|  |
| --- |
| [logo_gj01.gif](htto://joaquin.medina.name/) |
| Patrones de diseño |
| Patrón Mediador (Mediator) |
| Joaquin Medina Serrano |

|  |
| --- |
| El patrón Mediador define un objeto que encapsula cómo interactúan un conjunto de objetos. El mediador estimula la pérdida de acoplamiento ocultando las referencias explicitas entre los objetos, permitiendo variar su interacción de forma independientemente. Es decir, se define un objeto (mediador) que se encarga de gestionar a comunicación entre distintos objetos (Participantes), sin que éstos necesiten conocerse entre sí. |

**Información del documento**

|  |  |
| --- | --- |
|  | La Güeb de Joaquin (<http://joaquin.medina.name>) |
| Categoría | Programación .NET |
| Tema.: | Patrones de diseño |
| Titulo | Patrón de comportamiento Mediador (Mediator) |
| Autor.: | Joaquin Medina Serrano ([mailto:joaquin@medina.name](mailto:joaquin@medina.name?subject=Documento%20Patrón%20de%20comportamiento%20Mediador)) |
| Archivo | Informática/Lógica/Patrones de diseño |
| Sumario | El patrón Mediador define un objeto que encapsula cómo interactúan un conjunto de objetos. El mediador estimula la pérdida de acoplamiento ocultando las referencias explicitas entre los objetos, permitiendo variar su interacción de forma independientemente.  Dicho de otra forma: Se define un objeto (mediador) que se encarga de gestionar a comunicación entre distintos objetos (Participantes), sin que éstos necesiten conocerse entre sí. |
| Numero de paginas |  |
| Fecha de creación. |  |
| Fecha última modificación. |  |

Patrones de diseño - El mediador

Contenido

[Introducción 5](#_Toc248899376)

[Ejemplos donde se puede aplicar 5](#_Toc248899377)

[Estructura del patrón 6](#_Toc248899378)

[Estudio del ejemplo que propone Erich Gamma 7](#_Toc248899379)

[Ver un acoplamiento ligero 8](#_Toc248899380)

[¿Cuáles son los componentes de un mediador? 10](#_Toc248899381)

[Un ejemplo para implementar el patrón 11](#_Toc248899382)

[¿Cómo Implemento un mediador? 13](#_Toc248899383)

[Escribiendo los controles 13](#_Toc248899384)

[Definiendo la interfaz IParticipante 13](#_Toc248899385)

[Escribir los controles [Participantes] 14](#_Toc248899386)

[Escribir el objeto mediador 17](#_Toc248899387)

[Interfaz IMediador 17](#_Toc248899388)

[La case Mediador 19](#_Toc248899389)

[El Constructor 20](#_Toc248899390)

[Función Register 21](#_Toc248899391)

[Función UnRegister 22](#_Toc248899392)

[Función Notify 22](#_Toc248899393)

[Relaciones entre controles 23](#_Toc248899394)

[El código en el formulario 26](#_Toc248899395)

[El Evento Load 26](#_Toc248899396)

[Usar los eventos del formulario 27](#_Toc248899397)

[El problema de los menús 28](#_Toc248899398)

[Consideraciones finales 29](#_Toc248899399)

[Bibliografía 30](#_Toc248899400)

[Sobre el patrón Mediador 30](#_Toc248899401)

[General sobre Patrones 30](#_Toc248899402)

(Página en blanco)

## Introducción

La definición formal del mediador, dada en "Design Patterns, Elements of Reusable Object-Oriented Software" por Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides es:

*El patrón Mediador define un objeto que encapsula cómo interactúan un conjunto de objetos. El mediador estimula la pérdida de acoplamiento ocultando las referencias explicitas entre los objetos, permitiendo variar su interacción de forma independientemente.*

Dicho de otra forma: Se define un objeto (mediador) que se encarga de gestionar a comunicación entre distintos objetos (Participantes), sin que éstos necesiten conocerse entre sí.

## Ejemplos donde se puede aplicar

a) Para gestionar una red de ordenadores

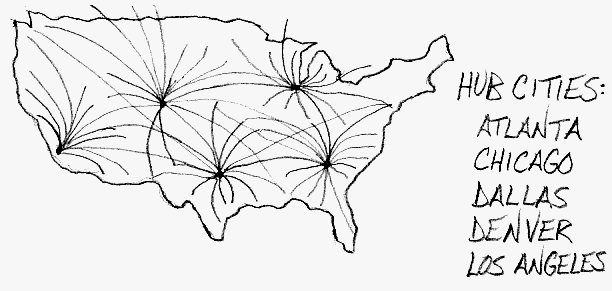


Imagen original en <http://www.vincehuston.org/dp/mediator.html>

b) Para gestionar el tráfico aéreo desde una torre de control

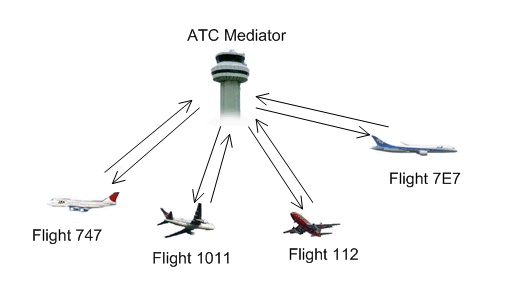


Imagen original en <http://sourcemaking.com/design_patterns/mediator>

c) Un ChatRoom

En la página [<http://www.dofactory.com/patterns/PatternMediator.aspx>.] existe un ejemplo, realizado en Java, en el que se muestra una sencilla aplicación de ChatRoom

## Estructura del patrón

El problema fundamental que resuelve este patrón es el de coordinar el funcionamiento de varios objetos que interactúan con otros objetos y que mantienen estructuras (muy) complicadas de relaciones entre sí. Para evitar esto el patrón Mediator encapsula el comportamiento de todo un conjunto de objetos en un solo objeto

La idea subyacente es conseguir que los objetos se puedan comunicar con otros sin tener que incluir referencias directas de código en sus clases.



Imagen original en: <http://kybele.escet.urjc.es/documentos/ISI/Patrones%20de%20Dise%C3%B1o.pdf>, Diapositiva 84

* Cuando muchos objetos interactúan con otros objetos, se puede formar una estructura muy compleja, con objetos con muchas conexiones con otros objetos. En un caso extremo cada objeto puede conocer a todos los demás objetos.

|  |  |
| --- | --- |
| Mediator example | Mediator example |

Imagen original en: <http://sourcemaking.com/design_patterns/mediator>

Fijémonos en el ejemplo de la torre de control de tráfico aéreo

Expresándolo en UML, el patrón mediador tiene la siguiente estructura

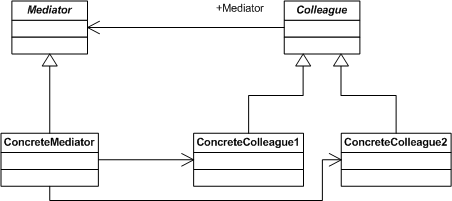


Imagen original en <http://pages.cpsc.ucalgary.ca/~heatond/mediator/>

Implementando este patrón en el ejemplo concreto del control de tráfico aéreo tendremos la siguiente estructura UML

