Michiel van Otegem (MVP) is softwarearchitect bij The Vision Web

ASP.NET 2.0: weer een grote stap voorwaarts

EEN NIEUWE REVOLUTIE IN DE ONTWIKKELING VAN WEBAPPLICATIES

ASP.NET versie 1.0 was met recht een revolutie te noemen ten opzichte van zijn voorganger ASP. Versie 2.0 lijkt hetzelfde te gaan doen ten opzichte van versie 1.x. Met de komst van versie 2.0 is ASP.NET veel meer geworden dan een gereedschapskist met onderdelen. Het biedt hele functieblokken met kant en klare functionaliteit die het mogelijk maken een applicatie te maken met een minimum aan programmeerwerk, zonder aan flexibiliteit te verliezen.

De 2.0 versie van ASP.NET is vooral toegespitst op het verminderen van de hoeveelheid werk die nodig is om een applicatie te maken. Het doel van Microsoft is de hoeveelheid te schrijven code te verminderen met zo'n 70%. Die doelstelling lijkt ruimschoots gehaald te worden doordat de basis van elke applicatie vrijwel geen code meer vergt. Overigens gaat de doelstelling absoluut niet ten koste van de snelheid, schaalbaarheid, en beheerbaarheid van applicaties; in tegendeel. Er is bijvoorbeeld een aantal wijzigingen aan het pagina/control-model waardoor het mogelijk is de ViewState te beperken zonder verlies aan functionaliteit. Er is wel enig verlies in dynamische lay-outmogelijkheden. Aangezien dit een notoir probleem is van versie 1.x, zal dit als muziek in de oren klinken. Beheerders zullen verder blij zijn met het nieuws dat je web.config niet meer handmatig hoeft aan te passen, maar dat hiervoor zowel een webinterface als een snap-in voor Microsoft Management Console zal komen. De laatste van de twee is momenteel nog niet beschikbaar, maar met de ASP.NET Web Site Administration Tool die te zien is in afbeelding 1 kunnen nu al zaken als beveiliging, de te gebruiken mailserver, de standaard foutafhandelingpagina en waarden in de <appSettings>-sectie van web.config veranderd worden.

Functieblokken

De drastische vermindering in code is met name toe te schrijven aan zogenaamde 'Building Block API's', zeg maar functieblokken met veelgebruikte functionaliteit voor de gemiddelde webapplicatie. Er is bijvoorbeeld een Membership API om de toegang tot een applicatie te regelen, en een Personalization API voor het verzorgen van personalisatie. Nu hoor ik je denken "dan ben ik zeker wel gebonden aan één gegevensbron, want anders is die API niet te gebruiken". Aangezien dat erg inflexibel zou zijn, is dat dus juist niet zo. De meeste functieblokken zijn gemaakt op basis van het zogenaamde 'Provider Design Pattern'. Dit betekent dat je als ontwikkelaar altijd te maken hebt met dezelfde API, ongeacht de onderliggende gegevensbron. De gegevensbron wordt aan de API gekoppeld via een Provider. De Provider verzorgt de opslag van de gegevens, en is er in verschillende smaken. Voor de meeste functieblokken is er een AccessProvider voor Microsoft Access databases en een SqlProvider voor SQL Server databases. De Membership API heeft bovendien een Provider voor Active Directory. Omdat de API's open zijn, kun je ook zelf een Provider implementeren, bijvoorbeeld met opslag in XML. Het Provider-model zorgt daardoor voor optimale flexibiliteit, zonder dat je veel extra programmeerwerk hebt als je een ander opslagmechanisme wilt gebruiken. Welke Provider gebruikt wordt, is een kwestie van configuratie, en kan dus veranderd worden zonder de code aan te passen. Dit betekent dat je met de bestaande API's kunt ontwikkelen in Microsoft Access, en vervolgens in productie kunt gaan met SQL Server. Welke Provider een API gebruikt kun je aangeven in web.config. Codevoorbeeld 1 laat zien hoe voor het Navigation-functieblok de XmlSiteMap Provider wordt ingesteld.

