Acceso a datos en Silverlight con WCF RIA Services

Luis Miguel Blanco Ancos

WCF RIA Services proporciona una plataforma para el desarrollo de aplicaciones basadas en el paradigma RIA, que permite aumentar la productividad del programador dentro del apartado relacionado con el acceso a datos. En el presente artículo se describen sus elementos principales, así como un ejemplo básico de utilización basado en Silverlight.

Un gran porcentaje de las aplicaciones que se desarrollan para entorno Web precisan de un mayor o menor grado de interacción con algún tipo de fuente de datos. En el contexto de este tipo de aplicaciones, basadas en una arquitectura de múltiples capas, los datos necesitan circular en ambos sentidos, desde la capa que representa el motor o fuente de datos en que se almacenan, hasta la capa intermedia, representada por la parte de la aplicación que se ejecuta en el servidor Web, para, finalmente, llegar a la capa cliente, representada por el navegador, en donde situamos la interfaz de usuario empleada para presentar los datos.

Para mejorar la experiencia de usuario, el programador puede aplicar el modelo de desarrollo RIA (Rich Internet Application) [1] en la creación de sus aplicaciones. Sin embargo, el desarrollo RIA supone añadir un mayor nivel de complejidad al proceso de creación de la aplicación, ya que implica agregar y controlar una cantidad adicional de elementos en sus capas cliente y servidora.

¿Qué es WCF RIA Services?

Si observamos las líneas generales de funcionamiento de una aplicación ASP.NET típica, veremos que en la capa intermedia situamos el grueso de las piezas que permiten su funcionamiento: lógica de la aplicación, acceso a datos, servicios y lógica de presentación; mientras que la capa cliente la dedicamos a albergar el navegador que visualizará el HTML generado por la lógica de presentación.

El esquema de desarrollo de aplicaciones RIA propone (ver figura 1), como medio de potenciar la experiencia de usuario, trasladar la lógica de presentación a la capa cliente. Esto supone incrementar la cantidad y complejidad del código que el programador debe escribir, ya que se verá obligado a desarrollar un conjunto de servicios en la capa intermedia, encargados de gestionar el flujo de datos entre esta capa y la capa cliente, lo cual incluye la creación y mantenimiento de los tipos encargados de transportar los datos en ambos extremos de esta línea de comunicación.

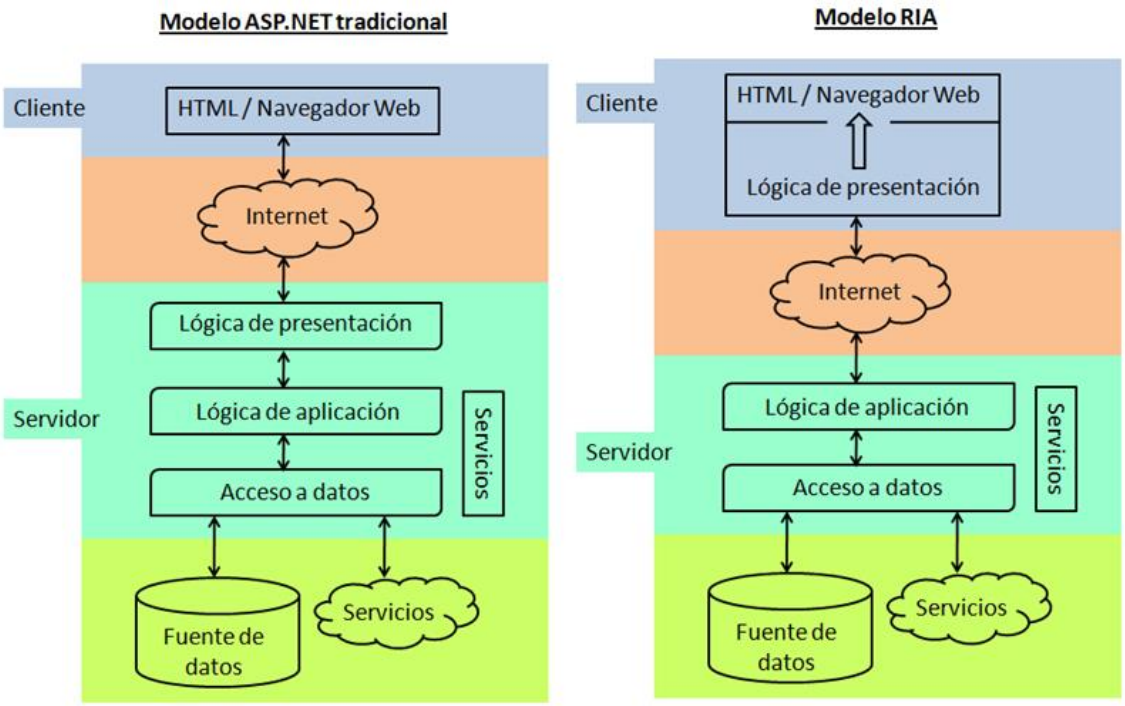


Figura 1. Modelos de ejecución ASP.NET tradicional y RIA.

