DomainDataSource

Un gestor de datos en Silverlight para la interfaz de usuario

DomainDataSource es un componente de WCF RIA Services que funciona como intermediario entre los datos proporcionados por el contexto de dominio y la interfaz de usuario de nuestras aplicaciones Silverlight, facilitando su manipulación. En este artículo abordaremos las principales características de este control, de modo que podamos explotar al máximo su potencial.

Luis Miguel Blanco

Aprendiz de brujo y redactor de dotNetManía (http://geeks.ms/blogs/lmblanco)

Al igual que el resto de elementos de la interfaz de usuario en una aplicación Silverlight, el control **DomainDataSource** se define declarativamente utilizando código XAML. No obstante, también será posible implementar determinados aspectos de su comportamiento desde *code behind*, como veremos en la aplicación de ejemplo que vamos a desarrollar.

Por otro lado, se trata de un control que no se limita simplemente a presentar los datos obtenidos desde el contexto de dominio, sino que está dotado de una serie de características adicionales que le permiten ordenar, filtrar, agrupar, etc. dichos datos.

Un ejemplo demostrativo

Con la finalidad de ilustrar las anteriormente mencionadas características en el uso del control **DomainDataSource**, vamos a crear en Visual Studio 2010 un nuevo proyecto de tipo "Silverlight Business Application" al que daremos el nombre **PruebasDDS**[1]. El motivo de utilizar este tipo de proyecto radica en que nos proporciona cierta infraestructura de la interfaz de usuario ya preparada, como es la posibilidad de tener varias páginas en la aplicación, realizando la navegación entre las mismas, lo que aprovecharemos para explicar cada uno de los aspectos del **DomainDataSource** en páginas distintas del proyecto.

Para adaptar esta interfaz prefabricada a nuestras necesidades, acomodando los estilos y recursos asignados a tipos de letra, literales, controles, etc., debemos editar el archivo de estilos **Styles.xaml**, situado en la carpeta **Assets** de la estructura del proyecto; y el archivo de recursos **ApplicationStrings.resx**, situado en la carpeta **Assets/Resources**. Cuando necesitemos añadir nuevas páginas al proyecto lo haremos dentro de la carpeta **Views** (figura 1).

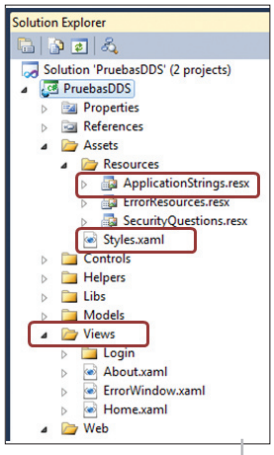


Figura 1. Estructura de carpetas del proyecto

Por otro lado, esta plantilla de proyecto también proporciona ya activado el soporte de WCF RIA Services [1], necesario para la interacción de los datos entre las capas cliente e intermedia de la

[1] Todo el código fuente desarrollado a lo largo de este artículo está disponible para su descarga en **http://www.dotnetmania.com**

aplicación [2]. Como requisito previo, necesitaremos tener instalado Silverlight 4 Tools for Visual Studio 2010 [3]. En cuanto a los datos de ejemplo, utilizaremos la base de datos Chinook [4], que podemos descargar desde CodePlex.

La primera tarea a realizar será la creación, en el proyecto Web de nuestra solución, de un "ADO.NET Data Model" (modelo de datos) con el nombre **ChinookModel**, donde a partir de la tabla **Invoice** de la base de datos se generará una entidad del mismo nombre. A continuación, crearemos un "Domain Service Class" (servicio de dominio) con el nombre **ChinookDomainService**, que configuraremos para que genere el código para las operaciones CRUD sobre la entidad **Invoice**. Estas operaciones de creación se explican con más detalle en [2].

