Edición de entidades con el control DataForm de Silverlight

Luis Miguel Blanco Ancos

El control DataForm de Silverlight es una herramienta mediante la que podemos crear formularios que nos permiten llevar a cabo la edición de la información contenida en nuestras fuentes de datos; desde objetos sencillos (Plain Old CLR Objects o POCO) hasta colecciones de entidades obtenidas a través de WCF RIA Services. En el presente artículo abordaremos éste y otros aspectos de interés relacionados con el uso de dicho control.

La aplicación de ejemplo

Entre los requisitos del ejemplo que usaremos en este artículo para ilustrar las funcionalidades y características del DataForm, además de Visual Studio 2010 como entorno de desarrollo, necesitaremos tener instalados en nuestro sistema Silverlight 4 Tools for Visual Studio 2010 [1], que incorpora diversas herramientas para el desarrollo de aplicaciones en Silverlight, tales como WCF RIA Services (que utilizaremos como tecnología de acceso a datos en este ejemplo); Silverlight 4 Toolkit [2], que instala un paquete de controles entre los que se encuentra el DataForm; y la base de datos para pruebas Chinook, disponible en Codeplex [3].

Una vez instaladas todas las herramientas necesarias, comenzaremos creando un nuevo proyecto en Visual Studio 2010 de tipo Silverlight Application, al que daremos el nombre EdicionDataForm (el código fuente está disponible en http://www.dotnetmania.com), y que configuraremos con soporte para WCF RIA Services en el cuadro de diálogo inicial New Silverlight Application.

Seguidamente, en el proyecto Web de la solución añadiremos los siguientes elementos: un modelo de datos (ADO .NET Entity Data Model) que contenga una entidad basada en la tabla Invoice de la base de datos Chinook, y un servicio de dominio (Domain Service Class) con el nombre ChinookDomainService, compuesto por la entidad Invoice, que configuraremos para habilitar su edición y generar clases de metadatos asociadas (figura 1). Recomendamos la consulta de [4] para un mayor detalle acerca de la creación del modelo de datos, servicio de dominio y contexto de dominio en una aplicación Silverlight basada en WCF RIA Services.

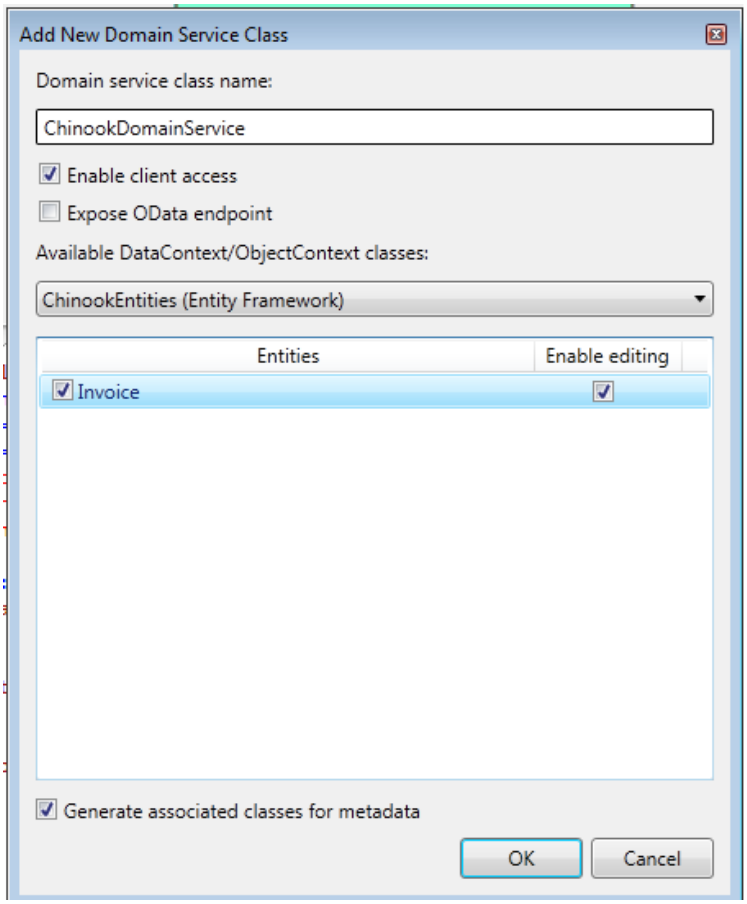

Figura 1. Configuración inicial del servicio de dominio.

