Moonlight: ¿una alternativa a Silverlight?

La luz de la luna también es plateada

El tema Silverlight recientemente se ha puesto caliente por las declaraciones de Bob Muglia en la pasada PDC (ver el artículo ilustrativo que publicó dotNetManía en el número del mes pasado [1])[1]. El asunto es si los que nos enamoramos de Silverlight por sus propias bondades (alta productividad de desarrollo entre ellas) podemos contar con algo similar para los sistemas y dispositivos para los que no se disponga de Silverlight. ¿Es Moonlight una alternativa? ¿Moonlight ejecuta en Linux todo lo que Silverlight ejecuta en Windows? Con este artículo pretendemos ofrecer un primer acercamiento a este tema.

¿Cuándo salió la luna? Un poco de historia

"No saber lo que ha ocurrido antes de nosotros es como seguir siendo niños"

Cicerón (escritor, orador y político romano, 106 AC-43 AC)

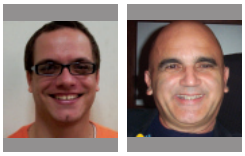
Miguel de Icaza, líder del proyecto Mono [2], dio a conocer allá por 2007, que presentaría una versión alpha de Moonlight a mediados de ese año [3]. Luego de un intenso trabajo por parte del equipo, la primera versión fue presentada precisamente en el ReMIX de Microsoft de París, el 21 de junio de 2007. Cuando se mostró la demo, los entusiastas de la tecnología .NET sentimos cada vez más posible de realizar el sueño de poder escribir todas las aplicaciones en .NET y que luego éstas pudiesen ejecutar en cualquier plataforma; "**write once, run everywhere**" de verdad.

Pero inicialmente todo no fue color de rosa. Para poder ejecutar en Moonlight una aplicación de Silverlight, había que de descargar de `http://ftp.novell.com/pub/mono/sources/moon` y compilar el código fuente de Moonlight, con el fin de probar cómo funcionaba la aplicación en nuestro escenario. Esto implicaba, por supuesto, disponer en Moonlight de todas las librerías necesarias, y enfrentar los frecuentes errores de ejecución de las aplicaciones por usar diferentes versiones.

Luego de esa presentación inicial en París, Microsoft empezó a cooperar con Novell en el proyecto, lo que le dio un gran impulso, ya que le proporcionaron a los desarrolladores de Moonlight tres aportes fundamentales:

- El paquete completo de testing de Silverlight.
- Los códecs para la reproducción de vídeo y audio para que pudiesen ser usados "sin cargo alguno" en Moonlight cuando se ejecutan en el navegador.
- Algunos de los detalles, especificaciones e interioridades de Silverlight que no estaban publicados en la web del producto.

El 9 de febrero de 2009 se lanza la versión 1.0 de Moonlight, el cual usaba el paquete de medios de Microsoft 1.0 para la reproducción de audio y vídeo, y que era compatible con Silverlight 1.0. Estaba implementado completamente en C/C++.



Camilo Abel Monreal y Miguel Katrib

Camilo Abel es desarrollador e instructor del grupo WEBOO. Dr. Katrib es líder del grupo WEBOO y profesor titular de Programación, ambos en la Universidad de La Habana.

[1] Usted puede ver también nuestro post al respecto, publicado en `www.weboomania.com`.

Posteriormente, en mayo del mismo año, cuando ya Microsoft había lanzado su Silverlight 2.0, aparece la versión alpha 2.0 de Moonlight, y más tarde en diciembre se lanza Moonlight 2.0. Esta versión, además de ser compatible con Silverlight 2.0, tenía algunas otras funcionalidades añadidas, como la capacidad de utilizar lenguajes que se ejecutan en el DLR (*Dynamic Language Runtime*), usar el paquete de medios de Microsoft 2.0 y ser completamente compatible con todos los controles Silverlight 2.0 MS-PL. Es bueno que algunos fundamentalistas conozcan el dato de que ya esta versión 2.0 contenía solo 142.000 líneas de código C/C++, en contraste con que contaba ya con 320.000 líneas de código C# (125.000 de las cuales provenían de los controles de Silverlight de Microsoft con código abierto).

Actualmente se está desarrollando la versión 3.0, de la cual en febrero de 2010 se lanzó la primera *preview* y que es una suerte de híbrido entre las versiones 3.0 y 4.0 de Silverlight. Esta versión ya deberá ofrecer una API completa. En la fecha en que se escribe este artículo, ya Moonlight tenía un 47.77% de ejecución satisfactoria de la *suite* de tests de Silverlight 4.

