Erwin Werkman is werkzaam bij The Vision Web

Atlas maakt Ajax simpel

DYNAMISCHE WEBPAGINA'S MET AJAX EN ATLAS

Ajax is hot. En dan bedoel ik het schoonmaakmiddel, noch de voetbalclub uit Amsterdam. Ajax is de naam van een verzameling van technologieën die webpagina's dynamischer en interactiever maakt. De interesse voor Ajax wordt aangejaagd doordat een aantal belangrijke sites er gebruik van maakt. Een paar voorbeelden zijn MSN Virtual Earth, start.com, Google Suggest, Google Maps en A9.com.

Kenmerkend aan deze sites is dat ze gebruik maken van DHTML en JavaScript om een pagina één keer in te laden en daarna alleen stukjes van de pagina te verversen. Dit maakt dat de pagina's veel interactiever en sneller werkend. DHTML en Javascript zijn lastige technieken om te leren, vaak browserafhankelijk, lastig te debuggen en maken niet altijd standaard onderdeel uit van de kennis van iedere ontwikkelaar. Geen nood. Er is nu al technologie die het eenvoudiger maakt om AJAX-enabled applicaties te maken en de toekomst ziet er nog rooskleuriger uit.

Wat is Ajax?

De naam Ajax is bedacht door Jesse James Garret van Adaptive Path en is een acroniem voor *Asynchronous Javascript + XML*. Ajax is geen echte technologie, maar meer een pattern voor een verzameling technologieën:

- XHTML en CSS voor de presentatie
- Document Object Model voor het manipuleren van wat je op het scherm ziet
- XML en XSLT voor het uitwisselen van gegevens tussen de browser en de server
- Asynchroon ophalen van gegevens met *XMLHttpRequest*
- JavaScript als lijm tussen al deze technologieën.

Zoals je ziet staat er in bovenstaand rijtje geen enkele technologie die we nu nog niet hebben. Je hoeft dus geen nieuwe browser te downloaden of een Java-applet of ActiveX-component te installeren om gebruik te kunnen maken van Ajax. Het is ook geen browserspecifieke technologie, de meeste moderne browsers bevatten alle elementen om gebruik te kunnen maken van Ajax. Wat maakt Ajax dan zo speciaal? Het eenvoudige antwoord is: eigenlijk niets. Voordat de naam Ajax bekend was, waren er al websites die van dezelfde technologie gebruikmaakten om een rijkere user-interface aan te bieden. Kijk bijvoorbeeld maar eens naar de Microsoft Exchange Web-client.

De interesse in Ajax hangt samen met een opbloei in de interesse voor Rich Internet Applicaties (RIA). RIA-applicaties verbeteren de user-interface van webapplicaties door een zogenaamde Single Page Interface (SPI) aan te bieden. Een SPI is een interface waarbij alleen de eerste pagina in zijn geheel wordt ingeladen, waarna op basis van clicks van de gebruiker, alleen nog stukjes HTML van de server opgehaald worden om delen van de pagina te vervangen. Toch is er iets gebeurd waardoor er nu een enorme aandacht is voor alles wat met Ajax te maken heeft. Waarschijnlijk is het gewoon een kwestie van juiste tijd en juiste plaats. Browsers hebben nu de mogelijkheden om goed met deze technologie om te gaan. Er zijn nog wel verschillen in de manier waarop browsers met 'standaarden' omgaan, maar de verschillen zijn aanzienlijk kleiner dan in het verleden.

Wat maakt Ajax anders? Ajax maakt het web interactiever. In een klassieke webapplicatie is de browser in feite een domme terminal die niets anders doet dan de HTML die door de server wordt geleverd aan de gebruiker tonen. De gebruiker klikt vervolgens op een hyperlink die een request oplevert voor de server. De server haalt vervolgens wat data op uit een database en levert een compleet nieuwe HTML-pagina aan de browser. Al die tijd heeft de gebruiker niets kunnen doen. Deze aanpak was voor het oorspronkelijke doel van HTML nog acceptabel, maar levert in de huidige ingewikkelde webapplicaties niet echt een bevredigende ervaring voor gebruikers. Een Ajax-applicatie haalt alleen de eerste keer een complete pagina op. Wanneer de gebruiker vervolgens op een link in de pagina klikt om bijvoorbeeld een product aan zijn winkelwagen toe te voegen, wordt niet een compleet nieuwe pagina opgehaald, maar wordt door middel van Javascript en DHTML slechts een stukje van de pagina opgehaald van de server. Door deze opzet is er veel

```
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class AuthenticationService : System.Web.Services.WebService
{
   [WebMethod]
   public bool Authenticate(String username, String password)
   {
      return (username.Equals("administrator") && password.Equals("atlas"));
   }
}
```

