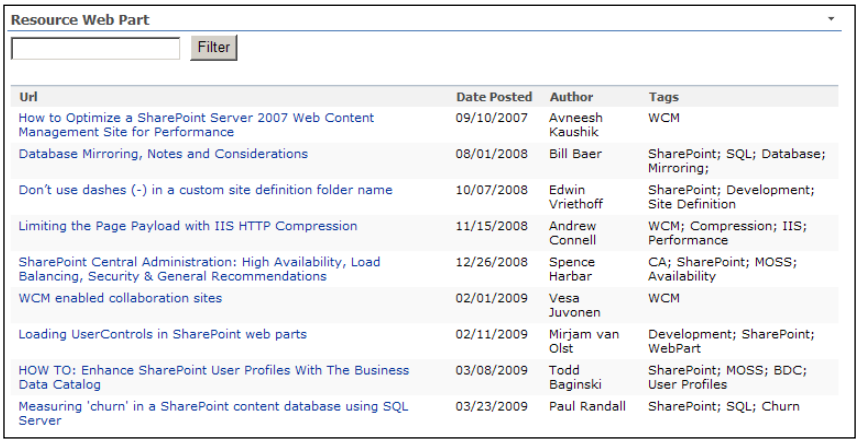
LINQ en SharePoint Development

Breed inzetbaar en zorgt voor uniforme interface

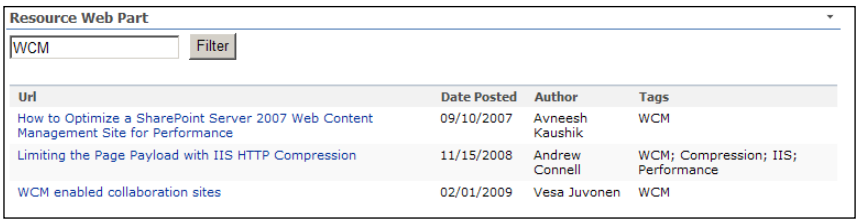
Mirjam van Olst

Met het .NET Framework 3.5 zijn de .NET-talen uitgebreid met Language Integrated Query, of LINQ. LINQ is een toolset waarmee je in de taal van jouw voorkeur (bijvoobeeld C# of VB) data uit diverse bronnen op een uniforme manier kunt benaderen en manipuleren. De databronnen kunnen bijvoorbeeld objecten in het geheugen zijn, maar het kunnen ook XML-files en databases zijn.

Het voordeel van LINQ is dat de LINQ-syntax altijd hetzelfde is. Het maakt niet uit of je nu arrays, XML-documenten, SQL-tabellen, SharePointlijsten of SharPoint-sites benadert. Een ander voordeel is dat je bij het gebruik van LINQ met strongly typed objecten werkt. Dit betekent dat de compiler je zal wijzen op eventuele fouten in je queries en dat je bij het schrijven van je query hulp van Visual Studio's IntelliSense krijgt. Dit artikel gaat in op de toepasbaarheid van LINQ voor het ontwikkelen van SharePoint oplossingen.



FIGUUR 1: EEN WEBPART VOOR HET TONEN EN FILTEREN VAN LIJST ITEMS.



FIGUUR 2: EEN WEBPART DAT DE GEFILTERDE LIJST MET ITEMS TOONT.

LINQ en SharePoint Development

Als je oplossingen voor SharePoint ontwikkelt, maak je over het algemeen veel gebruik van collecties. Denk bijvoorbeeld aan collecties van list items (SPListItemCollection), van lijsten (SPListCollection) en van sites (SPWebCollection). Deze collecties overerven, zoals de meeste collecties in het SharePoint object model, van de SPBaseCollection class. De SPBaseCollection class implementeert op zijn beurt de ICollection interface en LINQ to Objects heeft extension methods waarmee je queries kunt uitvoeren op objecten die ICollection implementeren. Het is dus mogelijk om collecties uit het SharePoint object model te benaderen met behulp van LINQ. Daarnaast kun je bij het ontwikkelen van maatwerk SharePoint oplossingen gebruik maken van de SharePoint web services. Deze geven XML terug en je kunt LINQ to XML gebruiken om deze XML te verwerken voor het gebruik van de data in je applicatie.