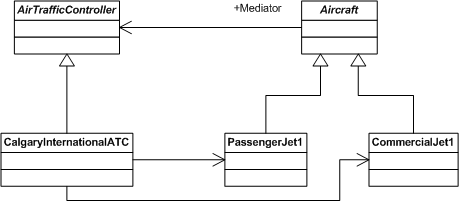


Imagen original en <http://pages.cpsc.ucalgary.ca/~heatond/mediator/>

Los eventos que ocurren entre los objetos implicados en el ejemplo de control de tráfico aéreo puede verse en el siguiente diagrama de secuencias

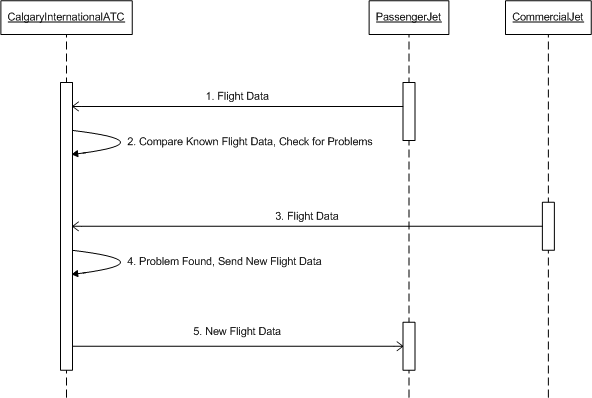


Imagen original en <http://pages.cpsc.ucalgary.ca/~heatond/mediator/>

## Estudio del ejemplo que propone Erich Gamma

En el libro **Patrones de Diseño** cuya ficha bibliográfica se muestra a continuación, en la página 251, habla de patrón Mediador, y en el apartado motivación dice (y copio literalmente)

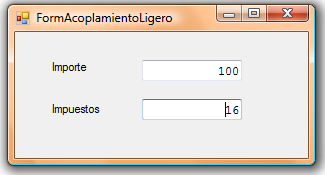
|  |
| --- |
| Datos de Catalogo Bibliográfico  **Gamma,E; Helm, R.; Johnson, R.; Vlissides, J.**  *Patrones de Diseño. Elementos de software orientado a objetos reutilizable*  PEARSON EDUCACION. S.A., Madrid 2003  ISBN 10:84-7829-059-1  ISBN 13: 978-84-7829-059-8  Materia: Informática 681.3  Formato 195x250. Páginas 384 |

*Un ejemplo típico es la implementación de los cuadros de dialogo en una interfaz de usuario, en la que las relaciones entre los distintos controles puede ser variada y complicada, por ejemplo, al seleccionar una opción en una lista desplegable, se puede cambiar el contenido de un campo de entrada. A la inversa, teclear texto en un campo de entrada puede seleccionar automáticamente una o más entradas de la lista desplegable. En cuanto haya texto en el campo de entrada, pueden activarse otros botones para permitir al usuario hacer algo con dicho texto, como cambiar o borrar aquello a lo que se refiere.*

Es decir, la idea es diseñar una interfaz de Windows que responda como una entidad única, habilitando e inhabilitando funciones y controles como respuesta a las acciones del usuario u opciones, o, de acuerdo con sus derechos y permisos.

## Ver un acoplamiento ligero

Por ejemplo, vamos a intentar resolver el problema de tener dos cuadros de texto en un formulario. Cuando se introduce un valor en uno, el otro debe mostrar el resultado de algún tipo de cálculo basado en este valor (por ejemplo, al introducir el precio en uno, mostrar el importe del IVA en el otro). Obviamente, si no hay valor en la fuente, no habría resultado en el destino; pero ¿cómo podemos asegurarnos que el cálculo, sólo se realiza si el valor es mayor que cero?



La solución más simple es usar el evento el evento Changed o bien la pareja de eventos Validating/Validated para incluir en los mismos, el código correspondiente que maneje la interacción de los dos controles, como, por ejemplo:

Public Class FormAcoplamientoLigero

Private Sub FormAcoplamientoLigero\_Load( \_

ByVal sender As System.Object, \_

ByVal e As System.EventArgs) \_

Handles MyBase.Load

'------------------------------------

'Apunte Táctico

'------------------------------------

' Si la propiedad CausesValidation se establece en

' false,los eventos Validating y Validated se suprimen.

' Ver http://msdn.microsoft.com/es-es/library/

' System.Windows.Forms.Control.Validating.aspx()

'------------------------------------

TextBoxImporte.CausesValidation = True

End Sub

#Region "Validacion del control [TextBoxImporte]"

'------------------------------------------------

' Empieza el proceso de validación

Private Sub TextBoxImporte\_Validating( \_

ByVal sender As Object, \_

ByVal e As System.ComponentModel.CancelEventArgs) \_

Handles TextBoxImporte.Validating

'------------------------------------

'Apunte Táctico

'--------------------------------------------------------

' Si la propiedad Cancel de CancelEventArgs se establece

' en true en el delegado del evento Validating, se

' suprimen todos los eventos que normalmente se

' producirían después del evento Validating.

' --> Validated y Focus

' Ver http://msdn.microsoft.com/es-es/library/

' System.Windows.Forms.Control.Validating.aspx()

'------------------------------------

' Si esta vacio(si no hay caracteres), es un error

If TextBoxImporte.Text.Length = 0 Then

e.Cancel = True

ErrorProvider1.SetError( \_

TextBoxImporte, "No hay ninguna cantidad")

Else

' Si llega aquí es que en el control hay caracteres

' si hay algún carácter que no sea un numero es un error

For Each caracter As Char In TextBoxImporte.Text

If Char.IsNumber(caracter) = False Then

e.Cancel = True

ErrorProvider1.SetError( \_

TextBoxImporte, \_

"Hay caracteres extraños que no son números")

Exit For

End If

Next

End If

End Sub

' Termina el proceso de validación

Private Sub TextBoxImporte\_Validated( \_

ByVal sender As Object, \_

ByVal e As System.EventArgs) \_

Handles TextBoxImporte.Validated

'------------------------------------

' Si todo esta correcto limpiar el mensaje de

' error del control [ErrorProvider1]

ErrorProvider1.SetError(TextBoxImporte, String.Empty)

'------------------------------------

' realizar el cálculo correspondiente a los impuestos

Dim impuestos As Double

Dim numero As Double = 0

If Double.TryParse(TextBoxImporte.Text, numero) = True Then

impuestos = (numero \* 16) / 100

Me.TextBoxImpuestos.Text = impuestos.ToString

Else

Me.TextBoxImpuestos.Text = "Problemas con el cálculo"

End If

End Sub

#End Region

End Class

Para más información, ver las siguientes referencias

* <http://msdn.microsoft.com/es-es/library/system.windows.forms.control.validated.aspx>
* <http://msdn.microsoft.com/es-es/library/system.windows.forms.control.leave.aspx>

Este tipo de "acoplamiento ligero" puede ser aceptable cuando estamos tratando con un cuadro de texto y un botón de comandos en un único formulario. Sin embargo, rápidamente tendremos problemas si tratamos de adoptar esta solución al tratar con controles múltiples que tienen que interactuar en combinaciones diferentes y complejas.