Codevoorbeeld 1.

```
<form id="form1" runat="server">
   <div id="authenticationPanel">
      <span id="errorMessage"></span>
      Username: <input type="text" id="username" runat="server" /><br />
      Password: <input type="password" id="password" runat="server" />
      <button id="submitButton" runat="server"
         onclick="AuthenticateUser();">Submit</button>
   </div>
   <div id="authenticatedPanel"
      style="visibility:hidden;">Hartelijk welkom!</div>
</form>
```

Codevoorbeeld 2.

```
<script language="JavaScript" src="/AuthenticationService.asmx/js"
        type="text/javascript"></script>

<script language="javascript" type="text/javascript">
   function AuthenticateUser()
   {
      var username = document.getElementById("username");
      var password = document.getElementById("password");

      requestAuthenticationService = AuthenticationService.Authenticate(
         username.value, password.value, OnComplete, OnTimeout);
   }

   function OnComplete(result)
   {
      if( result == true )
      {
         var authenticationPanel = document.getElementById("authentication
              Panel");
         var authenticatedPanel  = document.getElementById("authenticated
              Panel");

         authenticatedPanel.style.visibility = "visible";
         authenticationPanel.style.visibility = "hidden";
      }
      else
      {
         var errorMessage = document.getElementById("errorMessage");
         errorMessage.innerHTML = "Deze combinatie kon niet gevonden
            worden.<br/>";
      }
   }

   function OnTimeout(result)
   {
      alert("Timed out");
   }
</script>
```

Codevoorbeeld 3.

```
<head runat="server">
   <title>Atlas Demo</title>
   <atlas:Script ID="Script1" runat="server"
      Path="~/ScriptLibrary/AtlasCompat.js" Browser="Mozilla" />
   <atlas:Script ID="Script2" runat="server"
      Path="~/ScriptLibrary/AtlasCompat.js" Browser="Firefox" />
   <atlas:Script ID="Script3" runat="server"
      Path="~/ScriptLibrary/AtlasCompat.js" Browser="AppleMAC-Safari" />
   <atlas:Script ID="Script4" runat="server"
      Path="~/ScriptLibrary/AtlasCore.js" />
   <atlas:Script ID="Script5" runat="server"
      Path="~/ScriptLibrary/AtlasCompat2.js" Browser="AppleMAC-Safari" />
   <atlas:Script ID="Script6" runat="server"
      Path="~/ScriptLibrary/AtlasRuntime.js"/>
   <atlas:Script ID="Script7" runat="server"
      Path="~/ScriptLibrary/AtlasControls.js" />
</head>
```

Codevoorbeeld 4.

minder dataverkeer tussen de client en de server en reageert de interface ook sneller, omdat je gebruikmaakt van de kracht van de client.

Ajax kent vele verschijningsvormen. In zijn eenvoudigste vorm bestaat Ajax uit een javascriptfunctie die door middel van XmlHttpRequest en DHTML een aanroep naar de server doet en op basis van het resultaat een deel van de pagina aanpast. In zijn meest complexe vorm wordt het beheer van de user-interface overgenomen door een Ajax-engine. De Ajax-engine zorgt voor het updaten van de user-interface en voor de communicatie met de server. Er zijn op dit moment een aantal producten op de markt beschikbaar die een Ajax-engine aanbieden. Beide vormen zijn goed. Wat je wilt verbeteren aan je gebruikersinterface bepaalt welke vorm voor jou het meest geschikt is. De vraag blijft natuurlijk: hoe kun je in ASP.NET gebruikmaken van al dit moois? Als we kijken naar de architectuur van ASP.NET (1.1 en 2.0) dan is ASP.NET in essentie een server-based technologie. Het biedt allerlei faciliteiten om op de server heel krachtige dingen te doen. Als je leuke dingen op de client wilt doen ben je echter op jezelf aangewezen. Je zult zelf javascript moeten schrijven om de client dynamisch te maken met DHTML, maar met wat hulp is dat eenvoudiger dan je denkt.