LINQ en lijstitems

Eerst maar eens een voorbeeld. Ik heb een SharePointsite met daarin een linkslijst met links naar externe artikelen en blogs. Ik wil graag een custom webpart hebben dat deze lijst kan filteren op basis van een zoekterm die ingevuld wordt en de resultaten vervolgens in een grid in het webpart kan tonen. Het webpart bevat een textbox voor het invoeren van de tekst waarop gefilterd moet worden, een button en een grid.

Het grid toont vier velden uit de lijst namelijk de url, de datum waarop de blog of het artikel gepost is, de auteur en de tags. Figuur 1 laat zien hoe het webpart eruit ziet als er geen filterterm is ingevuld. Het webpart toont nu dus alle items uit de linkslijst.

Figuur 2 toont dezelfde lijst, maar nu is de data gefilterd op basis van de term 'WCM'.

De oplossing bevat een class MyResource die properties bevat voor de url, de auteur, de datum en de tags en een webpart dat niks anders doet dan het laden van een user control. Dit user control bevat de daadwerkelijke logica. Ik heb gekozen voor het gebruik van een user control, omdat in een user control de opmaak van de logica gescheiden is. De opmaak wordt opgeslagen in de codefront (.ascx file) en de logica in de codebehind file (.ascx.cs file) van het user control.
Codevoorbeeld 1 toont de klassieke manier om queries uit te voeren op SharePoint lijsten, met behulp van CAML queries. De CAML query zorgt ervoor dat alleen de data die aan de filter condities voldoet uit de database opgehaald wordt. Vervolgens wordt er door de opgehaalde items geïtereerd om voor elk item een MyResource object aan te maken en deze toe te voegen aan de lijst van MyResource objecten.
Hetzelfde webpart is ook te creëren met behulp van een LINQ-query zonder gebruik te maken van CAML. Codevoorbeeld 2 toont de code voor het ophalen en filteren van items uit een SharePoint lijst met behulp van een LINQ-query. De LINQ-query maakt gebruik van de query operators WHERE, CONTAINS, ORDERBY en SELECT. Deze operators zullen iemand die weleens een T-SQL query geschreven heeft bekend voor komen, omdat dezelfde operators ook in T-SQL gebruikt worden. In LINQ kunnen deze operators gebruikt worden om vrijwel elke willekeurige databron te benaderen.

Zelfs voor dit eenvoudige voorbeeld van het ophalen en filteren van een lijst is de code met de LINQ-query korter en gemakkelijker te lezen dan de code met de CAML-query. Bovendien is het schrijven van de LINQ-query minder foutgevoelig dan het schrijven van de CAML-query. Als je een fout maakt in de CAML-query zal je code gewoon compileren. Bij het uitvoeren van de code krijg je vervolgens de melding 'Cannot complete this action'. Ten eerste weet je dus pas dat je een fout hebt gemaakt op het moment dat je het webpart gaat testen en daarnaast krijg je ook nog een foutmelding die niet goed beschrijft wat je precies fout gedaan hebt.
Er zit echter ook een groot nadeel aan het gebruik van een LINQ- query voor het filteren van SharePoint collecties. In codevoorbeeld 2 wordt 'from SPListItem item in resourceList.Items' gebruikt. Dit betekent dat de LINQ query eerst alle objecten in de resourceList lijst op zal halen, om ze vervolgens pas te filteren op basis van de condities in de WHERE clausule. Voor een kleine lijst, zoals in het voorbeeld is dit geen probleem, maar als de lijst duizend items bevat, waarvan er uiteindelijk maar vijf getoond worden dan is dit erg veel onnodige overhead. Het gebruik van LINQ queries die op deze manier SharePoint data ophalen is dus niet aan te raden op collecties die meer dan een paar items bevatten.

