Stefan Kamphuis is werkzaam bij Capgemini

DotNetNuke

EEN VLIEGENDE START VOOR WEBAPPLICATIES

Zoals steeds meer ontwikkelaars ontdekken, is DotNetNuke een krachtig framework waarin men zijn eigen webapplicaties kan ontwikkelen. Een eigen webapplicatie draait in DotNetNuke en tijdens het ontwikkelen profiteer je van alle voordelen van het framework.

Het ontwikkelen van modules voor DotNetNuke is feitelijk niet anders is dan het ontwikkelen van een ASP.NET-applicatie. Dat laat natuurlijk onverlet dat enige kennis van DotNetNuke wel vereist is om te kunnen bepalen hoe een ontwikkelaar optimaal gebruik kan maken van het framework. Dit artikel beoogt de ontwikkelaar die nog onbekend is met DotNetNuke inzicht te geven in de mogelijkheden van het framework. Om dit doel te bereiken zal eerst een beeld worden gegeven van de opbouw van websites, pagina's en modules in DotNetNuke. Daarna worden achtereenvolgens de architectuur van DotNetNuke en de architectuur van de standaard modules toegelicht. Ten slotte wordt stilgestaan bij hoe eigen modules aansluiten op de standaard componenten van DotNetNuke.

Wat is DotNetNuke?

Geregeld blijkt dat er enkele hardnekkige misverstanden bestaan als het gaat om wat DotNetNuke eigenlijk is. Het is van belang deze misverstanden de wereld uit te helpen om er zo een beter beeld van te krijgen. Ten eerste is DotNetNuke geen contentmanagementsysteem (CMS). Het beschikt wel over contentbeheerfunctionaliteit, maar zaken als workflow en content-staging ontbreken vooralsnog. Daar staat tegenover dat DotNetNuke veel sneller een website up-and-running heeft dan de meeste contentmanagementsystemen, maar dat is dan ook niet het doel van een CMS. Ten tweede is DotNetNuke geen SharePoint-kloon. Waar SharePoint bijvoorbeeld erg sterk is in de integratie met Office, ontbreekt dit in DotNetNuke. Aan de andere kant laat DotNetNuke zich gemakkelijker dan SharePoint door een webmaster aanpassen en is men veel vrijer in het grafische ontwerp van de website. Ten slotte is DotNetNuke geen kant-en-klare website. Direct na installatie is DotNetNuke weliswaar een volledig functionele website, maar dan worden maar weinig van de mogelijkheden die DotNetNuke biedt, benut. Eigenlijk heeft DotNetNuke twee gezichten: voor de webmaster is het een hulpmiddel om eenvoudig een website mee te maken en te beheren, voor de ontwikkelaar is het een raamwerk waarin men eenvoudig eigen webapplicaties kan laten functioneren. Het voordeel is evident: maak je webapplicatie in DotNetNuke en je krijgt er een complete website bij.

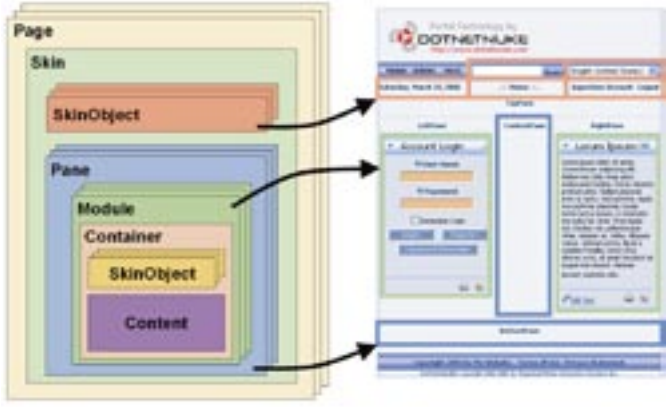
Tipje van de sluier

Om alvast een tipje van de sluier op te lichten is het aardig enkele voorbeelden te noemen van zaken die een ontwikkelaar uit handen wordt genomen, wanneer hij van DotNetNuke gebruikmaakt om een webapplicatie te realiseren.

