Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

19

CardSpace

Utilización con .NET 3.0

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Windows Presentation Foundation

La nueva forma de crear interfaces

Arquitectura de WPF - El lenguaje XAML

ISBN 978-987-1347-43-8

9 789871 347436 00019

RedUSERS

COMUNIDAD DE TECNOLOGIA

¡SUMATE!

EL SITIO Nº1 DE TECNOLOGIA

Noticias al instante // Entrevistas y coberturas exclusivas // Análisis y opinión de los máximos referentes // Reviews de productos // Trucos para mejorar la productividad // Registrate, participa, y comparte tus opiniones

SUSCRIBITE

SIN CARGO A CUALQUIERA DE NUESTROS **NEWSLETTERS** Y RECIBÍ EN TU CORREO ELECTRÓNICO TODA LA INFORMACIÓN DEL UNIVERSO TECNOLÓGICO ACTUALIZADA **AL INSTANTE**

INGRESÁ A
redusers.com/suscribirse-al-newsletter
¡Y REGISTRATE YA!

Foros · Encuestas · Tutoriales · Agenda de eventos · Videos · ¡Y mucho más!

redusers.com

Seguinos en:
www.facebook.com/redusers
www.twitter.com/redusers
www.youtube.com/redusersvideos

Aspectos importantes de CardSpace

Windows CardSpace abarca cuatro aspectos fundamentales, a los cuales nos referiremos a continuación.

Analicemos algunos de los aspectos más importantes de CardSpace:

***Apoyo a cualquier sistema de identidad**: Las múltiples identidades que usamos provienen de distintas fuentes y se expresan en diversas formas. En otras palabras, se basan en una serie de diferentes sistemas de identidades digitales, cada uno de las cuales puede usar otra tecnología subyacente. Tener en cuenta esta diversidad es útil para definir tres funciones distintas, que se describen en la Tabla 1. Teniendo en cuenta estas tres funciones, no es difícil entender cómo Windows CardSpace y el metasistema de identidad puede apoyar cualquier identidad digital. En la Figura 5, vemos la interacción que existe entre estas tres funciones. Como allí se indica, CardSpace es útil sólo si los proveedores de identidad y de las demás partes confían en los protocolos usados por el metasistema de identidad.

***Conciencia del usuario y control de identidad digital**: La existencia de protocolos estándar para la adquisición y la transmisión de fichas de seguridad es, sin duda, útil. Sin embargo, sin una forma para que los usuarios entiendan y tomen decisiones acertadas acerca de las identidades digitales, y lo que éstas representan, el sistema se derrumbaría fácilmente. En consecuencia, uno de los principales objetivos de CardSpace y del metasistema de identidad es capacitar a los usuarios de todo el mundo para tomar buenas decisiones acerca del uso de sus identidades digitales. Con este fin, CardSpace implementa una interfaz intuitiva de usuario para trabajar con identidades digitales. Cada tarjeta representa una identidad digital, mediante la cual el usuario puede presentar a la parte que confía, por ejemplo, un sitio web.

***Mejora de la confianza del usuario en la autenticación de aplicaciones web:** El hecho

Tabla 1 | Sistema de identidad

Usuario	También conocido como el sujeto, el usuario es la entidad que está asociada a una identidad digital. Los usuarios son a menudo las personas, pero las organizaciones, las aplicaciones, las máquinas y otras cosas también pueden tener identidades digitales.
Identidad	Un proveedor de identidad es justo lo que su nombre indica: algo que proporciona una identidad digital a un usuario. Un ejemplo puede ser una identidad digital asignada a nosotros por nuestro empleador; en este caso, el proveedor de identidad. Ésta usualmente consiste en un sistema como el de Active Directory de Windows Server.
Confianza	Una parte que confía a menudo utiliza una identidad (es decir, la información que figura en las reivindicaciones que conforman la identidad) para la autenticación de un usuario y, a continuación, tomar una decisión de autorización, tal como permitirle el acceso a cierta información.

de que los sitios web se independicen de la contraseña de inicio de sesión ayuda a reducir el problema del phishing, pero no lo elimina. Si un usuario es engañado en el acceso a la página del "phisher", el sitio podría aceptar cualquier token de seguridad que el usuario siempre envíe y, a continuación, pedir información confidencial, como el número de tarjeta de crédito. El phisher no adquirirá la contraseña del usuario para el sitio que está emulando, pero podría aprender otras cosas útiles. Para corregir este problema, se necesita:

- Mayor garantía de un sitio web para demostrar su identidad a los usuarios.
- Una manera de que los usuarios aprendan qué nivel de fiabilidad ofrece un sitio, como prueba de su identidad, y tomar una decisión explícita acerca de si confiar en ese sitio o no.

Para solucionar este tema, Microsoft está colaborando con otras empresas para crear un nuevo nivel de certificado.

Proceso de autenticación

Para explicar el proceso de autenticación, vamos a tomar como ejemplo el sitio web de una entidad bancaria, en el cual necesitamos autenticarnos para acceder al servicio de banca personal en línea. El proceso se inicia cuando ingresamos en el área de la banca personal; en este punto, la página verifica que estemos identificados. Para hacerlo, nos redirección a una página de inicio

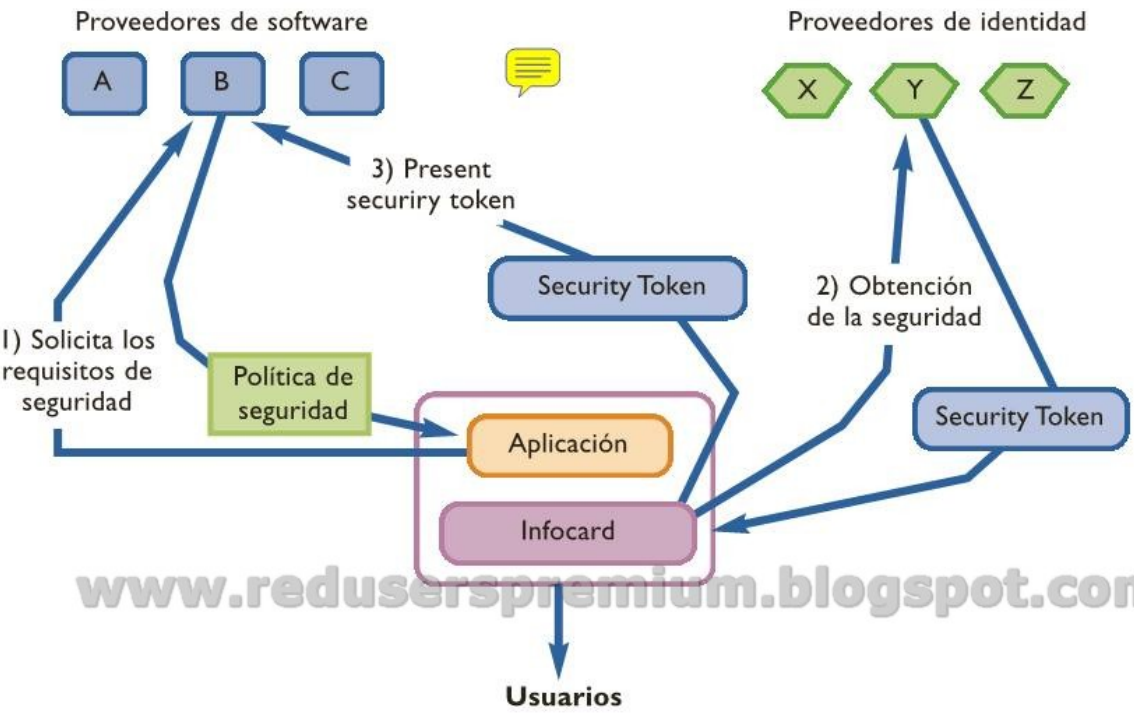

FIGURA 005 | Interacciones entre el usuario, el proveedor de identidad y la parte que confía.

de sesión, que contiene un formulario de inicio de sesión estándar, es decir, los clásicos cuadros de texto solicitando usuario y contraseña y, además, un objeto embebido mediante etiquetas XHTML que contiene la información que le indica a nuestro navegador que puede utilizar Windows CardSpace. En esta instancia, podemos seleccionar autenticarnos utilizando WCS, lo que hará que el navegador –en nuestro caso, Internet Explorer 7– requiera la implicación de Windows CardSpace en el proceso de inicio de sesión. Esto hará aparecer la pantalla de WCS, y entonces podremos seleccionar la identidad que vamos a utilizar con este sitio.