El control DomainDataSource

A continuación nos situaremos en el diseñador de la página XAML e insertaremos en su superficie un control DomainDataSource, que tomaremos del Cuadro de herramientas. Este control representa una fuente de datos cuya misión consiste en comunicar con el contexto y servicio de dominio creados en la aplicación. Una vez establecida dicha comunicación, el control creará internamente una colección, que rellenará con el resultado de la llamada a un método del servicio de dominio. Dispone, además, de la lógica necesaria para actualizar los valores que manipula sobre la fuente original de datos, así como la capacidad de filtrar, ordenar y agrupar dichos datos.

La forma de configurar el DomainDataSource consiste en asignar, a su propiedad QueryName, el nombre de un método del servicio de dominio que devuelva la colección de entidades que posteriormente editaremos desde el DataForm. Por otro lado, a la propiedad DomainContext le asignaremos el nombre del contexto de dominio (ChinookDomainContext) que es creado automáticamente por el servicio de dominio en el proyecto cliente de la solución.

El listado 1 muestra el código XAML necesario para crear este control en nuestra página. Observe el lector que para poder acceder al contexto de dominio, en la etiqueta UserControl declaramos el atributo xmlns:domainctx, que apunta al espacio de nombres en el que está situado el mencionado contexto.

```
<UserControl x:Class="EdicionDataForm.MainPage"
   xmlns:my="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.DomainServices"
   xmlns:domainctx="clr-namespace:EdicionDataForm.Web"
   <!--....-->

   <my:DomainDataSource x:Name="ddsInvoices" QueryName="GetInvoices">
      <my:DomainDataSource.DomainContext>
         <domainctx:ChinookDomainContext />
      </my:DomainDataSource.DomainContext>
   </my:DomainDataSource>
   <!--....-->
```

Listado 1.

El control DataForm

Nuestro siguiente paso consistirá en añadir a la página un control DataForm, que tomaremos igualmente del Cuadro de herramientas. Para que este control muestre la colección de entidades del DomainDataSource, deberemos asignar a la propiedad ItemsSource una expresión de enlace a datos (Data Binding), como vemos en el listado 2. En dicha expresión, el atributo ElementName se utiliza para asignar el objeto DomainDataSource que actúa como fuente de datos, mientras que el atributo Path sirve para establecer la propiedad Data del DomainDataSource, que contiene la colección de datos a editar.

```
<UserControl x:Class="EdicionDataForm.MainPage"
   <!--....-->
   xmlns:my1="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.DataForm.Toolkit"
   <!--....-->
   <StackPanel Background="Aquamarine">
      <my1:DataForm x:Name="frmDatos" Margin="10"
         ItemsSource="{Binding ElementName=ddsInvoices, Path=Data}" />
   </StackPanel>
   <!--....-->
```

Listado 2.

Si ejecutamos en este momento la solución obtendremos como resultado el DataForm en la ventana de nuestro navegador Web. Como por arte de magia, ya que nosotros no hemos especificado nada al respecto en el código XAML, se habrán generado automáticamente los controles pertenecientes a los campos del formulario; aunque lo cierto es que la causa reside en la propiedad AutoGenerateFields del DataForm, que por defecto tiene el valor True, produciendo este comportamiento.

Cada control generado para el formulario se adaptará en lo posible al tipo de dato que le toca editar. De esta manera, para valores de texto o numéricos se utilizará un TextBox; para fechas un DatePicker; para valores lógicos un CheckBox, etc. (figura 2).

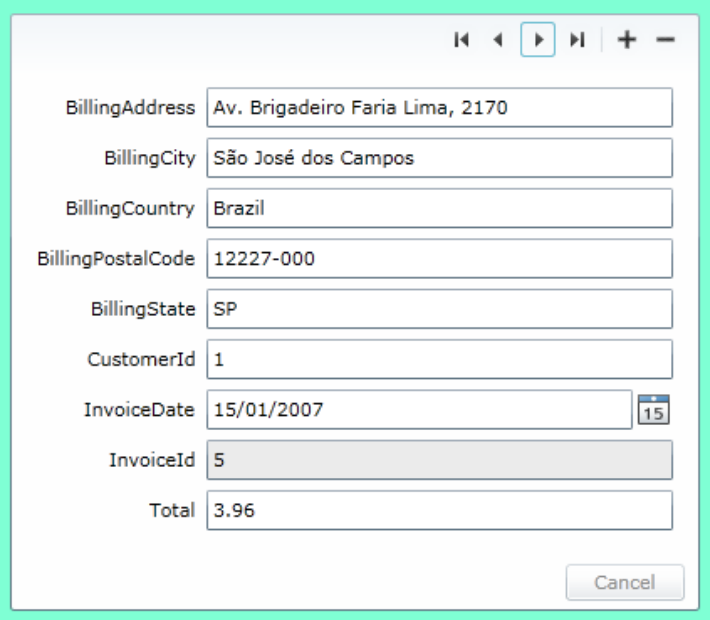

Figura 2. DataForm con presentación de contenido predeterminada.