Con el objetivo de facilitar el desarrollo de aplicaciones RIA, Microsoft ha desarrollado WCF RIA Services (proyecto conocido también por el nombre clave Alexandria y cuyo anterior nombre oficial fue .NET RIA Services), una tecnología que proporciona un patrón originalmente denominado “End-to-End Application Pattern” (ver figura 2), que podríamos definir como un patrón orientado, principalmente, a simplificar el desarrollo de las operaciones de manejo de datos situadas en ambos extremos (End-to-End) de las capas intermedia y cliente, pertenecientes a la arquitectura de una aplicación RIA.

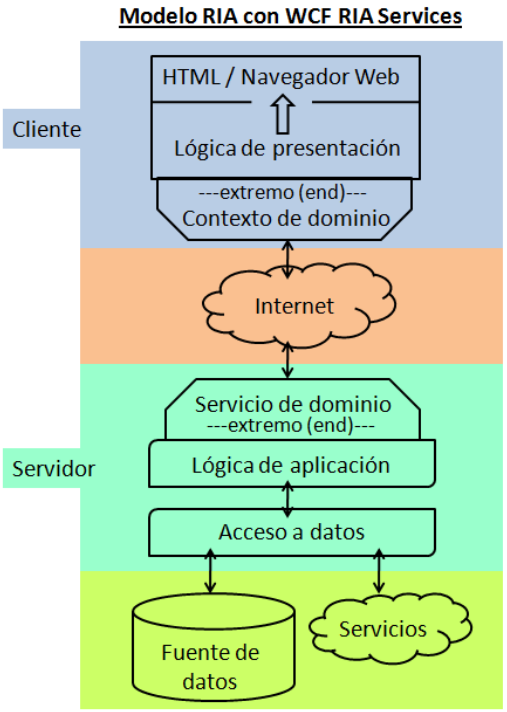


Figura 2. Modelo de ejecución RIA con WCF RIA Services.

La propuesta de WCF RIA Services consiste en ocuparse de toda esta complejidad que acabamos de describir, utilizando para ello un proceso de generación automática del código encargado de las operaciones de mantenimiento de los datos. Esta generación de código se realiza en ambas capas: intermedia y cliente, obteniendo como resultado la posibilidad de compartir metadatos, comportamiento, validación, etc., entre ambas capas de la aplicación; evitando así la necesidad de tener que implementar ciertas tareas dos veces. También se ofrecen servicios como autenticación, autorización y perfiles de usuario, los cuales facilitan el trabajo con la identidad del usuario de la aplicación. Según afirma el propio Nikhil Kothari [2], uno de los arquitectos del producto, todo este conjunto de características hará que WCF RIA Services se posicione como una herramienta RAD en el desarrollo RIA.

El cambio de denominación de .NET RIA Services hacia WCF RIA Services (al igual que ADO .NET Data Services ha pasado a llamarse WCF Data Services) ha sido debido al proceso de integración de dichas tecnologías en WCF, de forma que todo el potencial a nuestra disposición a la hora de desarrollar servicios se encuentre unificado en un lugar común: WCF, tal y como mencionan Brad Adams [3] e Ibon Landa [4] en sus respectivos blogs.

Las piezas del engranaje WCF RIA Services

Para implementar WCF RIA Services en nuestros desarrollos debemos crear un servicio de dominio (Domain Service), el cual quedará situado como extremo de la lógica de negocio de la aplicación, actuando como contenedor de las clases que proporcionan

dicha lógica. En el código de estas clases definiremos las operaciones que pueden realizar, tanto CRUD como personalizadas.

Es en esta fase del desarrollo de la aplicación, cuando WCF RIA Services se encarga de generar, a partir del modelo de entidades que previamente hayamos definido en nuestro proyecto, las clases del servicio de dominio, a las que posteriormente podremos añadir más funcionalidades. El modelo de entidades podemos crearlo mediante ADO .NET Entity Framework, LINQ to SQL, o puede ser un conjunto de tipos propio.

La otra fase de todo este proceso consiste en la creación, dentro del extremo correspondiente a la lógica de presentación, del contexto de dominio (Domain Context), o lo que es lo mismo, de aquellas clases que se comunicarán con la lógica de negocio de la aplicación, para presentar los datos obtenidos a partir de esta, y enviar las peticiones de actualización sobre la fuente de datos que se produzcan en la interfaz de usuario. La buena noticia en este caso es que WCF RIA Services también se ocupa de generar por nosotros todo este código, con lo que nuestro trabajo se ve gratamente aliviado.

Un mecanismo versátil

Aunque se adapta como un guante al funcionamiento con Silverlight, no es éste el único tipo de aplicación cliente al que WCF RIA Services puede ofrecer sus servicios. WPF, ASP.NET, Ajax, etc., se encuentran también entre los beneficiarios de esta tecnología, que a buen seguro facilitará la creación de aplicaciones RIA a toda la comunidad de desarrolladores.