Hasta aquí, la comunicación entre el navegador y WCS se realiza a través del protocolo de transferencia de Internet HTTP, pero una vez que seleccionamos la identidad, WCS se comunica con el proveedor de identidad a través de WS-Trust (servicio web de confianza) y obtiene el token de seguridad. Luego, éste es enviado al sitio utilizando el método POST del protocolo http; es decir, como si hiciera el proceso de autenticación clásico. Esto puede resultar confuso, pero veamos con código lo que debería figurar en la página HTML:

```
...
<form id="form1" method="post" action=
"login1.aspx">
  <button type="submit">Haga clic aquí para
  autenticarse usando WCS</button>
  <object type="application/x-information
  card" name="xmlToken">
    <param name="tokenType" value="urn:oasis:
    names:tc:SAML:1.0:assertion" />
    <param name="issuer" value="http://schemas.
    xmlsoap.org/ws/2005/05/identity/issuer/
    self" />
    <param name="requiredClaims" value=
    "http://schemas.xmlsoap.org/ws/2005/05/
    identity/claims/givenname http://schemas.
    xmlsoap.org/ws/2005/05/identity/claims/
    surname http://schemas.xmlsoap.org/ws/
    2005/05/identity/claims/emailaddress
    http://schemas.xmlsoap.org/ws/2005/05/
    identity/claims/privatepersonal
    identifier" />
  </object>
</form>
...
```

Analicemos un poco el código. Dentro de un formulario insertamos el objeto embebido, que es del tipo "**application/x-informationcard**", y necesita que se le especifiquen tres parámetros para su correcta implementación: **tokenType**, **issuer** y **requiredClaims**. Estos parámetros se detallan en la Tabla 2. Además, el formulario enviará los datos de la tarjeta de identidad a la página **login1.aspx**. Para describir un poco más el ejemplo anterior, veamos cómo utilizar los datos que nos envía Windows CardSpace, para así validar un usuario. WCS nos manda la información encriptada, y para recuperarla, debemos utilizar el objeto Request.Params. Entonces, pasamos el nombre del parámetro que queremos recuperar, en nuestro caso, "xmlToken".

En C#:

```
...
string xmlToken;
xmlToken = Request.Params["xmlToken"];
...
```

En VB.NET:

```
...
Dim xmlToken As String
xmlToken = Request.Params("xmlToken")
...
```


Ahora veamos cómo mostrar esta información que nos llega mediante POST a nuestra página login1.aspx. Ésta constará de una etiqueta, denominada Label1, en la que escribiremos lo que nos llega desde la página de login:

En C#:

```
...
public partial class login1 : System.Web.UI.
Page {
  protected void Page_Load(object sender,
  EventArgs e) {
    string xmlToken;
    xmlToken = Request.Params["xmlToken"];
    if (xmlToken == null || xmlToken.
    Equals(""))
      xmlToken = "*NULL*";
    //-
    Label1.Text = xmlToken;
    //-
  }
}
...
```

En VB.NET:

```
...
Partial Class login1
  Inherits System.Web.UI.Page
  Protected Sub Page_Load(ByVal sender As
  Object, ByVal e As System.EventArgs)
  Handles Me.Load
    Dim  xmlToken As String
    xmlToken = Request.Params("xmlToken")
    If (xmlToken = Nothing Or xmlToken.
    Equals(""))
      xmlToken = "*NULL*"
    //-
    Label1.Text = xmlToken
    //-
  End Sub
End Class
...
```

Afortunadamente, Microsoft nos provee de una serie de componentes con los cuales podemos procesar estos datos encriptados. Podemos usarlos importando los namespaces System.IdentityModel.Claims y Microsoft.IdentityModel.TokenProcessor. Veamos en código cómo proceder:

En C#:

```
...
    Token tkn = new Token(xmlToken);
    ClaimSet clset = tkn.IdentityClaims;

    foreach (Claim cl in clset) {
      Response.Write(cl.ClaimType + ":" +
      cl.Resource.ToString() + "<BR />");
    }
...
```

En VB.NET:

```
...
    Dim tkn As New Token(xmlToken)
    Dim clset As ClaimSet = tkn.Identity
    Claims;
    Dim cl As Claim

    For Each cl In clset
      Response.Write(cl.ClaimType + ":" +
      cl.Resource.ToString() + "<BR />")
    Next
...
```

Tabla 2 | Objeto "application/x-informationcard"

Propiedad	Descripción
tokenType	Controla el tipo de token que emite el Identity Selector; en este caso, un token SAML 1.0.
issuer	El proveedor de la URL de identidad del Identity Provider.
requiredClaims	Hace la petición del usuario para proveer de un token al Identity Provider con ciertas demandas.

Windows Presentation Foundation

La nueva plataforma

13

Contenidos

En este capítulo aprenderemos sobre la nueva plataforma de desarrollo de interfaces ricas, Windows Presentation Foundation. Para qué fue creada, nuevos actores dentro del mundo del desarrollo y un nuevo lenguaje de marcas, XAML.

Temas tratados

- Introducción
- XAML
- Controles simples
- Ventanas en WPF
- Gráficos y animaciones

Windows Presentation Foundation

Nos introduciremos en esta nueva herramienta de desarrollo que propone Microsoft, para generar aplicaciones de gran impacto. Ventajas, características y potencia.

Introducción

Analizaremos este nuevo paradigma de desarrollo: cómo nació, qué promete y qué herramientas conforman los nuevos actores que aparecen dentro del mundo del desarrollo.

- Interfaces visuales
- Qué nos proporciona
- La suite Expressions de Microsoft

XAML

Aprenderemos el lenguaje base de Windows Presentation Foundation, cómo se compone, su arquitectura y aspectos importantes.

- Concepto y definición
- Estructura
- Ventajas

Controles simples

Aprenderemos cuáles son los controles más comunes que se utilizan en el desarrollo con Windows Presentation Foundation.

- Tipos existentes
- Menús y barras de herramientas
- Todos los controles

Ventanas en WPF

Veremos cuáles son las mejoras al desarrollar aplicaciones de escritorio, ejemplos y guías prácticas para realizar nuestra primera aplicación de escritorio usando WPF.

- Ventajas
- Ejemplos
- Uniendo propiedades
- Elementos básicos de una interfaz de usuario

Gráficos y animaciones

Estudiaremos las herramientas que nos provee WPF para mostrar, crear y manipular imágenes en dos y tres dimensiones, como así también la manera de animarlas.

- Gráficos 2D simples
- Gráficos 3D
- Conceptos y generalidades sobre animación

Introducción a Windows Presentation Foundation

Analizaremos un concepto que fue dejado de lado por los desarrolladores durante mucho tiempo: la experiencia del usuario.

