PARTE 1

Introducción

- Introducción a Microsoft .NET
- Mi primera aplicación

CAPÍTULO 1

INTRODUCCIÓN A MICROSOFT .NET

.NET Framework proporciona un entorno de programación orientada a objetos y un entorno de ejecución para construir aplicaciones de escritorio o para la Web. Consta de dos componentes principales: el CLR (*Common Language Runtime*), que es el motor de ejecución que controla las aplicaciones en ejecución, y la biblioteca de clases de .NET Framework, que proporciona una biblioteca de código probado y reutilizable para el desarrollo de aplicaciones, de la que forman parte Windows Forms y WPF, entre otras.

El CLR no es más que una máquina virtual que administra la ejecución del código de una aplicación, de ahí que al código gestionado por esta máquina se le denomine "código administrado", a diferencia del resto de código (código más antiguo que no cumple todas las reglas del CLR pero que es compatible con este), que se conoce como "código no administrado". Este motor de ejecución se puede también hospedar en servidores como Microsoft SQL Server e IIS (*Internet Information Services*).

La biblioteca de clases de .NET es una biblioteca orientada a objetos que permite realizar tareas habituales de programación, como son la administración de cadenas, recolección de datos, conectividad de bases de datos y acceso a archivos, así como desarrollar los siguientes tipos de aplicaciones y servicios:

- Aplicaciones de consola.
- Aplicaciones Windows Forms (aplicaciones que muestran una interfaz gráfica).
- Aplicaciones WPF (aplicaciones que muestran una interfaz gráfica enriquecida).
- Aplicaciones de ASP.NET (aplicaciones para la Web). ASP.NET es una plataforma web que proporciona todos los servicios necesarios para compilar y ejecutar aplicaciones web.
- Aplicaciones de Silverlight. Silverlight es un complemento de Microsoft que nos permite desarrollar aplicaciones enriquecidas para la Web.
- Servicios Windows y servicios web.

PLATAFORMA .NET

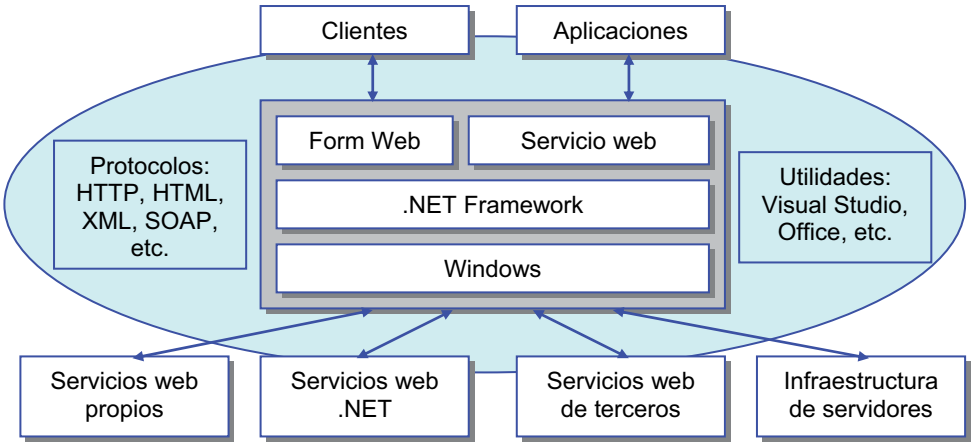
Microsoft .NET extiende las ideas de Internet y sistema operativo haciendo de la propia Internet la base de un nuevo sistema operativo. En última instancia, esto permitirá a los desarrolladores crear programas que transciendan los límites de los dispositivos y aprovechen por completo la conectividad de Internet y sus aplicaciones. Para ello proporciona una plataforma que incluye los siguientes componentes básicos:

- Herramientas de programación para crear los distintos tipos de aplicaciones especificados anteriormente.

- Una infraestructura de servidores; por ejemplo Windows Server, SQL Server, Internet Information Services, etc.

- Un conjunto de servicios (autentificación del usuario, almacén de datos, etc.) que actúan como bloques de construcción para el sistema operativo de Internet. Para entenderlo, compare los servicios con los bloques de Lego; al unir

bloques de Lego se pueden construir soluciones (una casa, un barco, etc.). De la misma forma, la unión de servicios web permite crear soluciones para realizar una tarea concreta.