Obtención, presentación y actualización de datos

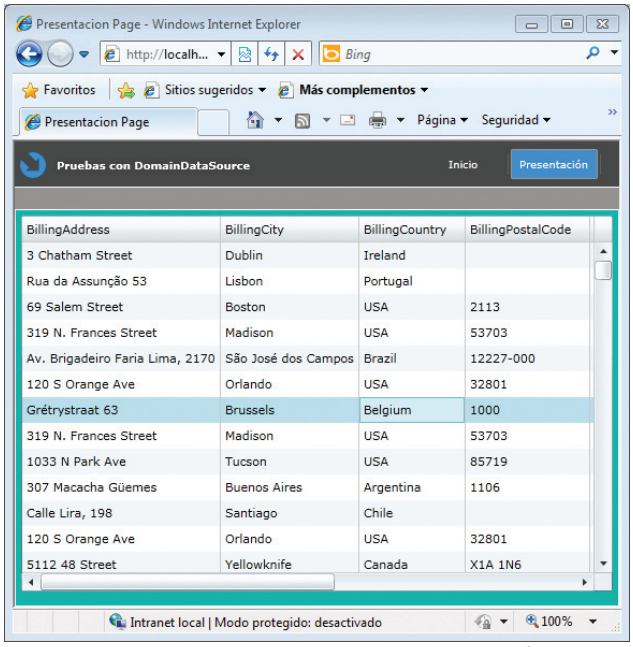
La principal tarea a cargo del control **DomainDataSource** consiste en obtener una colección de entidades desde el contexto de dominio mediante la llamada a uno de sus métodos. Una vez obtenida la colección, aplicaremos en los elementos de la interfaz de usuario expresiones de enlace a datos (*data binding*), para conseguir visualizar en dichos elementos los valores de las entidades.

Comenzaremos, como indica el título de este apartado, por las operaciones elementales de recuperación, visualización y edición de datos. Para ello, añadiremos al proyecto una página con el nombre **Presentacion.xaml**, a la que accederemos desde la página principal de la aplicación (**MainPage.xaml**) utilizando un control **HyperlinkButton**, como vemos en el listado 1.

```
<HyperlinkButton x:Name="lnkPresentacion"
    Style="{StaticResource LinkStyle}"
    NavigateUri="/Presentacion"
    TargetName="ContentFrame"
    Content="{Binding Path=ApplicationStrings.PresentacionPageTitle,
    Source={StaticResource ResourceWrapper}}"/>
```

Listado 1

Dentro del diseñador de la página **Presentacion.xaml** arrastraremos un **DomainDataSource** y un **DataGrid** desde la Caja de herramientas, asignándoles respectivamente los nombres **ddsInvoices** y **grdInvoices**. Debido a que el **DomainDataSource** necesita acceder al contexto de dominio, tenemos que especificar declarativamente dicho dominio añadiendo un espacio de nombres (**xmlns**) con el nombre **domainctx**, en la etiqueta **UserControl** de la página.



BillingAddress	BillingCity	BillingCountry	BillingPostalCode
3 Chatham Street	Dublin	Ireland	
Rua da Assunção 53	Lisbon	Portugal	
69 Salem Street	Boston	USA	2113
319 N. Frances Street	Madison	USA	53703
Av. Brigadeiro Faria Lima, 2170	São José dos Campos	Brazil	12227-000
120 S Orange Ave	Orlando	USA	32801
Grétrystraat 63	Brussels	Belgium	1000
319 N. Frances Street	Madison	USA	53703
1033 N Park Ave	Tucson	USA	85719
307 Macacha Güemes	Buenos Aires	Argentina	1106
Calle Lira, 198	Santiago	Chile	
120 S Orange Ave	Orlando	USA	32801
5112 48 Street	Yellowknife	Canada	X1A 1N6

Figura 2. DataGrid visualizando los datos del DomainDataSource

Pasando a la configuración del control **DomainDataSource**, la propiedad **QueryName** contendrá el nombre del método perteneciente al servicio de dominio (**GetInvoices**), que devuelve la colección de entidades al contexto de dominio, y éste a su vez al control. Para que el **DomainDataSource** tenga acceso al contexto de dominio, añadiremos al código de este control la etiqueta **DomainContext**, especificando el espacio de nombres **domainctx**, declarado en la etiqueta **UserControl**.

Finalmente, para que el **DataGrid** pueda mostrar datos, asignaremos a su propiedad **ItemsSource** una expresión de enlace a datos que conecte este control con el **DomainDataSource** (listado 2). El resultado lo vemos en la figura 2.

```
<navigation:Page x:Class="PruebasDDS.Views.Presentacion"
xmlns:domainctx="clr-namespace:PruebasDDS.Web"
<!-- .... -->
<riaControls:DomainDataSource x:Name="ddsInvoices" QueryName="GetInvoices">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>
</riaControls:DomainDataSource>
<!-- .... -->
<sdk:DataGrid x:Name="grdInvoices"
              ItemsSource="{Binding ElementName=ddsInvoices, Path=Data}"
              Margin="5" Height="350" />
```

Listado 2

En tiempo de ejecución, además de consultar los datos también es posible editarlos. Una vez finalizados los cambios, podemos grabarlos en la base de datos o descartarlos mediante los métodos **SubmitChanges** o **RejectChanges** del **DomainDataSource**, lo que en este ejem-

```
<Button x:Name="btnGrabar" Content="Grabar cambios"
    Width="110" Margin="5" Click="btnGrabar_Click" />
<Button x:Name="btnDeshacer" Content="Deshacer cambios"
    Width="110" Margin="5" Click="btnDeshacer_Click" />

//-----------
private void btnGrabar_Click(object sender, RoutedEventArgs e)
{
    this.ddsInvoices.SubmitChanges();
}

private void btnDeshacer_Click(object sender, RoutedEventArgs e)
{
    this.ddsInvoices.RejectChanges();
}
```

Listado 3

Figura 3. DataGrid ordenado mediante el DomainDataSource

plo llevaremos a cabo desde el evento **Click** de sendos controles **Button** (listado 3).