Requisitos para trabajar con WCF RIA Services

Para comenzar a dar nuestros primeros pasos con WCF RIA Services podemos utilizar Visual Studio 2008 SP1 (o la versión Express correspondiente) con Silverlight 3 [5]; o bien, Visual Studio 2010 Beta 2 con Silverlight 4 [6]. En las referencias situadas al final del artículo encontraremos enlaces para descargar las herramientas adecuadas según la versión de Visual Studio con la que vayamos a trabajar.

WCF RIA Services en la práctica

El mejor modo de comprobar todo lo anteriormente comentado consiste en desarrollar nuestro propio proyecto de prueba (los ejemplos están disponibles en http://www.dotnetmania.com), para el que en este caso emplearemos Visual Studio 2010 Beta 2 como entorno de desarrollo.

Una vez iniciado Visual Studio, crearemos un nuevo proyecto de tipo Silverlight Application con el nombre PruebaWCFRIAServices. Tras aceptar el cuadro de diálogo de creación del proyecto, y si tenemos WCF RIA Services instalado, el diálogo New

Silverlight Application (ver figura 3) mostrará la casilla Enable .NET RIA Services (este literal todavía no ha sido actualizado a WCF RIA Services), que marcaremos para habilitar las funcionalidades de WCF RIA Services en nuestra solución.

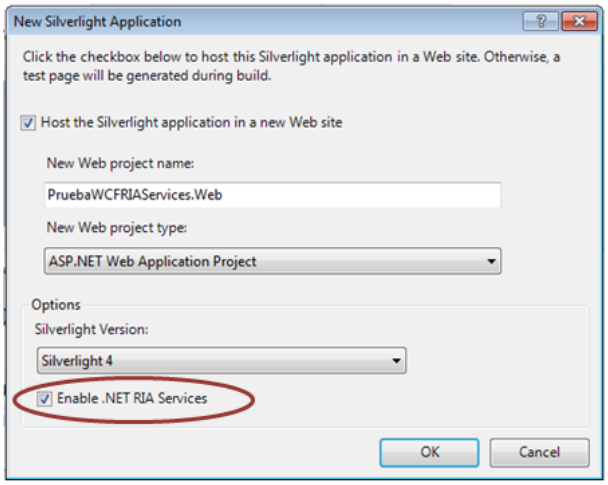


Figura 3. Habilitando WCF RIA Services en la aplicación Silverlight.

Si por algún motivo no seleccionamos inicialmente la opción Enable .NET RIA Services, siempre podremos hacerlo posteriormente; para ello, nos situaremos en el proyecto Silverlight de la solución, y seleccionando la opción de menú Project | <Nombre de proyecto> Properties se abrirá una ventana de propiedades del proyecto, en cuya pestaña Silverlight encontraremos la lista desplegable .NET RIA Services link, donde podremos establecer el proyecto Web correspondiente (ver figura 4).

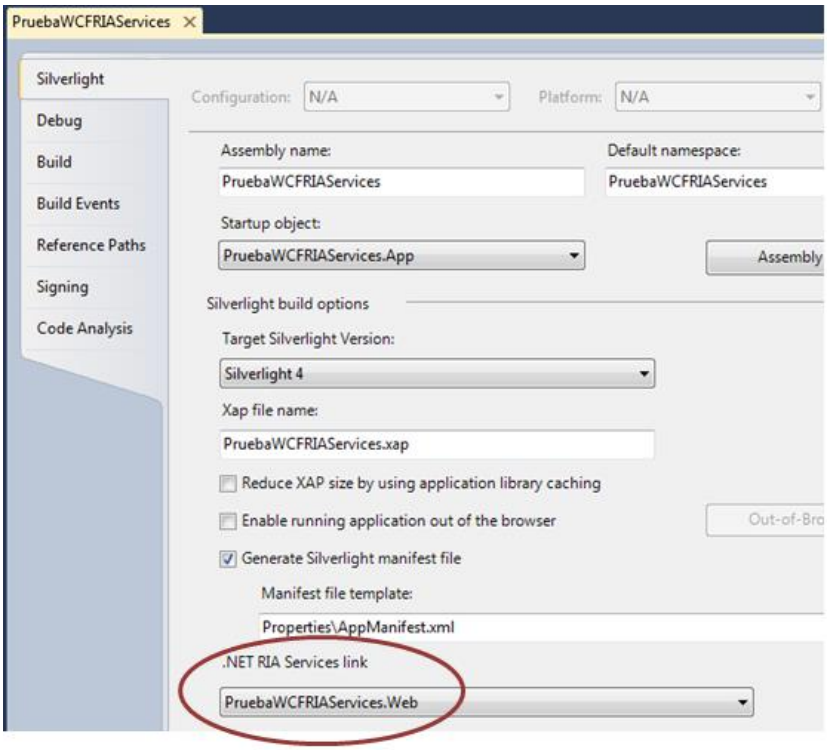


Figura 4. Establecer el proyecto Web para las funcionalidades de WCF RIA Services.