Para entender esta nueva tecnología, que de ahora en adelante nombraremos mediante sus siglas WPF (*Windows Presentation Foundation*), es importante entender el concepto de experiencia del usuario en software.
Muchas veces, cuando se fabricaban estas soluciones, los desarrolladores eran los encargados de generar el diseño gráfico para ellas, y cuando esto sucedía, se lo expresaba mediante la frase "diseñado por ingenieros". Sin embargo, la experiencia de usuario, en la actualidad, brinda un tremendo valor agregado a nuestro trabajo, y se representa en la satisfacción de los usuarios finales.
Hasta ahora, no era posible obtener un vínculo entre herramientas y APIs, que permitan interactuar tanto con la gente de desarrollo, como con el mundo de los diseñadores, en un solo ambiente. Mientras que aquellos encargados de brindar la riqueza de usuario de una manera espectacular utilizaban herramientas publicadas directamente en la Web, los que debían desarrollar la funcionalidad de dichas soluciones tenían que pasar por un sinnúmero de problemas para hacerlas realidad. Mediante WPF, ambos mundos pueden convivir en un solo escenario, en el que, a través de un único lenguaje, es posible interpretar el trabajo de diseño y programación en un solo producto, y tener una experiencia de usuario enriquecida. El resultado de esto es el actual

FIGURA 001 | La estrategia de Microsoft es apostar a la experiencia de usuario, y esto se puede ver en los lanzamientos de sus últimos productos.

sistema operativo de Microsoft Windows Vista y la nueva suite de oficina, Office 2007.
Esto nos lleva a pensar que antes, la mayoría de los responsables de construir aplicaciones de software se preocupaban por cómo éstas debían funcionar, pero no por cómo interactuaban con los usuarios. En resumen, en la actualidad, la interfaz de una aplicación es un componente fundamental de una completa experiencia de usuario. Entre otros beneficios, brinda un aumento de productividad en nuestras soluciones, como incrementar las ventas en un sitio Web, y mucho más.

Con la publicación de Microsoft .NET Framework 3.0, fueron liberadas al mercado cuatro nuevas tecnologías: *Windows Presentation Foundation* (WPF), *Windows Communication Foundation* (WCF), *Windows Wokrflow Foundation* (WF) y *Windows CardSpace* (InfoCard).
WPF proporciona una serie de herramientas que permiten construir interfaces que incluyen la incorporación de archivos de media, documentos, gráficos bi o tridimensionales, animaciones, características de interfaces web y muchas otras cosas más. Al ser un producto basado en .NET, esta tecnología está disponible para Windows Vista, XP y Server 2003. Dichas herramientas brindan tanto a diseñadores como a desarrolladores la posibilidad de crear aplicaciones ricas en experiencia de usuario, ofreciendo interfaces más amigables.
Este grupo de herramientas trabajan bajo el mismo protocolo de comunicación denominado XAML (se pronuncia zammel o xamel), lenguaje basado en XML que permite definir el diseño gráfico de una manera sencilla, tal como veremos más adelante.

De esta manera, con WPF podemos distinguir dos aspectos que, hasta el momento, estaban mezclados y provocaban cierta confusión al generar aplicaciones: el diseño de la interfaz, y el diseño de la parte lógica y de presentación de la aplicación. Mediante el uso de Microsoft Expression, los diseñadores podrán crear las interfaces de usuario, y los programadores, hacer su trabajo mediante Visual Studio, ambos utilizando un mismo lenguaje y obteniendo resultados no logrados hasta el momento.

Núcleo del .NET Framework 3.0

FIGURA 002 | El núcleo del .NET Framework 3.0 es una extensión de .NET Framework 2.0, cuyas novedades son InfoCard, WPF, WCF y WWF.

Características y capacidades de WPF

A continuación, vamos a mencionar los aspectos fundamentales de esta tecnología y, luego, nos referiremos a ellos en detalle.

WPF provee tres aspectos fundamentales al momento de construir una aplicación, en lo que respecta a experiencia del usuario. A continuación, vamos a mencionarlos y, luego, nos referiremos a ellos en detalle.

- Una plataforma unificada para interfaces de usuario modernas y ricas.
- Posibilidad de que desarrolladores y diseñadores trabajen juntos
- Una tecnología común para aplicaciones basadas en Windows y Web.

Una plataforma unificada para interfaces de usuario modernas y ricas

Tal como podemos observar en la Tabla 1, WPF abarca todas las características presentadas por esas tecnologías y herramientas; sin embargo, esto no significa que pretenda reemplazarlas. Por ejemplo, Windows Form continuará teniendo su valor tanto en su ámbito de aplicaciones como en el mundo de WPF; es totalmente claro que seguirán existiendo nuevas aplicaciones usando Windows Form. Así como Windows Media Player continuará en su uso para reproducir audio y video, PDF será, como lo es actualmente, una plataforma para presentación de documentos.

Para tener una idea mucho más clara acerca de lo que significa tener una plataforma unificada, podemos referirnos a una aplicación desarrollada que contenga texto, gráficos en 2D y 3D e, incluso, controles comunes, como cajas de texto, listas desplegables, etc., tal como se ilustra en la Figura 3. Lo importante en este punto es que el desarrollador no necesita tener conocimientos sobre creación de aplicaciones utilizando GDI+ o Direct3D.

Tabla 1 | Características y capacidades de WPF

	Windows Form	PDF	Windows Form/GDI+	Windows Media Player	Direct3D	WPF
Interfaz gráfica, formularios, controles	•					•
Documentos en pantalla	•	•				•
Imágenes			•			•
Video y audio				•		•
Gráficos en dos dimensiones			•			•
Gráficos en tres dimensiones					•	•

Por otro lado, construir interfaces modernas y ricas no significa utilizar necesariamente diferentes tecnologías; también incluye tomar las características de las tarjetas gráficas. Es por eso que WPF utiliza en su mayor capacidad la unidad de procesamiento gráfico (GPU, *Graphic Process Unit*). Esto también soluciona el problema del trabajo con imágenes basadas en bit maps, pues ahora WPF trabaja con imágenes basadas en vectores, que admiten ser redibujadas automáticamente para ajustarse al tamaño y a la resolución de la pantalla que corresponda.

FIGURA 003 | WPF nos ofrece la posibilidad de crear interfaces de usuario ricas y vistosas, y de separarlas de la lógica de nuestro negocio.

¿Por qué XAML?

Seguramente reflexionaremos sobre qué tiene de favorable utilizar XAML en vez de código tradicional. La respuesta es muy sencilla: las herramientas pueden consumir estas interpretaciones de manera mucho más transparente que al hacerlo mediante código, por lo que proporcionan un canal de comunicación directo y eficiente entre el diseñador y el desarrollador.

Posibilidad de que desarrolladores y diseñadores trabajen juntos

Con una herramienta como WPF, se podría esperar que las interfaces fueran construidas de una manera mucho más sencilla y rápida. De hecho, éste es uno de los objetivos de WPF; sin embargo, muchas de nuestras aplicaciones tienen un nivel de dificultad mayor, lo que implica incorporar destrezas de diseño que muchos de los desarrolladores no poseemos. Entonces es cuando necesitamos la ayuda de profesionales en el área de diseño gráfico.

La verdad es que existe una enorme problemática para que ambos perfiles trabajen juntos: mientras que el diseñador produce imágenes y estilos con sus herramientas, el desarrollador debe tomar estos recursos y traducirlos en una aplicación funcional, y no siempre esta traducción interpreta lo que el diseñador quiso expresar en sus bocetos o identidades gráficas. El problema aumenta cuando es necesario incorporar cambios por solicitudes del cliente, cuando casi siempre ambas partes salen afectadas al introducir una nueva funcionalidad, ya sea el código, el diseño gráfico o ambos.
Para eliminar este inconveniente, WPF introduce un nuevo lenguaje de comunicación entre ambos perfiles, **XAML** (*eXtensible Application Markup Language*). Está basado en XML, y utiliza etiquetas para botones, cajas de texto, etc., para definir cómo será presentada la interfaz. Más adelante, profundizaremos en este punto; por el momento, revisemos un pequeño ejemplo acerca de cómo incluir atributos gráficos en un botón con XAML:

```
<Button Bakcground="Red">
  No
</Button>
```

Cada elemento en XAML corresponde a una clase en WPF; por lo tanto, cada atributo corresponde a una propiedad de ella.

Haciendo una analogía acerca de cómo lograríamos esto en código C#, tendríamos algo así:

```
Button btn = new Button();
btn.Background = "Red";
btn.Content = "No";
```

> Con WPF, se podría esperar que las interfaces fueran construidas de una manera mucho más sencilla y rápida.