Pero ya se adelanta que la versión 4.0 deberá incluir todo el soporte para webcams y micrófonos, así como el soporte para los WCF RIA Services, entre otras características; además, seguramente en esta versión aparecerán algunas de las características que se incluyen en la versión 5.0 de Silverlight, si nos guiamos por el *roadmap* que han venido siguiendo ambos proyectos.

Moonlight 3.0 está en la actualidad disponible oficialmente como *plug-in* para Mozilla Firefox y Google Chrome. Miguel de Icaza comenta en su blog que tal vez antes de la salida de Moonlight 4.0 se publique una versión 3.5 con la mayor cantidad de características de Silverlight 4.0 que puedan incluir al inicio del año.

Alunizando

El proyecto Moonlight se puede dividir, a grandes rasgos, de la manera que se muestra en la figura 1. Los principales subsistemas que componen la arquitectura que se muestra en la figura son los siguientes:

- *User Code*. Es todo aquel código que incluimos o generamos en nuestras aplicaciones; desde el código HTML, JavaScript, CSS, controles creados en nuestra aplicación, librerías propias o de terceros, etc.
- *MS Code*. Es el conjunto de librerías de controles y funcionalidades de Silverlight. Aquí se usan los controles de Silverlight MS-PL. Se incluye también la implementación del DLR para la plataforma Mono, lo que permite ejecutar código de lenguajes dinámicos como IronPython, IronRuby, etc.; esta característica está presente desde Moonlight 2.0. El Media Codec Pack, por su tipo de licencia, no viene incluido de serie en Moonlight y se debe de descargar aparte del sitio de Microsoft. Este contiene todo lo relacionado con la reproducción de vídeo y audio (VC1, WMV, WMA, MP3, etc.).
- *Mono Code*. Aquí se encuentra todo lo desarrollado propiamente en el proyecto. Como se puede ver en la figura 1, consta de tres componentes fundamentales:
  - ***Core CLR***. Aquí se encuentra el subconjunto del *runtime* de Mono que se usa en Moonlight (en Moonlight 2.0 era un subconjunto de Mono 2.6, y en Moonlight 3.0 de Mono 2.9), la implementación de la API de Silverlight, controles, manejo de eventos, estructuras de datos, etc.
  - ***Mozilla, Chrome Plug-in***. Incluye todo lo relacionado con los *plug-ins* para los navegadores, como por ejemplo, la interacción con las API de cada uno.
  - ***Libmoon***. Aquí es donde se concentra todo lo relacionado con la interacción con el sistema operativo bajo el que se está ejecutando Moonlight. Es donde descansa todo lo relacionado con las llamadas a las funciones nativas donde se esté ejecutando la aplicación.

A partir de la versión 3.0, se creó **Moonlight Platform Abstraction Layer**, que es con la que se usa actualmente Moonlight en Android y OSX. Gracias a esta capa de abstracción, es mucho más fácil portar la plataforma a otros sistemas operativos.

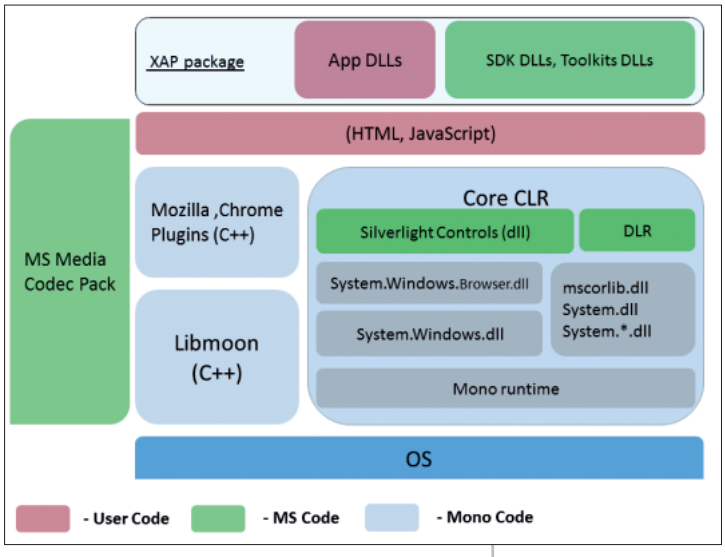


Figura 1. Arquitectura de Moonlight

Uno de los logros más significativos del proyecto Moonlight es lo que se ha logrado en el tema de la ace-