Ordenación

La siguiente característica del **DomainDataSource** que abordaremos será la capacidad de ordenar sus datos, para lo que añadiremos una nueva página al proyecto con el nombre **Ordenacion.xaml**, en la que al igual que en el caso anterior (y en los próximos ejemplos) añadiremos un **DomainDataSource** y un **DataGrid**; pero en esta ocasión reduciremos el número de columnas del **DataGrid**, creándolas manualmente dentro de la etiqueta **Columns**, a la que añadiremos tantos controles **DataGridTextColumn** como columnas necesitemos en la cuadrícula de datos. La propiedad **Binding**, mediante una expresión de enlace a datos, será la encargada de obtener del **DomainDataSource** el valor correspondiente para cada columna.

Para ordenar los datos en el **DomainDataSource** emplearemos la etiqueta **SortDescriptors**, que representa una colección de objetos **SortDescriptor**; este tipo de objeto contiene el modo de ordenación por uno de los campos de la colección de entidades devuelta por el **DomainDataSource**.

En esta parte del ejemplo ordenaremos los resultados por los campos **BillingCountry**, **BillingCity** y **Total** (listado 4), estableciendo también un sentido de la ordenación distinto para cada uno, me-

```
<riaControls:DomainDataSource x:Name="ddsInvoices" QueryName="GetInvoices"
    AutoLoad="True">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.SortDescriptors>
        <riaControls:SortDescriptor PropertyPath="BillingCountry" Direction="Descending" />
        <riaControls:SortDescriptor PropertyPath="BillingCity" Direction="Ascending" />
        <riaControls:SortDescriptor PropertyPath="Total" Direction="Descending" />
    </riaControls:DomainDataSource.SortDescriptors>
</riaControls:DomainDataSource>
<!-- .... -->
<sdk:DataGrid x:Name="grdInvoices"
              ItemsSource="{Binding ElementName=ddsInvoices, Path=Data}"
              AutoGenerateColumns="False" Margin="5" Height="350">

    <sdk:DataGrid.Columns>
        <sdk:DataGridTextColumn Header="Factura" Binding="{Binding InvoiceId}" />
        <sdk:DataGridTextColumn Header="Cliente" Binding="{Binding CustomerId}" />
        <sdk:DataGridTextColumn Header="País" Binding="{Binding  BillingCountry}" />
        <sdk:DataGridTextColumn Header="Ciudad" Binding="{Binding BillingCity}" />
        <sdk:DataGridTextColumn Header="Importe" Binding="{Binding Total}" />
    </sdk:DataGrid.Columns>
</sdk:DataGrid>
```

Listado 4

diante el atributo **Direction** de los objetos **SortDescriptor**.

Apreciaremos los campos ordenados porque en su cabecera se muestra un pequeño indicador del sentido de la ordenación (figura 3). Si bien el propio control **DataGrid** permite por defecto ordenar una columna haciendo clic en su cabecera, la ordenación desde el **DomainDataSource**, como acabamos de comprobar, presenta como ventaja la posibilidad de ordenar por más de una columna simultáneamente.

Ordenando desde code behind

Aunque la programación que habitualmente realicemos con el **DomainDataSource** sea eminentemente declarativa, también existe la posibilidad de implementar ciertos comportamientos desde *code behind*. Como muestra, añadiremos al diseñador de la página dos controles **TextBox**, dos **CheckBox** y un **Button** (listado 5), y en el evento **Click** de este último eliminaremos el orden establecido declarativamente en la página XAML. Seguidamente, crearemos dos nuevos objetos **SortDescriptor**, utilizando como campo de ordenación el valor de los **TextBox**, estableciendo el sentido del orden ascendente o descendente en función de si está marcado el **CheckBox** situado junto a cada **TextBox**. Finalmente, agregaremos estos objetos a la colección **DomainDataSource.SortDescriptors** (listado 6), con el resultado que vemos en la figura 4.

