Manipulación de datos en Silverlight mediante ADO.NET Data Services (y III)

Luis Miguel Blanco Ancos

En las anteriores entregas de este artículo se han tratado aquellos aspectos relacionados con la emisión de peticiones contra el servicio de datos y la posterior presentación de resultados en la interfaz de usuario. En esta tercera parte, que concluye la serie, abordaremos el otro gran aspecto en las operaciones de mantenimiento de datos con este tipo de servicios: la edición.

Edición de datos

Hasta el momento, nuestra labor se ha circunscrito a efectuar conexiones con la fuente de datos, recuperar su contenido y mostrarlo al usuario; pero en los próximos apartados del presente artículo, abordaremos aquellas operaciones encaminadas a realizar la modificación, inserción y borrado de datos, que junto a las consultas, constituyen los pilares fundamentales en cualquier proceso de tratamiento de datos.

Al igual que en anteriores ocasiones, agregaremos a la solución que estamos desarrollando como base de los ejemplos, un nuevo proyecto Silverlight con el nombre ServicioDatosConsultaSL5, compuesto por un DataGrid que emplearemos para realizar las tareas de edición, y un control Button para confirmar en la base de datos los cambios realizados desde la interfaz de usuario. El listado 1 muestra el código de marcado perteneciente a esta página.

```
<UserControl
  xmlns:data="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
  x:Class="ServicioDatosConsultaSL5.Page"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Width="610" Height="370">
  <Grid x:Name="LayoutRoot" Background="White">
    <StackPanel Background="LightGreen">
      <TextBlock HorizontalAlignment="Center">Edición de la tabla Suppliers</TextBlock>

      <data:DataGrid x:Name="grdDatos"
        ItemsSource="{Binding}" AutoGenerateColumns="False" Margin="5"
        HeadersVisibility="All" SelectionMode="Single" IsReadOnly="False"
        Height="300" Width="600" Background="Turquoise">
        <data:DataGrid.Columns>
          <data:DataGridTextColumn Header="Nombre" Binding="{Binding CompanyName}" />
          <data:DataGridTextColumn Header="Contacto" Binding="{Binding ContactName}" />
          <data:DataGridTextColumn Header="Ciudad" Binding="{Binding City}" />
          <data:DataGridTextColumn Header="Pais" Binding="{Binding Country}" />
```

```
            </data:DataGrid.Columns>
        </data:DataGrid>
        <Button x:Name="btnGrabar" Height="20" Width="100" Margin="10" Content="Grabar" />
    </StackPanel>
  </Grid>
</UserControl>
```

Listado 1. Código de marcado para la página de edición de datos.

A continuación entraremos en el code behind de la página para proceder a implementar las operaciones de edición. En primer lugar crearemos un tipo enumerado, que nos ayudará a saber la operación que estamos realizando en un momento dado.

El listado 2 muestra, además de la mencionada enumeración, la declaración de un conjunto de variables con ámbito a nivel de clase, que emplearemos como soporte de las diversas tareas a desarrollar. También vemos en el constructor de la página, la declaración de los diferentes manipuladores de evento que utilizaremos.

```
enum TipoOperacion
{
   Lectura = 1,
   Modificar = 2,
   Insertar = 3
}

public partial class Page : UserControl
{
   NorthwindEntities dtsvcNorthwind;
   TipoOperacion xTipoOperacion;
   ObservableCollection<Suppliers> cllSuppliers;
   int? nIndiceFilaEnEdicion = null;
   Suppliers oSupplierOriginal;

   public Page()
   {
      InitializeComponent();
      this.Loaded += new RoutedEventHandler(Page_Loaded);
      this.grdDatos.BeginningEdit += new
EventHandler<DataGridBeginningEditEventArgs>(grdDatos_BeginningEdit);
      this.grdDatos.CurrentCellChanged += new
EventHandler<EventArgs>(grdDatos_CurrentCellChanged);
      this.grdDatos.SelectionChanged += new
SelectionChangedEventHandler(grdDatos_SelectionChanged);
      this.grdDatos.KeyDown += new KeyEventHandler(grdDatos_KeyDown);
      this.grdDatos.KeyUp += new KeyEventHandler(grdDatos_KeyUp);
      this.btnGrabar.Click += new RoutedEventHandler(btnGrabar_Click);
   }
   //....
}
```

Listado 2. Declaraciones iniciales para los procesos de edición de datos.