- Software de dispositivos .NET para hacer posible una nueva generación de dispositivos inteligentes (ordenadores, teléfonos, consolas de juegos, etc.) que puedan funcionar en el universo .NET.

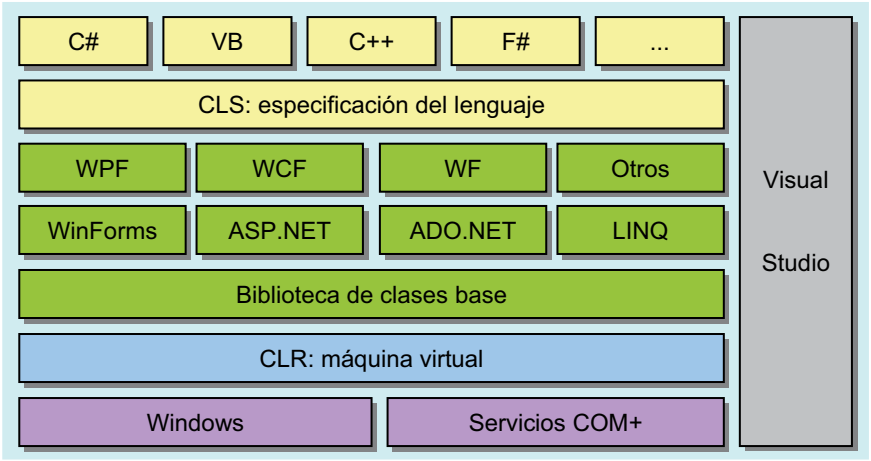
- Experiencias .NET utilizadas por los usuarios finales; por ejemplo, servicios que pueden leer las características del dispositivo que el usuario final está utilizando para acceder y activar así la interfaz más adecuada.

.NET Framework

Claramente, se requiere una infraestructura, no solo para facilitar el desarrollo de aplicaciones, sino también para hacer que el proceso de encontrar un servicio web e integrarlo en una aplicación resulte transparente para usuarios y desarrolladores: .NET Framework proporciona esa infraestructura, según se puede ver en la figura siguiente.

.NET Framework proporciona un entorno unificado para todos los lenguajes de programación. Microsoft ha incluido en este marco de trabajo los lenguajes C#, Visual Basic, C++ y F#, y, además, mediante la publicación de la especificación común para los lenguajes, ha dejado la puerta abierta para que otros fabricantes puedan incluir sus lenguajes (Object Pascal, Perl, Python, Fortran, Prolog, Cobol, PowerBuilder, etc., ya han sido escritos para .NET). Quizás, lo más atractivo de todo esto es la capacidad que ahora tenemos para escribir una misma aplicación utilizando diferentes lenguajes.

Para que un código pueda interactuar con cualquier otro independientemente del lenguaje utilizado, .NET Framework proporciona la "especificación común para los lenguajes" (CLS - *Common Language Specification*) que define las características fundamentales del lenguaje y las reglas de cómo deben ser utilizadas. Por ejemplo, el CLS define un conjunto de tipos de datos comunes (*Common Type System* o CTS) que indica qué tipos de datos se pueden manejar, cómo se declaran y cómo se utilizan. De esta forma, aunque cada lenguaje .NET utilice una sintaxis diferente para cada tipo de datos, por ejemplo, C# utiliza **int** para un número entero de 32 bits y Visual Basic utiliza **Integer**, estos nombres no son más que sinónimos del tipo común **System.Int32**. De esta forma, las bibliotecas que utilicen datos definidos en el CTS no presentarán problemas a la hora de ser utilizadas desde cualquier otro código escrito en la plataforma .NET, permitiendo así la interoperabilidad entre lenguajes.



También, además del CLR, el CLS y los lenguajes, forman parte de este marco de trabajo las bibliotecas que nos permiten crear aplicaciones Windows Forms (*WinForms*) o aplicaciones WPF (*Windows Presentation Foundation*), la plataforma de desarrollo web ASP.NET, la biblioteca para desarrollo de servicios web, la biblioteca ADO.NET o ADO.NET Entity Framework para acceso a bases de datos, la combinación de extensiones al lenguaje y bibliotecas (LINQ y sus proveedores de datos) que permiten expresar como parte del lenguaje consultas a datos, etc. Finalmente, para facilitar el desarrollo de aplicaciones utilizando estas bibliotecas disponemos del entorno de desarrollo integrado Visual Studio.