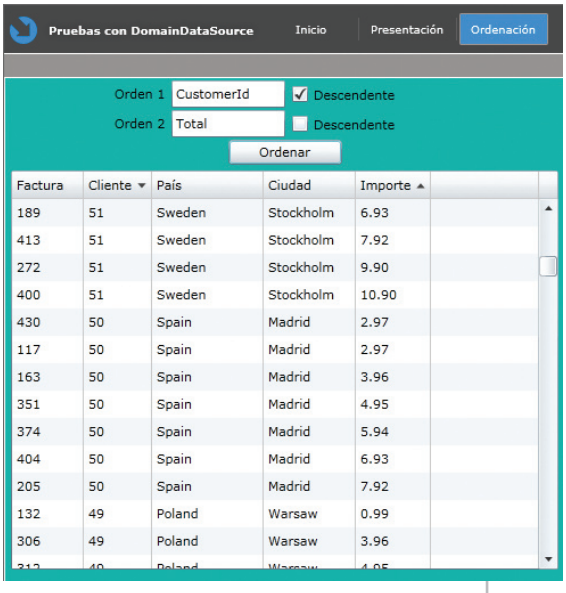


Figura 4. Ordenación mediante controles de la página

```
<StackPanel Width="300" >
    <StackPanel Orientation="Horizontal">
        <TextBlock Text="Orden 1" Margin="0,2,5,0"
                   VerticalAlignment="Center" />
        <TextBox x:Name="txtOrden1" Margin="0,2,5,0" Width="100" />
        <CheckBox x:Name="chkDescendente1" Content="Descendente"
                  Margin="0,2,10,0" VerticalAlignment="Center" />
    </StackPanel>

    <StackPanel Orientation="Horizontal">
        <TextBlock Text="Orden 2" Margin="0,2,5,0"
                   VerticalAlignment="Center" />
        <TextBox x:Name="txtOrden2" Margin="0,2,5,0" Width="100" />
        <CheckBox x:Name="chkDescendente2" Content="Descendente"
                   Margin="0,2,10,0" VerticalAlignment="Center" />
    </StackPanel>

    <Button x:Name="btnOrdenar" Content="Ordenar" Margin="0,2,0,0"
            Width="100"  HorizontalAlignment="Center"
            Click="btnOrdenar_Click" />
</StackPanel>
<!-- .... -->
```

Listado 5

```
private void btnOrdenar_Click(object sender, RoutedEventArgs e)
{
    this.ddsInvoices.SortDescriptors.Clear();

    SortDescriptor oSortDesc1 = new SortDescriptor(this.txtOrden1.Text,
        this.chkDescendente1.IsChecked.Value
            ? System.ComponentModel.ListSortDirection.Descending
            : System.ComponentModel.ListSortDirection.Ascending);

    this.ddsInvoices.SortDescriptors.Add(oSortDesc1);

    SortDescriptor oSortDesc2 = new SortDescriptor(this.txtOrden2.Text,
        this.chkDescendente2.IsChecked.Value
            ? System.ComponentModel.ListSortDirection.Descending
            : System.ComponentModel.ListSortDirection.Ascending);

    this.ddsInvoices.SortDescriptors.Add(oSortDesc2);
}
```

Listado 6

Filtrado

Si necesitamos establecer una condición sobre los datos devueltos por el **DomainDataSource**, de modo que sólo visualicemos un subconjunto de los mismos, aplicaremos sobre este control uno o varios filtros, representados por la clase (etiqueta XAML) **FilterDescriptor**, asignando a su propiedad **PropertyPath** el nombre del campo sobre el que se aplicará el filtro; la propiedad **Value** contendrá el valor del filtro, mientras que con la propiedad **Operator** (del tipo enumerado **FilterDescriptorLogicalOperator**) estableceremos el modo de aplicación del filtro (igual que, mayor que, menor o igual que, etc.). Todos los filtros deberán estar contenidos dentro de la etiqueta **FilterDescriptors**

```
<riaControls:DomainDataSource x:Name="ddsInvoices"
                              QueryName="GetInvoices" AutoLoad="True">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.FilterDescriptors>
        <riaControls:FilterDescriptor PropertyPath="BillingCountry"
            Operator="IsEqualTo" Value="Canada" />
        <riaControls:FilterDescriptor PropertyPath="BillingCity"
            Operator="IsContainedIn" Value="Halifax,Ottawa" />
        <riaControls:FilterDescriptor PropertyPath="Total"
            Operator="IsLessThan" Value="5" />
    </riaControls:DomainDataSource.FilterDescriptors>
</riaControls:DomainDataSource>
```

Listado 7

del **DomainDataSource**, que representa la colección de filtros creados en el control.