Todos conocemos la sensación de que el código que maneja los eventos de nuestro Formulario es inmanejable, monstruoso, y resulta difícil hacer modificaciones de comportamiento. Es aquí donde juega su papel el patrón mediador.

La idea básica es, que cada objeto se comunica con un "mediador". El "mediador" conoce a todos los objetos, y sabe cómo manipular su estado cuando un evento dado se dispara. De esta forma, evitamos todos los problemas asociados con colocar código específico dentro de un método asociado con un evento, ya que únicamente debe figurar una llamada al objeto mediador. Por ejemplo, en el ejemplo anterior, si usamos el evento [TextChanged ] del control quedaría el evento algo parecido a esto

Private Sub UcTextBoxParticipanteImporte\_TextChanged( \_

ByVal sender As System.Object, \_

ByVal e As System.EventArgs) \_

Handles UcTextBoxParticipanteImporte.TextChanged

objMediador.Notify(UcTextBoxParticipanteImporte)

End Sub

Observa que el control cuadro de texto no tiene idea de qué hará el mediador con la información, o incluso qué información desea. Todo lo que tiene que hacer es llamar al método "Notify" y pasar una referencia a sí mismo.

## ¿Cuáles son los componentes de un mediador?

Un mediador tiene dos requerimientos esenciales.

* Primero, necesitamos una clase "*mediador*" que defina la interfaz, y funcionalidad genérica, para los "*mediadores concretos*" que son subclases que definen varias posibles implementaciones.
* El segundo es, que todos los objetos que se encuentran junto a este, deben estar basados en clases que pueden comunicarse con su mediador. La estructura básica del patrón de mediador se muestra a continuación:

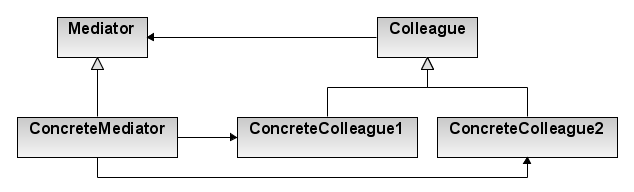


Imagen original en: <http://es.wikipedia.org/wiki/Archivo:Mediator_design_pattern.png>

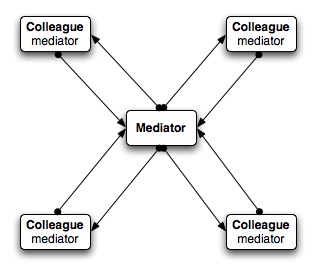


Imagen original en: <http://www.fancybread.com/blog/post.cfm/mediator-pattern-applied-to-javascript>

Si observamos el diagrama, lo primero que salta a la vista es que las clases que van a trabajar con un mediador necesitan guardar una referencia al objeto mediador. Lo segundo es, que el objeto mediador necesita guardar una referencia a todos los objetos colindantes.

El problema que se presenta es ¿cómo adquirir y guardar la referencia?, tenemos varias posibles respuestas, lo más sencillo de implementar es que cada control defina un método "*Registrame*", que verifica la presencia de un mediador y pasa una referencia de si mismo al mediador. Otra posibilidad es que el mediador defina el método "*DescubreObjetos()*", que busca en el entorno para descubrir objetos.

Otro problema es como guarda el mediador las referencias de los objetos, pues hay dos opciones, o bien utiliza una variable por objeto participante o bien utiliza una colección (un objeto Colección o una matriz).

Si utilizamos una colección de objetos, por ejemplo un objeto genérico Diccionario(Of Clave, referenciaObjeto), necesitaremos usar una clave diferente por cada control, hay varias alternativas, la más fácil es usar el nombre, (todos sabemos no puede haber en el mismo formulario dos controles con el mismo nombre)

## Un ejemplo para implementar el patrón

Vamos a implementar el patrón mediador en un formulario que tiene el aspecto que se muestra a continuación. Podemos considerar que es una calculadora de importes de IVA, de forma que en función de la base imponible y del tipo de IVA aplicable obtendremos el total de la factura

La interfaz tiene que realizar los caculos correspondientes y mostrar el resultado, también tiene que comprobar que los valores de los cuadros de texto sean adecuados al calculo que queremos realizar (es decir que sean números, y que haya algún numero) y en caso de no ser si debe mostrar un aviso al usuario mediante el control ErrorProvider

Las capturas de pantalla de esta ‘brillante’ aplicación son las siguientes

|  |  |
| --- | --- |
|  | Aspecto de la interfaz al pulsar el botón Borrar |
|  | Aspecto de la interfaz al pulsar el botón Calcular, sin datos en los controles de texto.  El control ErrorProvider muestra el icono de alerta al lado de los controles que tienen algún dato erróneo, (en este caso que están vacíos) |
|  | Aspecto de la interfaz al cambiar el contenido del ComboBox  Automáticamente se pone el valor del porcentaje de IVA correspondiente |
|  | Aspecto de la interfaz al pulsar el botón Calcular, con letras en lugar de un numero en la Base Imponible.  El control ErrorProvider muestra el icono de alerta al lado del control que tienen algún dato erróneo, (en este caso que contienen letras en lugar de números) |
|  | Aspecto de la interfaz al pulsar el botón Calcular, Con todos los datos correctos.  Se muestra en cada control la cantidad numérica correspondiente al cálculo realizado |

## ¿Cómo Implemento un mediador?

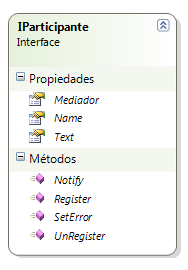
Para implementar un mediador en nuestro pequeño ejemplo necesitaremos lo siguiente

* Escribir las clases que van a ser mediadas. (Los participantes). En este ejemplo es un control TextBox y un control Button, tendremos que escribir dos controles personalizados que hereden de esas clases y que incluyan el código especifico para ser clases dirigidas por un mediador
* Incluir con el diseñador de Visual Estudio los nuevos controles que hemos escrito
* Escribir el objeto mediador
* Escribir código en el evento Load() de nuestro formulario principal para instanciar la clase medidora, y para dar a conoce al mediador los objetos participantes y a los objetos participantes quien es su mediador.

## Escribiendo los controles

### Definiendo la interfaz IParticipante

Antes que nada necesitamos definir una interfaz que contenga la funcionalidad que queremos que tengan todos los objetos participantes la interfaz tendrá la siguiente información:



''' <summary>

''' Interfaz de implementan los objetos participantes

''' </summary>

**Public Interface IParticipante**

Shadows Property Name() As String

Shadows Property Text() As String

Property Mediador() As IMediador

Sub Notify()

Sub Register()

Sub UnRegister()

Sub SetError(ByVal mensaje As String)

End Interface

Shadows Property Name() As String

Obtiene / establece el nombre de este objeto. En realidad expone la propiedad [Name] del objeto. La pongo en la interfaz porque se utiliza mucho y así evito el tener que instanciar el objeto original

Shadows Property Text() As String

Obtiene / establece el nombre de este objeto. En realidad expone la propiedad [Text] del objeto. La pongo en la interfaz porque se utiliza mucho y así evito el tener que instanciar el objeto original

Property Mediador() As IMediador

Es una propiedad que permite Establecer/obtener la referencia del objeto mediador de este participante

Sub Notify()

Funcion que llama al mediador pasando una referencia del participante

Sub Register()

El participante se registra en el mediador. Para que el [mediador] pueda usar un participante tiene que tener su referencia guardada en algún sitio, ese proceso de guardar la referencia del participante en el [mediador] es lo que denominamos 'registrar'

Sub UnRegister()

El participante se borra del registro del mediador, a partir de este momento, el mediador no lo tiene en cuenta, ya no es uno de sus participantes.