Aplicaciones de cliente

Para desarrollar aplicaciones basadas en Windows que se ejecuten localmente en los equipos de los usuarios podemos hacerlo mediante Windows Forms o mediante WPF (biblioteca de clases base de .NET para el desarrollo de interfaces gráficas de usuario vectoriales avanzadas).

Un formulario Windows no es más que una ventana que el desarrollador rellena con controles, para crear aplicaciones que muestran una interfaz gráfica al usuario, y con código, para procesar los datos.

Para diseñar aplicaciones que utilicen interfaces gráficas, además de la biblioteca Windows Forms, .NET proporciona otra biblioteca de clases denominada WPF. Esta biblioteca no ha sido creada para sustituir a Windows Forms, sino que simplemente proporciona otras posibilidades para el desarrollo de aplicaciones cliente; por ejemplo, facilita el desarrollo de aplicaciones en el que estén implicados diversos tipos de medios: vídeo, documentos, contenido 3D, secuencias de imágenes animadas, o una combinación de cualesquiera de los anteriores, así como el enlace con los datos.

Aplicaciones web

Podríamos decir que .NET conduce a la tercera generación de Internet, si pensamos que la primera generación consistió en trabajar con información estática que podía ser consultada a través de exploradores como si de un tablón de noticias se tratara, que la segunda generación se basó en que las aplicaciones pudieran interaccionar con las personas (sirva como ejemplo los famosos carros de la compra) y que la tercera generación se ha caracterizado por aplicaciones que puedan interaccionar con otras aplicaciones; por ejemplo, para programar una reunión de negocios, su aplicación de contactos puede interaccionar con su aplicación de calendario, que, a su vez, interaccionará con una aplicación de reserva de billetes para viajar en avión, que consultará a su aplicación preferencias de usuario, por si tuviera que cancelar alguna actividad ya programada.

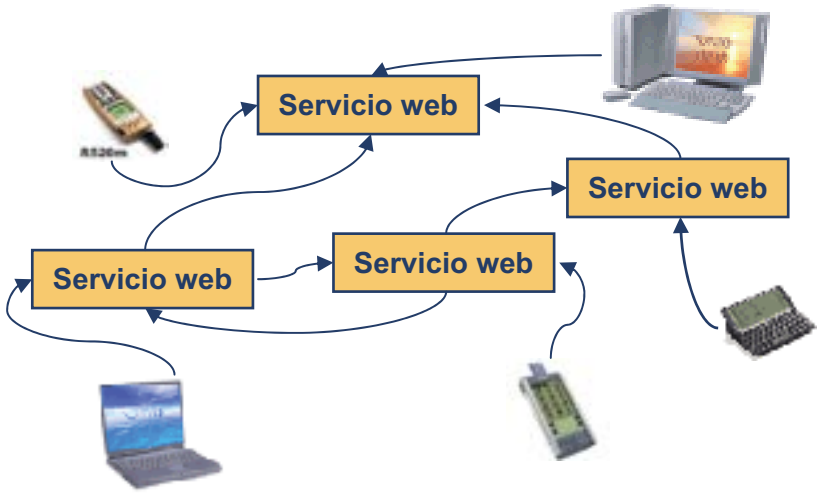
Precisamente, el principio de .NET es que los sitios web aislados de hoy en día y los diferentes dispositivos trabajen conectados a través de Internet para ofrecer soluciones mucho más ricas. Esto se ha conseguido gracias a la aceptación de los estándares abiertos basados en XML (*Extensible Markup Languaje* – lenguaje extensible para describir documentos). De esta manera, Internet se ha convertido en una fuente de servicios, no solo de datos.

XML no es, como su nombre podría sugerir, un lenguaje de marcado; es un metalenguaje que nos permite definir lenguajes de marcado adecuados para usos

determinados. Hay que desterrar ideas como que "XML es HTML mejorado". XML es un estándar para la *descripción* y el *intercambio* de información, principalmente en Internet.