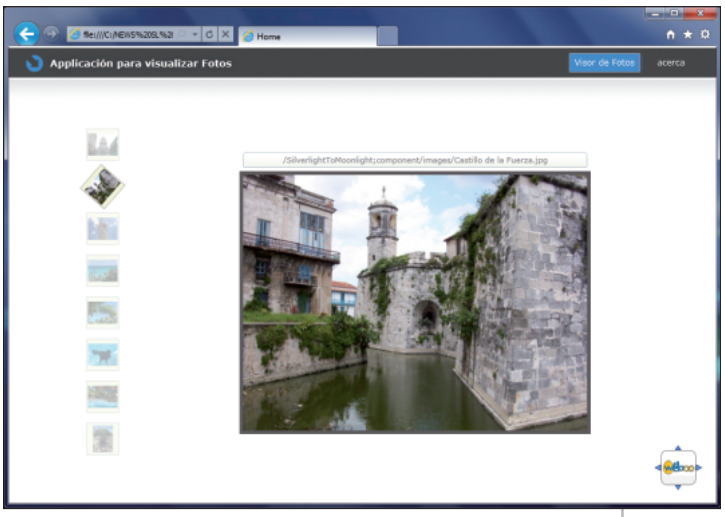


Figura 2. Visor de fotos en Silverlight

leración por hardware. Silverlight posee en estos momentos aceleración por hardware solamente para algunos *pixel shaders*; sin embargo, Moonlight 3.0 tiene aceleración por hardware para todos los *pixel shaders* [4]. Esto se debió fundamentalmente a que desde el inicio del proyecto, **David Reveman**, muy conocido por sus aportes en **Glitz** ([5], [6]), **Xgl** ([7]) y **Compiz** ([8]) trabajó en el diseño e implementación de esta funcionalidad[2].

No hay una "silver bullet" para la portabilidad

Como la mayoría de los lectores de **dotNetManía** seguramente están más familiarizados con Silverlight y Visual Studio, hagamos una demo en nuestro ambiente de desarrollo favorito y probémosla en Moonlight para ver qué pasa.

Desarrollemos una aplicación de visor de fotos. Para ello, creamos en Visual Studio un proyecto de tipo **Silverlight Navigation Application** para Silverlight 3.0. Agregamos todo el código que permite mostrar las imágenes, con las animaciones, etc., insertando dentro de los recursos de la aplicación una carpeta `images` con nuestras imágenes[3]. Cuando se ejecuta la aplicación en Windows, todo perfecto, como se muestra en la figura 2.

Lo que hace la aplicación es mostrar en un control `StackPanel` varios botones con las fotos. Estas se previsualizan de manera vertical a la izquierda, y cada vez que el cursor pasa por encima de una imagen, éstas muestran una animación, cambiando su tamaño y su opacidad. Cuando se selecciona una imagen haciendo clic sobre ella, entonces la imagen se muestra en el centro de la pantalla haciendo un efecto. Estas previsualizaciones se obtienen con el código del listado 1.

Un fragmento del código del ControlTemplate `TemplateImageBtn` se puede ver en el listado 2. En éste se ha eliminado todo lo relacionado con las animaciones y estados del botón.

```
private void CreatePreImages()
{
    var fotos = new string[] {"Capitolio.jpg", "Castillo de la Fuerza.jpg",
                "El Cristo de la Habana.jpg", "Fauna1.JPG", "Fauna2.JPG",
                "Fauna3.JPG", "Hotel.JPG", "John Lennon sentado en el Parque.jpg" };
    for (var i = 0; i < fotos.Length; i++)
    {
        var button = new Button
        {
            Template = (ControlTemplate)Application.Current.Resources["TemplateImageBtn"],
            Content = StrRelativeToComponent(string.Format("images/{0}", fotos[i]))
        };
        button.MouseEnter += BtnMouseEnter;
        button.MouseLeave += BtnMouseLeave;
        button.Click += BtnClick;
        ImageStackPanel.Children.Add(button);
    }
}

private static string StrRelativeToComponent(string resource)
{
    var name = Assembly.GetCallingAssembly().FullName;
    return (new Uri(String.Format("/{0};component/{1}",
        name.Substring(0, name.IndexOf(',')), resource),
        UriKind.RelativeOrAbsolute).ToString());
}
```

Listado 1. Código de creación de los botones de previsualización de las imágenes

[2] Vea "Hardware acceleration with Moonlight" (`http://www.youtube.com/watch?v=9gVSHaY0loY`) y "Moonlight 3D Perspective Support" (`http://www.youtube.com/watch?v=O53BSAruARE`).