Sub SetError(ByVal mensaje As String)

Establece la cadena de descripción del error para este control. Verifica la presencia del [Mediador]. Si existe, llama al método [Mediador].SetError(Me, mensaje) pasando la referencia de este objeto y el mensaje. Durante este proceso, El [Mediador] verifica la presencia del objeto [ErrorProvider]. Si existe, llama al método [ErrorProvider].SetError(). Si no existe no se hace nada

### Escribir los controles [Participantes]

En nuestro ejemplo utilizamos un control TextBox para introducir la cantidad. Si queremos que sea un participante de un objeto Mediador, necesitamos que implemente la interfaz que acabamos de definir, y escribir en la misma el código especifico para cada una de las funciones.

En clases normales utilizaríamos una clase base y luego la herencia, pero en los controles esa aproximación no funciona, lo que hay que hacer es escribir un control personalizado que herede del control TextBox, y que implemente la interfaz que hemos definido algo así

**Public Class UCTextBoxParticipante**

Inherits TextBox

Implements IParticipante

#Region "Implements interfaz IParticipante"

#Region "Exponiendo propiedades del control a traves de la interfaz"

**Public Shadows Property Text() As String Implements IParticipante.Text**

Get

Return MyBase.Text

End Get

Set(ByVal value As String)

MyBase.Text = value

End Set

End Property

''' <summary>

''' Obtiene / establece el nombre de este objeto

''' En realidad expone la propiedad [Name] del objeto

''' La pongo en la interfaz porque se utiliza mucho y

''' así evito el tener que instanciar el objeto original

''' </summary>

**Public Shadows Property Name() As String Implements IParticipante.Name**

Get

Return MyBase.Name

End Get

Set(ByVal value As String)

MyBase.Name = value

End Set

End Property

#End Region

#Region "El Objeto mediador (si existe)"

''' <summary>

''' Guardar una referencia de objeto mediador

''' Utilizo una interfaz para que este control

''' pueda ser usado por varios mediadores

''' por ejemplo en dos formularios diferentes

''' Valor por defecto: Nothing

''' </summary>

**Private \_ObjMediador As IMediador = Nothing**

''' <summary>

''' Guardar una referencia del objeto mediador

''' En realidad guarda una referencia a la interface

''' del objeto mediador

''' Utilizo una interfaz para que este control pueda

''' ser usado por varios mediadores

''' Valor por defecto: Nothing

''' </summary>

**Public Property ZMediador() As IMediador \_**

Implements IParticipante.Mediador

Get

Return \_ObjMediador

End Get

Set(ByVal value As IMediador)

\_ObjMediador = value

End Set

End Property

''' <summary>

''' Proceso de notificación al Mediador

''' Verifica la presencia del mediador. Si existe,

''' llama al método [Mediador].Notify(),

''' pasando la referencia de este objeto.

''' </summary>

**Public Sub ZNotify() Implements IParticipante.Notify**

If \_ObjMediador Is Nothing Then

Throw New ArgumentNullException( \_

"Problema [Notify]: Objeto [" & Me.Name & \_

"]: El MEDIADOR tiene el valor 'Nothing'.")

End If

\_ObjMediador.Notify(Me)

End Sub

''' <summary>

''' Proceso de registro en el Mediador

''' Verifica la presencia del mediador. Si existe,

''' llama al método [Mediador].Register(),

''' pasando la referencia de este objeto.

''' </summary>

**Public Sub ZRegister() Implements IParticipante.Register**

If \_ObjMediador Is Nothing Then

Throw New ArgumentNullException( \_

"Problema [Register]: Objeto [" & Me.Name & \_

"]: El MEDIADOR tiene el valor 'Nothing'.")

End If

\_ObjMediador.Register(Me)

End Sub

''' <summary>

''' Proceso de borrado del registro en el Mediador

''' Verifica la presencia del mediador. Si existe,

''' llama al método [Mediador].UnRegister(),

''' pasando la referencia de este objeto.

''' </summary>

**Public Sub ZUnRegister() Implements IParticipante.UnRegister**

If \_ObjMediador Is Nothing Then

Throw New ArgumentNullException( \_

"Problema [UnRegister]: Objeto [" & Me.Name & \_

"]: El MEDIADOR tiene el valor 'Nothing'.")

End If

\_ObjMediador.UnRegister(Me)

End Sub

#End Region

#Region "ErrorProvider (si existe)"

''' <summary>

''' Establece la cadena de descripción del error para este control

''' </summary>

''' <remarks>

''' <para>

''' Verifica la presencia del [Mediador]. Si existe,

''' llama al método [Mediador].SetError(Me, mensaje)

''' pasando la referencia de este objeto y el mensaje.

''' </para>

''' <para>

''' El [Mediador] verifica la presencia del objeto [ErrorProvider].

''' + Si existe, llama al método [ErrorProvider].SetError().

''' + Si no existe no se hace nada

''' </para>

''' </remarks>

**Public Sub ZSetError(ByVal mensaje As String) Implements IParticipante.SetError**

If \_ObjMediador Is Nothing Then

Throw New ArgumentNullException( \_

"Problema [ZSetError]: Objeto [" & Me.Name & \_

"]: El MEDIADOR tiene el valor 'Nothing'.")

End If

\_ObjMediador.SetError(Me, mensaje)

End Sub

#End Region

#End Region

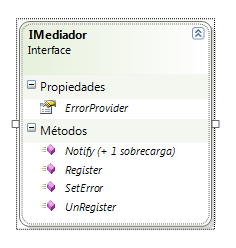
End Class

## Escribir el objeto mediador

Llega el momento de escribir el objeto mediador, en este punto se nos presenta un problema, si utilizamos la clase mediador en nuestros formularios, estamos ligando el formulario a la clase mediadora, y como eso no nos interesa, definimos una interfaz que será la que utilicemos en nuestro formulario

### Interfaz IMediador

La interfaz tendrá la siguiente estructura



''' <summary>

''' Interface que deben implementar las clases mediador

''' </summary>

''' <remarks>

''' Patron de Comportamiento Mediador

''' Utilizo una interfaz para que el control pueda ser

''' usado por varios mediadores

''' por ejemplo en dos formularios diferentes

''' <example>

''' http://msdn.microsoft.com/es-es/library/cd43d244.aspx

''' </example>

''' </remarks>

**Public Interface IMediador**

''' <summary>

''' Registra el control para que el

''' 'objeto Mediador' pueda usarlo

''' </summary>

''' <param name="participante">

''' El Objeto que se va a 'Register', recibido a

''' través de un objeto interfaz [IParticipante]