El siguiente paso consiste en definir el modelo de datos que utilizaremos en la aplicación, para lo que primeramente nos situaremos en el proyecto Web de la solución. Seleccionando la opción de menú Project | Add New Item, elegiremos, dentro del apartado Data, la plantilla ADO.NET Entity Data Model, a la que daremos el nombre NorthwindModel.edmx (ver figura 5). Dentro del asistente de configuración del modelo de datos crearemos una conexión contra la base de datos Northwind, a la que daremos el nombre NorthwindEntities (este nombre quedará grabado en el Web.config y será utilizado posteriormente como contexto de datos en la creación del servicio de dominio), por otro lado, añadiremos la tabla Customers como entidad al modelo, lo que producirá la creación de una clase con el mismo nombre, que representará a dicha entidad.

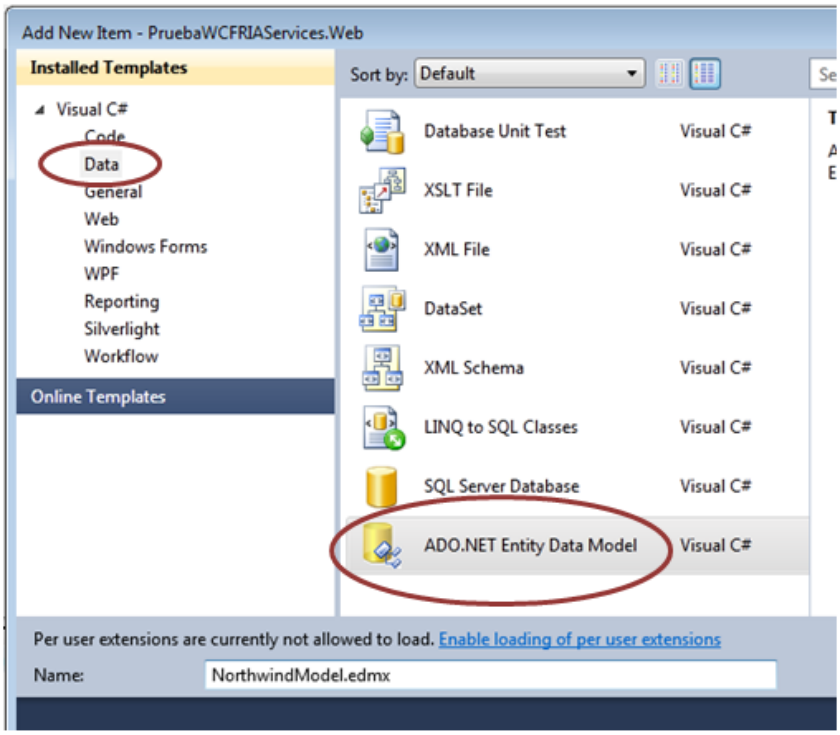


Figura 5. Añadir a la aplicación el modelo de datos.

Siguiendo posicionados en el proyecto Web, a continuación procederemos a crear el servicio de dominio, por lo que en esta ocasión, desde la categoría Web, añadiremos al proyecto un Domain Service Class, al que daremos el nombre NorthwindDomainService.cs (ver figura 6).

