windows.Forms.dll'.

Alle handelingen die we moeten uitvoeren om voor een Smart Device-applicatie een Windows Form aan te maken en te voorzien van event handling code, komen vertrouwd over (zie afbeelding 6).



Afbeelding 6. Visual Studio .NET is de ideale tool voor het aanmaken van .NET Compact Framework-applicaties. In de toolbox vinden we de compacte CF-controls. De Solution Explorer toont hier een Visual Studio .NET solution met twee projecten: een webservice-project en een tweede voor onze Smart Device-clientapplicatie

Visual Studio .NET is een ideale tool voor het aanmaken van .NET Compact Framework-applicaties. In de Solution Explorer maken we twee Window Forms aan: FormProducts en FormOneProduct. We definiëren een referentie naar een webservice met de namespace www.u2u.net en automatisch resulteert dat in een proxy class. Deze proxy class is geïmplementeerd in het bestand Reference.cs dat verbonden is via het bestand Reference.map aan de webservice, die beschreven is in het NorthWindSevice.wsdl-bestand.

In de toolbox kiezen we de gewenste compacte CF-controls, slepen deze op het Design Window en maken gebruik van het Properties Window. Hetzelfde geldt wanneer we de nodige event handlers implementeren. Kijk bijvoorbeeld naar de voorbeeldcode in afbeelding 5 met daarin de event handler van een GetProducts button. Er valt geen verschil op te merken met een event handler die geschreven zou zijn voor het standaard .NET Framework.

Het compact framework

De kracht van het compact framework ondervinden we snel, zodra we webservices willen consumeren in onze embedded applicaties, en ook wanneer we intensieve data – met name XML-manipulaties – willen uitvoeren. Om een Web Reference toe te voegen in een Smart Device Application-project, maken we gebruik van de Solution Explorer. Automatisch voegt Visual Studio .NET een proxy class toe (zie de voorbeeldcode in afbeelding 7), die we kunnen aanspreken in onze Smart Device-applicatie.

De code die gegenereerd werd voor deze proxy class is algemeen bruikbaar vanuit elke standaard .NET-applicatie. En het mooie is dat het .NET Compact Framework die standaard proxy-code ondersteunt. Voor onze Smart Device-applicatie betekent dit dat de Windows Form de webservice zogenaamd kan 'consumeren' op een uiterst eenvoudige manier: de form stuurt een request naar een instantie van de proxy class (zie de voorbeeldcode in afbeelding 8) en het proxy-object invokeert op zijn beurt de webservice.

Het opmerkelijke van de code in afbeelding 8 is haar eenvoud. Achter de schermen van de proxy class maakt de webservice call gebruik van SOAP-serialization om een DataSet-object terug te krijgen via de returnwaarde. In ons voorbeeld stuurt de webservice een iets complexere dataset door met twee tabellen, genaamd 'Products' respectievelijk 'Categories'. Net zoals bij het standaard framework, zorgt het compact framework ervoor dat de complexiteit transparant blijft. Pas wan-

```
[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.ComponentModel.DesignerCategoryAttribute("code")]
[System.Web.Services.WebServiceBindingAttribute(Name="NorthWindServiceSoap", amespace="http://www.u2u.net/")]
public class NorthWindService : System.Web.Services.Protocols.SoapHttpClientProtocol {

    public NorthWindService() {
        this.Url = "http://www.u2u.net/Northwind/Northwindservice.asmx";
    }
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://www.u2u.net/GetProductsAndCategories",
         RequestNamespace=" http://www.u2u.net/", ResponseNamespace=" http://www.u2u.net/",
         Use=System.Web.Services.Description.SoapBindingUse.Literal,
         ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public System.Data.DataSet GetProductsAndCategories() {
        object[] results = this.Invoke("GetProductsAndCategories", new object[0]);
        return ((System.Data.DataSet)(results[0]));
    }
    ...
}
```

Afbeelding 7. In een compact framework-applicatie kunnen webservices aangesproken worden met proxy classes. De code in dit voorbeeld werd gegenereerd door Visual Studio .NET. NorthWindService is hier een proxy class en GetProductsAndCategories een proxy-methode

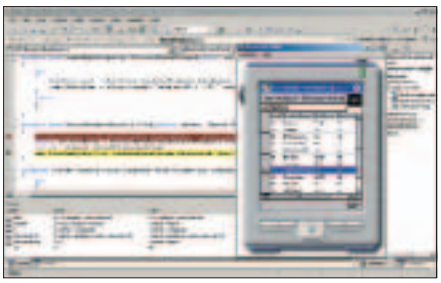
```
www.u2u.net.NorthWindService webservice = new www.u2u.net.NorthWindService();
DataSet dataset = webservice.GetProductsAndCategories();
```