Una tecnología común para aplicaciones basadas en Windows y Web

Tal como estuvimos mencionando, el hecho de construir aplicaciones ricas en interfaces de usuario es importante tanto para ambientes Windows como para aplicaciones Web. El desarrollo de este tipo de aplicaciones requiere, por lo general, la ayuda de muchas tecnologías usadas para trabajar de manera nativa en programas Windows, lo que implica que es necesario tener conocimientos para construir interfaces en ambientes Windows y en ambientes Web. La meta de WPF es, justamente, tener una única tecnología que permita trabajar en ambos ambientes sin tener que recurrir a varias tecnologías.

Un desarrollador puede crear una aplicación para navegadores con XAML (XBAP, *Xaml Browser APplication*) utilizando WPF, y hacer que se ejecute sobre él mismo. De igual manera, puede utilizar el mismo código para crear aplicaciones WPF basadas en Windows. Lo más importante de esta característica es que el desarrollador no tiene que preocuparse por modificar la interfaz gráfica de su aplicación si no necesita correr en un ambiente Windows pero sí en uno Web. Para él es totalmente independiente, y todo lo generado para Windows es utilizado también para Web, y viceversa.

Desarrolladores y diseñadores, juntos

FIGURA 004 | La introducción de WPF permite que diseñadores y desarrolladores trabajen en forma conjunta utilizando un mismo lenguaje.

XAML: Concepto y definición

Un lenguaje común de trabajo, que podrán utilizar tanto desarrolladores como diseñadores gráficos.

XAML se basa en la nueva tendencia de programación declarativa para escribir aplicaciones con WPF, según la cual se introduce un nuevo rol dentro del desarrollo de aplicaciones; tal como hemos mencionado anteriormente, estamos hablando del **diseñador gráfico**.
Por ejemplo, cuando generamos aplicaciones de la manera tradicional, los botones adoptan las formas y los colores del entorno en el que se ejecutan, sin que exista ningún elemento gráfico que provoque satisfacción al usuario final. Con la disponibilidad del modelo de la programación declarativa a través de XAML, nosotros podemos separar de una manera sencilla la lógica del negocio, de la lógica de presentación. XAML es un lenguaje de marcado que, implícitamente, se convierte en una herramienta independiente, tanto para los diseñadores como para los desarrolladores. Si analizamos la Figura 5, podemos resumir las principales ventajas de XAML en los siguientes puntos:

- Es un lenguaje mucho más expresivo que el código utilizado para crear las interfaces de usuario, lo que se resume en una simplificación de dicho componente.
- Al separar la lógica de programación de la de presentación, logramos que el trabajo hecho por los desarrolladores sea mucho más simple de entender. También conseguimos que el mantenimiento de la interfaz gráfica hecha por el diseñador sea más rápido.
- Al tener un lenguaje unificado de comunicación entre ambientes de desarrollo y de

Cómo declarar un objeto

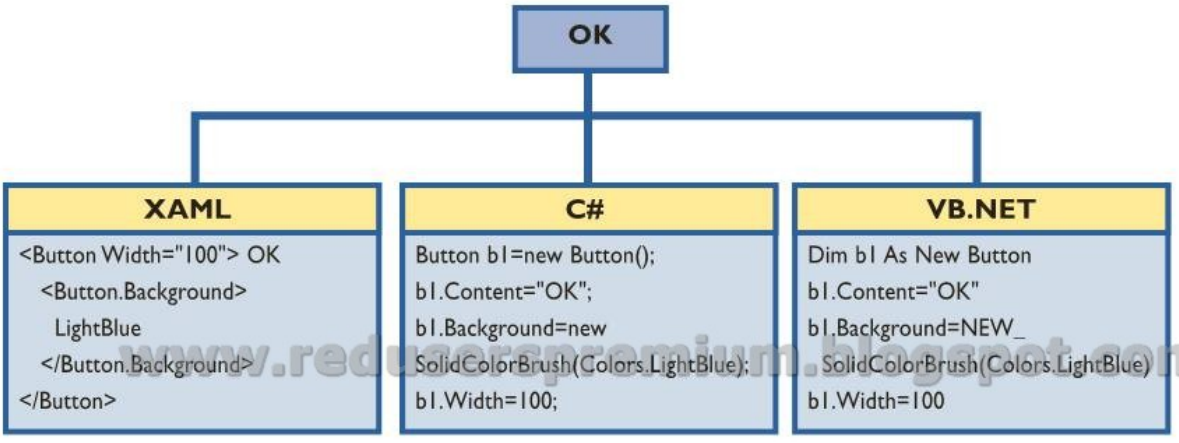

FIGURA 005 | Declarar un objeto y establecer sus propiedades gráficas resulta más sencillo con XAML que con otros lenguajes de .NET.

diseño gráfico, obtenemos un flujo de trabajo que parte de la definición de la identidad gráfica e interfaz de la aplicación, y finaliza con la inclusión del código y su funcionalidad por medio del desarrollador.

A partir de todo lo explicado anteriormente, podemos resumir que XAML simplifica la creación de una interfaz de usuario enmarcada dentro del .NET Framework 3.0. A diferencia de otros lenguajes de marcas, XAML representa directamente la instanciación de objetos administrables, hecho que facilita el código y el acceso a debugging para dichos objetos en este lenguaje. Esto significa que XAML se convierte en el lenguaje propio de la interfaz gráfica; su meta es generar la clase de código que, tradicionalmente, encontrábamos en el área oculta de nuestras aplicaciones, haciendo que su mantenimiento sea complicado para los desarrolladores y sus herramientas. Un aspecto que vale la pena mencionar es que XAML no es WPF, y WPF no es XAML, aun cuando ambos hayan sido desarrollados por Microsoft. El objetivo principal es trabajar juntos para lograr aplicaciones cuya riqueza de experiencia de usuario sea la mejor, y para que el esfuerzo entre desarrolladores y diseñadores sea mínimo al momento de integrar sus productos.
XAML forma parte del formato XML; de hecho, cada archivo XAML debe ser un archivo XML y, por lo tanto, hereda todas las definiciones y reglas de éste. Se lo podría considerar como una nueva tendencia basada en HTML, XHTML y otros tipos de lenguajes en beneficio de la interfaz de usuario. Lo que hace que XAML sea diferente del resto de los lenguajes de marcado es lo que representa en sí: cada elemento en XAML representa una clase del CLR de .NET, lo que permite extenderlo y trabajarlo con facilidad. Debido a que en XAML los elementos se representan como una clase, las propiedades pueden representarse como atributos. De este modo, la sintaxis es mucho más aerodinámica e intuitiva para los desarrolladores que han usado lenguajes de diseño en el pasado. En el siguiente ejemplo podemos distinguir fácilmente el beneficio de crear objetos visuales mediante XAML. En este caso, creamos un botón con texto rojo y fondo azul, y su contenido correspondiente:

> XAML simplifica la creación de una interfaz de usuario enmarcada dentro del .NET Framework 3.0.

```
<Button Background="Blue" Foreground="Red"
Content="This is a button"/>
```

Más adelante veremos que XAML es un lenguaje sencillo de utilizar, sobre todo, para quienes ya han tenido experiencia con el manejo de archivos en HTML, XHTML y XML, dado que su sintaxis es más simple y transparente para el diseñador y para el programador.

Arquitectura de WPF

Para entender la arquitectura global de las aplicaciones que se construyen bajo la plataforma WPF, veamos a continuación cómo se

§ ¿Qué significa XAML?