Para poner en práctica esta característica, después añadir una página con el nombre **Filtros.xaml** a nuestro proyecto y configurarla como en los anteriores casos, agregaremos una combinación de filtros por tres campos de las entidades manejadas por el **DomainDataSource** (listado 7).

Factura	Cliente	País	Ciudad	Importe
75	31	Canada	Halifax	4.95
177	31	Canada	Halifax	3.96
324	31	Canada	Halifax	3.96
342	30	Canada	Ottawa	3.96
386	30	Canada	Ottawa	3.96
308	30	Canada	Ottawa	4.95
124	30	Canada	Ottawa	3.96

Figura 5. Datos filtrados mediante el DomainDataSource

Al poner en ejecución esta página, comprobaremos la efectividad del código que acabamos de escribir, ya que el número de registros mostrados por el **DataGrid** se reducirá de acuerdo con las condiciones establecidas en los filtros (figura 5).

Como hemos comprobado, existen diversos modos de combinar los filtros y sus propiedades, a fin de acotar los resultados devueltos por el **DomainDataSource**: un filtro único, varios filtros sobre distintos campos, una colección de valores separados por comas, etc. Pero supongamos que necesitamos crear un filtro sobre un mismo campo numérico que contemple varios valores; si intentamos poner dichos valores en una lista separada por comas obtendremos un error, aunque esto no significa que no podamos establecer dicho filtro, ya que la solución estriba en emplear un filtro independiente por cada valor, añadiendo en la declaración del control **DomainDataSource** el valor **Or** al atributo **FilterOperator** (listado 8).

No obstante, los ejemplos anteriores resultan muy rígidos, ya que establecen un valor fijo para el filtro sin poder cambiarlo, mientras que lo deseable en este tipo de situación sería dar al usuario la posibilidad de introducir dicho valor.

Factura	Cliente	País	Ciudad	Importe
4	25	USA	Madison	5.95
8	25	USA	Madison	6.93
23	25	USA	Madison	6.94
92	25	USA	Madison	9.91
117	50	Spain	Madrid	2.97
153	25	USA	Madison	10.89
158	25	USA	Madison	5.94
163	50	Spain	Madrid	3.96

Figura 6. Filtrando desde un control de la página

```
<riaControls:DomainDataSource x:Name="ddsInvoices" QueryName="GetInvoices"
    FilterOperator="Or" AutoLoad="True">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.FilterDescriptors>
        <riaControls:FilterDescriptor PropertyPath="Total" Operator="IsEqualTo" Value="3,96" />
        <riaControls:FilterDescriptor PropertyPath="Total" Operator="IsEqualTo" Value="7,92" />
        <riaControls:FilterDescriptor PropertyPath="Total" Operator="IsEqualTo" Value="10,90" />
    </riaControls:DomainDataSource.FilterDescriptors>
</riaControls:DomainDataSource>
```

Listado 8

```
<riaControls:DomainDataSource.FilterDescriptors>
    <riaControls:FilterDescriptor
        PropertyPath="BillingCity"
        Operator="StartsWith"
        Value="{Binding ElementName=txtBillingCity,
        Path=Text}" />
</riaControls:DomainDataSource.FilterDescriptors>
<!-- .... -->
<StackPanel Orientation="Horizontal" Margin="5">
    <TextBlock Text="Ciudad:"
               VerticalAlignment="Center" />
    <TextBox x:Name="txtBillingCity" Width="200" />
</StackPanel>
<!-- .... -->
```

Listado 9

Podemos lograr este comportamiento desde el *code behind* de la página, como vimos en la ordenación de datos; sin embargo, en este caso optaremos por una solución distinta, pero igualmente efectiva (listado 9), consistente en enlazar la propiedad **FilterDescriptor.Value** con un control de la página que sea el que proporcione el valor de filtro, por ejemplo, un **TextBox**; empleando el operador de filtro **StartsWith**, para que los datos sean filtrados dinámicamente según escribimos (figura 6).