HTML es un lenguaje utilizado para definir la *presentación* de información en páginas web. Gracias a HTML hemos podido combinar texto y gráficos en una misma página y crear sistemas de presentación complejos con hiperenlaces entre páginas. Pero HTML no es útil en lo que se refiere a la descripción de información; XML sí. Por ejemplo, se puede utilizar HTML para dar formato a una tabla, pero no para describir los elementos de datos que componen la misma.

En definitiva, Internet y XML han dado lugar a una nueva fase de la informática en la que los datos del usuario residen en Internet, no en un ordenador personal, y se puede acceder a ellos desde cualquier ordenador de sobremesa, portátil, teléfono móvil o agenda de bolsillo (PDA: *Personal Digital Assistant*). Ello se debe fundamentalmente a que XML ha hecho posible que se puedan crear aplicaciones potentes, para ser utilizadas por cualquiera, desde cualquier lugar. En el corazón del nuevo enfoque de desarrollo está el concepto de servicio web (servicio web XML o WCF). Por ejemplo, en este contexto, el software no se instala desde un CD, sino que es un servicio, como la televisión por pago, al que suscribirse a través de un medio de comunicación.



Un servicio web es una aplicación que expone sus características de manera programática sobre Internet, o en una intranet, utilizando protocolos estándar de Internet como HTTP (*Hypertext Transfer Protocol* – protocolo de transmisión de hipertexto) para la transmisión de datos y XML para el intercambio de los mismos. Pues bien, .NET ha sido desarrollado sobre el principio de servicios web.

Finalmente, hay que decir que para facilitar la creación de aplicaciones web disponemos de la plataforma ASP.NET o bien de la tecnología Silverlight (un competidor directo de Flash).

ADO.NET

ADO.NET (*ActiveX Data Objects* para *.NET*) incluye un conjunto de clases que proporcionan servicio de acceso a bases de datos.

Biblioteca de clases base

.NET Framework incluye clases, interfaces y tipos que aceleran y optimizan el proceso de desarrollo y proporcionan acceso a la funcionalidad del sistema.

Entorno de ejecución común de los lenguajes

.NET Framework proporciona un entorno de ejecución llamado CLR (*Common Language Runtime*; es la implementación de Microsoft de un estándar llamado *Common Language Infrastructure* o CLI, creado y promovido por Microsoft, reconocido mundialmente por el ECMA). Se trata de una máquina virtual que administra la ejecución del código y proporciona servicios que hacen más fácil el proceso de desarrollo (en esencia, estamos hablando de una biblioteca utilizada por cada aplicación .NET durante su ejecución).

El proceso de ejecución de cualquier aplicación incluye los pasos siguientes:

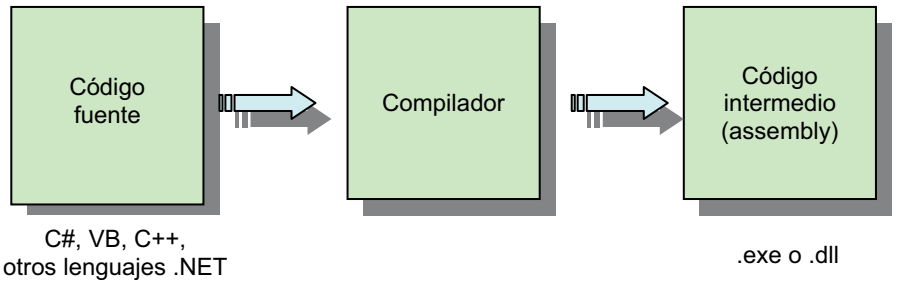
1. Diseñar y escribir el código fuente.
2. Compilar el código fuente a código intermedio.
3. Compilar el código intermedio a código nativo.
4. Ejecutar el código nativo.

Puesto que .NET Framework es un entorno de ejecución multilingüe, soporta una amplia variedad de tipos de datos y de características del lenguaje, que serán utilizadas en la medida que el lenguaje empleado las soporte y que el desarrollador adapte su código a las mismas. Esto es, es el compilador utilizado, y no el CLR, el que establece el código que se utiliza. Por lo tanto, cuando tengamos que escribir un componente totalmente compatible con otros componentes escritos en otros lenguajes, los tipos de datos y las características del lenguaje utilizado deben estar admitidos por la especificación de lenguaje común (CLS).