Afbeelding 1. Met de Web Administration tool kan de configuratie worden aangepast

```
<configuration>
  <system.web>
    <siteMap defaultProvider="XmlSiteMapProvider"
        enabled="true">
      <providers>
        <add name="XmlSiteMapProvider"
          description="Provider voor .sitemap XML bestand."
            type="System.Web.XmlSiteMapProvider, System.Web,
              Version=2.0.3600.0, Culture=neutral,
              PublicKeyToken=b03f5f7f11d50a3a"
          siteMapFile="web.sitemap"
            securityTrimmingEnabled="true"/>
        </providers>
      </siteMap>
  </system.web>
</configuration>
```

Codevoorbeeld 1.
Provider-instelling in web.config

Als je wilt, kun je de functieblokken direct gebruiken vanuit je code. Alleen dat al scheelt behoorlijk wat programmeerwerk, omdat je de meest triviale code niet meer hoeft te schrijven. Daarvoor gebruik je de functies die de functieblokken bieden. Omdat een functieblok een eenduidige manier biedt om bepaalde logica te verzorgen, is het ook mogelijk generieke controls te maken die gebruik maken van een functieblok. Je hoeft dan niet bevreesd te zijn dat je control niet meer werkt als je besluit een andere Provider te gebruiken. Microsoft heeft een hele verzameling controls aan ASP.NET toegevoegd die precies dat doen. Daardoor kun je veel gebruikte functionaliteit verzorgen op basis van een functieblok en de bijbehorende controls, zonder dat daar ook maar één regel code aan te pas komt!

```
<%@ Page Language="C#" %>
<html>
<head runat="server">
    <title>Login formulier</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Login ID="Login1" Runat="server">
        </asp:Login>
    </div>
    </form>
</body>
</html>
```

Codevoorbeeld 2.
Een pagina met een login-formulier

```
<configuration>
   <system.web>
      <profile>
         <properties>
            <add name="Winkelmandje" type="Shop.Basket, Shop"
                  allowAnonymous="true" />
            <group name="Adres">
               <add name="Straat" type="System.String" />
               <add name="Huisnummer" type="System.Int32" />
               <add name="HuisnummerToevoeging" type="System.String" />
               <add name="Postcode" type="System.String" />
               <add name="Plaats" type="System.String" />
            </group>
         </properties>
      </profile>
      <anonymousIdentification enabled="true" />
   <system.web>
</configuration>
```

Codevoorbeeld 3.
Personalisatie-instellingen in web.config

Membership

In ASP.NET 1.x vergt de Forms-beveiliging redelijk wat werk. Je dient een gegevensbron met gebruikers op te zetten, een login-procedure te schrijven, en een login-formulier te maken. Verder kun je ook nog denken aan functionaliteit waarmee gebruikers zich kunnen registeren, en een mogelijkheid om een vergeten wachtwoord op te vragen. Wil je dan ook nog dat pagina's er verschillend uitzien voor aangemelde gebruikers en niet-aangemelde gebruikers, en voor verschillende rollen, dan ben je echt wel een paar dagen zoet om het allemaal voor elkaar te krijgen. In ASP.NET 2.0 is dit alles letterlijk zo gepiept met behulp van de Membership API en de daaraan verwante Role Manager API. De Membership API bevat functies zoals CreateUser (gebruiker aanmaken), ValidateUser (gebruiker aanmelden), en FindUsersByEmail (gebruikers opzoeken aan de hand van het e-mailadres). De meest gebruikte functionaliteit is daardoor teruggebracht tot een enkele methodeaanroep, in plaats van een hele lap code om de database te openen, gegevens op te vragen, de gebruiker te valideren, enzovoort. ASP.NET 2.0 bevat verder een verzameling beveiligingscontrols (zie tabel 1) waardoor het niet meer nodig is om überhaupt code te schrijven voor dit soort functies. De hele login-functionaliteit kun je verzorgen door de Login-control op een pagina te plaatsen (zie codevoorbeeld 2) en je kunt gebruikers zich laten inschrijven met de CreateUserWizard-control. Dit is uitermate eenvoudig met slechts een tekst-editor, en door in Visual Studio een control op een pagina te slepen. Eenvoudiger kan haast niet. De controls zijn zo gemaakt dat ze je niet in een keurslijf dwingen. De controls kunnen vrijwel op alle manieren aangepast worden als het om de opmaak en teksten gaat. De meeste controls bieden bovendien de mogelijkheid om de opmaak te verzorgen via een verzameling templates, zodat je de opmaak helemaal zelf kunt verzorgen. Is dat nog niet genoeg voor je, dan kun je altijd nog de Membership API vanuit je code aanroepen.