Ajax in ASP.NET 1.1 & 2.0

ASP.NET 1.1 biedt standaard geen ondersteuning voor Ajax. Toch is er niets dat je tegenhoudt om het te gebruiken, al zul je zelf het meeste werk moeten doen. Er zijn enkele frameworks beschikbaar die een basis bieden voor het gebruik van Ajax in ASP.NET 1.1. Eén van de bekendste is Ajax.NET (beschikbaar voor ASP.NET 1.1 en 2.0) van Michael Schwarz. Michael heeft een framework ontwikkeld dat het erg simpel maakt om zelf Ajax-functionaliteit aan je pagina toe te voegen. Het framework doet dat doordat functies eenvoudig op de server aan te roepen zijn. Deze functies worden op een webservice-achtige manier gedeclareerd in je code-behind, waarna je ze na wat kleine aanpassingen aan de aspx-file, direct vanuit javascript kunt aanroepen. Ajax.NET helpt je echter niet bij het maken van het javascript en DHTML op de webclient. Ook de standaarduitvoering van ASP.NET 2.0 biedt geen directe ondersteuning voor Ajax. Wel biedt het in de vorm van script callbacks de mogelijkheid om op beperkte schaal te communiceren tussen de client en de server. Ook hier zul je weer een hoop zelf moeten schrijven. Maar wees niet bang: er is hulp onderweg.

ASP.NET Atlas

ASP.NET 2.0 biedt dus nog niet de uitgebreide ondersteuning voor Ajax die je zou willen hebben. Maar Microsoft heeft ook niet stil gezeten. Op de Professional Developers Conference (PDC) van september is dan ook de eerste alfaversie van wat nu nog ASP.NET Atlas heet verschenen. ASP.NET Atlas biedt een aantal nuttige hulpmiddelen.

Javascript

Een van die hulpmiddelen is het uitbreiden van javascript met de mogelijkheid om gebruik te maken van namespaces, overerving, interfaces en enums. Verder biedt ASP.NET Atlas een hulpbibliotheek voor strings en array's. Deze 'extensies' maken het bouwen en onderhouden van Atlas-applicaties eenvoudiger en gestructureerder. Al deze extensies, zoals het hele Atlas-framework, werken niet alleen in Internet Explorer, maar ook in de meeste andere moderne browsers.

Webservices

Webservices zijn bij uitstek geschikt om over het Internet aan te roepen. Ze zijn er op ingericht om via HTTP te werken en je kunt er allerlei data mee versturen. Het Atlas-team heeft er dan ook voor gekozen om webservices te gebruiken om aanroepen vanuit een webpagina te kunnen doen. In de volgende voorbeelden zien we hoe we kunnen controleren of een geldige gebruikersnaam en wachtwoord is ingevuld, zonder dat we een volledige round-trip naar de webserver maken. Codevoorbeeld 1 toont een simpele webservice die de authenticatie regelt en *true* of *false* teruggeeft, al naar gelang de gebruikersnaam en het wachtwoord zijn ingevuld. Codevoorbeeld 2 bevat een gedeelte van de ASPX-pagina die deze webservice gaat aanroepen. Het is een formulier waarin de gebruiker een gebruikersnaam en wachtwoord kan invullen. Het formulier bevat een hyperlink die de functie AuthenticateUser aanroept. Deze functie vind je in codevoorbeeld 3. De code uit codevoorbeeld 4 zorgt dat alle Atlas-scripts in de pagina opgenomen worden.

Codevoorbeeld 3 begint met een *script*-element met een merkwaardige aanroep van de webservice: */AuthenticationService.asmx/js*. Deze aanroep zorgt er voor dat een stukje javascript wordt opgehaald dat een proxy-functie bevat voor de webservice die we gemaakt hebben. Iets verder in dit voorbeeld wordt die functie aangeroepen: *AuthenticationService.Authenticate(...)* De proxy-functies die gebruikt worden om webservices aan te roepen, zijn altijd van de vorm *<volledige namespace>.<class naam>.<functienaam>*. De eerst twee parameters zijn de parameters die we in de webservice gedefinieerd hebben. De webservice

wordt asynchroon aangeroepen om te zorgen dat de user-interface niet bevriest. De laatste twee parameters zijn de functies die worden aangeroepen wanneer de webservice klaar is (*OnComplete*) of wanneer de webservice niet op tijd reageert (*OnTimeOut*).
De functie *OnComplete* ontvangt het resultaat van de aanroep naar de webservice en gaat hier vervolgens mee aan de slag om de gebruiker hartelijk welkom te heten of te melden dat de combinatie van gebruikersnaam en wachtwoord niet gevonden kon worden.
Onze authenticatie-webservice levert een simpele Boolean variabele op, maar het is ook mogelijk om een class te versturen. Atlas maakt gebruik van Javascript Object Notation (JSON) als protocol om objecten te versturen. Een class-object dat je op de client aanmaakt, wordt geserialiseerd naar een JSON-string, vervolgens naar de server verstuurd en op de server weer gedeserialiseerd naar .NET-class. Uiteraard werkt dit ook vanaf de server naar de webclient.

