Erwyn van der Meer is softwarearchitect en werkt bij LogicaCMG

Ontwikkelen van een interactieve tv-website

ASP.NET 2.0 VOOR WINDOWS MEDIA CENTER EN SET-TOP BOXEN

Een interactieve tv-website is een website die op jouw tv getoond kan worden en waar je met een afstandsbediening door heen kunt navigeren. Het ontwikkelen van zo'n website levert een aantal extra uitdagingen op. In dit artikel beschrijft de auteur wat deze uitdagingen zijn en hoe ASP.NET 2.0 bij een aantal hiervan kan helpen.

Interactieve tv is een mengvorm van videobeelden, tekst en grafische content. Je kunt als gebruiker via de afstandsbediening zelf beslissen welke informatie je wilt lezen en welke video je wilt bekijken. Om interactieve tv te kunnen gebruiken, moet je een Windows Media Center PC of een zogenaamde Set-Top Box (afgekort STB) op de tv aansluiten. De meeste STB's en Windows Media Center kunnen een user-interface tonen op basis van html en JavaScript. Ze halen via een internetaansluiting geschikte webpagina's op van een webserver en geven deze weer. Zo'n geschikte website noemen we een iTV-website. Een voorbeeld hiervan is Rabobank TV; zie afbeelding 1.

Aandachtspunten ontwikkelen iTV-website

Surfen over een website op een tv verloopt anders dan achter de pc. Surfen op een pc is een *lean-forward* activiteit, surfen op een tv is een *laid-back* bezigheid. Een iTV-website moet je kunnen gebruiken als je lekker op de bank hangt, op twee meter of meer van de tv met alleen een afstandsbediening in je hand. Daarom moet je bij het ontwikkelen rekening houden met een aantal zaken:

- De letters moeten groot en duidelijk zijn en de teksten kort. De resolutie van een standaard tv is slechts 640x576 en de tekst moet op twee meter afstand nog leesbaar zijn.
- Er moet altijd één schermelement zijn dat de focus heeft. De focus moet met een consistente kleur worden aangegeven. Bij Rabobank TV is dat bijvoorbeeld oranje. In afbeelding 1 heeft het item 'Erven en Schenken' de focus.
- De focus moet verplaatst kunnen worden met de pijltjestoetsen op de afstandsbediening, op een toetsenbord en door er met de muiscursor overheen te bewegen. In afbeelding 2 zie je een voorbeeld van een afstandsbediening.
- Een tv geeft over het algemeen meer licht dan een monitor. Witte vlakken op een tv-scherm lichten daardoor zeer fel op. Het is daarom beter het gebruik van donkere en lichte kleuren om te draaien ten opzichte van een normale website: lichte tekst op een donkere achtergrond werkt beter dan donkere tekst op een lichte achtergrond.
- Je moet tekst kunnen invoeren via de afstandsbediening. Dat is omslachtiger dan via een toetsenbord, dus tekstinvoer moet zoveel mogelijk beperkt worden.

Afbeelding 1. Screenshot van Rabobank TV

Verschillen tussen set-top boxen

Er zijn diverse STB's in omloop in Nederland. Bijvoorbeeld die van Versatel, KPN Mine, MediaMall, UPC en Casema. Er zijn grote verschillen tussen deze devices. De belangrijkste verschillen voor het ontwikkelen van een iTV-website zijn:

- De manier waarop toetsindrukken op de afstandsbediening kunnen worden uitgelezen.
- De resolutie waarop webpagina's ontworpen moeten worden: 1024x768, 800x600 of 640x576.
- Welk deel van het scherm gebruikt mag worden om content te tonen.
- Welke browser in de STB zit: Internet Explorer, ANT Galio, ANT Fresco of Mozilla.
- De manier waarop je video's kunt starten en stoppen.
- Welke videoformaten ondersteund worden: Windows Media Video (WMV) of MPEG-4-varianten.
- Welke bugs er in de Cascading Style Sheet-ondersteuning zitten.
- Verschillen in eventhandling in JavaScript.
- Of transparantie in afbeeldingen in gif- of png-formaat ondersteund wordt.
- Of de device een goede cache heeft voor html, JavaScript, CSS-bestanden en plaatjes.