Codevoorbeeld 3 gebruikt het beste uit beide werelden. De data wordt in eerste instantie gefilterd met behulp van een CAML-query, dus alleen de list items die nodig zijn worden uit de database opgehaald. De LINQ query zorgt vervolgens voor de sortering van de items en voor het toevoegen van de opgehaalde data aan MyResource objecten. De meerwaarde van het gebruik van LINQ is in dit voorbeeld minder duidelijk, omdat je nu toch een CAML query nodig hebt. Dit is echter wel de aan te raden aanpak, omdat de query op deze manier geen onnodige data op zal halen. Doordat de LINQ query zorgt voor het sorteren en voor het toevoegen van de data aan MyResource objecten is de code wel nog iets gebruiksvriendelijker dan die uit codevoorbeeld 1.

```
SPList resourceList = SPContext.Current.Web.Lists["Links"];

SPQuery query = new SPQuery();
query.Query = String.Format("<Where><And><Contains><FieldRef
Name='Tags'/><Value Type='Text'>{0}</Value></
Contains><IsNotNull><FieldRef Name='URL'></FieldRef></IsNotNull>
</And></Where><OrderBy><FieldRef Name='PostedOn' Ascending='TRUE'
/></OrderBy>", FilterTextBox.Text);
SPListItemCollection listItemsColl = resourceList.GetItems(query);

List<MyResource> resourceCollection = new List<MyResource>();

foreach (SPListItem item in listItemsColl)
{
MyResource res = new MyResource();
res.Url = item["URL"].ToString();
res.Author = item["Author0"].ToString();
res.PostedOn = Convert.ToDateTime(item["PostedOn"]).ToString("MM/dd/
yyyy");
res.Tags = item["Tags"].ToString();

resourceCollection.Add(res);
}

ResourcesGrid.DataSource = resourceCollection;
ResourcesGrid.DataBind();
```

CODEVOORBEELD 1: LIST ITEMS OPHALEN MET BEHULP VAN EEN CAML QUERY.

```
SPList resourceList = SPContext.Current.Web.Lists["Links"];

// Get items, filter on tag, order by PostedOn date and assign to
MyResource object
var resourceListItems = from SPListItem item in resourceList.Items
where item["Tags"].ToString().Contains(FilterTextBox.Text)
&& item["URL"].ToString().Length > 0
orderby item["PostedOn"] ascending
select new MyResource
{
Url = item["URL"].ToString(),
Author = item["Author0"].ToString(),
PostedOn = Convert.ToDateTime(item["PostedOn"]).ToString("MM/dd/
yyyy"),
Tags = item["Tags"].ToString()
};

ResourcesGrid.DataSource = resourceListItems;
ResourcesGrid.DataBind();
```

CODEVOORBEELD 2: BAD PRACTICE - LIST ITEMS OPHALEN MET EEN DURE LINQ QUERY.

```
SPList resourceList = SPContext.Current.Web.Lists["Links"];

SPQuery query = new SPQuery();
query.Query = String.Format("<Where><And><Contains><FieldRef
Name='Tags'/><Value Type='Text'>{0}</Value></
Contains><IsNotNull><FieldRef Name='URL'></FieldRef></IsNotNull>
</And></Where>", FilterTextBox.Text);

// Filter items on tag with CAML, order by PostedOn date and as-
sign to MyResource object using LINQ
var resourceListItems = from SPListItem item in resourceList.
GetItems(query)
orderby item["PostedOn"] ascending
select new MyResource
{
Url = item["URL"].ToString(),
Author = item["Author0"].ToString(),
PostedOn = Convert.ToDateTime(item["PostedOn"]).ToString("MM/dd/
yyyy"),
Tags = item["Tags"].ToString()
```

```
};

ResourcesGrid.DataSource = resourceListItems;
ResourcesGrid.DataBind();
```

CODEVOORBEELD 3: GOOD PRACTICE - LIST ITEMS OPHALEN MET EEN COMBINATIE VAN CAML EN LINQ QUERIES.

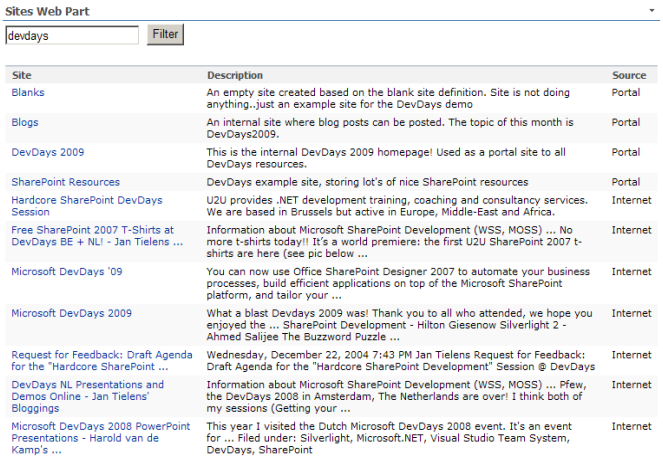
LINQ en sites uit meerdere databronnen

Voor het volgende voorbeeld heb ik een nieuwe class MyWebsite aangemaakt met daarin drie properties van websites, de titel met url, een omschrijving en een property om aan te geven of een site een portal site of een internet site is. Codevoorbeeld 4 toont wederom een gecombineerde CAML en LINQ query. De CAML query haalt uit de sites lijst in de SiteDirectory van de portal een verzameling sites op die gefilterd zijn op basis van een zoekwoord dat moet voorkomen in de titel of in de omschrijving van de site. De LINQ query sorteert deze sites vervolgens op titel en voegt ze toe aan een MyWebsite object.