Personalizando la presentación de los campos

Al ejecutar por primera vez el DataForm, nos percataremos de que los campos del formulario se muestran según el orden alfabético de las propiedades correspondientes al tipo de los objetos contenidos en la colección (clase Invoice). Por otra parte, las etiquetas o títulos de los campos corresponden a los nombres de las propiedades de dicha clase, lo cual es posible que no se ajuste a nuestros requerimientos de diseño.

Podemos personalizar este comportamiento mediante los diferentes tipos de plantilla que suministra el DataForm, sin embargo, en esta ocasión dejaremos que los campos se generen automáticamente, adaptando el orden y los títulos mediante el atributo Display, perteneciente al espacio de nombres DataAnnotations, que aplicaremos de la siguiente manera:

Dentro del proyecto Web de la solución editaremos la clase InvoiceMetadata, situada en el archivo ChinookDomainService.metadata.cs. Esta clase contiene los metadatos de la clase entidad Invoice, y será aquí donde deberemos añadir el código necesario en el caso de que necesitemos personalizar algún aspecto de la misma, para que sea propagado al contexto de dominio, y de ahí a la interfaz de usuario.

En este caso, calificaremos cada una de las propiedades de InvoiceMetadata con el atributo Display, al que pasaremos el parámetro Order, consistente en un número que establece el orden en el que la propiedad será mostrada en el DataForm; y Name, una

cadena para establecer el título de la etiqueta asociada al campo del formulario mediante el que editamos la propiedad. También podemos utilizar en alguna de las propiedades el parámetro Description, que es un texto que generará, cuando el DataForm esté en modo de edición, un icono al lado del campo, de forma que al situar encima el cursor, visualizará dicho texto dentro de una etiqueta flotante. El listado 3 muestra el uso de estos atributos.

```
[MetadataTypeAttribute(typeof(Invoice.InvoiceMetadata))]
public partial class Invoice
{
  //....
  internal sealed class InvoiceMetadata
  {
    private InvoiceMetadata()
    {
    }

    [Display(Order = 4, Name = "Dirección de factura:", Description = "Dirección de entrega del pedido")]
    public string BillingAddress { get; set; }

    [Display(Order = 5, Name = "Ciudad de factura:")]
    public string BillingCity { get; set; }

    [Display(Order = 7, Name = "País de factura", Description = "Nación de residencia")]
    public string BillingCountry { get; set; }

    [Display(Order = 8, Name = "Código postal de factura")]
    public string BillingPostalCode { get; set; }

    [Display(Order = 6, Name = "Provincia de factura:")]
    public string BillingState { get; set; }

    [Display(Order = 2, Name = "Cliente:")]
    public int CustomerId { get; set; }

    [Display(Order = 3, Name = "Fecha de factura:")]
    public DateTime InvoiceDate { get; set; }

    [Display(Order = 1, Name = "Factura:")]
    public int InvoiceId { get; set; }

    [Display(Order = 9, Name = "Importe:")]
    public decimal Total { get; set; }
  }
}
```

Listado 3.

Para completar estos retoques de presentación del formulario volveremos al código XAML, agregando al DataForm las siguientes propiedades (listado 4):

- Header. Cadena con un título para el formulario.
- BorderThickness. Valor numérico que indica el grosor del borde del formulario.

- LabelPosition. Valor de la enumeración DataFieldLabelPosition, que permite establecer la posición de la etiqueta asociada a un campo del formulario.
- DescriptionViewerPosition. Valor de la enumeración DataFieldDescriptionViewerPosition, que permite establecer la posición del icono de descripción asociado a un campo del formulario.

```
<my1:DataForm x:Name="frmDatos"
   ItemsSource="{Binding ElementName=ddsInvoices, Path=Data}"
   Margin="10" Header="Edición de facturas" BorderThickness="5"
   LabelPosition="Top" DescriptionViewerPosition="BesideLabel"  />
```

Listado 4.

Volviendo a ejecutar la solución, el DataForm mostrará un aspecto ligeramente distinto del original (figura 5).