XAML es un acrónimo que se pronuncia xammel o zámel, y que significa lenguaje de marcas para aplicaciones extensibles (*eXtensible Application Markup Language*). Se utiliza con Windows Presentation Foundation para crear interfaces de usuario.

encuentran estructurados sus componentes principales (Figura 0[illegible]). Podemos observar que todos los tipos de datos de medios (en amarillo) son soportados por WPF; es decir, vectores, bitmaps, 3D, audio y video, texto y efectos. El segundo punto por considerar es que con WPF podemos expandir todas las capacidades de animación a través de todos los tipos de medios, y así podemos animar cualquier clase de contenido. El motor de composición WPF (WPF, *Composition Engine*) es una de las capacidades más interesantes de esta tecnología, al ser la forma por la cual obtenemos contenidos vivos dentro de contenido estático u otro tipo de contenido. Esta cualidad de una estructura absorbente está disponible para cualquier tipo de contenido y control provisto por WPF. Es muy importante destacar que WPF está relacionado no sólo con el desarrollo de interfaces ricas en experiencia de usuario, sino también con la fidelidad de la información y la conexión para obtener los datos. Controles, Layouts y Databinding son ejemplos de este tipo de características que presta WPF. Encontramos también el formato XPS, una definición de documentos ricos que nos permiten disfrutar de lo mejor de las capacidades de WPF dentro de documentos.

Arquitectura de WPF

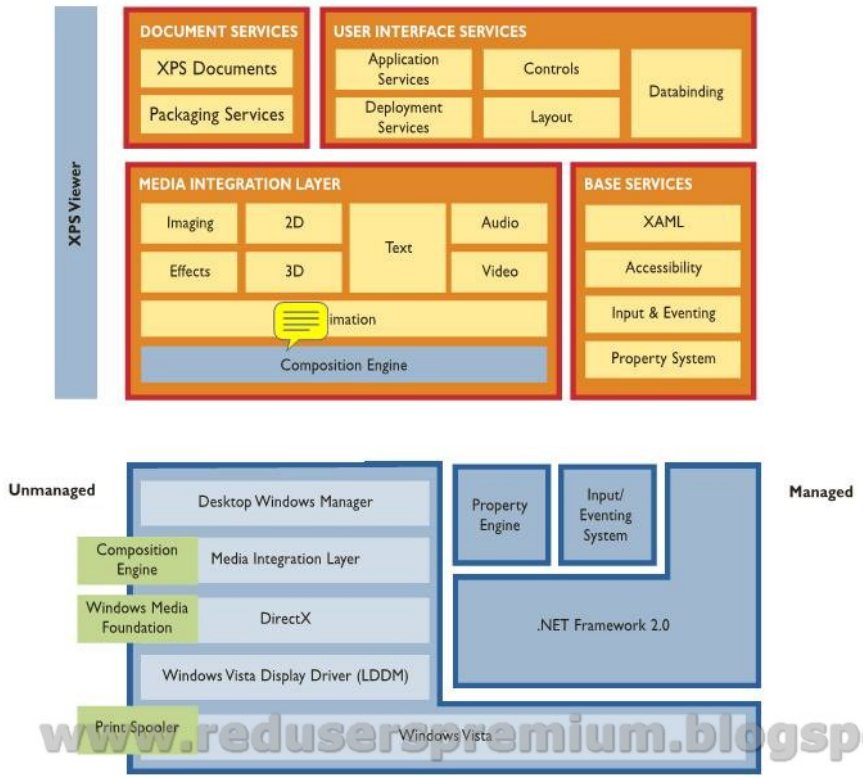

FIGURA 006 | La arquitectura de WPF nos permite expandir las capacidades gráficas de las aplicaciones que desarrollemos.

Principales espacios de nombres en WPF

Vamos a conocer los principales Namespaces o espacios de nombres de WPF, ubicados en su forma jerárquica.

System.Object

El punto de inicio para el modelo de programación en WPF es la exposición a través del código administrado. El CLR provee un número de características que hacen que el desarrollo sea más robusto y productivo (incluyen administración de memoria, manejo de errores y sistemas de tipos comunes).

En la Figura 007 podemos observar los componentes más importantes dentro de la tecnología WPF. La sección roja del diagrama (**PresentationFramework**, **PresentationCore** y **Milcore**) representa las secciones de código más importantes en WPF. De todos ellos, el único componente no administrado es **Milcore**. La razón por la cual Milcore fue desarrollado en código no administrado es la interacción que este componente debe tener con **DirectX**. Es preciso tener en cuenta que todo el despliegue de las aplicaciones en WPF se realiza a través del motor de DirectX, ya que éste lleva a cabo una optimización de hardware y software. Es por esto que WPF necesita un buen control sobre el uso de memoria y ejecución.

Componentes que trabajan fuera del CLR

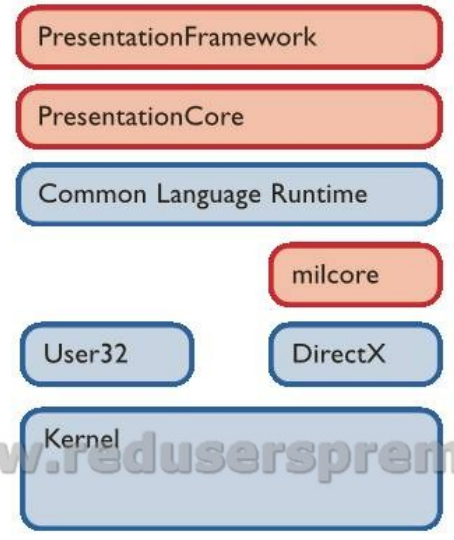

FIGURA 007 | WPF contiene componentes que trabajan fuera del CLR. Uno de los motivos de esto es su interacción exhaustiva con DirectX.

La composición del motor de Milcore es extremadamente sensitiva, lo que incremente las ventajas del CLR en general.

System.Threading.DispatcherObject

Muchos de los objetos en WPF derivan del **DispatcherObject**, el cual provee los aspectos primordiales para la construcción de concurrencia y threading. WPF se basa en un sistema de mensajería implementado por el Dispatcher. Existen dos puntos importantes que debemos tener en cuenta para comprender la concurrencia dentro de WPF: dispatcher y thread affinity.

Cuando WPF fue diseñado, uno de los objetivos fue trasladarse a un simple hilo (*thread*) de ejecución, pero en un modelo *non-thread affinity*. Éste se ejecuta cuando un componente usa la identidad de un hilo de ejecución para almacenar un tipo de estado. La razón principal de utilizar este tipo de procesamiento fue la interoperabilidad entre sistemas –como OLE 2.0, el Portapapeles e Internet Explorer–, ya que éstos requieren un *single thread afinity* (SAT).

System.Windows.DependecyObject

Una de las principales filosofías en la construcción de aplicaciones WPF es la preferencia de usar propiedades sobre métodos o eventos. Las propiedades son declarativas y nos permiten ejecutar la funcionalidad deseada de una manera mucho más sencilla. Es por este motivo que en el CLR se ha incluido un sistema conducido mediante propiedades; dentro de WPF, se lo conoce como un sistema de propiedades ricas, derivado desde el tipo **DependecyObject**. El sistema de propiedades es, claramente, de dependencia, y sigue las dependencias entre las expresiones de las propiedades y los valores de las propiedades, revalidadas automáticamente cuando las dependencias cambian. Para verlo de una

Tratamiento de objetos visuales

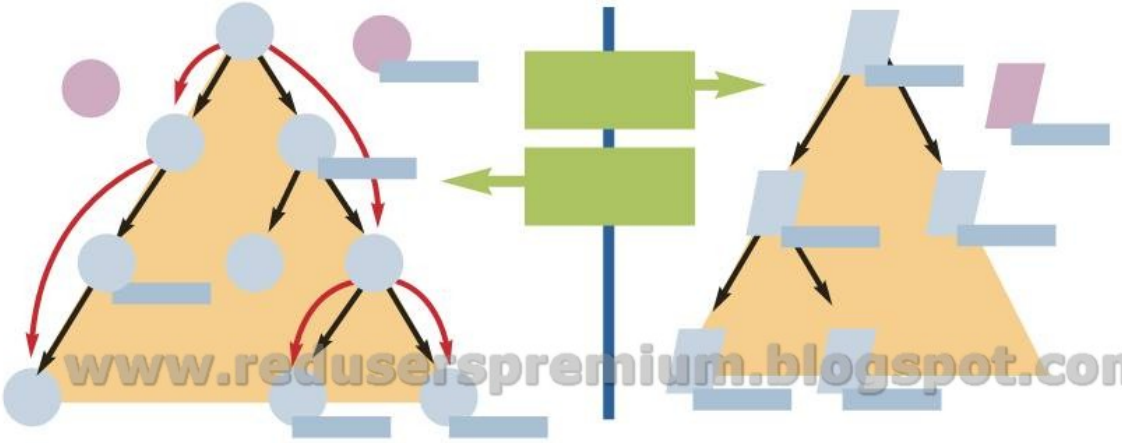

FIGURA 008 | Cuando los objetos visuales en WPF son redibujados o repintados, el código de usuario no se bloquea.

manera más clara, pensemos en tener una propiedad heredada (FontSize, por ejemplo). El sistema es automáticamente actualizado si la propiedad cambia en un padre de un elemento que hereda el valor.