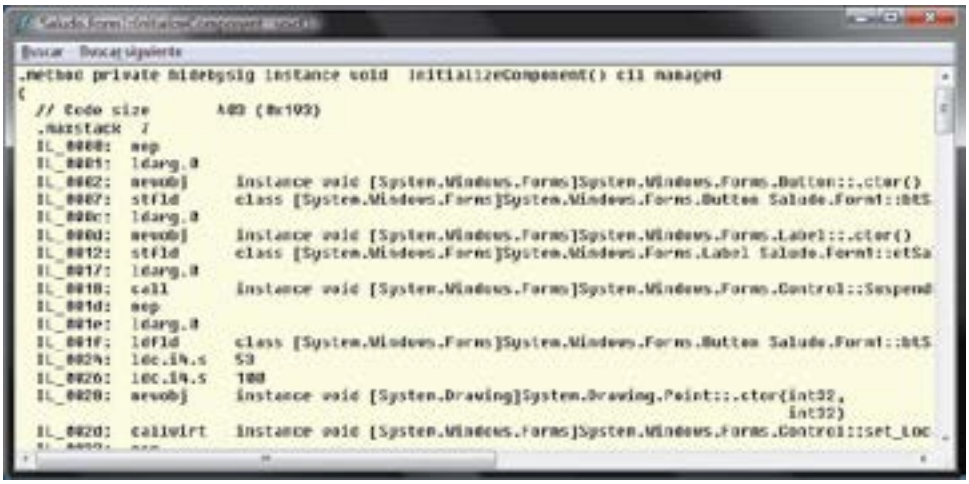
Cuando se compila el código escrito, el compilador lo traduce a un código intermedio denominado MSIL (*Microsoft Intermediate Language*) o simplemente IL, correspondiente a un lenguaje independiente de la unidad central de proceso

(UCP). Esto quiere decir que el código producido por cualquier lenguaje .NET puede transportarse a cualquier plataforma (Intel, Sparc, Motorola, etc.) que tenga instalada una máquina virtual de .NET y ejecutarse. Pensando en Internet esta característica es crucial ya que esta red conecta ordenadores muy distintos.

IL incluye instrucciones para cargar, iniciar y llamar a los métodos de los objetos, así como para operaciones aritméticas y lógicas, control del flujo, acceso directo a memoria, manipulación de excepciones y otras operaciones.



La siguiente figura muestra el aspecto que tiene el código intermedio de una aplicación. Este código puede obtenerlo a través del desensamblador *ildasm.exe* que viene con la plataforma .NET.

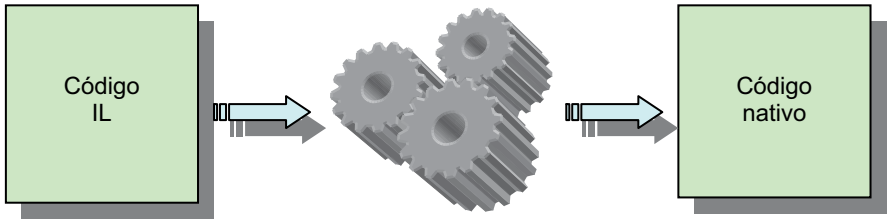


Cuando el compilador produce IL también produce *metadatos*: información que describe cada elemento manejado por el CLR (tipo, método, etc.). Esto es, el código a ejecutar debe incluir esta información para que el CLR pueda proporcionar servicios tales como administración de memoria, integración de múltiples lenguajes, seguridad, control automático del tiempo de vida de los objetos, etc. Tanto el código intermedio como los metadatos son almacenados en un fichero ejecuta-

ble y portable (.exe o .dll), denominado *ensamblado* (*assembly* en inglés), que permite que el código se describa a sí mismo, lo que significa que no hay necesidad de bibliotecas de tipos o de lenguajes de definición de interfaces.

Un *ensamblado* es la unidad fundamental de construcción de una aplicación .NET y básicamente incluye dos partes diferenciadas: el *manifiesto* y el código IL. El manifiesto incluye los metadatos que describen completamente los componentes en el *ensamblado* (versión, tipos, dependencias, etc.) y el código describe el proceso a realizar.