Dado que ya hemos tratado en las anteriores entregas sobre la forma de conectar con la base de datos mediante el servicio, y obtener el contenido de una tabla para llenar

un DataGrid, evitaremos repetir dicho proceso aquí, entrando directamente en la primera de las operaciones a implementar: la modificación.

Modificación de datos

El comportamiento que pretendemos conseguir en este proceso consiste en cambiar el valor de alguna o todas las celdas pertenecientes a la fila sobre la que estamos posicionados, pulsando el botón Grabar para actualizar los cambios realizados en el origen de datos. En el caso de que queramos cancelar dichas modificaciones, restaurando la fila a sus valores originales, pulsaremos la tecla Escape, o bien, haremos clic directamente en una fila distinta a la que estábamos editando.

Para comenzar con la modificación de una fila haremos doble clic sobre cualquiera de sus celdas, o si ya estamos situados en la fila requerida pulsaremos la tecla F2, lo cual provocará que la celda que tiene el foco entre en modo de edición. Dicha acción desencadenará el evento BeginningEdit en el DataGrid, siendo en el manipulador de dicho evento, donde la variable de tipo TipoOperacion (que inicialmente tiene el valor de lectura) cambie a estado de modificación.

También guardaremos el índice de la fila del DataGrid sobre la que hemos comenzado a editar, a fin de poder detectar posibles cambios de fila durante el proceso de modificación; y ya para finalizar con este evento, guardaremos una copia del objeto Suppliers seleccionado, por si decidimos cancelar las modificaciones efectuadas en las celdas de la fila. Ver el listado 3.

```
void grdDatos_BeginningEdit(object sender, DataGridBeginningEditEventArgs e)
{
   if (xTipoOperacion == TipoOperacion.Lectura)
   {
      xTipoOperacion = TipoOperacion.Modificar;
      nIndiceFilaEnEdicion = this.grdDatos.SelectedIndex;

      oSupplierOriginal = new Suppliers();
      oSupplierOriginal.SupplierID = ((Suppliers)this.grdDatos.SelectedItem).SupplierID;
      oSupplierOriginal.CompanyName = ((Suppliers)this.grdDatos.SelectedItem).CompanyName;
      oSupplierOriginal.ContactName = ((Suppliers)this.grdDatos.SelectedItem).ContactName;
      oSupplierOriginal.City = ((Suppliers)this.grdDatos.SelectedItem).City;
      oSupplierOriginal.Country = ((Suppliers)this.grdDatos.SelectedItem).Country;
   }
}
```

Listado 3. Comienzo del proceso de edición de una celda del DataGrid.

Al cambiar de celda dentro de la misma fila, debemos mantener en modo de edición la celda que recibe el foco. Dado que al realizar esta acción se produce el evento CurrentCellChanged, en su manipulador comprobaremos que la operación de edición

sea correcta y no se haya cambiado de fila dentro del DataGrid, como vemos en el listado 4.

```
void grdDatos_CurrentCellChanged(object sender, EventArgs e)
{
  if (xTipoOperacion == TipoOperacion.Modificar || xTipoOperacion == TipoOperacion.Insertar)
  {
    if (this.grdDatos.SelectedIndex == nIndiceFilaEnEdicion)
    {
      this.grdDatos.BeginEdit();
    }
  }
}
```

Listado 4. Cambio de foco en celda del DataGrid.

Una vez que hayamos terminado de modificar todas aquellas celdas que necesitemos, pulsaremos el botón btnGrabar. En el evento Click de este botón haremos uso de nuestro servicio de datos llamando a su método UpdateObject, de esta forma le indicaremos que debe actualizar el objeto que pasamos como parámetro, el cual corresponde al elemento de la colección visualizada por el DataGrid, que hemos modificado mediante dicho control.

No obstante, la llamada al método UpdateObject no produce por sí misma la actualización del contenido del objeto en la base de datos, siendo necesario ejecutar el método BeginSaveChanges, perteneciente también al servicio, y que como su nombre indica, inicia el proceso asíncrono que actualiza sobre la fuente de datos los cambios realizados desde la interfaz de usuario.