Atlas-controls

Het kunnen aanroepen van een webservice is een zeer krachtige uitbreiding van de mogelijkheden die je op de webclient hebt om je applicaties dynamischer te maken. Maar zoals je in het vorige voorbeeld hebt kunnen zien, ben je nog steeds aangewezen op javascript en DHTML om de user-interface aan te passen. ASP.NET Atlas breidt de client dan ook uit met een op ASP.NET lijkend model om op de client te kunnen programmeren. In codevoorbeeld 5 zie je de code die wat betreft functionaliteit gelijk is aan de code uit codevoorbeeld 4, maar nu gebruik maakt van de faciliteiten die Atlas je biedt.
Ten eerste zie je dat er, net als in ASP.NET, een *pageLoad*-functie is toegevoegd. Deze functie wordt na het laden van de pagina automatisch als eerste aangeroepen en dit is dus de plaats om initialisatie van je controls te doen. In de pageLoad-functie worden de verschillende elementen uit de pagina aan een eigen control toegekend. Dat gaat door middel van een verwijzing naar het id van het element dat de vorm $('*<elementid>*') heeft, wat gelijkstaat aan het aanroepen van *document.getElementById*. De aangemaakte controls kun je vervolgens gebruiken om properties te zetten, functies aan te roepen en events aan te hangen zoals je in het voorbeeld ziet bij de button: *submitButton.click.add(onSubmitButtonClick)*. In coderegel hieronder zie je dan ook dat het niet meer nodig is om een *onclick*-attribuut aan de button toe te voegen.

```
<script language="JavaScript" src="/AuthenticationService.asmx/js"
        type="text/javascript"></script>

<script language="javascript" type="text/javascript">
   var g_usernameTextBox;
   var g_passwordTextBox;
   var g_authenticationPanel;
   var g_authenticatedPanel;
   var g_errorMessage;

   function pageLoad()
   {
      g_usernameTextBox = new Web.UI.TextBox($('username'));
      g_usernameTextBox.initialize();

      g_passwordTextBox = new Web.UI.TextBox($('password'));
      g_passwordTextBox.initialize();

      g_authenticationPanel = new Web.UI.Control($('authenticationPanel'));
      g_authenticationPanel.initialize();

      g_authenticatedPanel = new Web.UI.Control($('authenticatedPanel'));
      g_authenticatedPanel.initialize();

      g_errorMessage = new Web.UI.Label($('errorMessage'));
      g_errorMessage.initialize();

      var submitButton = new Web.UI.Button($('submitButton'));
      submitButton.initialize();
      submitButton.click.add(onSubmitButtonClick);
   }

   function onSubmitButtonClick()
   {
      var username = g_usernameTextBox.get_text();
      var password = g_passwordTextBox.get_text();

      AuthenticateUser( username, password);
   }

   function AuthenticateUser( username, password)
   {
      requestAuthenticationService = AuthenticationService.Authenticate(
         username, password, OnComplete);
   }

   function OnComplete(result)
   {
      if( result == true )
      {
         g_authenticatedPanel.set_visible( true);
         g_authenticationPanel.set_visible( false);
      }
      else
      {
         g_errorMessage.set_text("Deze combinatie kon niet gevonden
              worden.<br/>");
      }
   }
</script>
```

Codevoorbeeld 5.

```
<button id="submitButton" runat="server">Submit</button>
```