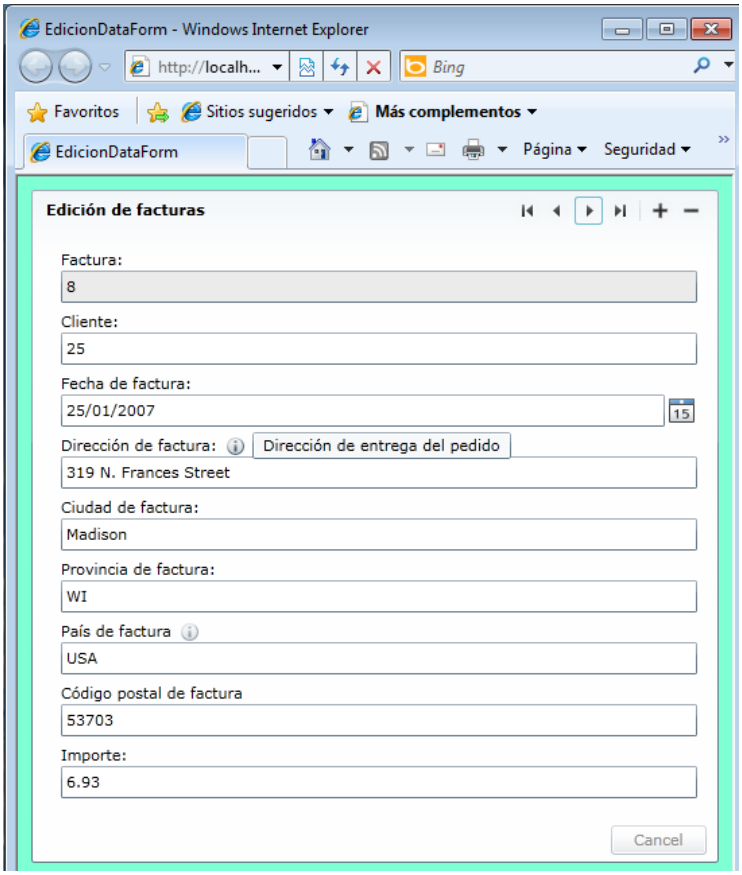

Figura 5. DataForm con campos reorganizados.

La barra de herramientas. Navegación y edición de datos

Uno de los elementos destacables del DataForm lo constituye la barra de herramientas. Situada en su zona superior, ofrece al usuario un conjunto de controles para realizar la navegación y edición de la colección de entidades enlazada al formulario.

En lo que respecta a los controles de edición, podemos observar que de forma predeterminada, el DataForm permite editar directamente los campos del formulario (su propiedad DataFormMode tiene el valor Edit), de manera que en cuanto es detectado el más mínimo cambio en cualquiera de ellos, se habilita el botón Cancel para permitir deshacer las modificaciones realizadas a la entidad actualmente en edición. En el caso de hacer clic en cualquiera de los botones de navegación se confirmarán los cambios en el objeto, y como ayuda visual, cada vez que volvamos a pasar por un objeto modificado, en la parte superior del DataForm se mostrará un asterisco indicando dicha situación.

Esta facilidad de edición puede no ser conveniente en todos los escenarios, ya que podemos encontrarnos con situaciones en las que sea preferible ofrecer el formulario en un modo inicial de solo lectura, y que sea el usuario quien active la capacidad de editar la entidad actualmente en curso.

Asignando a la propiedad AutoEdit del DataForm el valor False, los campos del formulario se mostrarán inicialmente sin capacidad de edición, debiendo el usuario hacer clic en el botón con forma de lapicero que aparecerá en la barra de herramientas, para cambiar el formulario a modo de edición.

Si queremos ser todavía más restrictivos, también podemos asignar el valor False a AutoCommit, otra de las propiedades del DataForm. El efecto que esto tendrá cuando cambiemos a modo de edición será la aparición, en la parte inferior del formulario, del botón OK (inicialmente deshabilitado) junto al ya conocido Cancel. Con el formulario configurado de esta manera, en cuanto realicemos algún cambio en los campos se habilitará el botón OK y se deshabilitará toda la barra de herramientas, de modo que solamente tendremos la opción de aceptar o cancelar las modificaciones.

El listado 5 muestra el código para crear el DataForm con las anteriores propiedades, incluyendo además CommitButtonContent y CancelButtonContent, propiedades mediante las cuales podemos asignar un texto alternativo a estos botones (figura 6).

```
<my1:DataForm x:Name="frmDatos"
   ItemsSource="{Binding ElementName=ddsInvoices, Path=Data}"
   Margin="10" Header="Edición de facturas" BorderThickness="5"
   LabelPosition="Top" DescriptionViewerPosition="BesideLabel"
   CommitButtonContent="Aceptar" CancelButtonContent="Deshacer cambios"
   AutoEdit="False" AutoCommit="False" />
```

Listado 5.