Tot zover lijkt dit voorbeeld erg op het vorige, maar behalve portal sites zou het webpart natuurlijk ook internet sites kunnen tonen die met dezelfde zoekterm gefilterd of doorzocht worden. Hiervoor gebruiken we de Live Search API web service. De url van deze web service is http://api.search.live.net/searchwsdl?AppID=<jouwAppID>. Je kunt op de Live Search site een eigen AppID aanvragen door je eenvoudigweg te registreren. Vervolgens wordt door het toevoegen van een webreference naar de Live Search service in Visual Studio door Visual Studio een proxy class aangemaakt. Met deze proxy class kunnen we de Live Search service typed aanroepen. Aan de service geven we een Query object mee waarin we onze zoekopdracht definiëren. Het resultaat van de service is een collectie van WebResult objecten die onder andere properties voor de titel, de url en de omschrijving van de sites bevatten. Op basis van LINQ kunnen we deze data eenvoudig samenvoegen met die van onze SharePoint lijst. We doen dit door gebruik te maken van de UNION query operator van LINQ. Het resultaat is dat data uit twee volledig verschillende databronnen samengevoegd wordt in één collectie en getoond kan worden in één webpart. Figuur 3 toont het webpart, dat dus zowel portal als internet sites kan filteren en tonen.

```
 private IEnumerable<MyWebsite> _listOfWebsites;

SPSite site = SPContext.Current.Site;
SPList sitesList = site.OpenWeb("SiteDirectory").Lists["Sites"];

SPQuery query = new SPQuery();
query.Query = String.Format("<Where><Or><Contains><FieldRef
Name='Title'/><Value Type='Text'>{0}</Value></
Contains>><Contains><FieldRef Name='Description'/><Value
Type='Text'>{0}</Value></Contains>
</Or></Where>", FilterTextBox.Text);


// Get filtered webs from site directory, order by Title and as-
sign to MyWebsite object
var sites = from SPListItem item in sitesList. GetItems(query)
orderby item.Title ascending
select new MyWebsite
{
Site = item["URL"].ToString(),
Description = item["Description"].ToString(),
WebSiteSource = "Portal"
};

_listOfWebsites = sites;


//Build the Live Search request
SearchRequest request = new SearchRequest();

// Common request fields (required)
request.AppId = "<jouwAppID>";
request.Query = String.Format("sharepoint {0} ", FilterKeyword.
Text);
request.Sources = new SourceType[] { SourceType.Web };

// Common request fields (optional)
request.Version = "2.0";
request.Market = "en-us";

// Web-specific request fields (optional)
request.Web = new WebRequest();
request.Web.Count = 7;
request.Web.CountSpecified = true;
request.Web.Offset = 0;
request.Web.OffsetSpecified = true;

// Send the request; display the response.
SearchResponse response = service.Search(request);

var externalSites = from WebResult result in response.Web.Results
select new MyWebsite
 {
 Site = String.Format("<a href='{0}'>{1}</a>", result.DisplayUrl, re-
sult.Title),
 Description = result.Description,
 WebSiteSource = "Internet"
 };

_listOfWebsites = _listOfWebsites.Union<MyWebsite>(externalSites);

SitesGrid.DataSource = _listOfWebsites;
SitesGrid.DataBind();
```

CODEVOORBEELD 4: HET COMBINEREN SHAREPOINT DATA EN LIVE SEARCH RESULTATEN MET LINQ.



FIGUUR 3: WEBPART DAT DE GEFILTERDE LIJST MET ZOWEL PORTAL ALS INTERNET SITES TOONT.