De grootste uitdaging bij de ontwikkeling van Rabobank TV was om vanuit één codebase zowel Windows Media Center als andere STB's te kunnen ondersteunen. De eerste release was specifiek bedoeld voor

Afbeelding 2. Windows Media Center-afstandsbediening

Windows Media Center. Een belangrijk ontwerpcriterium was dat er in volgende releases gemakkelijk ondersteuning voor andere STB's zou kunnen worden toegevoegd. De recentste versie ondersteunt inmiddels verscheidene devices.

Om Windows Media Center te kunnen gebruiken, heb je een geschikte versie van Windows nodig: Windows XP Media Center Edition 2005, Windows Vista Home Premium of Windows Vista Ultimate. Windows Media Center ondersteunt verschillende applicatiemodellen. In dit artikel gaan we alleen in op iTV-websites. Een iTV-website wordt door Microsoft een *hosted HTML application* genoemd.

Windows Media Center SDK

Microsoft ondersteunt het ontwikkelen van websites voor Windows Media Center via de Windows Media Center SDK. De SDK bevat bijvoorbeeld JavaScript-bestanden waardoor je op eenvoudige wijze elementen op je webpagina de focus kunt geven of deze focus via de afstandsbediening kunt verplaatsen. Ook laat de SDK zien hoe je video's kunt weergeven op de pagina. We zijn begonnen met het gebruiken van de voorbeelden uit deze SDK voor een proof-of-concept. Maar bij bestudering van de code uit de SDK bleek dat gebruik wordt gemaakt van Internet Explorer-specifieke extensies:

- htc-bestanden om onmouseover-effecten in te definiëren.
- Filter-property voor opacity in CSS.
- Gebruik van custom attributen als MCFocusable voor html-elementen om aan te geven of een element de focus kan krijgen.
- Gebruik van de document.all-collectie in dhtml-code in plaats van het standaard Document Object Model.
- Uitlezen van toetsaanslagen via window.event.keyCode.

De code uit de SDK kan dus niet zonder meer gebruikt worden voor andere browsers. Een ander punt dat ons opviel, is dat er nergens

```
<browsers>
  <browser id="WMCShell" parentID="IE6to9">
    <identification>
      <capability name="extra" match="Media Center PC" />
      <capability name="extra" match="MediaCenter" />
    </identification>
    <capture>
      <capability name="extra"
        match="Media Center PC (?'mceversion'(\d+\.\d+))" />
      <capability name="extra"
        match="MediaCenter (?'mceshell'(\d+\.\d+\.\d+\.\d+))" />
    </capture>
    <capabilities>
      <capability name="mceVersion" value="${mceversion}" />
      <capability name="mceShellVersion" value="${mceshell}" />
    </capabilities>
  </browser>
</browsers>
```