Naast javascript biedt Atlas ook een declaratieve manier om met controls om te gaan. Deze declaratieve manier wordt Atlas-script genoemd en lijkt erg op de manier waarop in XAML de user-interface wordt gedeclareerd. XAML is de declaratieve taal die in Microsoft Vista wordt gebruikt voor de user-interface. Codevoorbeeld 6 bevat een voorbeeld van het gebruik van Atlas-script.
Dit voorbeeld toont een textbox met daarnaast twee buttons, waarmee respectievelijk de zichtbaarheid van de textbox wordt aangepast en de textbox enabled of disabled wordt. In het gedeelte dat begint met *<script type="text/xml-script">* vind je de declara-

```
<form id="form1" runat="server">
   <input id="textBox" type="text"/>
     
   <button id="visibilityButton"  class="buttonstyle">
      Toggle Visibility Property</button>
     
   <button id="enabledButton" class="buttonstyle">
      Toggle Enabled Property</button>
</form>

<script type="text/xml-script">
   <page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
      <references>
         <add src="/ScriptLibrary/AtlasUI.js" />
         <add src="/ScriptLibrary/AtlasControls.js" />
      </references>
      <components>
         <textBox targetElement="textBox"
                  text="Dit is een textbox control." cssClass="textBox">
            <bindings>
               <binding  id="setEnabled" dataContext="textBox"
                         dataPath="enabled" property="enabled"
                         transform="Invert"
                         automatic="false" />
               <binding  id="setVisibility" dataContext="textBox"
                         dataPath="visible"
                         property="visible" transform="Invert"
                         automatic="false" />
            </bindings>
         </textBox>
         <button targetElement="visibilityButton">
            <click>
               <invokeMethod target="setVisibility" method="evaluateIn" />
            </click>
         </button>
         <button targetElement="enabledButton">
            <click>
               <invokeMethod target="setEnabled" method="evaluateIn" />
            </click>
         </button>
      </components>
   </page>
</script>
```

Codevoorbeeld 6.

```
public class User
{
   public string Username;
   public string Emailaddress;

   public User(string username, string emailaddress)
   {
      this.Username = username;
      this.Emailaddress = emailaddress;
   }
}
```
Codevoorbeeld 7.

```
[WebMethod]
public User[] GetUsers()
{
   ist<User> users = GetAllUsers();

   return users.ToArray();
}

List<User> GetAllUsers()
{
   List<User> allUsers = new List<User>();

   allUsers.Add( new User( "ppuk", "ppuk@hotmail.com"));
   allUsers.Add( new User( "jklaassen", "jklaassen@hotmail.com"));
   allUsers.Add( new User( "dduck", "dduck@hotmail.com"));

   return allUsers;
}
```
Codevoorbeeld 8.

tieve code die hiervoor zorgt. Het root-element is *page*. Dit element bevat twee delen: het *references*-gedeelte bevat verwijzingen naar het javascript dat gebruikt wordt. Het *components*-gedeelte bevat de declaratie van de delen uit de user-interface die gebruikt gaan worden.

Er worden drie *components* gedeclareerd: een textbox en twee buttons. Deze componenten worden via het attribuut *targetElement* aan de userinterface-elementen met dezelfde id gekoppeld. Het *textBox*-element bevat twee *bindings*. Met bindings kun je de properties van een component aan data-elementen koppelen. In dit voorbeeld worden de *enabled*- en *visible*-eigenschappen van de textbox aan zichzelf gekoppeld, via de *dataPath*- en *propertyI*-attributen. Het *transform*-attribuut geeft aan dat wanneer deze binding 'aangeroepen' wordt, de waarde van de eigenschap geïnverteerd moet worden (true wordt false en andersom). De bindings worden aangeroepen door de twee buttons. De *button*-elementen bevatten ieder een *click*-element met daarin een *invokeMethod*-element dat zorgt dat de bindings van de textbox worden aangeroepen, wanneer de gebruiker op de button klikt. De Atlas runtime-engine zorgt verder zelf voor het binden van het Atlas-script aan de elementen in je user-interface. Atlas-script is erg krachtig, maar vergt wel een andere manier van denken.

Data-binding

Het derde onderdeel van ASP.NET Atlas is data-binding. Met ASP.NET Atlas is het mogelijk om datasources direct aan userinterface-elementen te koppelen. We gaan onze inlogpagina uitbreiden. Wanneer de gebruiker één keer een onbekende gebruikersnaam en wachtwoord heeft ingevoerd, tonen we een lijstje met alle bekende gebruikersnamen. Dit lijstje wordt opgehaald via de webservice. We breiden de webservice uit met de class *User* (zie codevoorbeeld 7) die een naam en een e-mailadres bevat. We breiden de webservice verder uit met een method die alle bekende users in een array doorgeeft; zie codevoorbeeld 8.