Personalisatie

Veel huidige applicaties passen zich in meer of mindere mate aan de gebruiker aan. Dit kan expliciet zijn, door de gebruiker bijvoorbeeld de opmaak of indeling van pagina's te laten bepalen, maar ook impliciet, door bijvoorbeeld de gebruiker keuzes te bieden waarvan te verwachten is dat die voor de gebruiker relevant zijn. Aan dat laatste hangt meestal een hele berg logica die bepaalt wanneer iets relevant is, en dat is dan ook zeer gebonden aan het doelgebied van de applicatie. Expliciete personalisatie daarentegen is vrij eenvoudig, en daarom af te handelen met een standaardmechanisme. ASP.NET 2.0 biedt hiervoor een personalisatie-functieblok dat even ingenieus als doeltreffend is. Om gegevens van een gebruiker op te slaan, moet je eerst bepalen welke gegevens je wilt opslaan. Dit kunnen eenvoudige gegevens zijn, zoals adresgegevens, maar mag ook een complex object zijn, zoals bijvoorbeeld een winkelwagentje. Welke gegevens je wilt opslaan leg je vast onder de <profile>-sectie in web.config, zoals je kunt zien in codevoorbeeld 3.

In codevoorbeeld 3 zie je een aantal eigenschappen dat te maken heeft met adresgegevens. Van de eigenschappen zijn de naam en het type vastgelegd. Bovendien zijn de betreffende eigenschappen gegroepeerd onder de groep 'Adres'. Verder is er buiten deze groep nog een eigenschap 'Winkelmandje', waarvan het volledige type, inclusief de naam van de Assembly is aangegeven. Dat is nodig, omdat de Shop.Basket class een custom type is. Verder nog van belang is dat deze eigenschap gemarkeerd is met allowAnonymous="true". Hiermee wordt aangegeven dat deze eigenschap te gebruiken is voor gebruikers die nog niet aangemeld zijn, zodat een gebruiker al zaken in het winkelmandje kan doen voordat hij/zij is aangemeld. Wanneer de gebruiker zich aanmeldt, kan het object worden gekopieerd naar het profiel van de aangemelde gebruiker, en zijn de andere eigenschappen ook beschikbaar. Merk op dat de mogelijkheid tot anonieme identificatie ingeschakeld is om anoniem gebruik van het winkelmandje mogelijk te maken.

Control	Omschrijving
Login	Laat een formulier zien waarin gebruikers hun gebruikersnaam en wachtwoord invoeren. Ook validatie van de gebruiker.
LoginName	Laat de naam van de aangemelde gebruiker zien.
LoginStatus	Laat zien of de huidige gebruiker is aangemeld, en geeft een link weer die de aangemelde gebruiker afmeldt.
LoginView	Laat een weergave zien die afhankelijk is van of een gebruiker aangemeld is of niet, en eventueel op basis van de rol van de gebruiker.
ChangePassword	Laat een formulier zien waarmee gebruikers hun wachtwoord kunnen aanpassen.
CreateUserWizard	Laat een serie formulieren zien waarmee een gebruiker aangemaakt wordt.
PasswordRecovery	Laat een formulier zien waarmee gebruikers hun wachtwoord (of een nieuw wachtwoord) naar hun e-mailadres kunnen laten sturen.

Tabel 1.
Beveiligingscontrols

Afbeelding 2. Intellisense op profiel met typeaanduiding

De eigenschappen die zijn gedefinieerd in web.config zijn in de code expliciet te benaderen met behoud van type. Dit in tegenstelling tot bijvoorbeeld waarden die worden opgeslagen in het Session-object of de ViewState. Zoals je kunt zien in afbeelding 2 werkt in Visual Studio zelfs de Intellisense helemaal mee, en geeft deze het type van de eigenschap weer. Dit betekent dat je gevrijwaard bent van typefouten die met bijvoorbeeld het Session-object heel gewoon zijn. Een ander voordeel ten opzichte van het Session-object is dat de benodigde gegevens pas worden ingelezen als ze nodig zijn, terwijl alle Session-variabelen automatisch worden geladen op het moment dat een pagina wordt uitgevoerd. Doordat je eigenschappen kunt groeperen en zelfgemaakte types kunt gebruiken, is het profielmechanisme uitermate flexibel en voor vrijwel iedere vorm van personalisatie te gebruiken.