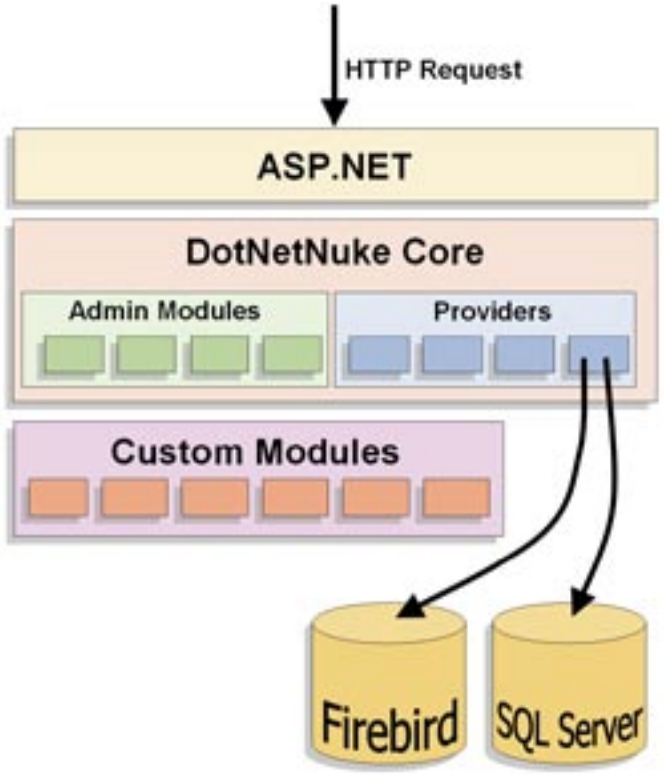
- Uitgebreide mogelijkheden voor de webmaster: in principe kan iemand die met Word kan werken ook de inhoud van een website met DotNetNuke onderhouden.
- Er is een groot aantal classes opgenomen in DotNetNuke die de ontwikkelaar veel werk uit handen neemt: van het zippen tot unzippen, van Ajax-toepassingen tot caching, van een wizard-framework tot een User Control Library.
- Het installeren van nieuwe versies van zowel DotNetNuke zelf, als van (al dan niet zelf gemaakte) modules, kan volledig geautomatiseerd worden uitgevoerd.
- De ontwikkelaar blijft volledig vrij om de architectuur van zijn voorkeur te kiezen.
- Meertaligheid (lokalisatie) is standaard ingebouwd. Dit is al sinds begin 2005 geïmplementeerd op de standaard ASP.NET 2.0-manier.
- Eigen modules kunnen eenvoudig worden 'aangesloten' op de zoekmachine van DotNetNuke, zodat ook deze inhoudelijk doorzocht kunnen worden.

Opbouw van een DotNetNuke website

In afbeelding 1 is te zien hoe een website in DotNetNuke is opgebouwd. Allereerst bestaat een site uit één of meer pagina's (*Pages*). Op iedere pagina is een *Skin* van toepassing. In een skin ligt het uiterlijk van een pagina vast. Een skin omvat in DotNetNuke veel meer dan alleen kleur en lettertype. Ook de indeling van de pagina en de plaats van vaste objecten zijn in de skin vastgelegd. Deze vaste objecten, *SkinObjects*, zijn (vaak kleine) stukjes functionaliteit die altijd op een pagina dienen te staan en waar de webmaster geen invloed meer op hoeft te hebben. Een voorbeeld van een dergelijk *SkinObject* is het 'Login'-linkje. Naast *SkinObjects* zijn in een skin ook één of meer *Panes* opgenomen. Een pane is een deel van de pagina dat door de webmaster kan worden ingevuld met één of meer *Modules*. Dit zijn de voor de webmaster beschikbare brokken functionaliteit. Er is een aantal modules standaard meegeleverd, maar er zijn er nog veel meer op internet te vinden, zowel gratis als tegen een geringe vergoeding. Een eigen webapplicatie in DotNetNuke, zal ook een module worden. Zoals een page een skin heeft die het uiterlijk bepaalt, heeft een module een *Container*. Net als een skin, kan een container *SkinObjects* bevatten. Ten slotte is uiteraard de *Content* (functionaliteit) van de module opgenomen.



Afbeelding 1. Opbouw van een pagina in DotNetNuke en een voorbeeld van een implementatie



Afbeelding 2. Architectuur van DotNetNuke

Verscheidene websites op één installatie

Een aspect van DotNetNuke dat nog niet aan bod is geweest, is de mogelijkheid om verscheidene website op dezelfde installatie te draaien. Wanneer requests voor verschillende domeinnamen of subdomeinnamen op dezelfde virtual directory in IIS (op dezelfde webserver) uitkomen, zorgt DotNetNuke er voor dat de gebruiker de juiste website te zien krijgt. Deze sites hebben in DotNetNuke geen enkele relatie met elkaar en staan volledig los van elkaar. Hoewel beide sites van dezelfde database en webserver gebruikmaken, zal een bezoeker van de ene website niets merken van de aanwezigheid van de andere website.