La finalización de este proceso de actualización se produce en el método de devolución de llamada GrabarSuppliersCompletado, llamando al método EndSaveChanges del servicio, al que pasaremos como parámetro el objeto IAsyncResult que recibe el método actual. Por último, volvemos a establecer la variable que contiene el modo de edición a lectura, y asignamos nulo a la variable que indica la fila en edición. Todo ello lo vemos en el listado 5.

```
void btnGrabar_Click(object sender, RoutedEventArgs e)
{
  switch (xTipoOperacion)
  {
    case TipoOperacion.Modificar:
      dtsvcNorthwind.UpdateObject(cllSuppliers[(int)nIndiceFilaEnEdicion]);
      break;
    //....
  }
  dtsvcNorthwind.BeginSaveChanges(GrabarSuppliersCompletado, null);
}

void GrabarSuppliersCompletado(IAsyncResult oResultado)
{
  dtsvcNorthwind.EndSaveChanges(oResultado);
```

```
    xTipoOperacion = TipoOperacion.Lectura;
    nIndiceFilaEnEdicion = null;
}
```

Listado 5. Grabación del registro modificado mediante el servicio.

En la figura 1 podemos ver el DataGrid en proceso de edición.

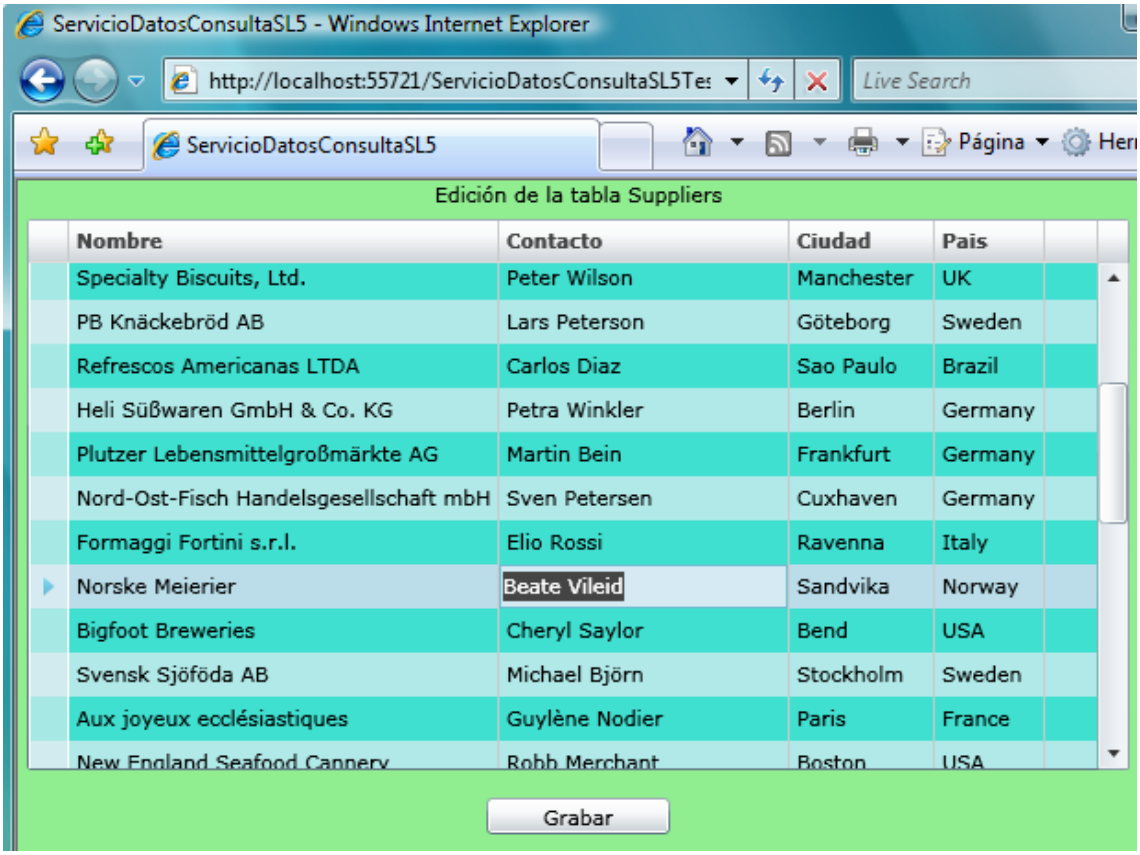

Figura 1. Modificando una fila del DataGrid.