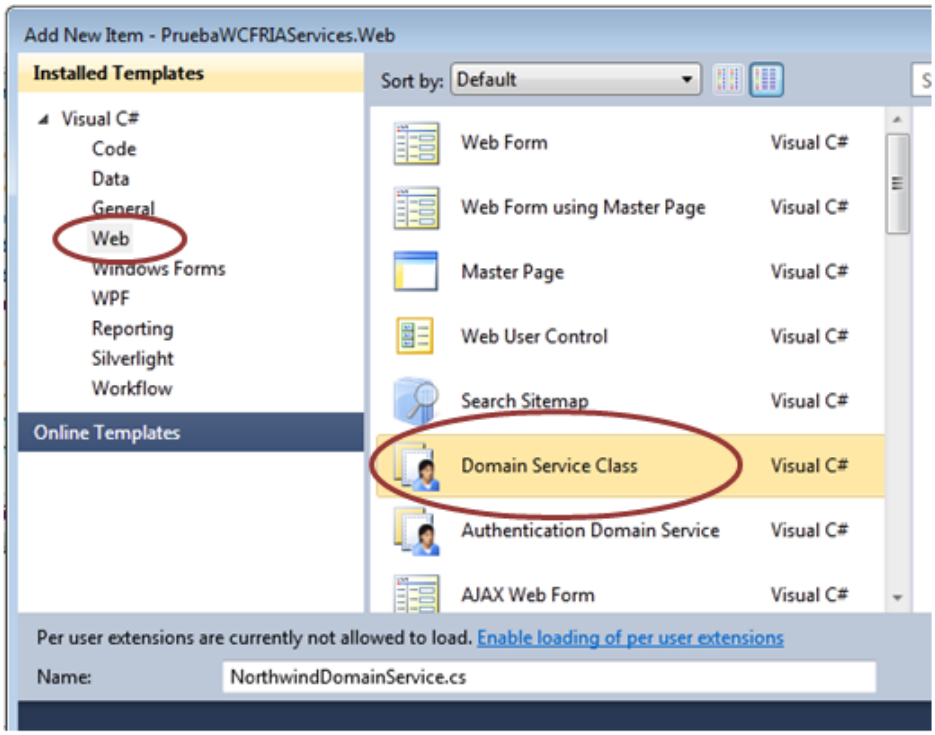


Figura 6. Creación del servicio de dominio.

Justo en este momento se abrirá el cuadro de diálogo Add New Domain Service Class (ver figura 7), que utilizaremos para establecer diversos aspectos de la generación de código, entre los que destacamos los siguientes:

--Enable client access. Cuando esta casilla está marcada, WCF RIA Services también generará, en el proyecto Silverlight (capa cliente), las clases que representan a las entidades del modelo de datos.

--Available DataContexts/ObjectContexts. Desde esta lista desplegable elegiremos uno de los modelos de datos que hayamos creado en el proyecto Web de la solución. Actualmente disponemos de un único modelo: NorthwindEntities.

--Entities. Esta columna muestra los nombres de las entidades pertenecientes al modelo de datos seleccionado.

--Enable editing. Esta columna permite establecer qué entidades del modelo (clases de entidad) dispondrán de la capacidad de realizar operaciones de edición. Con el fin de mantener el ejemplo lo más sencillo posible, no permitiremos la edición de la entidad Customers, quedando la misma limitada a operaciones de consulta.

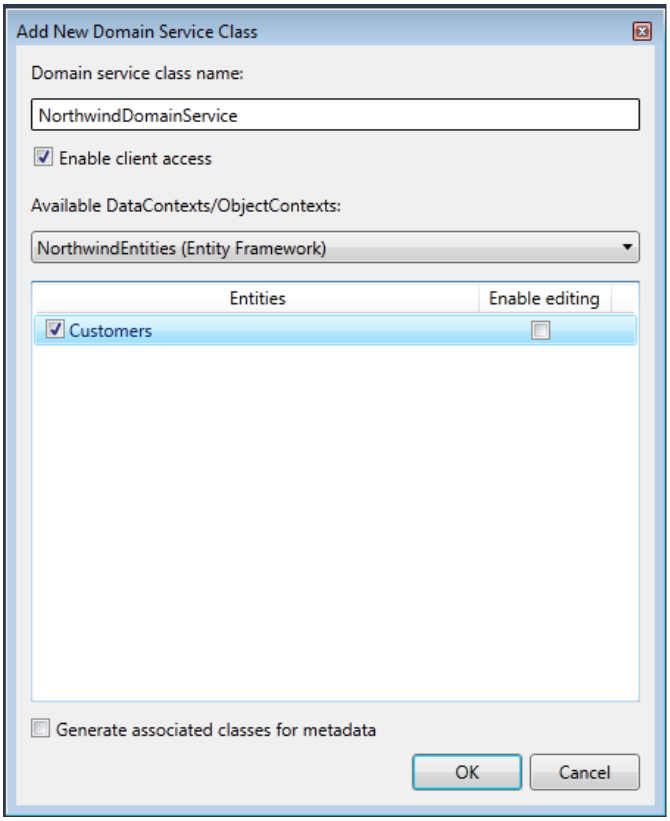


Figura 7. Configuración del servicio de dominio.

Al aceptar este cuadro de diálogo se generará el código del listado 1, donde vemos la clase NorthwindDomainService, que representa al servicio de dominio. Esta clase hereda de LinqToEntitiesDomainService<NorthwindEntities>.

```
[EnableClientAccess()]
public class NorthwindDomainService : LinqToEntitiesDomainService<NorthwindEntities>
{
   public IQueryable<Customers> GetCustomers()
   {
      return this.ObjectContext.Customers;
   }
}
```

Listado 1.

Esta clase será la que contenga todos los métodos de operación (consulta, inserción, borrado y modificación) contra las entidades del modelo de datos. Para identificar el tipo de operación que realiza cada método, se utiliza una convención basada en añadir un prefijo a su nombre; en el caso de las consultas, dicho prefijo es Get.

Al crear el servicio de dominio hemos indicado que el uso que daremos a la entidad Customers será de sólo lectura, por lo tanto, la clase NorthwindDomainService será generada con un único método, GetCustomers, que devolverá una colección de entidades Customers, las cuales representan los registros de la tabla del mismo nombre existente en la base de datos Northwind.