Afbeelding 8. Het .NET Compact Framework ondersteunt de standaard proxy-code voor webservices. Dit betekent dat we webservices kunnen blijven 'consumeren' met behulp van de dot-operator

```
DataTable datatableCategories = dataset.Tables["Categories"];
DataView dataviewCategories = datatableCategories.DefaultView;
comboBoxCategories.DataSource = dataviewCategories;
comboBoxCategories.ValueMember = "CategoryID";
comboBoxCategories.DisplayMember = "CategoryName";

DataView dataviewProducts = dataset.Tables["Products"].DefaultView;
dataviewProducts.RowFilter = String.Format("CategoryID = '{0}'",datatableCategories.Rows[0]["CategoryID"]);
dataGridProducts.DataSource= dataviewProducts;
```

Afbeelding 9. Het .NET Compact Framework ondersteunt ADO.NET

Afbeelding 10. Smart Device-applicaties worden gecompileerd in IL, die met de tool ILDASM.EXE kan worden bestudeerd



Afbeelding 11. Visual Studio .NET heeft de mogelijkheid om de Smart Device-applicatie te debuggen met een Pocket PC-emulator of met een Windows CE .NET-emulator

neer we de dataset analyseren, komen de tabellen en hun relaties te voorschijn.

In het codevoorbeeld van afbeelding 9 verwerken we de dataset met de compacte versie van ADO.NET. Er komt een eerste DataTable en een eerste DataView-object bij te pas, genaamd 'dataviewCategories', die we verbinden met de DataSource-property van het ComboBox-object 'comboBoxCategories'. Uit onze dataset halen we een tweede DataTable en een tweede DataView-object, met name 'dataviewProduct'. We formuleren een RowFilter, die we opstellen met de statische Format-functie van de String class. Dit zal bij de .NET-programmeur vertrouwd overkomen.

Visual Studio.NET 2003 laat ons standaard toe om het .NET Compact Framework aan te spreken vanuit zowel C# als Visual Basic .NET, zonder beperkingen op te leggen aan andere programmeertalen. Na compilatie van onze code komt de Portable Executable in Intermediate Language (IL) tot stand. Deze is overigens alleen bruikbaar op Windows CE-omgevingen waarop de binaire files van het .NET Compact Framework zijn geïnstalleerd. De na compilatie verkregen IL-instructies kunnen we bestuderen met de IL disassembler tool ILDASM.EXE (zie afbeelding 10).

Deployment en debugging

Nog mooier wordt het wanneer we onze Smart Device-applicatie willen installeren op een Pocket PC of ander Windows CE-toestel. Vanuit Visual Studio .NET is het mogelijk om zowel in Release-mode als in Debug-mode de applicatie te installeren op een extern verbonden Windows CE-toestel. Het volstaat om in de Solution Explorer de nodige CAB-files aan te maken via een popup-menu, dat standaard aanwezig is. Het is eveneens mogelijk om de applicatie vanuit Visual Studio .NET op te starten, waarbij de studio alles voor zijn rekening neemt: het aanmaken van de CAB-files, de installatie op de Pocket PC en het remote opstarten van de applicatie.

Bij het .NET Compact Framework worden twee emulatoren geleverd, één voor de Pocket PC en een ander voor Windows CE .NET. Beide emulatoren leveren het comfort om applicaties te ontwikkelen en te debuggen zonder de verplichting om steeds verbonden te zijn met de werkelijke toestellen. Het volstaat dat in Visual Studio .NET de nodige breakpoints worden aangebracht in de source code en dat de Smart Device-applicatie wordt opgestart in de Debug-mode, waarbij we kiezen voor emulatie (zie afbeelding 11).

Vertrouwd

Microsoft biedt eindeloos veel mogelijkheden met het .NET Compact Framework. Want met behulp van Visual Studio .NET 2003 wordt elke.NET developer in staat gesteld om op een vertrouwde manier Smart Device-applicaties aan te maken.

Microsoft Press

Titel: Microsoft® .NET Compact Framework (Core Reference)
ISBN: 0-7356-1725-2
Auteur: A. Wigley, S. Wheelwright, R. Burbidge, R. MacLeod, M. Sutton

Nuttige internetadressen

- http://www.u2u.net (de volledige code van het in dit artikel gebruikte voorbeeld)
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dncenet/html/choose_api.asp
- http://msdn.microsoft.com/vstudio/device/compactfx.asp
- http://msdn.microsoft.com/theshow/Episode029/default.asp