Respecto a la cancelación de un proceso de modificación (como hemos apuntado anteriormente) será implementada de dos maneras: al cambiar de fila sin haber pulsado el botón de grabación, lo que desencadenará el evento SelectionChanged del DataGrid; y al detectar la pulsación de la tecla Escape, situación que manejaremos mediante el evento KeyUp, como vemos en el listado 6.

```
void grdDatos_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if ((nIndiceFilaEnEdicion != null) && (this.grdDatos.SelectedIndex != nIndiceFilaEnEdicion))
    {
        cllSuppliers.RemoveAt((int)nIndiceFilaEnEdicion);

        if (xTipoOperacion == TipoOperacion.Modificar)
        {
            cllSuppliers.Insert((int)nIndiceFilaEnEdicion, oSupplierOriginal);
        }

        nIndiceFilaEnEdicion = null;
```

```
      xTipoOperacion = TipoOperacion.Lectura;
    }
}

void grdDatos_KeyUp(object sender, KeyEventArgs e)
{
   if (e.Key == Key.Escape)
   {
      if (nIndiceFilaEnEdicion != null)
      {
         switch (xTipoOperacion)
         {
            case TipoOperacion.Modificar:
               cllSuppliers.RemoveAt((int)nIndiceFilaEnEdicion);
               xTipoOperacion = TipoOperacion.Lectura;
               cllSuppliers.Insert((int)nIndiceFilaEnEdicion, oSupplierOriginal);
               this.grdDatos.SelectedIndex = (int)nIndiceFilaEnEdicion;
               break;
               //....
         }
         nIndiceFilaEnEdicion = null;
      }
   }
}
```

Listado 6. Cancelación de la operación de modificación de una fila.

Inserción de datos

La próxima operación a desarrollar será la inserción de nuevos registros, proceso que iniciaremos con la pulsación de la tecla Insert, mediante la que añadiremos un nuevo elemento a la colección subyacente de datos del DataGrid. El resto de la operativa será muy similar al proceso de modificación: teclear los valores en las diferentes celdas de la nueva fila, pulsar el botón btnGrabar para agregar la fila en la base de datos, o cancelar la inserción, bien pulsando Escape o cambiando de fila.

Para implementar este comportamiento en el DataGrid crearemos un manipulador para su evento KeyDown, donde al detectar la pulsación de Insert, añadiremos un nuevo objeto Suppliers a la colección de elementos del control, guardaremos la posición de fila en la que vamos a trabajar y estableceremos el tipo de operación en insertar.

Una vez agregada la nueva fila y antes de asignarle el foco, lo deseable sería que la misma fuera visualizada en la cuadrícula de datos; sin embargo esto es algo que no sucede de forma predeterminada; para conseguirlo debemos recurrir al método ScrollIntoView.

ScrollIntoView obliga al DataGrid a visualizar la fila y/o columna pasados como parámetros; siendo preciso en nuestro caso utilizar solamente el primero (la fila) para posicionarnos en el lugar de la cuadrícula que necesitamos. Como podemos observar

en el listado 7, la forma de indicar la fila consiste en pasar como parámetro el elemento de la colección sobre el que queremos posicionarnos.

Finalmente, estableceremos el foco en la fila mediante las propiedades SelectedItem y CurrentColumn, haciendo que entre en modo de edición gracias al método BeginEdit.

```
void grdDatos_KeyDown(object sender, KeyEventArgs e)
{
   if (e.Key == Key.Insert)
   {
      cllSuppliers.Add(new Suppliers());
      nIndiceFilaEnEdicion = cllSuppliers.Count - 1;
      xTipoOperacion = TipoOperacion.Insertar;

      this.grdDatos.ScrollIntoView(cllSuppliers[cllSuppliers.Count - 1], null);

      this.grdDatos.SelectedItem = cllSuppliers[cllSuppliers.Count - 1];
      this.grdDatos.CurrentColumn = this.grdDatos.Columns[0];
      this.grdDatos.BeginEdit();
   }
}
```

Listado 7. Inicio del proceso de inserción.

Para añadir la nueva fila en la base de datos (al igual que en el proceso de modificación) utilizaremos el botón btnGrabar, como vemos en el listado 8, pero en este caso llamaremos a AddToSuppliers, un método del servicio con la misión específica de añadir un nuevo objeto de tipo Suppliers a la base de datos.

```
void btnGrabar_Click(object sender, RoutedEventArgs e)
{
   switch (xTipoOperacion)
   {
      //....
      case TipoOperacion.Insertar:
         dtsvcNorthwind.AddToSuppliers(cllSuppliers[(int)nIndiceFilaEnEdicion]);
         break;
   }
   dtsvcNorthwind.BeginSaveChanges(GrabarSuppliersCompletado, null);
}
```

Listado 8. Grabación de una fila insertada en el DataGrid.