In het tweede voorbeeld gebruik ik de UNION query operator om twee collecties van hetzelfde type samen te voegen. De mogelijkheden van LINQ zijn echter veel breder. Zo zou je een scenario kunnen hebben met twee verschillende collecties die slechts één veld gemeen hebben. Deze twee collecties zou je dan kunnen joinen op basis van deze kolom. Ik zou bijvoorbeeld bij het eerste voorbeeld met de artikelen en de blogs ook een lijst kunnen hebben met informatie over de auteurs. Ik zou dan de informatie over de blog of het artikel aan kunnen vullen met informatie over de auteur door de twee lijsten samen te voegen met behulp van de JOIN query operator van LINQ.

LINQ en SharePoint web services

Een ander scenario waarin LINQ gebruikt zou kunnen worden

voor het ontwikkelen van maatwerk oplossingen voor SharePoint is als er gebruik gemaakt wordt van de SharePoint web services. De meeste web services geven data terug in de vorm van XML en deze XML moet verwerkt worden om de data te kunnen gebruiken in de maatwerk oplossing. Een voorbeeld van een scenario waarin de SharePoint web services gebruikt zullen worden is bij het bouwen van Silverlight web parts waarin SharePoint data getoond wordt. Silverlight applicaties leven immers op de client en niet op de server en kunnen dus geen gebruik maken van het SharePoint object model. Zij kunnen echter wel de SharePoint web services aanroepen en op deze manier SharePoint data gebruiken.
Voor een project waarin de SharePoint list web service (http://<siteurl>/_vti_bin/lists.asmx) is toegevoegd als Service Reference in Visual Studio 2008 toont codevoorbeeld 5 de code die nodig is om data (in dit geval foto's) uit een lijst met de naam "Barcelona" te halen. Het aantal items dat er maximaal teruggegeven kan worden is met behulp van de rowlimit property beperkt tot 10. De web service geeft XML terug die door een LINQ to XML query verwerkt wordt. De url van de foto's en de url van de thumbnail van de foto's worden in een MyPhoto object opgeslagen. Deze worden vervolgens door het Silverlight web part getoond, maar dat is voor dit voorbeeld niet van belang. Duidelijk te zien is dat de LINQ query die nodig is voor het verwerken van de XML die teruggegeven wordt door de web service zeer eenvoudig is. De structuur van de XML die teruggegeven wordt door de lists web service is weergegeven in codevoorbeeld 6.

```
 private Action<IEnumerable<MyPhoto>> _processPhotos;

 /// <summary>
 /// Start an asynchronous service call to get my photos
/// and register a callback method to process the  result when it
comes in.
 /// </summary>
 /// <param name="processPhotos">A method that takes one parameter
and no result</param>
public void GetListItemsFromSharePoint(Action<IEnumerable<MyPhot
o>> processPhotos)
{
      _processPhotos = processPhotos;
      ListsSoapClient proxy = new ListsSoapClient();

      proxy.GetListItemsCompleted += proxy_GetListItemsCompleted;

      // Build XML elements for querying SharePoint list
      XElement fieldRef = new XElement("FieldRef");
      fieldRef.SetAttributeValue("Name", "Created");
      fieldRef.SetAttributeValue("Ascending", "FALSE");

      XElement orderBy = new XElement("OrderBy", fieldRef);
      XElement query = new XElement("Query", orderBy);
      XElement viewFields = new XElement("ViewFields");
      XElement queryOptions = new XElement("QueryOptions");

      proxy.GetListItemsAsync("Barcelona", "",
                                 query,
                                  viewFields,
                                  "10",
                                  queryOptions,
                                  "");
}

private void proxy_GetListItemsCompleted(object sender, GetListI-
temsCompletedEventArgs e)
{
      if (e.Error != null)
      {
           MessageBox.Show(e.Error.ToString());
      }
      else
      {
           XElement result = e.Result;

           var photos = from x in result.Elements().First().Ele-
ments()
                        select new MyPhoto
                        {

Url = x.Attribute("ows_EncodedAbsWebImgUrl").Value,

ThumbUrl = x.Attribute("ows_EncodedAbsThumbnailUrl").Value
                        };
           _processPhotos(photos);
      }
}
```