[3] Por razones de espacio, el código completo no se incluye aquí, pero el lector podrá descargarlo del sitio web de dotNetManía.

Como se puede apreciar de los listados, el contenido de los botones es un texto de la forma: "**/SilverlightToMoonlight;component/images/#.jpg**";

```
<ControlTemplate x:Key="TemplateImageBtn" TargetType="Button" >
<Grid Width="50" Height="50" Margin="5,5,5,5" x:Name="grid" Opacity="0.3"
    RenderTransformOrigin="0.5,0.5">
    <!-- Aquí va el código del StateManager y los StoryBoard -->
    <Rectangle Margin="6,6,0,0" x:Name="rect" Fill="#00000000"/>
    <Rectangle Fill="Beige" Margin="2,2,2,2" Stroke="#4C2F2F2F"/>

<Image Margin="4,6,4,6" Source="{TemplateBinding Content}" IsHitTestVisible="False" />
</Grid>
</ControlTemplate>
```

Listado 2. Fragmento de código del ControlTemplate

de esta manera, en la línea del listado 2 que utiliza enlace a datos, el elemento **Source** de la imagen en el ControlTemplate **TemplateImageBtn** tomará el valor de ese texto.

Luego, cuando se hace clic en una previsualización, lo que se ejecuta es el código del listado 3. Este lo que hace no es más que guardar cuál fue el último botón en el que se hizo clic, cambiar el estado visual de los botones de la previsualización y luego, como en el ControlTemplate del botón **btnVisualPlayer** se hace igual que en los botones de las previsualizaciones, se tiene una imagen cuyo **Source** es un *binding* al contenido del control, como se ve en la línea correspondiente del listado 4. También hay un botón cuyo contenido es el del control; esto es lo que permite ver cuál es la imagen que se está mostrando.

Pero, ¡sorpresa!: cuando intentamos ejecutar con Moonlight 3.0 (usando la Preview 2.99.0.10), obtenemos el resultado que nos muestra la figura 3.

Evidentemente, hay algo con los *bindings* del **Source** de la imagen de **btnVisualPlayer** que no está funcionando bien, ya que en el control botón que tenemos encima de la imagen sí se muestra el texto con el camino al archivo de la imagen que supuestamente debería mostrarse.

Sin embargo, cambiando un poco la aplicación, podemos lograr que esto funcione. Para ello, eliminamos el ControlTemplate de **btnVisualPlayer** y modificamos **btnVisualPlayer** como se muestra en el listado 5.

Luego, en el código que maneja el clic del botón de previsualización se hacen otros cambios, como se muestra en el listado 6.

Esto puede parecer un clásico parche, pero es un ejemplo práctico que nos confirma que no podemos confiarnos al 100% de que no haya diferencias en lo que probemos con Silverlight en Windows y en Linux con Moonlight. La solución adoptada aquí no obedece a otra lógica que la de tratar de solucionar el problema de una manera más rudimentaria, porque ponemos directamente cuál es la imagen seleccionada en lugar de usar un recurso más evolucionado como es el **ControlTemplate**, perdiendo de este modo flexibilidad para hacer cambios posteriores.

```
private void BtnClick(object sender, RoutedEventArgs e)
{
    var btn = sender as Button;
    if (btn == null)
        return;
    if (btn == LastFocus)
        return;
    if (LastFocus != null)
        VisualStateManager.GoToState(LastFocus, "MouseLeave", true);
    LastFocus = btn;
    var txt = btn.Content.ToString();
    VisualStateManager.GoToState(btn, "Selected", true);
    DropImageStb.Begin();
    btnVisualPlayer.Content = txt;
}
```

Listado 3. ¿Qué ocurre cuando se hace clic en una previsualización?

```
<ControlTemplate x:Key="VisualPlayerTemplate" TargetType="Button">
<Grid>
    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
        <Button Margin="5" Content="{TemplateBinding Content}"  Background="#FFAABB72"/>

    <Grid Width="500" Height="375">
            <Rectangle Fill="#FF605A5A"/>
            <Image Source="{TemplateBinding Content}" IsHitTestVisible="False"/>
        </Grid>
    </StackPanel>
</Grid>
</ControlTemplate>
```