System.Windows.Media.Visual

Teniendo ya un sistema definido, el siguiente punto es obtener los píxeles dibujados en la pantalla. La clase **Visual** provee la construcción de un árbol de objetos visuales, cada uno los cuales contiene instrucciones y metadatos para ejecutar varias instrucciones (transformaciones, truncaciones, etc.). Esta clase es diseñada para ser lo más flexible y ligera posible, y la mayoría de sus características no tienen exposición pública de sus APIs, sino que son manejadas a través del llamado de funciones protegidas.

System.Windows.UIElement

UIElement define subsistemas de base incluyendo layouts (diseños), entradas y eventos. Son claros los puntos referentes a las entradas y los eventos, pero WPF define un componente denominado Layout. Por ejemplo, tenemos varios sistemas que poseen modelos basados en layout (HTML suporta tres modelos de layout: flujo, absoluto y tablas). Como lo mencionamos anteriormente, WPF otorga a los diseñadores y programadores un modelo de diseño flexible y extensible. Por tanto, el objetivo principal de este concepto es proveer de un modelo extensible que permite a los programadores crear su propio Framework para cubrir sus necesidades presentadas en el desarrollo de una aplicación.

System.Windows. FrameworkElement

FrameworkElement define un conjunto de políticas y customizaciones de subsistemas en las capas iniciales del WPF. La política primaria definida dentro del WPF es para el entorno de la aplicación, construyendo un contrato dentro del modelo layout (definido por UIElement) y añadiendo una noción de ranuras dentro del layout. Esto proporciona a los creadores de dichos layouts un conjunto de propiedades para manejar su semántica de una manera mucho más simple (por ejemplo, MinWidth, Margin, etc.). Un aspecto que debemos destacar es que **FrameworkElement** también incluye el enlace de datos y el estilo utilizado dentro de la aplicación.

> Debemos tener en cuenta que todo el despliegue de las aplicaciones en WPF se realiza a través del motor de DirectX, ya que éste lleva a cabo una optimización de hardware y software.

System.Windows.Controls.Control

Una de las características más importantes en lo que respecta a la clase Control es su capacidad de trabajar bajo templates o plantillas, que permiten describir la función del control de una manera declarativa. La clase Control otorga una serie de propiedades –tales como Foreground, Background, Padding, etc.–, de tal manera que los autores pueden determinar cómo será desplegado dicho control. La implementación de un control permite tener un modelo de datos y de interacción entre ellos. Por lo tanto, esta completa conexión entre modelo de datos, de interacción y de despliegue nos brinda un control completo sobre cómo se debería visualizar y cómo debería trabajar cada uno de estos elementos.

Herencia de clases del framework

La mayoría de las clases utilizadas dentro de WPF son derivadas de las cuatro más importantes. Véremos cómo trabajar con ellas.

Estas clases son comúnmente referidas en la documentación de SDK disponible para los elementos base del WPF. Éstas son:

- UIElement
- FrameworkElement
- ContentElement
- FrameworkContentElement

APIS de los elementos base en WPF

Las clases **UIElement** y **ContentElement** son derivadas de **DependecyObject** a través de diferentes métodos. A continuación, veremos cómo interactúan estos elementos en la presentación de una aplicación.
La clase **UIElement** también tiene su representación en la jerarquía de la clase Visual, que se encarga de exponer en un nivel inferior los gráficos subyacentes que soporta Windows Presentation Foundation. La clase **Visual** provee de un framework capaz de definir regiones rectangulares independientes en una pantalla. Refiriéndonos ya a la construcción de aplicaciones, **UIElement** está diseñado para manejar elementos que soporten el modelo de objetos de gran tamaño, utilizados para las disposiciones de las regiones que puedan ser representadas en forma rectangular; por lo tanto, hace que el modelo de contenido sea más abierto y permite una diferente combinación entre elementos. Por otro lado, **ContentElement** no deriva de la clase Visual; su forma de uso es que **ContentElement** debe ser consumido por alguien más, como un **reader** o un **viewer**, los cuales se ocuparán de interpretar los elementos para producir una representación visual en WPF y consumirla posteriormente. Algunas clases de **ContentElement** fueron desarrolladas para ser hosts de contenidos (almacenamiento de contenidos); ellas proporcionan este almacenamiento y son representadas por una o más clases de tipo **ContentElement** (**DocumentViewer** es un ejemplo). **ContentElement** es usada como una clase base para elementos de modelo de objetos un poco más pequeños, los cuales pueden ser almacenados dentro de un **UIElement** con características tales como texto, información y contenido del documento.

Nivel del framework y nivel del core

UIElement se usa como la clase base para **FrameworkElement**; de la misma manera, **ContentElement** es la clase base para **FrameworkContentElement**. La razón por la cual existe este siguiente nivel de clases es apoyar el nivel del core de WPF, que está totalmente separado del nivel del framework de WPF. Con esta división podemos identificar que las APIs son divididas en los assemblies de **Presentation-**

Core y **PresentationFramework**. El nivel del framework de WPF ofrece más que una solución completa para las necesidades de aplicaciones básicas, incluyendo funcionalidad tal como la administración de layouts para la presentación. El nivel del core brinda la forma en la cual se puede utilizar WPF sin caer en el *overhead* de assemblies adicionales. Pero es importante destacar que la diferencia entre estos niveles es raramente identificada en la mayoría del desarrollo de aplicaciones y, por lo tanto, es responsabilidad del desarrollador definir qué APIs van a ser utilizadas. Los desarrolladores deben conocer las diferencias entre los niveles antes mencionados si el diseño de la aplicación que se está construyendo necesita reemplazar código sustancial del nivel de framework del WPF; por ejemplo, si la aplicación tiene sus propias implementaciones de interfaces de usuario y layouts. La manera más sencilla de crear una clase personalizada que extienda funcionalidad de WPF es derivando de una clase de WPF, de acuerdo con su jerarquía. En la Tabla 2 se enumeran las principales clases para extender.

Tabla 2 | Principales clases de WPF para extender

1	**Si se crea una clase que derive de DependencyObject, se heredará la siguiente funcionalidad:**	
	a.	Soporte de GetValue y SetValue, y soporte al sistema general de propiedades
	b.	Posibilidad de usar las propiedades de dependencia y adjuntar propiedades que son implementadas de esa manera.
2	**Si se crea una clase que derive de UIElement, se heredará la siguiente funcionalidad, adicional a la definida en la clase DependecyObject:**	
	a.	Soporte básico para valores de propiedades animadas.
	b.	Soporte para eventos de entradas.
3	**Si se crea una clase que derive de FrameworkElement, se heredará la siguiente funcionalidad, adicional a la definida en la clase UIElement:**	
	a.	Soporte para estilos y storyboard.
	b.	Soporte para databinding.
	c.	Soporte para referencias de recursos dinámicos.
	d.	Soporte para herencia en valores de propiedades.
	e.	Concepto de árbol lógico.
	f.	Soporte para el nivel de framework del sistema de layout.
4	**Si se crea una clase que derive de ContentElement, se heredará la siguiente funcionalidad adicional a la explicada en DependecyObject:**	
	a.	Soporte para animaciones.
	b.	Soporte para eventos de entradas.
5	**Si se crea una clase que derive de FrameworkContentElement, se heredará la siguiente funcionalidad adicional a la definida en ContentElement:**	
	a.	Soporte para estilos y storyboard.
	b.	Soporte para databinding.
	c.	Soporte para referencias de recursos dinámicos.
	d.	Soporte para herencia en valores de propiedades.