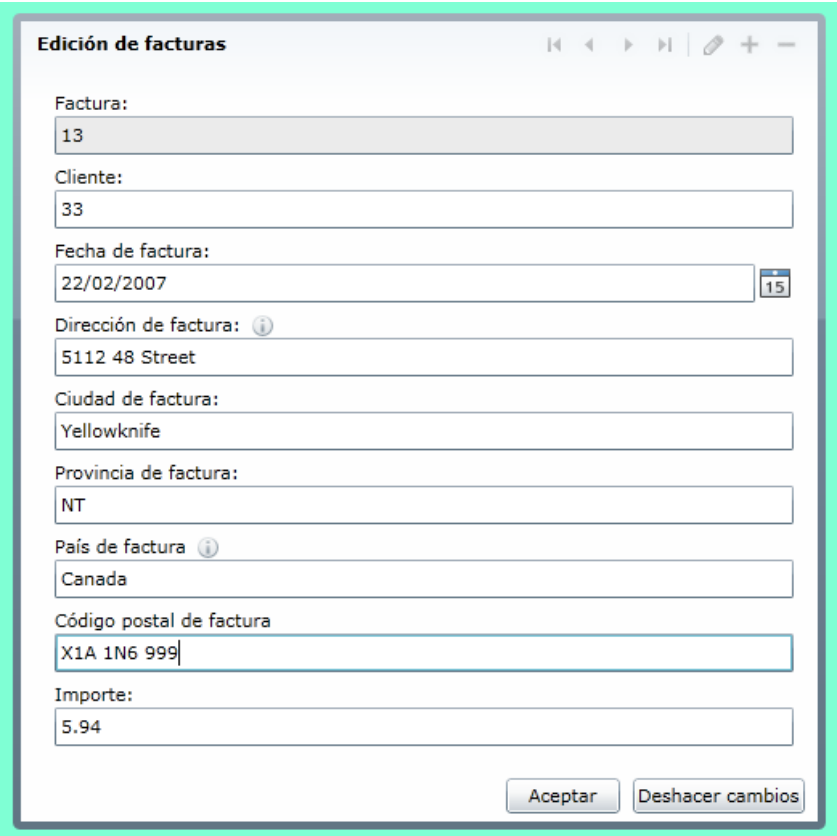

Figura 6. DataForm con edición automática deshabilitada.

Actualizar los cambios en la fuente de datos origen

Las operaciones de creación, modificación y borrado que realicemos a través del DataForm, no se actualizan inmediatamente en la base de datos original de forma predeterminada, sino que debe ser el programador quien proporcione explícitamente los medios para llevar este paso a cabo. Para ello debemos ejecutar el método SubmitChanges, perteneciente al DomainDataSource enlazado al DataForm, y que contiene los datos que estamos editando a través del formulario. Dicho método será el encargado de actualizar sobre la base de datos los cambios realizados por el usuario desde el DataForm. El listado 6 muestra la creación de un nuevo botón en la página XAML, y el código de su evento Click, en el que se realiza esta llamada.

```
<Button x:Name="btnActualizarFuenteDatos"
   Content="Actualizar fuente de datos" Width="200"
   Click="btnActualizarFuenteDatos_Click" />
//--------
private void btnActualizarFuenteDatos_Click(object sender, RoutedEventArgs e)
{
   this.ddsInvoices.SubmitChanges();
}
```

Listado 6.

Conclusión

El empleo del control DataForm puede suponer al desarrollador un importante ahorro de tiempo en la creación de sus formularios de edición de datos para Silverlight, ya que incorpora de serie una maquinaria con todas aquellas funcionalidades esenciales, que de otro modo tendríamos que implementar manualmente al desarrollar nuestros procesos de mantenimiento de datos. Confiamos en que este artículo haya despertado la curiosidad del lector, y le animamos a exprimir todo el potencial de este interesante control.

REFERENCIAS

[1] Silverlight 4 Tools for Visual Studio 2010

http://www.microsoft.com/downloads/details.aspx?FamilyID=eff8a0da-0a4d-48e8-8366-6ddf2ecad801&displaylang=en

[2] Silverlight 4 Toolkit

http://silverlight.codeplex.com/releases/view/43528

[3] Base de datos Chinook

http://chinookdatabase.codeplex.com/

[4] Blanco, Luis Miguel. “Acceso a datos en Silverlight con WCF RIA Services”. dotNetManía nº 68, marzo 2010