Listado 4. ControlTemplate VisulaPlayerTemplate

Pero cuando vemos la aplicación ejecutándose de manera correcta en Moonlight (figura 4), apreciamos que bien valió la molestia de detectar el fallo

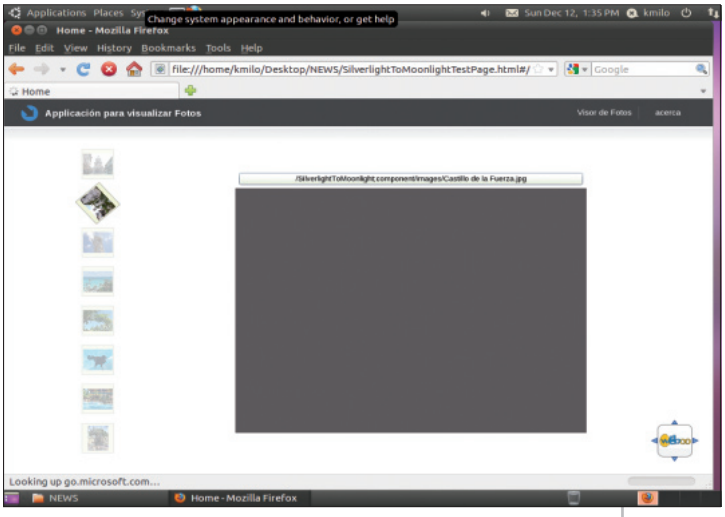


Figura 3. Bug de la aplicación Visor de fotos usando Moonlight, en Ubuntu 10.10

No solo la luz de la luna tiene sombras

Pero los *bugs* no son solo cosa de Moonlight; por ejemplo, el famoso problema del goteo de memoria (*memory leak*) en Silverlight 4 [9], ha provocado que varios productos desarrollados usando dicha versión hayan tenido que buscar alternativas para mitigarlo. Aún después de 9 meses de haberse detectado el problema y con todos los recursos de Microsoft, éste no se ha solucionado: vea el post del error, que es el más visitado del fórum (más de 7 millones de visitas en el momento en que escribimos este trabajo).

Desarrollar aplicaciones que valgan lo mismo para entornos Windows que para entornos Linux y para diferentes dispositivos, es un objetivo de muchas empresas que quieren atraer clientes de ambos segmentos o que están condicionadas por las exigencias de los diferentes clientes. Desarrollar en Silverlight teniendo en cuenta que los productos deberían también poder ejecutarse en Moonlight, tiene la ventaja de que habrán muchos problemas detectados posiblemente y ya resueltos, ya que por lo general el *roadmap* de Silverlight va más adelantado que el de Moonlight.

```
<Button x:Name="btnVisualPlayer" Grid.Column="1" RenderTransformOrigin="0.5,0.5"
    Margin="0,0,0,0" IsEnabled="False" Opacity="0" Background="#FF33495C">
    <Button.RenderTransform>
        <TransformGroup>
            <ScaleTransform/>
            <SkewTransform/>
            <RotateTransform/>
            <TranslateTransform/>
        </TransformGroup>
    </Button.RenderTransform>
    <Button.Content>
        <Grid>
            <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
                <Button Margin="5"  x:Name="txtVisualPlayer"   Background="#FFAABB72"/>
                <Grid Width="500" Height="375">
                    <Rectangle Fill="#FF605A5A"/>
                    <Image Margin="5" x:Name="ImgVisualPlayer" IsHitTestVisible="False" />
                </Grid>
            </StackPanel>
        </Grid>
    </Button.Content>
</Button>
```

Listado 5. Botón btnVisualPlayer modificado

> Uno de los logros más significativos del proyecto Moonlight es lo que se ha logrado en el tema de la aceleración por hardware

Una buena práctica, si se desarrolla teniendo en cuenta la compatibilidad de nuestros productos tanto con Moonlight como con Mono, es la de subscribirse a la lista de correos de los *bugs*. Para esto, visite **http://lists.ximian.com/mailman/listinfo/mono-bugs** (para subscribirse a la lista de Mono) y **http://lists.ximian.com/mailman/listinfo/moonlight-list** (para subscribirse a la lista de Moonlight). De esta manera, se y cambiar el código. Del lobo un pelo, porque hasta ahora ¿cuándo habíamos tenido una "portabilidad" similar?