Objetos "freezables"

A continuación veremos qué son exactamante y cómo aprovecharlos en nuestro desarrollo.

Conozcamos primero a qué se denomina objeto "freezable". Se trata de un tipo especial de objeto que tiene dos estados: congelado y no congelado (*frozen* y *unfrozen*). Cuando un objeto de este tipo se encuentra en estado unfrozen, se comporta de la misma manera que cualquier otro; en cambio, cuando está frozen, no puede ser modificado.
Un objeto freezable provee un evento de cambio (*changed event*) para notificar cualquier modificación sobre él. Uno frozen podría ser compartido a través de los diferentes hilos de ejecución, en tanto que uno unfrozen no aplica en este contexto.

Las clases freezable tienen muchos usos, la mayoría de los cuales se relaciona con subsistemas gráficos. Estas clases simplifican el uso de ciertos objetos dentro de sistemas gráficos y ayudan a **mejorar la performance** de una aplicación. Algunos ejemplos de tipos que heredan de clases freezable son **Brush**, **Transform** y también clases de tipo **Geometry**.

A continuación, veamos cómo podemos crear un **SolidColorBrush** y usarlo para pintar el fondo de un botón.

```
...
Button myButton = new Button();
SolidColorBrush myBrush = new SolidColorBrush
(Colors.Yellow);
myButton.Background = myBrush;
...
```

Cuando el botón es generado, el subsistema de gráficos de WPF utiliza la información que se proporcionó para pintar un grupo de píxeles y, así, crear el aspecto de un botón. Aunque se utilizó **SolidColorBrush** para describir de qué manera debería ser pintado el botón, dicho componente, en realidad, no hace este trabajo. El sistema de gráficos genera los objetos de una manera rápida, y dichos objetos son los que, en verdad, son presentados en la pantalla.

Un objeto freezable es del tipo perteneciente a **DependecyObject** y, por lo tanto, utiliza el sistema de dependencia de propiedades. Sus propiedades de las clases no poseen propiedades de dependencia, pero las usan para reducir la cantidad de código que se debe escribir; esto se debe a que fueron diseñadas con propiedades de dependencia.

Las clases freezable tienen muchos usos, la mayoría de los cuales se relaciona con subsistemas gráficos. Estas clases simplifican el uso de ciertos objetos dentro de sistemas gráficos y ayudan a mejorar la perfomance de una aplicación.

Estructura de XAML

Ahora verémos qué características posee un documento construido en formato XAML.

Un documento construido en formato XAML contiene información referente a un contenedor de interfaz de usuario, tal como lo especifica WPF. En la mayoría de los casos, esta información corresponde a elementos utilizados para interactuar con una aplicación, tales como ventanas y cuadro de diálogo, entre otros. Tomando en cuenta que XAML es un documento basado en XML, hereda todos los atributos y propiedades para formar de una manera correcta este tipo de archivos; por lo tanto, debe ser un documento bien formado en XML (*well-formed XML*). Una de las características principales de este tipo de documentos es que contiene un único elemento raíz (elemento padre), que incluye al resto de elementos. En el caso de WPF, cuando tenemos una interfaz declarada mediante XAML, el elemento padre corresponderá al componente especificado en él.

Analicemos este concepto mediante el ejemplo inicial en la mayoría de las aplicaciones, Hola Mundo, el cual representa un mensaje dentro de una ventana a través de un componente de texto que se declara de la siguiente manera:

```
<Window xmlns="http://schemas.microsoft.com/
2003/xaml" Visible="true">
   <SimpleText Foreground="Black" FontSize=
   "14">
      Hola Mundo!
   </SimpleText>
</Window>
```

El nodo Window es la raíz del documento, y especifica que su contenedor es una ventana; dentro de él encontramos otro componente encargado de desplegar el texto (**SimpleText**). El trabajo de WPF es interpretar las etiquetas en XAML y convertirlas en código gestionado por parte de las librerías del CLR. De este modo, distintos elementos se corresponden con propiedades de los objetos instanciados. Si revisamos el caso anterior, un ejemplo de esto sería la propiedad Foreground del elemento **SimpleText** del documento. La clase **SimpleText** a la que corresponde la etiqueta especificada en el archivo XAML posee una propiedad con el mismo nombre que el atributo allí especificado. Una vez creada la instancia de dicha clase, se podría acceder a esta propiedad y establecer el valor deseado.

Revisemos el mismo ejemplo basado en C# para poder mostrar su equivalencia:

```
// Crear la ventana principal

mainWindow = new MSAvalon.Windows.Window ();

// Añadir el elemento con el texto "Hola
Mundo!",
```

Un documento construido en formato XAML contiene información referente a un contenedor de interfaz de usuario, tal como lo especifica WPF.

```
// con letra negra de tamaño 14

txtElement = new MSAvalon.Windows.Controls.
SimpleText();
txtElement.Text = "Hola Mundo!";
txtElement.Foreground = new MSAvalon.Windows.
Media.SolidColorBrush(Colors.Black);
txtElement.FontSize = new FontSize(14,
FontSizeType.Point);
mainWindow.Children.Add(txtElement);
mainWindow.Show();
```

Antes de finalizar con esta breve explicación sobre los elementos que interactúan en un archivo de XAML resumamos tres puntos importantes, que se describen en la Tabla 3.

Ventajas de WPF

Venimos revisando las principales características y los distintos tipos de funcionalidad que podemos encontrar al construir aplicaciones basadas en WPF. En nuestras mentes tenemos claros el concepto y las ventajas que generaríamos al desarrollar aplicaciones de esta manera; sin embargo, puntualicemos y resumamos los beneficios que presenta la creación de aplicaciones bajo este marco de trabajo:

- Brindar una tecnología de desarrollo mediante la cual podamos crear una experiencia de usuario rica y diferente de la actual, de una manera fácil y rápida.
- Exponer una única plataforma de construcción de interfaz de usuario en donde se puedan integrar todos los componentes gráficos disponibles en el mercado, como gráficos en 2D, 3D, animaciones, audio, video y textos enriquecidos, entre otros servicios avanzados.
- Comunicación simple y uniforme entre los desarrolladores y los diseñadores gráficos a través de un solo medio de interpretación como lo es XAML, con el fin de optimizar el trabajo de ambas partes, mejorar la actualización de componentes gráficos y de programación cuando una aplicación así lo requiera, y evitar conflictos de versiones y daños entre la interacción de los productos de desarrollo y diseño.
- La arquitectura de WPF se basa en un modelo declarativo, lo cual permite, a través de XAML, obtener todas las capacidades de la experiencia de usuario utilizando cualquier elemento disponible dentro de WPF. De este modo se potencia el papel del diseñador gráfico en el desarrollo de aplicaciones.
- Al incrementar la riqueza en una experiencia de usuario, como resultado implícito

Tabla 3 | Elementos más importantes dentro de XAML

Recursos	Un recurso es un objeto que puede ser accedido desde el elemento al que pertenece o desde sus hijos. No hay que olvidar que dichos recursos tienen su equivalencia en código generado y podrán ser accedidos por los datos de las aplicaciones.
Orígenes de datos	Posibilidad de utilizar orígenes de datos para establecer el valor de propiedades de los elementos de las interfaces; se deben tomar en cuenta los conceptos globales para establecer un origen de datos en cualquier aplicación.
Gestión de eventos	Un simple fragmento de código permite indicar que, ante cierto evento producido sobre un objeto, se debe hacer referencia a un evento para generar la funcionalidad deseada en dicho proceso.

de estas características tenemos una aplicación mucho más amigable, sencilla de usar y, sobre todo, extremadamente agradable a la vista. Dependiendo de la naturaleza de la aplicación, esta nueva característica puede incrementar las ventas y la fidelización de los clientes, obtener una lectura muy similar a la impresa en un documento de presentación, y muchas características más.