Codevoorbeeld 1. WMC.browserbestand om Windows Media Center mee te herkennen

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head>
    <title>Browser Sniffing</title>
  </head>
  <body>
    <h1>Browser Sniffing</h1>
    <p>Browser ID: <%= Request.Browser.Id %></p>
    <asp:Panel runat="server" Visible="false"
        WMCShell:Visible="true">
      Alleen zichtbaar in Windows Media Center<br />
      Media Center versie:
      <%= Request.Browser.Capabilities["mceVersion"]%>
    </asp:Panel>
    <asp:Label ID="Label1" runat="server" Visible="false"
        VersatelGalio:Visible="true">
      Alleen zichtbaar op Versatel STB met ANT Galio browser
    </asp:Label>
  </body>
</html>
```

Codevoorbeeld 2. ASP.NET-pagina BrowserSniffing.aspx om browser sniffing te testen

html-hyperlinks ( ) gebruikt worden voor de links tussen pagina's. Navigatie wordt volledig via JavaScript-code geregeld. Dit maakt het lastig om de pagina's te kunnen beheren met een contentmanagementsysteem (CMS). Het was een wens van de klant om een CMS in te kunnen zetten. We hebben er daarom voor gekozen de JavaScript-code uit de SDK geheel te herschrijven. We hebben de JavaScript-code gesplitst in een cross-browser compatibel deel en een relatief klein stuk code dat specifiek per STB wordt geschreven. In dit artikel ga ik verder niet in op de JavaScript-code en richt ik me specifiek op ASP.NET-technologie die het mogelijk maakt aan elke STB andere code te serveren.

ASP. NET 2.0 browser sniffing

Vanwege de verschillen tussen STB's is het belangrijk op de webserver te kunnen detecteren welk device de website opvraagt. Vervolgens kunnen webpagina's, JavaScript en CSS-bestanden geserveerd worden die aangepast zijn voor dat device. ASP.NET 2.0 biedt uitstekende faciliteiten voor browser sniffing. ASP.NET gebruikt hiervoor de User-Agent HTTP-header die door de browser meegestuurd wordt met elk HTTP-verzoek. Voor Windows Media Center op Windows Vista is de User-Agent-string bijvoorbeeld "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.0; SLCC1; .NET CLR 2.0.50727; **Media Center PC 5.0;** .NET CLR 3.0.04506; **MediaCenter 6.0.6000.16386)**". In deze string zitten twee substrings waaraan je Media Center kunt herkennen. Dit zijn de vetgedrukte gedeeltes.
In de ASP.NET-website kun je een folder aanmaken met de naam App_Browsers. Daarin kun je bestanden plaatsen met de browserextensie. Dat zijn XML-bestanden met een bepaald formaat. In zo'n browserbestand kun je browsers laten herkennen op basis van regular expression matching op de User-Agent-string. In codevoorbeeld 1 zie je het WMC.browserbestand waarmee Windows Media Center te detecteren is. ASP.NET 2.0 detecteert zelf standaard al veel verschillende browsers en kent een hiërarchisch model van browserherkenning. Internet Explorer 7.0 wordt automatisch herkend door ASP.NET 2.0 en krijgt als id *IE6to9*. De User-Agent-string van Windows Vista Media Center is een verbijzondering van *IE6to9*. Daarom zie je in codevoorbeeld 1 *parentID="IE6to9"* staan.

Codevoorbeeld 2 toont een eenvoudige ASP.NET-pagina die aangeeft welke browser is gedetecteerd via *Request.Browser.Id*. Deze pagina maakt gebruik van een ASP.NET 2.0-feature om per browsertype andere waarden aan properties van webcontrols te kunnen geven. De pagina bevat twee Panel-controls die standaard onzichtbaar zijn vanwege *Visible="false"* in de mark-up. De eerste Panel is desondanks zichtbaar in Windows Media Center vanwege de override *WMCShell: Visible="true"*. Het tweede Panel zal alleen zichtbaar zijn op de device VersatelGalio. Dit is een van de mogelijkheden van ASP.NET 2.0 om het gedrag van een pagina te laten variëren per browsertype. In afbeelding 3 is het resultaat te zien van deze pagina in Internet Explorer 7.0 en afbeelding 4 toont het resultaat in Windows Vista Media Center.

XHTML transformeren met device-specifieke XSLT

Een geavanceerdere mogelijkheid is om XSLT te gebruiken. Het idee is dat je uitgaat van een generieke XHTML-pagina zonder device-specifieke zaken. Zo'n pagina kan gemakkelijk door een contentmanagementsysteem worden geproduceerd. Per device selecteer je een XSLT die je gebruikt om de generieke pagina te transformeren naar een device-specifieke pagina. Om de juiste XSLT te selecteren, gebruik je ASP.NET browser sniffing. Het voert helaas te ver om deze mogelijkheid hier verder te bespreken.

CSS genereren met ASP. NET

STB's hanteren verschillende schermresoluties waar je rekening mee moet houden. Helaas zitten er vaak bugs in de ondersteuning in browsers voor lay-out met afmetingen in percentages in Cascading Style Sheets (CSS). Je ontkomt er daarom niet aan de afmetingen van schermelementen in pixels op te geven. Je zou deze afmetingen

Afbeelding 3. Resultaat van browser sniffing in IE 7.0

Afbeelding 4. Resultaat van browser sniffing in WMC op Vista

per resolutie vooraf kunnen uitrekenen en per devicetype in aparte CSS-bestanden kunnen opslaan. Maar dat is niet zo onderhoudsvriendelijk. Je kunt beter deze berekening on-the-fly laten uitvoeren door ASP.NET op basis van één source-bestand. Normaal gesproken gebruik je ASP.NET-pagina's om (X)HTML-output te genereren. ASP.NET kan ook gebruikt worden om andere typen output te genereren, bijvoorbeeld CSS. Je kunt dit aangeven aan de browser door de MIME-type van de output op text/css te zetten in de code-behind van de ASPX-pagina; zie codevoorbeeld 3. In de ASPX-pagina kun je methodes uit de code-behind aanroepen. Bijvoorbeeld de PixSize-methode; zie codevoorbeeld 4. Deze methode schaalt de gespecificeerde afmeting van 24 pixels op basis van de resolutie van de gedetecteerde browser. In plaats van 24px zou er bijvoorbeeld 18px uit de berekening kunnen komen.

Caching per device

Dynamisch gegenereerde output kan prima cachebaar zijn. Bijvoorbeeld als je ASP.NET gebruikt om content device-specifiek te maken, terwijl de onderliggende bron statisch is. Om pagina's sneller te laten laden, is het aan te raden al deze output op zowel de client als de server te laten cachen. ASP.NET heeft een zogenaamde output-cache voor serverside caching. Het is vele malen sneller om een pagina uit de output-cache te serveren dan om hem opnieuw te genereren. Je moet hiervoor wel expliciet aangeven dat er per devicetype gecached moet worden. Anders krijgen alle devices de output voor het type dat toevallig de eerste request heeft gedaan. ASP.NET biedt hiervoor een speciale mogelijkheid. Je moet daarvoor in elke pagina die je per devicetype cachebaar wil maken het volgende opnemen:

```
Response.Cache.SetVaryByCustom("itvdevice");
```

Een geschikte plek hiervoor is de methode *OnPreRender*. Je kunt deze regel ook in een gemeenschappelijke baseclass voor jouw cachebare pagina's opnemen. Dit geeft ASP.NET het signaal dat er aan jouw *HttpApplication* gevraagd wordt hoe er gecached moet worden. In je *Global.asax* kun je daarvoor de methode *GetVaryByCustomString* overriden. Deze methode moet - per versie van de pagina die gecached moet worden - een unieke cache-identifier retourneren. In codevoorbeeld 5 zie je een implementatie van deze methode. We genereren hier een cache-identifier op basis van de identifier van het browsertype. Bij elk verzoek voor een pagina roept ASP.NET de methode *GetVaryByCustomString* aan. Nadat de cache-identifier verkregen is, wordt gekeken of er voor deze pagina al output gecached is met deze identifier. Als dat het geval is, dan wordt de gecachede versie geretourneerd. Zo niet, dan wordt de ASP.NET-pagina uitgevoerd om output te genereren.

```
protected override void OnPreRender(EventArgs e)
{
   base.OnPreRender(e);
   Response.ContentType = "text/css";
}
```

Codevoorbeeld 3. Bepalen dat CSS als output gegenereerd wordt

```
body
{
   color: #FFF;
   font: <%= PixSize(24) %> Arial, Verdana, sans-serif;
   font-weight: bold;
}
```

Codevoorbeeld 4. Berekening met behulp van methode PixSize

```
/// <summary>
/// Overrides the default application-wide implementation
/// of the VaryByCustom property to implement a custom
/// output caching policy.
/// </summary>
/// <param name="context">The <see cref="HttpContext"/> of the
/// current request.</param>
/// <param name="custom">The custom string that specifies which
/// cached response is used to respond to the current
/// request.</param>
/// <returns>If the value of <paramref name="custom"/> is
/// "itvdevice" returns the id of the sniffed browsertype,
/// otherwise returns the value returned by the base
/// implementation.</returns>
public override string GetVaryByCustomString(HttpContext context,
                                             string custom)
{
   if (custom == "itvdevice")
      return "ITVDEVICE=" + context.Request.Browser.Id;
   else
      return base.GetVaryByCustomString(context, custom);
}
```

Codevoorbeeld 5. Implementatie van de methode GetVaryByCustomString in Global.asax

Extra uitdagingen

Het ontwikkelen van een iTV-website zorgt voor een aantal extra uitdagingen. ASP.NET 2.0 biedt geen speciale out-of-the-box ondersteuning voor het bouwen van iTV-websites die geschikt zijn voor verschillende STB's. Maar het framework is dusdanig krachtig en flexibel dat het relatief eenvoudig is STB-detectie toe te voegen en om device-specifieke output te genereren. In dit artikel werden hiervoor een aantal mogelijkheden aangereikt.

Erwyn van der Meer is softwarearchitect en werkt sinds 1999 bij de divisie Financial Services van LogicaCMG (http://www.logicacmg.com/nl). Voor vragen en opmerkingen kun je hem bereiken op erwyn.van.der.meer@logicacmg.com. Zijn blog kun je vinden op http://bloggingabout.net/blogs/erwyn/.

Referenties

Windows Media Center SDK: http://www.microsoft.com/windowsxp/mediacenter/developer/

Rabobank TV: http://itv.rabobank.nl

Blog over Rabobank TV: http://bloggingabout.net/blogs/erwyn/archive/2006/02/14/11053.aspx