Globale architectuur van DotNetNuke

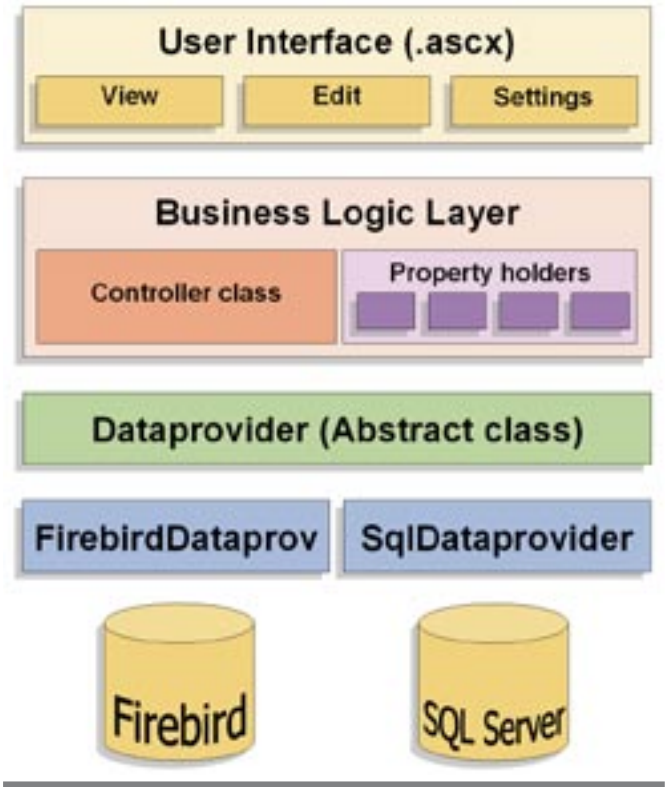
Na de opbouw van pagina's en het bekend raken met de verschillende componenten op een pagina is het goed om wat meer te weten te komen over de architectuur van DotNetNuke zelf. In afbeelding 2 is een schematische weergave hiervan te zien.

De DotNetNuke-core is zelf niets meer en niets minder dan een ASP.NET-applicatie, die gebruikmaakt van de mogelijkheden die het .NET Framework en objectoriëntatie bieden. Voor het beheer van de applicatie is een aantal *Admin Modules* opgenomen. Een deel van de functies van DotNetNuke wordt via p*roviders* beschikbaar gesteld, zodat een beheerder zelf een keuze kan maken uit de beschikbare implementaties van een provider. Voor de inhoud van de website zelf zijn *Custom Modules* bedoeld. Zoals gezegd is DotNetNuke zelf gewoon een ASP.NET-applicatie. De applicatie bevat echter slechts één aspx-bestand: Default.aspx. Daarnaast bevat DotNetNuke (uiteraard) een groot aantal classfiles, usercontrols en custom controls. Een belangrijk deel van de core van DotNetNuke wordt gevormd door alle zogenaamde Admin Modules. Dit zijn alle modules die een vast onderdeel van DotNetNuke zijn en waarmee de website wordt beheerd. Het is, zonder de core te wijzigen, niet mogelijk deze functionaliteit aan te passen of te verwijderen. Maar het is een koud kunstje om een Admin Module te kopiëren, aan te passen en in plaats van de standaardmodule op te nemen in een website. Vermoedelijk zal niemand hier snel behoefte aan hebben, maar het geeft wel aan dat DotNetNuke zeer open en flexibel is.