CODEVOORBEELD 5: HET AANROEPEN VAN DE WEB SERVICE EN HET VERWERKEN VAN DE XML DIE DOOR DE WEB SERVICE TERUGGEGEVEN WORDT.

```
<listitems …>
   <rs:data ItemCount="10">
      <z:row ows_ID="3"
         ows_Created="2009-02-18T14:41:09Z"
         ows_EncodedAbsWebImgUrl ="http://<siteurl>/
Barcelona/_w/<imagename1>.jpg"
         ows_EncodedAbsThumbnailUrl =" http://<siteurl>/
Barcelona/_t/<imagename1>.jpg "
         … />
      <z:row ows_ID="4"
         ows_Created="2009-02-18T17:15:58Z"
         ows_EncodedAbsWebImgUrl ="http://<siteurl>/
Barcelona/_w/<imagename2>.jpg"
         ows_EncodedAbsThumbnailUrl =" http://<siteurl>/
Barcelona/_t/<imagename2>.jpg "
         …/>
         …
   </rs:data>
</listitems>
```

CODEVOORBEELD 6: STRUCTUUR VAN DE XML DIE DOOR DE WEB SERVICE TERUGGEGEVEN WORDT.

LINQ extensions

Zoals eerder al genoemd is LINQ standaard in staat om verschillende soorten data uit verschillende soorten databronnen op een uniforme manier te benaderen en manipuleren. Daarnaast kan LINQ op twee verschillende manieren uitgebreid worden.

- Ten eerste is het mogelijk om standaard LINQ implementaties zoals LINQ to Objects en LINQ to XML uit te breiden of aan te passen met eigen query operators
- Ten tweede kun je geheel eigen LINQ providers schrijven

In de meeste gevallen zal het schrijven van een eigen LINQ provider niet nodig zijn en zul je voldoende hebben aan het aanpassen of uitbreiden van de LINQ query operators. Dit kan met behulp van extension methods. Je kunt extension methods gebruiken om bijvoorbeeld je eigen implementatie van de WHERE operator te schrijven als je vindt dat deze iets anders moet doen dan de standaard WHERE operator. Je kunt echter ook een eigen operator schrijven. In beide gevallen geldt dat je ervoor moet zorgen dat jouw eigen implementatie het LINQ query expression patroon moet implementeren. Ook alle standaard operators implementeren dit patroon. Door het LINQ patroon te volgen zal ook jouw eigen implementatie naadloos geïntgreerd zijn in de taal (bv C#) en gebruik kunnen maken van de IntelliSense features van Visual Studio.

Het schrijven van een eigen LINQ provider is meer werk dan het schrijven of aanpassen van een query operator. Om het gebruik van de LINQ provider in meerdere scenario's mogelijk te maken zul je minimaal een aantal query operators moeten implementren zoals de WHERE, SELECT en ORDERBY query operators.

Verschillende mensen hebben al eigen LINQ query providers geschreven. Een aantal van deze custom LINQ providers zijn te vinden op Codeplex, voor bijvoorbeeld LINQ to Active Directory, LINQ to Ebay en…jawel…LINQ to SharePoint.

LINQ to SharePoint

Bart de Smet heeft LINQ to SharePoint geschreven en dit op codeplex geplaatst. LINQ to SharePoint is een uitbreiding op LINQ die je kunt installeren op je ontwikkelomgeving. Als je dit doet worden er een aantal add-ins aan Visual Studio toegevoegd. Eén van deze add-ins zorgt ervoor dat je bij het toevoegen van een nieuw item aan een project kunt kiezen voor een LINQ to SharePoint file. Bij het aanmaken van een file van dit filetype start er een wizard die helpt om de SharePoint lijsten en bibliotheken te selecteren die je wilt gebruiken in je applicatie. De geselecteerde lijsten en bibliotheken kun je vervolgens met LINQ benaderen en LINQ to SharePoint vertaalt de LINQ-queries naar CAML-queries. Hiermee is LINQ to SharePoint eigenlijk een abstractielaag op SharePoint.
Omdat LINQ to SharePoint LINQ-queries vertaalt naar CAML queries haalt LINQ to SharePoint alleen de data op uit de database die ook daadwerkelijk opgevraagd wordt. Dit is dus een wezenlijk verschil ten opzichte van het schrijven van LINQ-queries die direct queries op SharePoint collecties uitvoeren. LINQ to SharePoint heeft eigenlijk het beste van de CAML en de LINQ werelden in zich. Het biedt de mogelijkheid om gebruik te maken van LINQ-queries en de bijbehorende IntelliSense features van Visual Studio en het is strongly typed. Daarnaast zorgt het ervoor dat alleen de data die voldoet aan de filter criteria uit de database wordt opgehaald. Het voorkomt dus dat er grote hoeveelheden data eerst opgehaalt en vervolgens gefiltered worden.
LINQ to SharePoint is gebouwd om als voorbeeld voor het bouwen van een eigen LINQ-provider te dienen. Het is dan ook helaas geen volledig afgerond product, de laatste versie op Codeplex is een alpha versie. Als je kiest voor het gebruik van LINQ to SharePoint zorg er dan dus voor dat je de oplossing die je ermee bouwt goed doortest.
Er is inmiddels ook een ander initiatief dat de moeite waard is om te bekijken, Linq4SP, dat gebouwd is door het Hongaarse bedrijf L&M solutions. Dit is een andere implementatie dan LINQ to SharePoint, maar maakt wel gebruik van hetzelfde gedachtegoed.