Si observamos la clase que representa el servicio de datos generada por el EDM, comprobaremos cómo existe un método de este tipo (AddToNombreEntidad) para cada una de las entidades contenidas en el modelo de datos. La figura 2 muestra la página de este ejemplo durante la creación de una nueva fila.

Figura 2. Añadiendo una fila en el DataGrid.

En lo concerniente a la cancelación del proceso de inserción, centraremos nuestra atención en el evento KeyUp; donde la llamada al método RemoveAt de la colección subyacente al DataGrid desencadenará a su vez el evento SelectionChanged. Es en este último donde se puede producir un error, ya que se intentaría volver a borrar el elemento de la colección previamente eliminado en el evento KeyUp.

Para evitar este problema, en el evento KeyUp asignaremos el valor de la variable nIndiceFilaEnEdicion a la variable intermedia nIndiceFila y restaremos uno a nIndiceFilaEnEdicion. Al llamar a RemoveAt pasaremos como parámetro la variable nIndiceFila, de forma que al producirse el evento SelectionChanged “engañaremos” a la expresión que comprueba si la fila del DataGrid seleccionada es igual al valor de nIndiceFilaEnEdicion, para que no intente ejecutar nuevamente el método RemoveAt. El listado 9 muestra el código del evento KeyUp.

```
void grdDatos_KeyUp(object sender, KeyEventArgs e)
{
   if (e.Key == Key.Escape)
   {
      if (nIndiceFilaEnEdicion != null)
      {
         switch (xTipoOperacion)
         {
            //....
            case TipoOperacion.Insertar:
               int nIndiceFila = (int)nIndiceFilaEnEdicion;
               nIndiceFilaEnEdicion--;
               xTipoOperacion = TipoOperacion.Lectura;
               cllSuppliers.RemoveAt((int)nIndiceFila);
               break;
```

```
        }
        nIndiceFilaEnEdicion = null;
      }
    }
}
```

Listado 9. Cancelación de una inserción de fila en el DataGrid.

Borrado de datos

Y finalmente llegamos a la última de las operaciones de edición, el borrado, que se realizará cuando el usuario pulse la tecla Suprimir, lo que nos lleva de nuevo a recalar en el evento KeyUp.

Realizar un borrado es, en principio, algo tan sencillo como llamar al método DeleteObject del servicio de datos (pasando como parámetro el objeto a borrar de la colección) y a continuación al método BeginSaveChanges, para iniciar el proceso asíncrono de actualización sobre la fuente de datos.

Lamentablemente las cosas no suelen ser tan simples, ya que debemos tener en cuenta que las relaciones existentes entre las tablas de la base de datos (Suppliers con Products por ejemplo) harán que se produzca un error si intentamos borrar un registro de la tabla Suppliers, debido precisamente a la mencionada correspondencia con otras tablas del origen de datos.

Para evitar que se produzca el error que comentamos, en primer lugar debemos obtener los elementos relacionados con el elemento que queremos borrar; recorrer dichos elementos eliminando el enlace existente; y finalmente, proceder al borrado.

La forma de implementar este proceso consiste en llamar al método asíncrono BeginLoadProperty del servicio de datos, cuyo funcionamiento ya explicamos en la segunda parte del presente artículo.

Una vez realizada la llamada al método BeginLoadProperty, será durante la ejecución de su método de devolución de llamada (CargarProductsCompletado), donde recuperaremos el objeto Suppliers a borrar desde la propiedad AsyncState del parámetro IAsyncResult que recibe este método. A continuación recorreremos la colección existente en la propiedad Products del objeto Suppliers, llamando, para cada elemento de la colección, al método DeleteLink del servicio de datos, que eliminará el enlace entre el objeto Suppliers y cada uno de los objetos Products. Finalmente, ejecutaremos el método DeleteObject del servicio de datos para borrar el objeto Suppliers, y acto seguido BeginSaveChanges, para iniciar el proceso asíncrono de borrado en la base de datos. En el método de devolución de llamada BorrarSupplierCompletado, llamaremos al método del servicio de datos EndSaveChanges para confirmar el borrado, y eliminaremos el elemento de la