Voor een toenemend aantal functionaliteiten is in DotNetNuke het Provider Model geïmplementeerd. In het kort komt het provider model er op neer dat DotNetNuke zelf een abstracte klasse bevat, die door één of meer providers geïmplementeerd wordt. In de web.config wordt vervolgens geconfigureerd welke van de implementaties wordt gebruikt. Het belangrijkste voorbeeld van dit providermodel is zonder twijfel de DataProvider. DotNetNuke en alle standaard modules bevatten een abstracte klasse DataProvider. De in deze klasse gedefinieerde methoden worden geïmplementeerd in een klasse SqlDataProvider. Op dit moment (mei 2006) wordt er gewerkt aan een FirebirdProvider, zijn er plannen voor een MySqlProvider en is een OracleProvider commercieel verkrijgbaar. Het providermodel zorgt ervoor dat alleen de web.config hoeft te worden aangepast om een ander dbms te gebruiken.
Hetzelfde providermodel wordt in DotNetNuke ook gebruikt voor onder andere:

- Logging: standaard is er de keus tussen logging in de database en logging in een XML-bestand.
- HtmlEditor: hiermee is de WYSIWYG-editor die in de website gebruikt wordt te configureren. Er wordt er één meegeleverd, maar er zijn er verschillende gratis en commercieel verkrijgbaar. Ook bekende componentenleveranciers als Telerik stellen bij hun editor een DotNetNuke-HtmlEditorprovider beschikbaar.
- FriendlyUrl: deze provider zorgt dat zoekmachinevriendelijke URL's worden gebruikt voor een website www.dotnenuke.com/Blogs/Default.aspx in plaats van www.dotnenuke.com/default.aspx?tabid=825).
- Caching: de meegeleverde implementaties van de cachingprovider maken het mogelijk de inhoud van de cache niet in het geheugen, maar op een bestandssysteem te bewaren. Hierdoor is DotNetNuke ook in een webfarm-omgeving te gebruiken.

Onder *Custom Modules* worden die stukken functionaliteit verstaan die de webmaster tot zijn beschikking heeft om de website van inhoud te voorzien. Denk hierbij aan een Custom Module voor tekst, voor afbeeldingen of een FAQ, maar bijvoorbeeld ook een weblog of forum. Custom Modules staan in DotNetNuke los van de core. Hoewel ASP.NET vereist dat de Module Controls (ascx-bestanden) in een folder in de webapplicatie staan, wordt de code wel in een eigen assembly gecompileerd. Een Custom Module



Afbeelding 3. Architectuur van standaardmodules

kan in DotNetNuke worden geïnstalleerd door eenvoudigweg een zipfile te uploaden. Een dergelijke zipfile wordt ook wel *Private Assembly* genoemd. DotNetNuke zorgt na het uploaden ervoor dat de module voor de webmaster beschikbaar komt, zodat deze de nieuwe module op de website kan toepassen.

Architectuur Modules

In DotNetNuke is een groot aantal modules standaard beschikbaar. Deze modules zijn alle volgens dezelfde architectuur opgezet. Deze architectuur wordt ook gevolgd door veel van de beschikbare niet-standaard modules. In afbeelding 3 is deze architectuur op hoofdlijnen weergegeven.

Het is duidelijk dat hier gekozen is voor een drielagen model, met een User Interface-laag, een Business Logic-laag en een Data-laag. Opvallend is dat ook hier, in de datalaag, gebruik wordt gemaakt van het providermodel, net als in DotNetNuke zelf. DotNetNuke vereist dat een module over ten minste één User Control beschikt die een overerving van de DotNetNuke-basisklasse PortalModuleBase is. De standaardmodules beschikken doorgaans over drie User Controls:

- Een User Control met de functionaliteit voor de bezoeker van de website (View).
- Een User Control waarmee de webmaster de inhoud van de module beheert (Edit).
- Een User Control die, geïntegreerd met de standaard instellingenpagina voor modules, dient om de instellingen van de module te wijzigen (Settings).

Samen vormen deze User Controls de *User Interface Layer (UI)* van de module. Vanuit de User Controls wordt uitsluitend gecommuniceerd met de *Business Logic Layer (BLL)*. De BLL bevat doorgaans één *Controller Class* die alle methoden bevat waarmee de UI de gegevens uit de database kan benaderen en bijwerken. De functies in de Controller Class waarmee gegevens worden opgehaald uit de database, geven gevulde *Property Holders* terug aan de UI. Deze worden doorgaans gevuld door het *Common Business Object*. Deze *Property Holders* zijn eenvoudige objecten waarin geen functionaliteit is geïmplementeerd. Zij representeren een tabel of view, zodat de UI niets van de namen van databasevelden hoeft te weten.

Common Business Object

De *Controller Class* maakt gebruikt van het *Common Business Object (CBO)*. Het CBO is een goed voorbeeld van een onderdeel van DotNetNuke waarmee de ontwikkelaar tijd kan besparen. Door aan het CBO een *DataReader* en een *Property Holder Class* te geven, kan met één statement een object of arraylist van objecten worden gevuld, rekeninghoudend met eventuele Null-waarden in de database. De DataReader hoeft dus niet record-voor-record en veld-voor-veld te worden uitgelezen.