WebParts

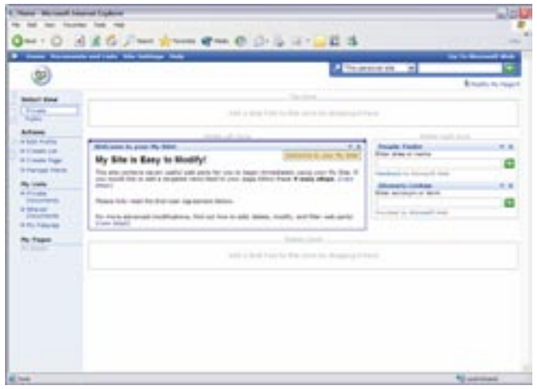
Een onderdeel dat gebruik maakt van de personalisatiemogelijkheden van ASP.NET 2.0 zijn WebParts. WebParts zijn ook al bekend uit Sharepoint, waar ze gebruikers de mogelijkheid geven een pagina naar smaak in te delen. Een WebPart is een soort control met bepaalde inhoud. Denk bijvoorbeeld aan een verzameling links, een agenda, het laatste nieuws, enzovoort. De gebruiker kan één of meer WebParts in daarvoor gedefinieerde WebPartZones plaatsen. Als er meer zones op een pagina staan, kan een WebPart in elk van die zones geplaatst worden. Je kunt dit zien in afbeelding 3, waarin vier zones zijn gedefinieerd: Top, Bottom, Left en Right. In edit-modus zijn de zones zichtbaar zoals in afbeelding 3. De People Finder WebPart staat nu in de Right-zone, maar zou evengoed verplaatst kunnen worden naar een van de andere zones. Wanneer de gebruiker klaar is met het indelen van de pagina, worden de instellingen opgeslagen in het profiel van de gebruiker, en zal de indeling de volgende keer dat de gebruiker op de site komt weer zo zijn.

ASP.NET 2.0 biedt alle controls die nodig zijn voor deze functionaliteit. Hoewel je dit dus volledig op basis van controls opbouwt, is het nog best een klusje om door te krijgen hoe het allemaal werkt,

```
<%@ Master %>
<html >
<head runat="server">
    <title>.NET Magazine Master Page</title>
</head>
<body>
    <form runat="server">
        <h1>.NET Magazine Master Page</h1>
        <hr />
        <asp:contentplaceholder id="Body" runat="server">
        </asp:contentplaceholder>
        <hr />
        <asp:contentplaceholder id="Footer" runat="server">
        </asp:contentplaceholder>
    </form>
</body>
</html>
```

Codevoorbeeld 4.

Master Page definitie



Afbeelding 3. Een pagina indelen met WebParts

en valt daarom buiten het bestek van dit artikel. Wanneer je het eenmaal onder de knie hebt, is het een heel krachtig gereedschap om een mooie portal te maken. Zelf WebParts aanmaken is overigens wel weer heel eenvoudig; gewoon een User Control maken. Binnen de User Control kun je eigenschappen markeren met het PersonalizableAttribute zodat ook die eigenschappen in het profiel kunnen worden opgeslagen. Zo heb je bovenop de standaard functionaliteit van een User Control dus wat extra mogelijkheden binnen het WebPart Framework.

Eenduidige opmaak

Een van de zaken die je heel erg mist in ASP.NET 1.x is de mogelijkheid een sjabloon te definiëren voor meer pagina's. De meeste ontwikkelaars lossen dit op door aan het begin en einde van iedere pagina een (user) control te plaatsen die de algemene opmaak van de pagina verzorgt. ASP.NET 2.0 biedt een veel elegantere oplossing in de vorm van Master Pages. Een Master Page is een paginasjabloon. Iedere pagina kan definiëren welk sjabloon hij gebruikt. Tot zover niets bijzonders. Het aardige is echter dat je ook sjablonen kunt maken op basis van een sjabloon. Zo kun je dus voor delen van een applicatie een enigszins afwijkende opmaak gebruiken, maar toch eventuele wijzigingen aan de algehele opmaak meenemen. Je kunt als het ware een hele hiërarchie van sjablonen maken. Ook kun je tijdens het uitvoeren bepalen welke Master Page een pagina gebruikt, zodat je bijvoorbeeld gebruikers hun eigen opmaak kunt laten kiezen. Ook kun je vanuit een pagina het sjabloon programmatisch benaderen en wijzigen. Een Master Page werkt met zogenaamde ContentPlaceHolder-controls waarmee één of meer regionen in het sjabloon kunnen worden aangegeven. In de pagina plaats je vervolgens Content-controls die aan de hand van de ContentPlaceHolderID aangeven voor welke regio ze content bevatten. Codevoorbeeld 4 bevat een simpel voorbeeld van een Master Page, en codevoorbeeld 5 is een voorbeeld van een pagina die daar gebruik van maakt.

Naast Master Pages zijn er nog meer mogelijkheden om een eenduidige opmaak te verzorgen voor een applicatie: Themes en

```
<%@ Page Language="C#" MasterPageFile="~/NetMagazine.master"%>
<asp:Content ContentPlaceHolderID="Body" Runat="server">
   <asp:Label ID="lbl1" Runat="server" BackColor="Green">
      Een label in de Body ContentPlaceHolder
   </asp:Label>
</asp:Content>

<asp:Content ContentPlaceHolderID="Footer" Runat="Server">
   &copy 2004 .NET Magazine
</asp:Content>
```

Codevoorbeeld 5.