Podemos ampliar la clase NorthwindDomainService, bien modificando el método GetCustomers existente, o bien añadiendo nuevos métodos, como GetCustomersTipoContacto, que vemos en el listado 2, donde gracias al uso de LINQ, se devuelven las entidades de la tabla Customers que cumplan una determinada condición, pasada como parámetro a dicho método.

```
//....
public IQueryable<Customers> GetCustomersTipoContacto(string sTipoContacto)
{
   return this.ObjectContext.Customers.Where(oCustomer => oCustomer.ContactTitle == sTipoContacto);
}
```

Listado 2.

Clases cliente intermediarias

En este momento compilaremos la solución, y seguidamente, en la ventana Solution Explorer, pulsaremos el botón Show All Files para el proyecto Silverlight, observando que la carpeta Generated_Code contiene un archivo llamado PruebaWCFRIAServices.Web.g.cs. Este archivo contiene un conjunto de clases denominado “clases cliente intermediarias” (client proxy classes), entre las que destacamos las siguientes:

--Customers. Se trata de la versión, para la capa cliente de la aplicación, de la clase que representa la entidad del mismo nombre generada por el diseñador del modelo de datos. Para cada entidad existente en el modelo se generará una clase de estas características en el cliente.

--NorthwindDomainContext. Clase que representa al contexto de dominio (hereda de DomainContext), ofreciendo en el lado cliente las operaciones existentes en el lado servidor contenidas en el servicio de dominio.

Todo este código es obtenido gracias a que en el momento de crear nuestra solución hemos marcado la casilla Enable .NET RIA Services (en el diálogo inicial de creación de la aplicación), y a que la clase NorthwindDomainService está calificada con el atributo EnableClientAccess. El listado 3 muestra los métodos del contexto de dominio encargados de obtener las correspondientes colecciones de entidades Customers. Observe el lector que al ser generados estos métodos, su nombre está basado en el existente dentro del servicio de dominio, añadiéndose el sufijo Query.

```
public sealed partial class NorthwindDomainContext : DomainContext
{
   //....
   public EntityQuery<Customers> GetCustomersQuery()
   {
      this.ValidateMethod("GetCustomersQuery", null);
      return base.CreateQuery<Customers>("GetCustomers", null, false, true);
   }

   public EntityQuery<Customers> GetCustomersTipoContactoQuery(string sTipoContacto)
```

```
    {
      Dictionary<string, object> parameters = new Dictionary<string, object>();
      parameters.Add("sTipoContacto", sTipoContacto);
      this.ValidateMethod("GetCustomersTipoContactoQuery", parameters);
      return base.CreateQuery<Customers>("GetCustomersTipoContacto", parameters, false, true);
    }
    //....
  }
```

Listado 3.

Resulta muy importante tener en cuenta, que todo este código creado para la capa cliente por WCF RIA Services, será generado de nuevo en cuanto realicemos cualquier modificación, bien en el modelo de datos o en el servicio de dominio de la capa intermedia de nuestra aplicación. No es conveniente, por lo tanto, que realicemos cambios manuales en este código, ya que los perderemos al volver a compilar la solución.

Visualizando las entidades en el lado cliente

Para presentar la información de algunas columnas de la tabla Customers en nuestra interfaz de usuario Silverlight utilizaremos un control DataGrid, que llenaremos durante la carga de la página XAML. El listado 4 muestra el código de marcado con la definición de la interfaz.

```
<UserControl
xmlns:my="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
  x:Class="PruebaWCFRIAServices.MainPage"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  mc:Ignorable="d" d:DesignHeight="500" d:DesignWidth="600">

  <Grid x:Name="LayoutRoot" Background="White">
    <StackPanel Background="LightSkyBlue">
      <my:DataGrid x:Name="grdDatos" Margin="5" Height="400" Width="580"
AutoGenerateColumns="False" IsReadOnly="True">
        <my:DataGrid.Columns>
          <my:DataGridTextColumn Header="Código" Binding="{Binding CustomerID}" />
          <my:DataGridTextColumn Header="Compañía" Binding="{Binding CompanyName}" />
          <my:DataGridTextColumn Header="Persona contacto" Binding="{Binding ContactName}" />
          <my:DataGridTextColumn Header="Categoría" Binding="{Binding ContactTitle}" />
        </my:DataGrid.Columns>
      </my:DataGrid>
    </StackPanel>
  </Grid>
</UserControl>
```

Listado 4.

Pasando al code-behind de esta página (ver listado 5), para acceder a la colección de entidades Customers, en primer lugar declararemos el espacio de nombres correspondiente al proyecto Web de la solución, y seguidamente instanciaremos, con ámbito de clase, un tipo NorthwindDomainContext, que nos permitirá invocar desde el

lado cliente, a los métodos del servicio de dominio encargados de realizar las diversas operaciones de la lógica de negocio.