''' </param>

''' <remarks>

''' Para que el [Mediador] pueda usar un objeto tiene

''' que tener su referencia guardada en algún sitio,

''' ese proceso de guardar la referencia del objeto

''' en el [mediador] es lo que denominamos 'registrar'

'''</remarks>

**Sub Register(ByVal participante As IParticipante)**

''' <summary>

''' Borra el control del registro del Mediador

''' </summary>

''' <param name="participante">

''' El Objeto que se va a 'UnRegister', recibido a

''' través de un objeto interfaz [IParticipante]

''' </param>

''' <remarks>

''' Si el objeto se elimina de la lista de objetos

''' registrados, el [Mediador] no podrá usarlo

''' </remarks>

**Sub UnRegister(ByVal participante As IParticipante)**

''' <summary>

''' Esta es la función que maneja la información de como

''' se relacionan los objetos (los participantes) entre si

''' </summary>

''' <param name="participante">

''' El objeto participante que llama a esta función

''' Recibido a través de un objeto interfaz [IParticipante]

''' </param>

**Overloads Sub Notify(ByVal participante As IParticipante)**

''' <summary>

''' Esta es la función que maneja la información de como

''' se relacionan los objetos (los participantes) entre si

''' </summary>

''' <param name="objeto">

''' El objeto participante que llama a esta función

''' Recibido a través de un objeto interfaz [IParticipante]

''' </param>

''' <remarks>

''' Es la que realmente hace todo el trabajo

''' </remarks>

**Overloads Sub Notify(ByVal objeto As System.Object)**

''' <summary>

''' Guardar una referencia del objeto ErrorProvider

''' la idea es que el objeto pueda llamarlo

''' Puede tener el valor Nothing

''' </summary>

**Property ErrorProvider() As ErrorProvider**

''' <summary>

''' Establece la cadena de descripcion del error para esta control

''' Verifica la presencia del objeto ErrorProvider.

''' Si existe, llama al método [ErrorProvider].SetError(),

''' pasando la referencia del participante y el mensaje.

''' </summary>

''' <param name="participante">

''' El objeto participante que llama a esta función

''' Recibido a través de un objeto interfaz [IParticipante]

''' </param>

''' <param name="mensaje">

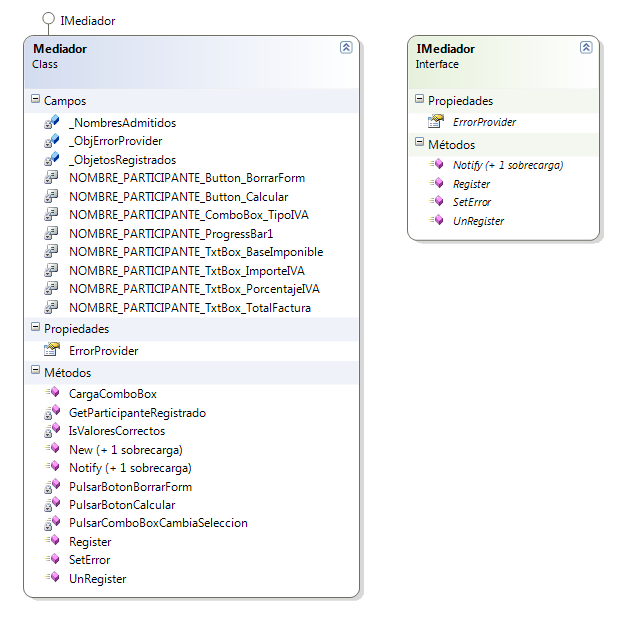
''' El mensaje que se envia al control ErrorProvider</param>

**Sub SetError(ByVal participante As IParticipante, ByVal mensaje As String)**

End Interface

### La case Mediador

La clase mediador tiene el siguiente esquema



#### Las variables de la clase

#Region "Variables de la clase"

'-----------------------------------------------------

' Contiene los nombres de los controles participantes

' Es una lista cerrada, si no esta aquí no se puede registrar

' Observa la secuencia de formación de los nombres de las constantes

'-----------------------------------------------------

Private Const NOMBRE\_PARTICIPANTE\_ComboBox\_TipoIVA As String = \_

"UcComboBoxParticipanteTipoIVA"

Private Const NOMBRE\_PARTICIPANTE\_TxtBox\_BaseImponible As String = \_

"UcTextBoxParticipanteBaseImponible"

Private Const NOMBRE\_PARTICIPANTE\_TxtBox\_PorcentajeIVA As String = \_

"UcTextBoxParticipantePorcentajeIVA"

Private Const NOMBRE\_PARTICIPANTE\_TxtBox\_ImporteIVA As String = \_

"UcTextBoxParticipanteImporteIVA"

Private Const NOMBRE\_PARTICIPANTE\_TxtBox\_TotalFactura As String = \_

"UcTextBoxParticipanteTotalFactura"

Private Const NOMBRE\_PARTICIPANTE\_ProgressBar1 As String =

"UcProgressBarParticipante1"

Private Const NOMBRE\_PARTICIPANTE\_Button\_Calcular As String = \_

"UcButtonParticipanteCalcular"

Private Const NOMBRE\_PARTICIPANTE\_Button\_BorrarForm As String = \_

"UcButtonParticipanteBorrarForm"

' ¡¡¡OBSERVACION !!!

' si añades una variable aqui, acuerdate de añadirla tambien en el constructor

'

'--------------------------------------------------------

''' <summary>

''' Lista de los controles que se pueden registrar

''' </summary>

''' <remarks>

''' <para>

''' Contendrá la lista de los nombres que hay en las

''' constantes, es decir la lista de los controles

''' que se pueden registrar

''' </para>

''' <para>

''' El objeto es que no se me cuele un nombre de

''' control que no corresponda exactamente con el

''' nombre registrado en la constante, que debe

''' ser el nombre del control en el formulario,

''' el valor de la propiedad [name]

''' </para>

''' <para>

''' Ese problema da lugar a errores extraños, sobre

''' todo cuando se cambia un nombre de un control y

''' no te acuerdas de cambiarlo en el mediador

''' </para>

'''</remarks>

Private \_NombresAdmitidos As List(Of String) = Nothing

'-----------------------------------------------------

''' <summary>

''' Contiene la lista de objetos registrados

''' </summary>

Private \_ObjetosRegistrados As Dictionary(Of String, IParticipante) = Nothing

#End Region

Observa que por cada control participante que hay en el formulario hay una variable interna que contendrá su referencia, y además, hay una constante que contiene el nombre que el control participante tiene en el formulario concretamente en su propiedad [Name]. De esta forma, puedo identificarlos fácilmente.