colección para que se refleje en el DataGrid. El listado 10 muestra el código de todo este proceso.

```
void grdDatos_KeyUp(object sender, KeyEventArgs e)
{
   //....
   if (e.Key == Key.Delete && xTipoOperacion == TipoOperacion.Lectura)
   {
      if (MessageBox.Show("¿Borrar fila?", string.Empty, MessageBoxButton.OKCancel) ==
MessageBoxResult.OK)
      {
         Suppliers oSupplier = (Suppliers)this.grdDatos.SelectedItem;
         dtsvcNorthwind.BeginLoadProperty(oSupplier, "Products", CargarProductsCompletado,
oSupplier);
      }
   }
}

void CargarProductsCompletado(IAsyncResult oResultado)
{
   Suppliers oSupplier = (Suppliers)oResultado.AsyncState;
   dtsvcNorthwind.EndLoadProperty(oResultado);

   foreach (Products oProduct in oSupplier.Products)
   {
      dtsvcNorthwind.DeleteLink(oSupplier, "Products", oProduct);
   }

   dtsvcNorthwind.DeleteObject(oSupplier);
   dtsvcNorthwind.BeginSaveChanges(BorrarSupplierCompletado, oSupplier);
}

void BorrarSupplierCompletado(IAsyncResult oResultado)
{
   dtsvcNorthwind.EndSaveChanges(oResultado);
   cllSuppliers.Remove((Suppliers)oResultado.AsyncState);
}
```

Listado 10. Borrando un registro mediante el servicio de datos.

La figura 3 muestra un momento en la ejecución de esta opción de borrado.

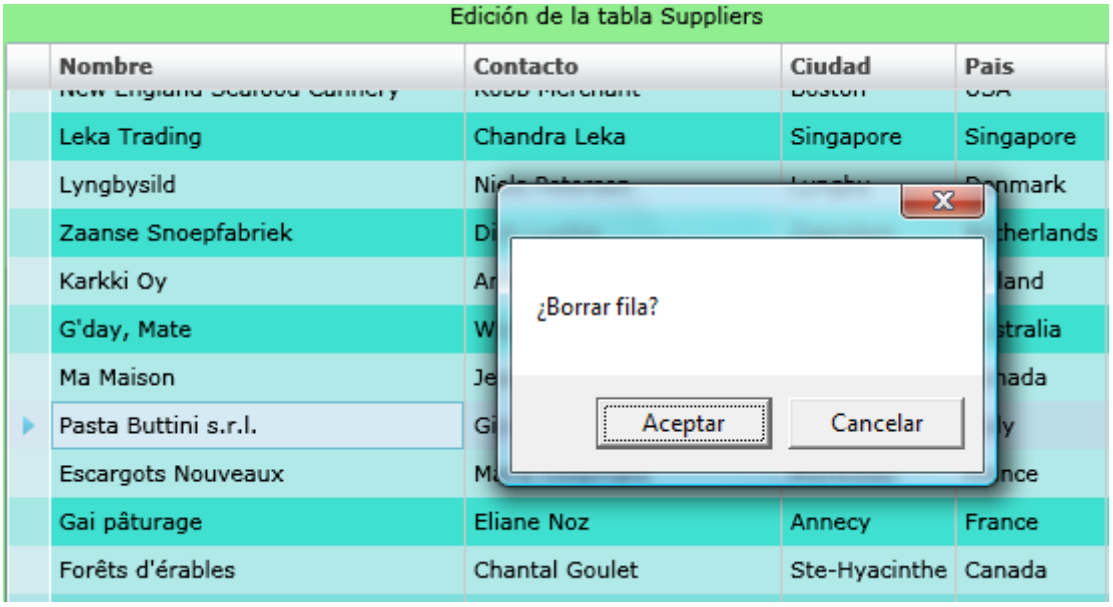

Figura 3. Borrando una fila desde el DataGrid.

El método DeleteLink, como hemos comprobado en este ejemplo, solamente elimina la relación entre el objeto Suppliers y todos los objetos Products asociados, a fin de poder borrar el registro de la tabla Suppliers. Como resultado, los registros de la tabla Products en los que se ha eliminado la relación aparecerán con el campo SupplierID a nulo. Si necesitáramos eliminar dichos registros relacionados, tendremos que ejecutar también el método DeleteObject para cada uno de esos registros, pero en tal caso, será necesario eliminar igualmente las relaciones que pudieran existir con terceras tablas, por ejemplo entre Products y Order Details.

Conclusión

Y tras este último ejemplo en las operaciones de edición, concluimos este artículo en el que hemos abordado los principales aspectos del manejo de datos con ADO.NET Data Services, desde sus conceptos básicos hasta la obtención y edición de un origen de datos mediante Silverlight. Los ejemplos expuestos en esta entrega están disponibles, como siempre, en http://www.dotnetmania.com/.