A continuación, dentro del evento Loaded de la página Silverlight, asignaremos a la propiedad ItemsSource del control DataGrid la colección de entidades a visualizar utilizando la propiedad Customers del contexto de dominio. Para cargar de datos la mencionada colección, también debemos llamar al método Load del contexto de dominio, que espera como parámetro un tipo EntityQuery<Entity>, el cual obtendremos llamando al método GetCustomersQuery del tipo NorthwindDomainContext.

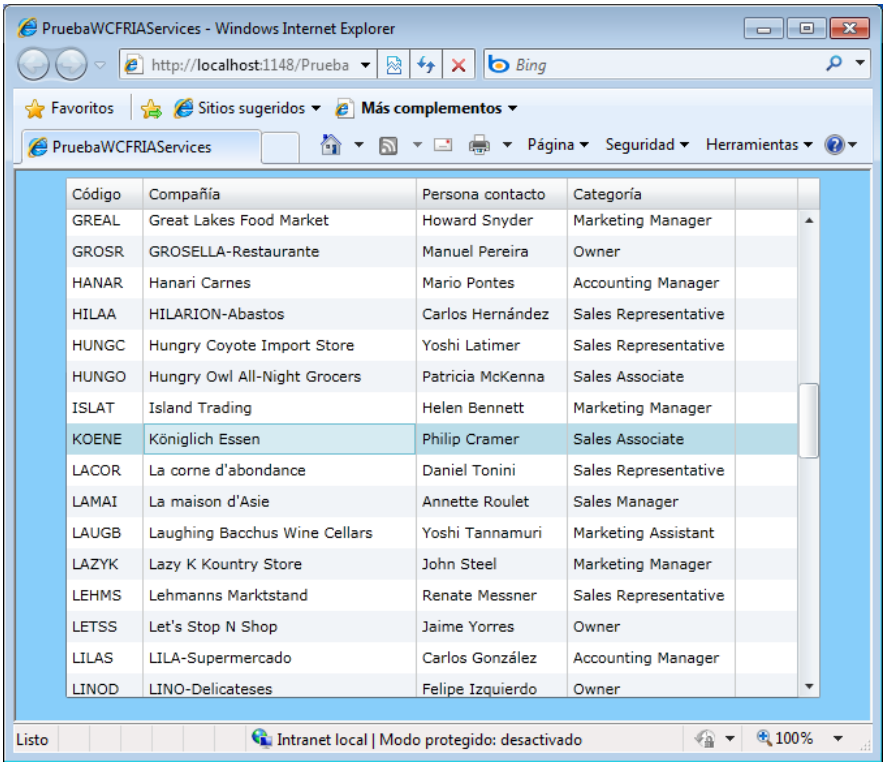
```
using PruebaWCFRIAServices.Web;

namespace PruebaWCFRIAServices
{
   public partial class MainPage : UserControl
   {
      NorthwindDomainContext oNorthwindDomainContext = new NorthwindDomainContext();

      //....
      void MainPage_Loaded(object sender, RoutedEventArgs e)
      {
         grdDatos.ItemsSource = oNorthwindDomainContext.Customers;
oNorthwindDomainContext.Load(oNorthwindDomainContext.GetCustomersQuery());
      }
      //....
```

Listado 5.

El método NorthwindDomainContext.Load se ejecuta de forma asíncrona, por lo que una vez que haya finalizado, la propiedad Customers estará poblada de valores, los cuales serán mostrados en el DataGrid (ver figura 8).



Código	Compañía	Persona contacto	Categoría
GREAL	Great Lakes Food Market	Howard Snyder	Marketing Manager
GROSR	GROSELLA-Restaurante	Manuel Pereira	Owner
HANAR	Hanari Carnes	Mario Pontes	Accounting Manager
HILAA	HILARION-Abastos	Carlos Hernández	Sales Representative
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	Sales Representative
HUNGO	Hungry Owl All-Night Grocers	Patricia McKenna	Sales Associate
ISLAT	Island Trading	Helen Bennett	Marketing Manager
KOENE	Königlich Essen	Philip Cramer	Sales Associate
LACOR	La corne d'abondance	Daniel Tonini	Sales Representative
LAMAI	La maison d'Asie	Annette Roulet	Sales Manager
LAUGB	Laughing Bacchus Wine Cellars	Yoshi Tannamuri	Marketing Assistant
LAZYK	Lazy K Kountry Store	John Steel	Marketing Manager
LEHMS	Lehmanns Marktstand	Renate Messner	Sales Representative
LETSS	Let's Stop N Shop	Jaime Yorres	Owner
LILAS	LILA-Supermercado	Carlos González	Accounting Manager
LINOD	LINO-Delicateses	Felipe Izquierdo	Owner

Figura 8. DataGrid mostrando las entidades cargadas.

Supongamos ahora que queremos filtrar la colección de entidades en base al campo ContactTitle, utilizando el método GetCustomersTipoContacto que a tal efecto existe en el servicio de dominio.