### El Constructor

''' <summary>

''' Constructor estandard

''' </summary>

''' <remarks>

''' Instanciar la lista (genérica) de controles participantes

''' Instanciar y cargar la lista (genérica) de nombres de controles permitidos

''' </remarks>

Public Sub New()

' Instanciar la lista (genérica) de controles

If \_ObjetosRegistrados Is Nothing Then

\_ObjetosRegistrados = New Dictionary(Of String, IParticipante)

End If

' Instanciar y cargar la lista (genérica) de nombres

' de controles permitidos

If \_NombresAdmitidos Is Nothing Then

\_NombresAdmitidos = New List(Of String)

' dar de alta los nombres de controles permitidos

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_ComboBox\_TipoIVA)

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_TxtBox\_BaseImponible)

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_TxtBox\_PorcentajeIVA)

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_TxtBox\_ImporteIVA)

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_TxtBox\_TotalFactura)

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_Button\_Calcular)

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_Button\_BorrarForm)

\_NombresAdmitidos.Add(NOMBRE\_PARTICIPANTE\_ProgressBar1)

End If

End Sub

### Función Register

El concepto de registrar implica que el [Mediador] conozca a sus [Participantes], y para ello, el [Mediador] guarda de alguna forma las referencias de los [Participantes]

La función que registra los controles lo primero que hace es localizar el nombre del control (propiedad [Name]). A continuación mira a ver si es uno de los controles permitidos para ser registrados y por ultimo añade la referencia a la colección siempre y cuando no exista previamente, es decir un participante solo se registra una vez

''' <summary>

''' Registra un participante

''' </summary>

''' <param name="participante">

''' El Objeto que se va a 'Register', recibido a

''' través de un objeto interfaz [IParticipante]

''' </param>

''' <remarks>

''' Para que el [mediador] pueda usar un objeto tiene

''' que tener su referencia guardada en algún sitio,

''' ese proceso de guardar la referencia del objeto

''' en el [mediador] es lo que denominamos 'registrar'

'''</remarks>

**Public Sub Register(ByVal participante As IParticipante) \_**

**Implements IMediador.Register**

' La clave de acceso a este objeto en

' la colección [\_ObjetosRegistrados]

Dim clave As String = participante.Name

' Primero ver si el control puede registrarse

' ver si está en la lista [\_NombresAdmitidos]

If \_NombresAdmitidos.Contains(clave) = False Then

Throw New ArgumentOutOfRangeException( \_

"Register - Ese control [" & clave & \_

"] no puede registrarse porque no esta " & \_

"en la lista de controles registrables")

End If

' Segundo registrar el objeto

' Solo se registra una vez

If \_ObjetosRegistrados.ContainsKey(clave) = False Then

\_ObjetosRegistrados.Add(clave, participante)

End If

End Sub

### Función UnRegister

''' <summary>

**''' Borrar del registro un participante**

''' </summary>

''' <param name="participante">

''' El Objeto que se va a 'UnRegister', recibido a

''' través de un objeto interfaz [IParticipante]

''' </param>

''' <remarks>

''' Si el objeto se elimina de la lista de objetos

''' registrados, el [Mediador] no podrá usarlo

''' </remarks>

**Public Sub UnRegister(ByVal participante As IParticipante) \_**

**Implements IMediador.UnRegister**

' si esta registrado se borra del registro

' si no esta registrado no se hace nada

If \_ObjetosRegistrados.ContainsKey(participante.Name) = True Then

\_ObjetosRegistrados.Remove(participante.Name)

End If

End Sub

### Función Notify

Esta función es la que averigua que participante es el que ha llamado y una vez averiguado lanza la función que ‘encapsula’ las operaciones que hay que realizar para ese participante especifico

''' <summary>

''' Esta es la función que identifica que participante

''' es el que ha llamado

''' Tambien sabe cual es el proceso a realizar segun

''' el participante que llame

''' </summary>

''' <param name="objeto">

''' El objeto participante que llama a esta función

''' Recibido a través de un objeto System.Object

''' </param>

''' <remarks>

''' Maneja la información de como se relacionan

''' los objetos (los participantes) entre si

''' </remarks>

**Public Overloads Sub Notify(ByVal objeto As System.Object) \_**

**Implements IMediador.Notify**

'----------------------------------------------------

' Es la que realmente hace todo el trabajo

'----------------------------------------------------

Dim objControl As System.Windows.Forms.Control = Nothing

objControl = CType(objeto, System.Windows.Forms.Control)

Dim clave As String = objControl.Name

Try

If \_ObjetosRegistrados.ContainsKey(clave) = True Then

Select Case objControl.Name

Case Is = NOMBRE\_PARTICIPANTE\_ComboBox\_TipoIVA

Call CambiaSeleccionComboBox()

Case Is = NOMBRE\_PARTICIPANTE\_TxtBox\_BaseImponible

' no hacer nada

Case Is = NOMBRE\_PARTICIPANTE\_TxtBox\_PorcentajeIVA

' no hacer nada

Case Is = NOMBRE\_PARTICIPANTE\_TxtBox\_ImporteIVA

' no hacer nada

Case Is = NOMBRE\_PARTICIPANTE\_TxtBox\_TotalFactura

' no hacer nada

Case Is = NOMBRE\_PARTICIPANTE\_ProgressBar1

' no hacer nada

Case Is = NOMBRE\_PARTICIPANTE\_Button\_Calcular

Call PulsarBotonCalcular()

Case Is = NOMBRE\_PARTICIPANTE\_Button\_BorrarForm

Call PulsarBotonBorrarForm()

Case Else

' este error solo puede darse si hay algún

' control registrado que no esta en la lista

' de [cases] del [Select Case]

Throw New ArgumentOutOfRangeException( \_

"Notify - Ese control [" & clave & \_

"] esta registrado pero no esta en la lista de cases")

End Select

Else

Throw New ArgumentOutOfRangeException( \_

"Notify - Ese control [" & clave & "] no esta registrado")

End If

Catch ex As Exception

Throw

End Try

End Sub

### Relaciones entre controles

Las tareas que hay que realizar cuando se llama un participante las he recogido en funciones, porque así es más fácil su mantenimiento y el código queda más limpio, a modo de ejemplo vamos a ver la función que encapsula las tareas que hay que hacer cundo se pulsas el botón [Calcular]

'

''' <summary>

''' Proceso a seguir cuando llama el botón Calcular

''' </summary>

''' <remarks>

''' <para>En primer lugar se busca el participante

''' [TxtBoxBaseImponible] y se comprueba que exista un valor

''' y que sea numérico. Si no es así se emite un mensaje a

''' través del ErrorProvider (con la función SetError de la

''' interfaz del participante) y se termina la ejecución

''' de la función</para>

''' <para>Localizar al participante [TxtBoxPorcentajeIVA] y

''' realizar las mismas operaciones y recuperar su

''' valor numérico</para>

''' <para> Localizar [TxtBoxImporteIVA] y situar en el mismo el

''' importe del IVA que se calcula en la función</para>

''' <para>Localizar [TxtBoxTotalFactura] y situar el importe

''' total de la futura que se calcula en la función</para>

''' <para>Utilizar un bucle para simular una operación larga y

''' activar con su valor un control [TxtBoxProgressBar1] </para>

'''</remarks>

**Private Sub PulsarBotonCalcular()**

'-----------------------------------------------------------

'La función IsValoresCorrectos

'-----------------------------------------------------------

'Es un proceso independiente que mira a ver si los

'valores de los participantes son correctos antes

'de realizar cualquier tipo de cálculo

'Devuelve Un valor lógico que indica:

'TRUE: Si los valores son correctos

'FALSE: NO los valores no son buenos, porque o bien

' hay letras en lugar de numeroso o porque

' no hay nada los campos, (están vacios)

'----------------------------------------------------------

If IsValoresCorrectos() = False Then Exit Sub

'------------------------------------------------------------------

' La interfaz que contendrá el objeto

Dim objInterfazParticipante As IParticipante = Nothing

'------------------------------------------------------------------

' La función [GetParticipanteRegistrado]

'------------------------------------------------------------------

' Busca un objeto registrado en la lista

' [\_ObjetosRegistrados] usando su clave (su nombre)

' Si lo encuentra devuelve un objeto interfaz con el valor pedido

' Si no está dispara un error porque se supone que en este

' punto del código todos los objetos con los que trabajo deben

' estar(registrados)

'------------------------------------------------------------------

**' Recuperar el objeto BaseImponible**

objInterfazParticipante = GetParticipanteRegistrado( \_

NOMBRE\_PARTICIPANTE\_TxtBox\_BaseImponible)

'Recuperar el valor numérico

Dim baseImponible As Double = 0

If Double.TryParse( \_

objInterfazParticipante.Text, baseImponible) = False Then

objInterfazParticipante.SetError("No es un número")

Exit Sub

End If

' escribir el número perfectamente formateado

objInterfazParticipante.Text = baseImponible.ToString( \_

"N", System.Globalization.CultureInfo.CurrentCulture)

''-------------------------------------

'' Apunte táctico

'' ejemplo de la forma de recuperar el objeto para

'' acceder a otras propiedades del mismo

''-------------------------------------

Dim objTextBoxParticipante As UCTextBoxParticipante

objTextBoxParticipante = \_

CType(objInterfazParticipante, UCTextBoxParticipante)

'Recuperar el valor numérico

If Double.TryParse(objTextBoxParticipante.Text, baseImponible) = False Then

objTextBoxParticipante.ZSetError("No es un número")

Exit Sub

End If

''-------------------------------------

'' / FIN Apunte táctico

''-------------------------------------

'------------------------------------------------------------------

**' Recuperar el objeto PorcentajeIVA**

objInterfazParticipante = GetParticipanteRegistrado( \_

NOMBRE\_PARTICIPANTE\_TxtBox\_PorcentajeIVA)

'Recuperar el valor numérico

Dim porcentajeIva As Double = 0

If Double.TryParse( \_

objInterfazParticipante.Text, porcentajeIva) = False Then

objInterfazParticipante.SetError("No es un número")

Exit Sub

End If

objInterfazParticipante.Text = porcentajeIva.ToString( \_

"N", System.Globalization.CultureInfo.CurrentCulture)

'------------------------------------------------------------------

**' recuperar el objeto ImporteIVA**

objInterfazParticipante = GetParticipanteRegistrado( \_

NOMBRE\_PARTICIPANTE\_TxtBox\_ImporteIVA)

Dim importeIva As Double = 0

importeIva = (baseImponible \* porcentajeIva) / 100

objInterfazParticipante.Text = importeIva.ToString( \_

"N", System.Globalization.CultureInfo.CurrentCulture)

'------------------------------------------------------------------

**' recuperar el objeto TotalFactura**

objInterfazParticipante = GetParticipanteRegistrado( \_

NOMBRE\_PARTICIPANTE\_TxtBox\_TotalFactura)

Dim totalFactura As Double = 0

totalFactura = baseImponible + importeIva

objInterfazParticipante.Text = totalFactura.ToString( \_

"N", System.Globalization.CultureInfo.CurrentCulture)

'------------------------------------------------------------------

**' Recuperar el objeto ProgressBar**

objInterfazParticipante = GetParticipanteRegistrado( \_

NOMBRE\_PARTICIPANTE\_ProgressBar1)

' y simular una operación larga

' Observación, se instancia el objeto para acceder

' a sus propiedades [Maximum], [Maximum] y [Value]

Dim objProgreesBar As ProgressBar

objProgreesBar = CType(objInterfazParticipante, ProgressBar)

objProgreesBar.Minimum = 0

objProgreesBar.Maximum = 2000

For i As Integer = objProgreesBar.Minimum To objProgreesBar.Maximum

objProgreesBar.Value = i

Next

' borrar el progressBar

objProgreesBar.Value = objProgreesBar.Minimum

End Sub

## El código en el formulario

Además de todo lo que hemos escrito (los controles (participantes) y la clase mediador) también necesitamos escribir código en el formulario, es allí donde se instancia el mediador, y donde se instancian los controles participantes, y es allí donde tenemos que cumplir con las normas de funcionamiento básicas del patrón, es decir, el mediador tiene que conocer a los participantes y los participantes tiene que conocer a su mediador, el mejor sitio para poner el código es evento Load del formulario

### El Evento Load

**Private Sub Form\_Load( \_**

ByVal sender As System.Object, \_

ByVal e As System.EventArgs) \_

Handles MyBase.Load

' activar el objeto mediador

Call ConfigurarMediador()

End Sub

'------------------------------------------------------

' El Objeto Mediador que maneja los controles

' (Los participantes) de este formulario

Private objInterfacMediador As IMediador = Nothing

''' <summary>

''' Define y carga el objeto Mediador cOn los participantes

''' </summary>

''' <remarks></remarks>

**Public Sub ConfigurarMediador()**

'------------------------------------------------------

' Aqui se realizan las siguientes tareas

' + Se define e instancia el objeto mediador

' + Los participantes deben conocer a su mediador

' + Los participantes deben conocer al ErrorProvider (si existe)

' + El mediador debe conocer a los participantes con los que media

' OBSERVA que se trabaja con la INTERFACE del objeto mdediador

'------------------------------------------------------

Try

'--------------------------

**' Crear una instancia de la clase mediador**

Dim objMediador As New Mediador

'--------------------------

**' Asignar a la interface la instancia de la clase**

objInterfacMediador = objMediador

'--------------------------

**' los participantes deben conocer a su mediador**

Me.UcComboBoxParticipanteTipoIVA.ZMediador = objInterfacMediador

Me.UcTextBoxParticipanteBaseImponible.ZMediador = objInterfacMediador

Me.UcTextBoxParticipantePorcentajeIVA.ZMediador = objInterfacMediador

Me.UcTextBoxParticipanteImporteIVA.ZMediador = objInterfacMediador

Me.UcTextBoxParticipanteTotalFactura.ZMediador = objInterfacMediador

Me.UcButtonParticipanteCalcular.ZMediador = objInterfacMediador

Me.UcButtonParticipanteBorrarForm.ZMediador = objInterfacMediador

Me.UcProgressBarParticipante1.ZMediador = objInterfacMediador

'--------------------------

' El mediador debe conocer al ErrorProvider (si existe)