Conclusie

LINQ is heel breed inzetbaar en zorgt voor een uniforme interface voor het benaderen van verschillende soorten data in verschillende soorten databronnen. Ook bij het ontwikkelen van maatwerk applicaties voor SharePoint is LINQ te gebruiken. Het voordeel van het gebruik van LINQ, vooral voor niet-SharePoint developers, is dat er gebruik gemaakt kan worden van een bekende taal, waarbij ondersteuning in de vorm van IntelliSense en strongly typed objecten beschikbaar is.

Bij het gebruik van LINQ in combinatie met de collecties van objecten in het SharePoint object model is het van belang om in het oog te houden welke data je nodig hebt en welke data er daadwerkelijk opgevraagd wordt uit de database. Je kunt met het gebruik van LINQ-queries ongemerkt veel meer data ophalen dan dat je nodig hebt. Het is dus van belang om goed te kijken welke data je nodig hebt en hoe je deze zo efficiënt mogelijk op kunt halen, bijvoorbeeld door LINQ te combineren met CAML-queries.
LINQ kan het combineren van data uit meerdere databronnen erg gemakkelijk en aantrekkelijk maken, omdat de data uit alle bronnen op dezelfde wijze te benaderen is en omdat de data uit meerdere bronnen gemakkelijk samen te voegen is met behulp van LINQ.
Daarnaast kan LINQ ingezet worden bij het gebruik van de SharePoint web services voor het verwerken van de XML die door de web services teruggegeven wordt.
Er zijn dus vele scenario's te bedenken waarbij het gebruik van LINQ bij het ontwikkelen van maatwerk SharePoint oplossingen een daadwerkelijke meerwaarde kan hebben, zolang je maar goed in de gaten houdt of de query die door LINQ uitgevoerd wordt wel efficiënt is en of deze niet onnodig veel data uit de database ophaalt.

LINQ kan het combineren van data uit meerdere databronnen erg makkelijk en aantrekkelijk maken

Er zijn ook een aantal custom LINQ providers voor SharePoint te vinden op het internet, die interessant kunnen zijn om te gebruiken of om te bekijken om te zien hoe ze geïmplementeerd zijn. Het voordeel van deze custom LINQ providers is dat ze LINQ queries omzetten naar CAML queries en dat ze alleen de gefilterde data ophalen uit de database. Door gebruik te maken van deze custom providers hoef je zelf geen CAML meer te schrijven en loop je ook niet het risico dat er onnodig veel data opgehaald wordt uit de database.

Als je een SharePoint ontwikkelaar bent en je nog niet in LINQ verdiept is het zeker de moeite waard om ernaar te kijken en om ermee te spelen. Je zult merken dat het gemakkelijk is om met LINQ te werken, omdat het ingebakken zit in de taal die je al kent (C# of VB.NET) en qua syntax erg lijkt op een andere taal die je al kent (T-SQL). Ik acht de kans groot dat de rol die LINQ gaat spelen in het ontwikkelen van maatwerk SharePoint oplossingen in de toekomst steeds groter zal worden, dus tijd die je besteedt aan het bekend raken met LINQ is zeker geen verloren tijd!

Links

o LINQ to SharePoint
- http://linqtosharepoint.codeplex.com/
- http://msevents.microsoft.com/cui/WebCastEventDetails.aspx?culture=en-US&EventID=1032352642&CountryCode=US

o Linq4SP
- http://lmsolutions.web.officelive.com/linq4sp.aspx

Mirjam van Olst, is SharePoint architect/consultant bij het Information Worker Solutions Center van Macaw. Zij blogt op http://www.sharepointblogs.com/mirjam