Codevoorbeeld 9 bevat de code voor de webpagina. Het formulier is uitgebreid met een Atlas *ListView*-control. Deze control biedt een lay-out-template waarin gedefinieerd wordt hoe een rij uit de datasource wordt weergegeven. De template bevat twee Label-controls die allebei een binding hebben met een eigenschap uit de datasource. De binding van *usernameLabel* geeft aan dat de waarde van de eigenschap *Username* gekoppeld moet worden aan de *text*-eigenschap van *usernameLabel*. De *text*-eigenschap wordt op dezelfde manier aan de eigenschap *EmailAddress* gekoppeld.

```
<form id="form1" runat="server">
   <div id="content">
      <div class="left">
         <atlas:ListView   ID="userList" runat="server"
                           ItemTemplateControlID="row">
            <LayoutTemplate>
               <ul id="Ul1" runat="server">
                  <li id="row" runat="server">
                     <b>
                        <atlas:Label ID="usernameLabel" runat="server">
                           <bindings>
                              <atlas:Binding DataPath="Username"
                                             Property="text" />
                           </bindings>
                        </atlas:Label>
                     </b>
                     <br />
                     <atlas:Label ID="emailAddressLabel" runat="server">
                        <bindings>
                           <atlas:Binding DataPath="Emailaddress"
                                          Property="text" />
                        </bindings>
                     </atlas:Label>
                  </li>
               </ul>
            </LayoutTemplate>
         </atlas:ListView>
      </div>
   </div>
   <div id="authenticationPanel">
      <div id="errorMessage"></div>
      Username: <input type="text" id="username" runat="server" /><br />
      Password: <input type="password" id="password" runat="server" />
      <button id="submitButton" runat="server"
              onclick="AuthenticateUser();">Submit</button>
   </div>
   <div id="authenticatedPanel"
        style="visibility:hidden;">Hartelijk welkom!</div>
</form>

<script language="JavaScript" src="/AuthenticationService.asmx/js"
        type="text/javascript"></script>

<script language="javascript" type="text/javascript">
   function AuthenticateUser()
   {
      var username = document.getElementById("username");
      var password = document.getElementById("password");

      AuthenticationService.Authenticate( username.value,
         password.value, OnComplete);
   }

   function OnComplete(result)
   {
      if( result == true )
      {
         var authenticationPanel = document.getElementById("authentication
            Panel");
         var authenticatedPanel  = document.getElementById("authenticated
            Panel");

         authenticatedPanel.style.visibility = "visible";
         authenticationPanel.style.visibility = "hidden";
      }
      else
      {
         var errorMessage = document.getElementById("errorMessage");
         errorMessage.innerHTML = "Deze combinatie kon niet gevonden
            worden.";

         AuthenticationService.GetUsers( OnGetUsersComplete);
      }
   }

   function OnGetUsersComplete(results)
   {
      var userList = document.getElementById("userList");
      userList.control.set_data(results);
   }
</script>
```
Codevoorbeeld 9.

Het binden van de data aan de ListView gebeurt in de functie *OnGetUsersComplete*. Deze functie wordt aangeroepen wanneer de aanroep naar *AuthenticationService.GetUsers* een resultaat heeft teruggegeven. In *OnGetUsersComplete* wordt het resultaat van de webservice (een array van User-object) als datasource van de ListView gezet. Deze wordt vervolgens door de Atlas-engine gerenderd en op het scherm getoond. Dit alles gebeurt zonder dat de hele pagina ververst hoeft te worden, wat voor de gebruiker een veel interactievere ervaring oplevert.

Nog even geduld...

Al deze voorbeelden zijn gemaakt met de PDC-previewversie van ASP.NET Atlas, die overigens alleen werkt met Visual Studio 2005 Beta 2. Deze preview is zeker geen bètaversie, maar meer een vroege alfaversie. Het Atlas-team denkt ergens in 2006 een bètaversie beschikbaar te hebben die ook voor productiedoeleinden mag worden gebruikt. Nog even geduld dus. Tot die tijd zullen we ons moeten vermaken met de mogelijkheden die ASP.NET 1.1 en 2.0 ons bieden.

Erwin Werkman is werkzaam bij The Vision Web (http://www.thevisionweb.nl) en houdt zich bezig met e-commerce-projecten. Hij is te bereiken via erwin.werkman@thevisionweb.nl.

Nuttige internetadressen

Ajax: A New Approach to Web Applications - Jesse James Garret:
http://www.adaptivepath.com/publications/essays/archives/000385.php
Ajax.NET - A free library for the Microsoft .NET Framework:
http://ajax.schwarz-interactive.de/csharpsample/default.aspx
Atlas Beta site: http://atlas.asp.net