' para que los participantes puedan usarlo

objInterfacMediador.ErrorProvider = Me.ErrorProvider1

'--------------------------

**' El mediador debe conocer a los participantes con los que media**

' dos formas de hacerlo

' UNA los participantes se registran en el mediador

' (una vez conocido el mediador)

' Ejemplo --> Me.UcTextBoxParticipanteBaseImponible.ZRegister()

' DOS el mediador registra a los participantes

objMediador.Register(Me.UcTextBoxParticipanteBaseImponible)

objMediador.Register(Me.UcComboBoxParticipanteTipoIVA)

objMediador.Register(Me.UcTextBoxParticipantePorcentajeIVA)

objMediador.Register(Me.UcTextBoxParticipanteImporteIVA)

objMediador.Register(Me.UcTextBoxParticipanteTotalFactura)

objMediador.Register(Me.UcButtonParticipanteCalcular)

objMediador.Register(Me.UcButtonParticipanteBorrarForm)

objMediador.Register(Me.UcProgressBarParticipante1)

'--------------------------

'' Cargar el ComboBox con datos. Se hace en el objeto

'' mediador para mantener la consistencia de datos

objMediador.CargaComboBox()

Catch ex As Exception

Throw

End Try

End Sub

### Usar los eventos del formulario

Lo último que nos queda es usar todo esto, por ejemplo,

* El evento ComboBox.SelectedIndexChanged que se produce cuando cambia el texto seleccionado en el combo Box, el evento quedará así

Private Sub UcComboBoxParticipanteTipoIVA\_SelectedIndexChanged( \_

ByVal sender As System.Object, ByVal e As System.EventArgs) \_

Handles **UcComboBoxParticipanteTipoIVA.SelectedIndexChanged**

objInterfacMediador.Notify(sender)

End Sub

* Evento Click de los botones, Con un solo controlador de eventos manejamos dos botones [BorrarFormulario], y [Calcular] este evento quedará así:

Private Sub UcButtonParticipanteBorrarForm\_Click( \_

ByVal sender As System.Object, ByVal e As System.EventArgs) \_

Handles **UcButtonParticipanteBorrarForm.Click, \_**

**UcButtonParticipanteCalcular.Click**

objInterfacMediador.Notify(sender)

End Sub

* El evento del botón terminar quedará de la forma tradicional

Private Sub ButtonTerminar\_Click( \_

ByVal sender As System.Object, ByVal e As System.EventArgs) \_

Handles **ButtonTerminar.Click**

Me.Close()

End Sub

## El problema de los menús

Existe un problema más tonto que es aquel en el que una misma acción esta en un botón y en una opción de menú. Por ejemplo, el botón [Terminar] realiza la misma operación que la opción de menú [Archivo.Terminar].

Este problema se resuelve creando una función por ejemplo [AccionBotonTerminar()] que contenga todas las operaciones que hay que hacer para terminar la aplicación y esta función se llama desde los sitios que haga falta, el botón , el menú, etc.

El problema surge cuando al acción que ejecuta el botón esta manejada por el [mediador] en este caso el [menú] no puede llamar al mediador porque entre otras cosas no está registrado y el mediador no sabe qué hacer con el. La forma en la que lo he resuelto ha sido creando una función OnClick() para simular el evento OnClick, del que carecen los botones (curiosamente) de esta forma, en el menú solo tenemos que llamar al evento OnClick del botón para que se simule su pulsación

Al haber habido una pulsación de un botón el mediador es el que se ocupa de realizar las tareas asociadas a ese botón.

**Código a incluir en el control Botón participante [UcButtonParticipanteCalcular]**

''' <summary>

''' Simula la pulsación del botón

''' </summary>

**Protected Overloads Overrides Sub OnClick(ByVal e As EventArgs)**

MyBase.OnClick(e)

End Sub

''' <summary>

''' Simula la pulsación del botón

''' </summary>

''' <remarks>

''' Esta propiedad no la tienen los botones,

''' y se la añado porque me resulta util

''' </remarks>

**Public Overloads Sub OnClick()**

MyBase.OnClick(New EventArgs)

End Sub

**Código de la opción de menú**

Private Sub CalcularToolStripMenuItem\_Click( \_

ByVal sender As System.Object, ByVal e As System.EventArgs) \_

**Handles CalcularToolStripMenuItem.Click**

' Simular la pulsación del botón [Calcular]

UcButtonParticipanteCalcular.OnClick()

End Sub

## Consideraciones finales

Uno se fija en todo lo que hemos escrito y se pregunta… ¿realmente merece la pena todo el esfuerzo?

La respuesta es, como siempre, ambigua, por una parte, los controles una vez creados, ahí están, no hay que volver a trabajar para usarlos, simplemente se usan y listo por lo que esa parte del esfuerzo es reutilizable, pero la realidad es que la clase ‘mediador’ es específica para el formulario, y esa que es la complicada no es reutilizable, y es aunque se admiten opiniones, yo creo que es más complicada que escribir las relaciones entre controles dentro del código del formulario

En teoría (y siempre hablando del ejemplo del formulario) hemos ganado flexibilidad porque toda la interacción entre los objetos del formulario esta fuera del formulario,

En resumen, mi opinión personal es que en .NET y específicamente para el tema de la aplicación del patrón Mediador para manejar los controles de un formulario, parece que no tiene mucho sentido, aunque en otros lenguajes puede que si lo tenga. Evidentemente, en otros supuestos este patrón sigue siendo útil y su aplicación puede reportar beneficios importantes, por ejemplo en un proceso de distribución de mensajes

Por último y no menos importante, es algo ¡¡chulo!! ¿No? Y además ¡¡funciona!!

## Bibliografía

### Sobre el patrón Mediador

* <http://www.portalfox.com/index.php?name=Sections&req=viewarticle&artid=68>
* <http://www.portalfox.com/index.php?name=Sections&req=viewarticle&artid=159>
* <http://pages.cpsc.ucalgary.ca/~heatond/mediator/>

* <http://stevenblack.com/PTN-Mediator.ASP>..

* <http://www.dofactory.com/patterns/PatternMediator.aspx>

* <http://www.vincehuston.org/dp/mediator.html>
* <http://sourcemaking.com/design_patterns/mediator>
* <http://www.blackwasp.co.uk/Mediator.aspx>
* <http://www.patterndepot.com/put/8/mediator.pdf>
* <http://www.java2s.com/Code/Java/Design-Pattern/MediatorPattern2.htm>
* <http://www.as3dp.com/2008/07/07/the-mediator-design-pattern-a-minimalist-example/>
* <http://sourcemaking.com/design_patterns/mediator>

### General sobre Patrones

* <http://blogs.vbcity.com/jspano/category/44.aspx>
* <http://www.oodesign.com/>
* <http://patterns.cs.up.ac.za/examples.shtml#ch2>
* <http://www.vbdotnetheaven.com/Code/Jun2003/2014.asp> Abstract Factory Pattern in VB .Net
* <http://blogs.vbcity.com/jspano/articles/198.aspx>. Implementing the Command Pattern in .Net
* <http://blogs.vbcity.com/jspano/articles/199.aspx>. Implementing the Visitor Design Pattern In .Net

(Página en blanco)