Agrupación de datos

Mediante el uso combinado de la colección **GroupDescriptors** y objetos **GroupDescriptor**, podemos agrupar los datos del **DomainDataSource** en base a un campo de la colección de entidades de este último, de forma que en el **DataGrid** todos los registros con el mismo valor para el campo de agrupamiento muestren una fila en la cuadrícula de datos a modo de cabecera para dicho campo; al hacer clic en esa fila de cabecera se contraerán o expandirán las filas dependientes (figura 7). Para conseguir esta funcionalidad, al declarar el grupo en el código de la página XAML, asignaremos el nombre del campo utilizando la propiedad **GroupDescriptor.PropertyPath**. Todas estas operaciones las realizaremos añadiendo una página al proyecto con el nombre **Grupos.xaml** (listado 10).

```
<riaControls:DomainDataSource x:Name="ddsInvoices"
    QueryName="GetInvoices"
    AutoLoad="True">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.GroupDescriptors>
        <riaControls:GroupDescriptor
            PropertyPath="BillingCountry" />
    </riaControls:DomainDataSource.GroupDescriptors>
</riaControls:DomainDataSource>
```

Listado 10

Factura	Cliente	País	Ciudad	Importe
411	7	Austria	Vienne	3.97
◢ BillingCountry: Belgium (7 items)				
423	8	Belgium	Brussels	0.99
333	8	Belgium	Brussels	13.88
334	8	Belgium	Brussels	3.96
453	8	Belgium	Brussels	11.89
208	8	Belgium	Brussels	2.97
76	8	Belgium	Brussels	6.93
7	8	Belgium	Brussels	3.96
▷ BillingCountry: Brazil (45 items)				
▷ BillingCountry: Canada (61 items)				
◢ BillingCountry: Chile (9 items)				
24	57	Chile	Santiago	7.92
14	57	Chile	Santiago	6.94

Figura 7. Datos agrupados en el DataGrid mediante el DomainDataSource

Si añadimos varios objetos **GroupDescriptor** (listado 11) conseguiremos un agrupamiento anidado a varios niveles (figura 8).

```
<riaControls:DomainDataSource.GroupDescriptors>
    <riaControls:GroupDescriptor
       PropertyPath="BillingCountry" />
    <riaControls:GroupDescriptor
       PropertyPath="BillingCity" />
</riaControls:DomainDataSource.GroupDescriptors>
```

Listado 11

También podemos realizar la operación de expandir/contraer los grupos desde *code behind*, lo cual confiere a nuestra aplicación una mayor flexibilidad. Si por ejemplo añadimos un **CheckBox** a la página (listado 12), en su evento **Click** (listado 13) obtenemos, de la propiedad **DomainDataSource.Data**, los datos en forma del tipo **ICollectionView**. Recorriendo la colección **ICollectionView.Groups**, por cada uno de los objetos **CollectionViewGroup** que ésta contiene podremos expandir o contraer el grupo en el **DataGrid** llamando a los métodos **CollapseRowGroup** o **ExpandRowGroup** de este control, pasándoles como parámetro el objeto **CollectionViewGroup**. Previamente ejecutaremos el método **DataGrid.ScrollIntoView** para posicionarnos en el grupo requerido (figura 9).

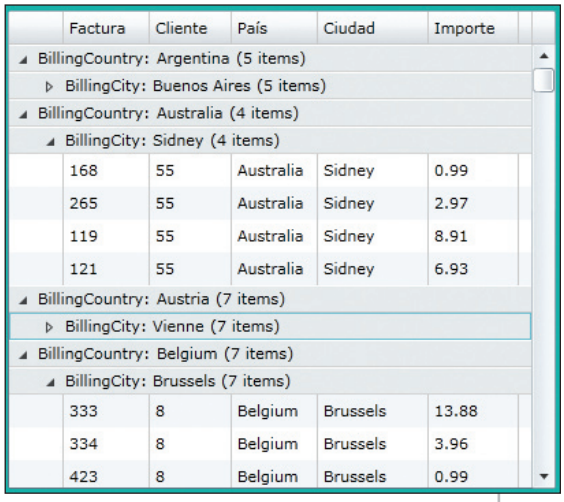


Figura 8. Grupos anidados.

Selección mediante parámetros

Además del código generado automáticamente por el servicio de dominio, podemos escribir nuestros propios métodos en dicho servicio, que retornen los datos en función de los parámetros que reciban, por lo que necesitamos disponer en el **DomainDataSource** de algún tipo de mecanismo que nos permita enviar los valores de tales parámetros cuando el método a utilizar en este control así lo requiera.

Para crear un parámetro utilizaremos la clase **Parameter**, asignando a la propiedad **ParameterName** el nombre del parámetro perteneciente al método, mientras que en la propiedad **Value** asignaremos el valor que recibirá el parámetro. Los parámetros tendrán que estar contenidos dentro de la colección **DomainDataSource.QueryParameters**.