En primer lugar modificaremos la interfaz de usuario (ver listado 6), añadiendo algunos controles adicionales que permitan introducir el valor para realizar el filtro.

```
<StackPanel Orientation="Horizontal" Margin="5" HorizontalAlignment="Center">
  <TextBlock VerticalAlignment="Center">Tipo de contacto:</TextBlock>
  <TextBox Name="txtTipoContacto" Width="200" Height="22" Margin="10,0,0,0" />
  <Button Name="btnCargarTipoContacto" Content="Cargar por tipo contacto" Width="200"
Margin="5" Click=”btnCargarTipoContacto_Click” />
</StackPanel>
```

Listado 6.

A continuación, en el code-behind, escribiremos el código para cargar los datos desde el evento Click del control Button, pero empleando una técnica ligeramente distinta a la explicada anteriormente, ya que ahora recuperaremos, al ejecutar el método NorthwindDomainContext.Load, el objeto LoadOperation<Entidad> que devuelve como resultado, y que como su nombre indica, representa una operación de carga de un conjunto de entidades (Customers en nuestro caso) que obtendremos de la propiedad LoadOperation.Entities.

Para poder utilizar más cómodamente LoadOperation<Entidad>, declararemos el espacio de nombres System.Windows.Ria en nuestro código. Cuando el usuario haya escrito algún valor en el TextBox de la página, pasaremos como parámetro a NorthwindDomainContext.Load la llamada a GetCustomersTipoContactoQuery, y en caso contrario llamaremos a GetCustomersQuery (ver el listado 7).

```
using System.Windows.Ria;

//....

void btnCargarTipoContacto_Click(object sender, RoutedEventArgs e)

{

   LoadOperation<Customers> oLoadOpCust;

   if (this.txtTipoContacto.Text != string.Empty)

   {

      oLoadOpCust =
oNorthwindDomainContext.Load(oNorthwindDomainContext.GetCustomersTipoContactoQuery(this.txt
TipoContacto.Text));

   }

   else

   {

      oLoadOpCust =
oNorthwindDomainContext.Load(oNorthwindDomainContext.GetCustomersQuery());

   }

   this.grdDatos.ItemsSource = oLoadOpCust.Entities;

}
```

Listado 7.

En la figura 9 vemos la aplicación en ejecución tras filtrar los datos a visualizar.

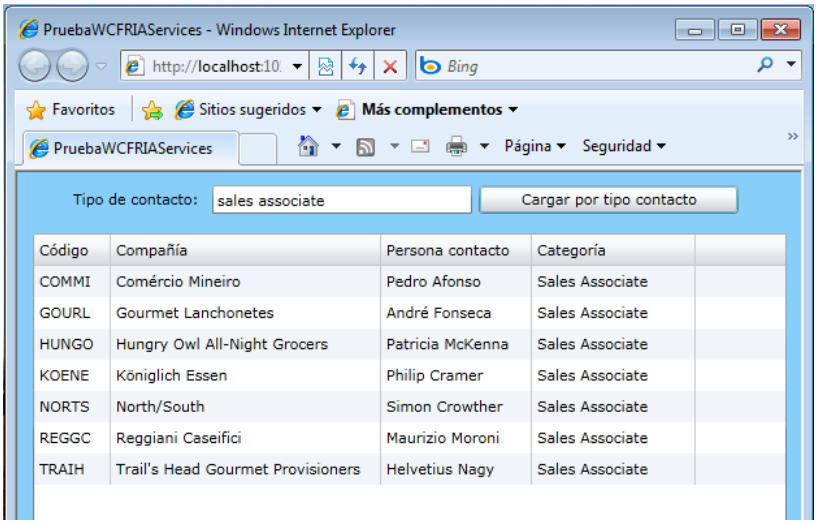


Figura 9. Datos obtenidos al ejecutar el filtrado.

Conclusión

WCF RIA Services supone un importante avance, como producto orientado a simplificar y acelerar los tiempos de desarrollo de las aplicaciones RIA que necesiten interactuar con orígenes de datos. La introducción realizada en este artículo representa una pequeña muestra de la potencia de esta herramienta, que con toda seguridad ganará un buen número de adeptos entre la comunidad de desarrolladores.

REFERENCIAS

[1] Wikipedia: Rich Internet Application.

http://en.wikipedia.org/wiki/Rich_Internet_application

[2] .NET RIA Services: From Vision to Architecture (Nikhil Kothari).

http://www.nikhilk.net/NET-RIA-Services-Vision-Architecture.aspx

[3] Welcome to WCF RIA Services Beta (Brad Adams)

http://blogs.msdn.com/brada/archive/2009/11/18/welcome-to-wcf-ria-services-beta.aspx

[4] Por fin un poco de cordura (Ibon Landa)

http://geeks.ms/blogs/ilanda/archive/2009/11/20/por-fin-un-poco-de-cordura.aspx

[5] Silverlight.net. WCF RIA Services get started.

http://silverlight.net/getstarted/riaservices/

[6] Silverlight.net. Silverlight 4 get started.

http://silverlight.net/getstarted/silverlight-4-beta/#tools