Pagina op basis van Master Page in codevoorbeeld 4


```
<asp:Button Runat="server" SkinID="Annuleer" BackColor="Red"
   Font-Italic="True" Font-Size="Large"
   BorderWidth="0px" ForeColor="White" />

<asp:Button Runat="server" SkinID="OK" BackColor="Green"
   Font-Italic="True" Font-Size="Large"  Font-Bold="True"
   BorderWidth="0px" ForeColor="White" />
```

Codevoorbeeld 6.

Een Theme-definitie met Skins

```
<asp:Button ID="Button1" Runat="server" SkinID="Annuleer"
   Text="Annuleer" />
<asp:Button ID="Button2" Runat="server" SkinID="OK"
   Text="Bestel" />
```

Codevoorbeeld 7.

Fragment dat gebruik maakt van de Theme in Codevoorbeeld 6

Skins. Een theme is vergelijkbaar met een theme in Windows. Een theme is een totale look & feel voor je applicatie, dus niet alleen kleuren en lettertypes, maar ook afbeeldingen en dergelijke. Standaard zal ASP.NET 2,0 enkele kant en klare themes bevatten, maar helaas zijn deze in de huidige bèta niet beschikbaar. Je kunt echter ook zelf themes definiëren aan de hand van een skin-bestand in een themes\[Theme-naam] map. Als je een skin-bestand definieert, zoals in codevoorbeeld 6, dan kun je deze als volgt aangeven in de Page-directive van een pagina (of Master Page):

```
<%@ Page Theme="NetMagazine" %>
```

Je kunt een theme ook tijdens het uitvoeren van de pagina instellen door de Page.Theme-eigenschap in te stellen, zodat deze bijvoorbeeld afhankelijk van de gebruiker kan zijn.
In een skin-bestand zoals in codevoorbeeld 6 definieer je voor iedere control een opmaak, net zoals je dat in een pagina zou doen. Het enige dat je weg laat is de ID-eigenschap. Je kunt hierbij gebruik maken van de standaard opmaakmogelijkheden, CSS, en referenties aan afbeeldingen die je binnen een theme kunt definieren. Binnen een theme kun je ook skins definiëren. Een skin is een aparte weergave voor een control, zodat je bijvoorbeeld verscheidene weergaven kunt maken van een knop, elk met een eigen SkinID; zoals gedaan in codevoorbeeld 6. In de pagina's bepaal je vervolgens welke knop welke skin gebruikt, zodat bijvoorbeeld de 'terug'-knop er anders uit ziet dan een 'selecteer'-knop. Het codefragment in codevoorbeeld 7 doet dit met twee knoppen die een SkinID uit codevoorbeeld 6 gebruiken. Het resultaat is te zien in afbeelding 4.

Nog veel meer...

In dit artikel is slechts een fractie van de nieuwe functionaliteit besproken. Een greep uit de onderwerpen die niet aan de orde zijn gekomen:

- Navigatie-controls
- Databinding zonder code
- Groeperen van validatie-controls
- Een postback naar een andere pagina
- Een nieuwe manier om opmaak en code te scheiden
- Een nieuw (dynamisch) compilatiemodel
- Een nieuwe aanpak bij de ondersteuning van browsers
- Cache-validatie gekoppeld aan databasetabellen

Je ziet dat het onmogelijk is alles te behandelen, zeker als je bedenkt dat de mogelijkheden van Visual Studio voor ASP.NET 2.0 vrijwel helemaal buiten beschouwing zijn gelaten. Het is dan ook niet onverstandig om al eens met de bèta te experimenteren. Daarmee is de leercurve straks minder, en kun je bovendien ontwikkelingen die je doet voor de definitieve versie al enigszins op de leest schoeien van ASP.NET 2.0.

Afbeelding 4. Resultaat van Codevoorbeeld 7

Nuttige Internetadressen
http://beta.asp.net/quickstart/aspnet/
http://lab.msdn.microsoft.com/vs2005/
http://www.asp.net
http://www.aspnl.com/aspnl/nl/forums/

Michiel van Otegem (MVP) is softwarearchitect bij The Vision Web (www.thevision-web.com) en voorzitter van dotNED, de .NET Gebruikersgroep Nederland (www.dotned.nl). Michiel is te bereiken via: michiel@aspnl.com.