- WPF puede ser usado con cualquier framework AJAX para optimizar el procesamiento de solicitudes a las páginas y mejorar el tiempo de respuesta de una aplicación web. Por lo tanto, no depende de ninguna implementación de clientes en AJAX.
- Finalmente, WPF constituye una revolución con respecto a la forma en que se diseñan, crean y entregan las aplicaciones. El reto está en mejorar nuestras aplicaciones para incrementar la satisfacción del cliente.

Ejemplos

A continuación, veremos un par de ejemplos acerca de cómo crear y construir una aplicación basada en WPF, utilizando como herramientas Visual Studio 2008 –incorporado en el .NET Framework 3.5–; el lenguaje de programación utilizado por defecto será C#.

Crear un proyecto en WPF

Para crear una aplicación en WPF, lo primero que debemos hacer es abrir Visual Studio 2008 y realizar los siguientes pasos:

1. Archivo
2. Nuevo Proyecto
3. Visual C#, Templates
4. Seleccionar Aplicación WPF
5. Ingresar los datos requeridos en el wizard:
   a. Nombre de la aplicación
   b. Directorio en el cual será almacenada
   c. Si se va a crear una nueva solución o se va a adjuntar a una existente
   d. Nombre de la solución

Una vez que hayamos creado la solución, podemos notar que, automáticamente, la herramienta agrega las referencias a los namespaces utilizados dentro de WPF (PresentationCore y PresentationFramework, entre otros), así como crea un componente por defecto basado en XAML (tal como se crearía una página por defecto en aplicaciones ASP.NET), en donde podemos ingresar código descriptivo para empezar a definir los elementos que antes habíamos mencionado. El código insertado en esta pantalla (ventana) es el siguiente:

```
<Window x:Class="WpfApplication1.Window1"
    xmlns="http://schemas.microsoft.com/
    winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/
    winfx/2006/xaml"
```

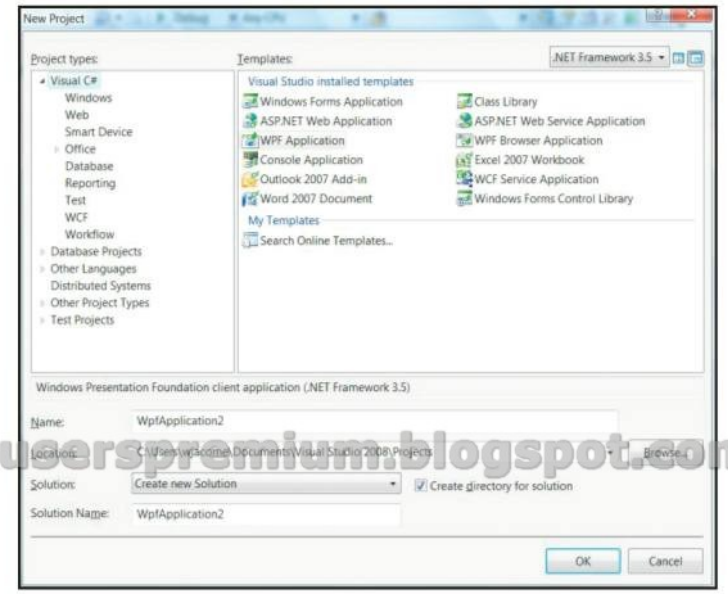

FIGURA 009 | Creando una aplicación en WPF con Visual Studio 2008.

```
    Title="Window1" Height="300" Width="300">
    <Grid>

    </Grid>
</Window>
```

De ahora en adelante, podemos entender que todo el formato basado en HTML es reemplazado por XAML, los archivos de estilos CSS son sustituidos por XAML styles, y el code-behind se sigue usando como tal.

Unir propiedades

En el siguiente ejemplo veremos cómo es posible unir una propiedad de un elemento con otra. En este caso específico, el valor del elemento **slider** es unido al contenido de la caja de texto **TextBox**. Es importante mencionar que, como adicional a esta funcionalidad, se ha añadido la palabra "**value**" en la parte frontal del valor numérico. Esto demuestra cómo se usa a manera de un convertidor en **code-behind**.

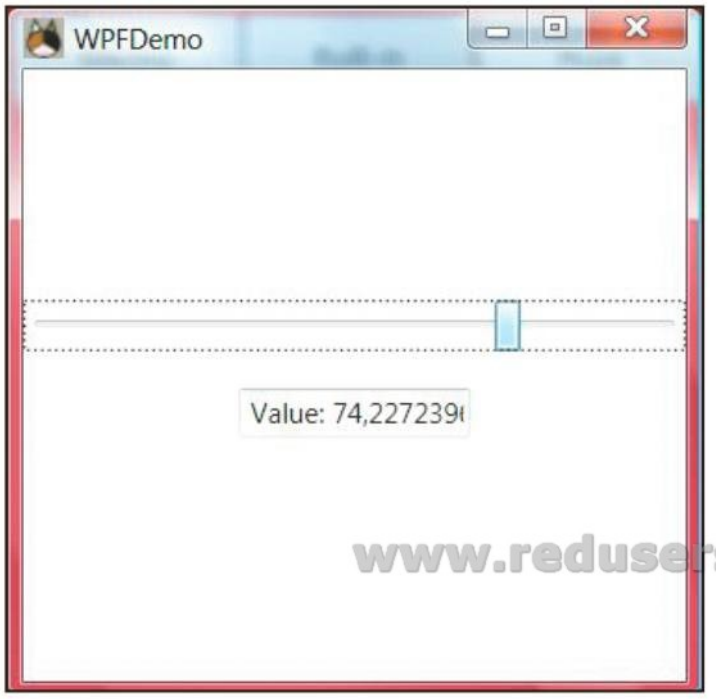

FIGURA 010 | Uniendo propiedades de slider para mostrar el contenido de avance en la caja de texto.

XAML:

```
<StackPanel>
    <Slider Name="sliderTest" Minimum="1"
    Maximum="100" Value="50" Margin="0,100,
    0,0" ValueChanged="sliderTest_Value
    Changed" />

    <TextBox Text="{Binding ElementName=
    sliderTest, Path=Value, Converter={Static
    Resource oIntToString}}" Width="100"
    Margin="0,16,0,0"/>

</StackPanel>
```

Code-behind en C#:

```
public class CIntToStringConverter :
IValueConverter
{
  #region IValueConverter Members

  public object Convert( object value, Type
  targetType, object parameter, System.
  Globalization.CultureInfo culture )
  {
    return "Value: " + value.ToString();
  }

  public object ConvertBack( object value,
  Type targetType, object parameter,
  System.Globalization.CultureInfo culture )
  {
    return null;
  }

  #endregion
}
```


Como resultado de este código, tenemos una ventana en la que, al hacer **clic** sobre el **slider**, podemos ir viendo el valor en el cual está ubicado el desplazamiento actual.

Argentina $8,90 (recargo al interior $0,20) / México: $45

USERS

Microsoft

Curso teórico y práctico de programación

Desarrollador .net

Con toda la potencia de **Visual Basic .NET** y **C#**

La mejor forma de aprender a programar desde cero

★★★★★

Basado en el programa Desarrollador Cinco Estrellas de Microsoft

19

CardSpace

Utilización con .NET 3.0

SALIDA LABORAL
La mejor preparación para rendir los exámenes
GARANTIZADA

Windows Presentation Foundation

La nueva forma de crear interfaces

Arquitectura de WPF - El lenguaje XAML

ISBN 978-987-1347-43-8

9 789871 347436 00019