mantendrá actualizado de los *bugs* encontrados hasta el

```
private void BtnClick(object sender, RoutedEventArgs e)
{
    var btn = sender as Button;
    if (btn == null)
        return;
    if (btn == LastFocus)
        return;
    if (LastFocus != null)
        VisualStateManager.GoToState(LastFocus, "MouseLeave", true);
    LastFocus = btn;
    var txt = btn.Content.ToString();
    VisualStateManager.GoToState(btn, "Selected", true);
    DropImageStb.Begin();
    btnVisualPlayer.Template = null;
    txtVisualPlayer.Content = txt;
    ImgVisualPlayer.Source = (ImageSource)ImageSourceConverter.ConvertFromString(txt);
}
```

Listado 6. Manejador del clic del botón

momento, así como de sus soluciones o alternativas.

Existen herramientas que permiten saber si el código que desarrollamos en MS .NET será compatible con .NET Mono; una de éstas es **MoMa** (lo que ya fue ilustrado en un artículo anterior que publicamos en esa revista [2]). Lamentablemente, no se dispone aún de una herramienta similar para Moonlight. Por lo tanto, de momento solo podemos echar mano a las "buenas prácticas" e ir desarrollando en Silverlight y paralelamente ir probando también en Moonlight. Las experiencias que vayamos adquiriendo las compartiremos con los lectores de **dotnetManía**.

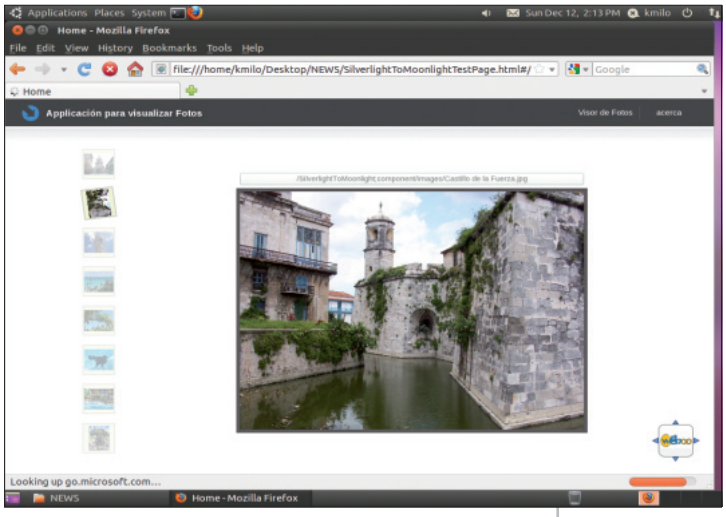


Figura 3. Bug de la aplicación Visor de fotos usando Moonlight, en Ubuntu 10.10

Referencias

[1] **Díez, Braulio** y **García, Reyes**. "El caso SilverGate". En **dotNetManía** nº 76, diciembre de 2010.

[2] **Tamayo, Alejandro** y **Katrib, Miguel**."Mono. Una implementación multiplataforma de .NET". En **dotNetManía** nº 73, septiembre de 2010.

[3] **Gardner, Dana**. "Expect a June demo of Silverlight on Linux, sans browser". En **http://fastforwardblog.com/2007/06/01/expect-a-june-demo-of-silverlight-on-linux-sans-browser/**.

[4] **de Icaza, Miguel**. "David Reveman lands GPU acceleration for Moonlight". En **http://tirania.org/blog/archive/2010/Nov-23.html**.

[5] **Glitz**. En **http://es.wikipedia.org/wiki/Glitz**.

[6] **Nillson, Peter** y **Reveman, David**. "Glitz: hardware accelerated image compositing using OpenGL". En *Proceedings of the Annual Technical Conference on USENIX* (ATEC '04). Berkeley, CA, USA, 2004.

[7] **Xgl**. En **http://es.opensuse.org/Xgl**.

[8] **Compiz**. En **http://www.compiz.org/**.

[9] Silverlight Forum. "Memory Leak Issue with Inline DataTemplates in XAML". En **http://forums.silverlight.net/forums/p/171739/389243.aspx**.

[10] Plataforma Web de Microsoft **http://http://www.microsoft.com/web**.