Voor interactie met de database vinden we het providermodel terug. De module bevat een abstract class *DataProvider* en ten minste één implementatie van deze abstract class. Er is een implementatie per ondersteund dbms (SQL Server, Firebird, Oracle, enzovoort). Doorgaans worden deze dataprovider-implementaties gecompileerd in een eigen assembly, maar dat is niet noodzakelijk. DotNetNuke biedt ook de mogelijkheid om databasescripts te maken die tijdens de installatie van de module worden uitgevoerd. Op deze manier kunnen tabellen en stored procedures worden gecreëerd. Daarmee kan altijd een soepele upgrade naar een nieuwe versie van een module worden verzekerd.

De eigen modules

Bij het realiseren van een webapplicatie in DotNetNuke dient met een aantal zaken rekening gehouden te worden. Zoals eerder gezegd, zal een applicatie in DotNetNuke een module worden. Dat betekent dat de module geïnstantieerd zou kunnen worden in iedere website van een DotNetNuke-installatie (DotNetNuke kan immers in één installatie verscheidene websites bevatten), op verschillende pagina's in een website en zelfs meer keren op dezelfde pagina. Vooraf zal moeten worden bedacht hoe de applicatie zich moet gedragen in dergelijke gevallen. Hoe een module zich gedraagt wanneer hij op verschillende plaatsen wordt gebruikt, is natuurlijk volledig afhankelijk van de functionele eisen, maar in ieder geval zal de ontwikkelaar zich bewust moeten zijn van deze situatie en moeten bedenken hoe de applicatie hierop moet reageren. De meeste standaardmodules houden de inhoud van verschillende instanties in een website gescheiden. De reden hiervoor ligt voor de hand: een tekst op de ene pagina dient vaak los te worden beheerd van de tekst op een andere pagina. Een voorbeeld van een module die de inhoud slechts op het niveau van de website gescheiden houdt, is de *Forum Module*. Een webmaster kan de forum-module op verschillende pagina's van zijn website gebruiken, maar hij heeft op zijn website maar één set forums en groepen. Wel kan hij per instantie van de module bepalen welke groepen zichtbaar zijn. Doordat handig gebruik wordt gemaakt van de mogelijkheden van DotNetNuke, heeft de webmaster veel meer mogelijkheden om zijn website en zijn forum naar eigen inzicht te gebruiken, dan bij losstaande forumapplicaties het geval zou zijn,

Eigen architectuur

Ten slotte is het van groot belang dat een ontwikkelaar zich realiseert dat DotNetNuke geen architectuur oplegt aan de applicatie. De modulearchitectuur zoals eerder beschreven is slechts de architectuur zoals deze in de meeste standaardmodules wordt gehanteerd. Het staat de ontwikkelaar volledig vrij om een andere architectuur te kiezen.

Samenvatting

DotNetNuke is een zeer uitgebreid webapplicatie-framework dat, door zijn flexibiliteit en uitbreidbaarheid, de ontwikkelaar in staat stelt om in minder tijd meer functionaliteit op te leveren. Dat is mogelijk doordat veel van het gebruikelijke 'loodgieterswerk' niet meer hoeft te gebeuren, maar daarnaast ook doordat er al veel functionaliteit kant-en-klaar beschikbaar is, hetzij gratis, hetzij tegen (vaak) geringe vergoeding. Kortom, ga naar www.dotnetnuke.com/ en download de starterkits voor Visual Studio 2005 (of Visual Web Developer) en je bent in no-time aan de slag.

Stefan Kamphuis is werkzaam bij Capgemini. De afgelopen zes jaar heeft hij zich vrijwel uitsluitend met het Microsoft-platform beziggehouden. De ontwikkeling van DotNetNuke volgt hij sinds 2003 op de voet. Als actief lid van de Nederlandse DotNetNuke-gebruikersgroep heeft Stefan gesproken op DevDays 2006. Bij Capgemini maakt Stefan sinds ruim een jaar ook professioneel gebruik van DotNetNuke, tot volle tevredenheid van de betrokken klanten.

Referenties

http://www.dotnetnuke.com/
http://netherlands.dnn-usergroup.net/
http://forums.asp.net/