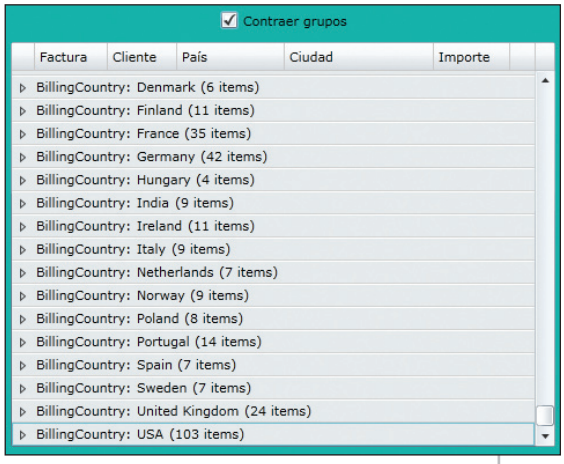


Figura 9. DataGrid con todos los grupos contraídos

```
<CheckBox x:Name="chkContraer"
          Content="Contraer grupos"
          HorizontalAlignment="Center"
          Margin="5"
          Click="chkContraer_Click" />
```

Listado 12

```
private void chkContraer_Click(object sender, RoutedEventArgs e)
{
    ICollectionView oCollectionView = (ICollectionView)this.ddsInvoices.Data;

    if ((bool)this.chkContraer.IsChecked)
    {
        foreach (CollectionViewGroup  oCollectionViewGroup in oCollectionView.Groups)
        {
            this.grdInvoices.ScrollIntoView(oCollectionViewGroup, null);
            this.grdInvoices.CollapseRowGroup(oCollectionViewGroup, true);
        }
    }
    else
    {
        foreach (CollectionViewGroup oCollectionViewGroup in oCollectionView.Groups)
        {
            this.grdInvoices.ScrollIntoView(oCollectionViewGroup, null);
            this.grdInvoices.ExpandRowGroup(oCollectionViewGroup, true);
        }
    }
}
```

Listado 13

```
public IQueryable<Invoice> GetInvoicesBillingCountry(string sBillingCountry)
{
    return this.ObjectContext.Invoices.Where(oInvoice => oInvoice.BillingCountry == sBillingCountry);
}
```

Listado 14

```
<riaControls:DomainDataSource x:Name="ddsInvoices" QueryName="GetInvoicesBillingCountry" AutoLoad="True">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.QueryParameters>
        <riaControls:Parameter ParameterName="sBillingCountry" Value="Canada" />
    </riaControls:DomainDataSource.QueryParameters>
</riaControls:DomainDataSource>
```

Listado 15

Para ilustrar el uso de parámetros mediante un ejemplo, añadiremos al servicio de dominio el método **GetInvoicesBillingCountry** (listado 14), que recibe un parámetro con el que devuelve las entidades **Invoice** que tengan dicho valor en la propiedad **BillingCountry**. A continuación, añadiremos al proyecto la página **Parametros.xaml**, agregando a ésta un **DomainDataSource** con el nombre del método recién creado en la propiedad **QueryName**, y la definición de un parámetro para dicho método (listado 15). Los datos resultantes se ajustarán al valor del parámetro (figura 10).

Por otro lado, al igual que ocurre con los filtros, mediante una expresión de enlace a datos (listado 16) podemos conseguir que el usuario, a través de un control de la página, sea quien introduzca el valor del parámetro (figura 11).

Factura	Cliente	País	Ciudad	Importe
13	33	Canada	Yellowknife	5.94
25	3	Canada	Montréal	3.96
51	15	Canada	Vancouver	5.95
55	31	Canada	Halifax	6.94
57	30	Canada	Ottawa	7.92
61	15	Canada	Vancouver	2.97
63	3	Canada	Montréal	10.89
64	15	Canada	Vancouver	5.94
70	29	Canada	Toronto	2.97

Figura 10. Datos resultantes del método parametrizado

País: Italy

Factura	Cliente	País	Ciudad	Importe
43	47	Italy	Rome	5.94
120	47	Italy	Rome	2.97
182	47	Italy	Rome	3.97
243	47	Italy	Rome	7.92
258	47	Italy	Rome	11.89
302	47	Italy	Rome	2.97
338	47	Italy	Rome	3.97
444	47	Italy	Rome	4.95
457	47	Italy	Rome	7.93

Figura 11. Envío del parámetro desde un control de la página

Paginación

Si tenemos que trabajar con una cantidad considerable de entidades, resultará recomendable su visualización paginada en el **DataGrid**, facilitando así la consulta al usuario. No obstante, el trabajo con este modo de visualización no depende única y exclusivamente del **Da-**

```
<riaControls:DomainDataSource x:Name="ddsInvoices"
     QueryName="GetInvoicesBillingCountry"
     AutoLoad="True">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.QueryParameters>
        <riaControls:Parameter
          ParameterName="sBillingCountry"
          Value="{Binding
            ElementName=txtBillingCountry,Path=Text}" />
    </riaControls:DomainDataSource.QueryParameters>
</riaControls:DomainDataSource>
<!-- .... -->
<StackPanel Orientation="Horizontal"
           HorizontalAlignment="Center">
    <TextBlock Text="País" Margin="5" />
    <TextBox x:Name="txtBillingCountry" Width="200"
           Margin="5" />
</StackPanel>
```

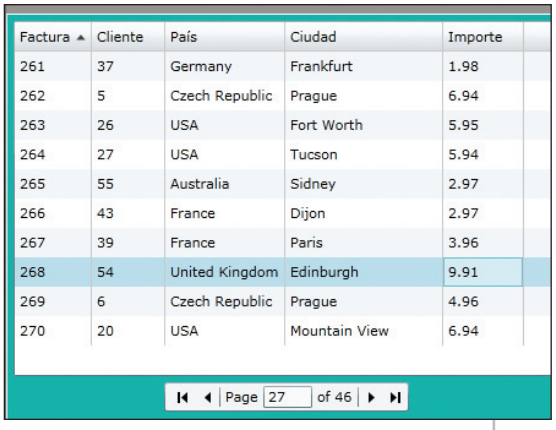
Listado 16

```
<riaControls:DomainDataSource x:Name="ddsInvoices"
      QueryName="GetInvoices" AutoLoad="True"
      PageSize="10" LoadSize="30">
    <riaControls:DomainDataSource.DomainContext>
        <domainctx:ChinookDomainContext />
    </riaControls:DomainDataSource.DomainContext>

    <riaControls:DomainDataSource.SortDescriptors>
        <riaControls:SortDescriptor
            PropertyPath="InvoiceId" />
    </riaControls:DomainDataSource.SortDescriptors>
</riaControls:DomainDataSource>
<!-- .... -->
<sdk:DataGrid x:Name="grdInvoices"
<!-- .... -->
<sdk:DataPager x:Name="pgrInvoices"
    Source="{Binding ElementName=ddsInvoices,Path=Data}"
           Width="200" />
```

Listado 17

taGrid, sino que también recae sobre el **DomainDataSource** y el control **DataPager**. Para aplicar esta característica, añadiremos al proyecto de ejemplo una nueva página llamada **Paginacion.xaml**.



Factura	Cliente	País	Ciudad	Importe
261	37	Germany	Frankfurt	1.98
262	5	Czech Republic	Prague	6.94
263	26	USA	Fort Worth	5.95
264	27	USA	Tucson	5.94
265	55	Australia	Sidney	2.97
266	43	France	Dijon	2.97
267	39	France	Paris	3.96
268	54	United Kingdom	Edinburgh	9.91
269	6	Czech Republic	Prague	4.96
270	20	USA	Mountain View	6.94

Page 27 of 46

Figura 12. Paginación de datos

En las propiedades **PageSize** y **LoadSize** (listado 17), pertenecientes al **DomainDataSource**, estableceremos respectivamente la cantidad de elementos a mostrar por página y la cantidad de elementos a cargar. El **DataPager** se encargará de proporcionar la interfaz de usuario para navegar por las páginas mostradas en el **DataGrid**. Precisamente por el hecho de utilizar el **DataPager**, hemos de definir un orden en el **DomainDataSource** para que la paginación funcione adecuadamente, debido a que Entity Framework no soporta paginación si no tiene establecido previamente un orden sobre los datos (figura 12).

Conclusiones

En este artículo hemos abordado las principales posibilidades que ofrece el **DomainDataSource**, un interesante componente de WCF RIA Services, que facilita la interacción con los datos entre la interfaz de usuario y el servicio/contexto de dominio de la aplicación. Para aquellos lectores interesados en seguir profundizando en todos los aspectos del funcionamiento de este componente, les recomendamos visitar el blog de **Jeff Handley** [5], uno de los principales miembros de su equipo de desarrollo.

Referencias

[1] WCF RIA Services: **https://www.silverlight.net/getstarted/riaservices**.

[2] **Blanco, Luis Miguel**. "Acceso a datos en Silverlight con WCF RIA Services". En **dotNetManía** nº 68, marzo de 2010.

[3] Silverlight 4 Tools for Visual Studio 2010 y Silverlight Toolkit: **https://www.silverlight.net/getstarted**.

[4] Base de datos Chinook: **http://chinookdatabase.codeplex.com**.

[5] Blog de **Jeff Handley**: **http://jeffhandley.com**.

[6] Recursos para desarrolladores en MSDN: **http://msdn.microsoft.com/es-ES/**.