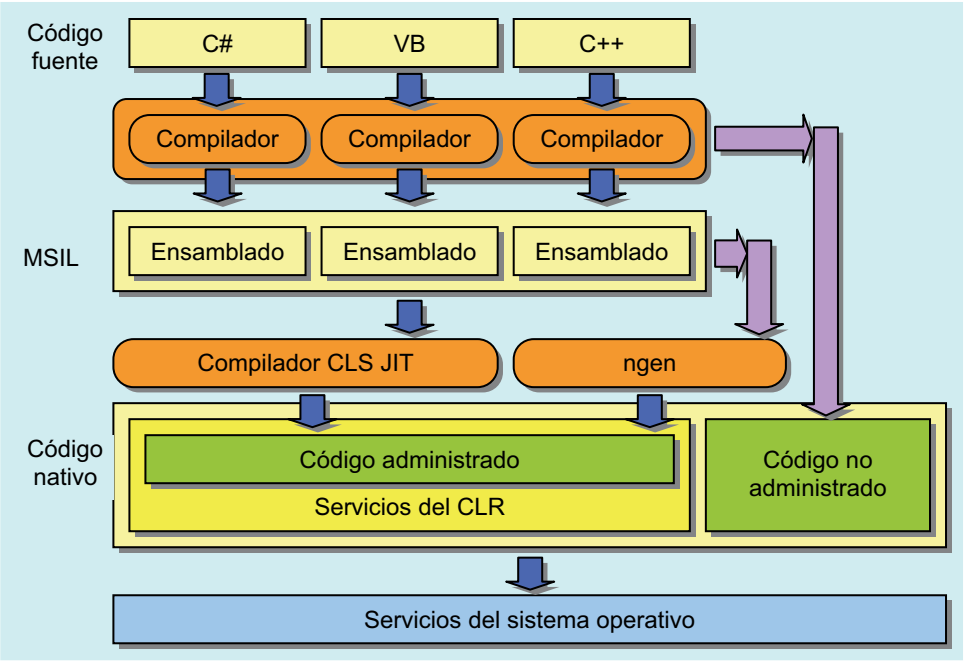
Antes de que el código intermedio pueda ser ejecutado, debe ser convertido por un compilador JIT (*Just in Time*: al instante) a código nativo, que es código específico de la UCP del ordenador sobre el que se está ejecutando el JIT.



La máquina virtual no convierte todo el código MSIL a código nativo y después lo ejecuta, sino que lo va convirtiendo bajo demanda con el fin de reducir el tiempo de ejecución; esto es, cada método es compilado a código nativo cuando es llamado por primera vez para ser ejecutado, y el código nativo que se obtiene se guarda para que esté accesible para subsiguientes llamadas. Este código nativo se denomina "código administrado" si cumple la especificación CLS, en otro caso recibe el nombre de "código no administrado".

La máquina virtual (el CLR) proporciona la infraestructura necesaria para ejecutar el código administrado así como también una variedad de servicios que pueden ser utilizados durante la ejecución (administración de memoria –incluye un recolector de basura para eliminar un objeto cuando ya no esté referenciado–, seguridad, interoperatividad con código no administrado, soporte multilenguaje para depuración, soporte de versión, etc.).

El código no administrado es código creado sin tener en cuenta la especificación de lenguaje común (CLS). Este código se ejecuta con los servicios mínimos del CLR (por ejemplo, sin recolector de basura, depuración limitada, etc.). Los componentes COM, las interfaces ActiveX y las funciones de la API Win32 son ejemplos de código no administrado.



La utilidad *ngen.exe* (*Native Image Generator*) es un generador de código nativo. Permite crear una imagen en código nativo de los *ensamblados* especificados: `ngen [opciones] [ensamblado [...]]`.

.NET Framework y COM+

El software reutilizable no es una idea nueva. El modelo COM (*Component Object Model*) introdujo el concepto de "componente": un fragmento de código reutilizable en cualquier aplicación Windows. OLE (*Object Linking and Embedding*) fue el primer lugar en el que los desarrolladores experimentaron COM.

El problema con COM, desde una perspectiva de desarrollo, era que requería escribir muchas interfaces para convertir la lógica de negocio de una aplicación en un componente reutilizable. Además de eso, COM también obligaba a los desarrolladores a manejar manualmente complejidades como limpieza de la memoria cuando un componente no se va a utilizar más, recuento del número de veces que un componente está en uso, establecimiento y eliminación de hilos y procesos, y manejo de versiones, lo que se traducía en errores de aplicación, fallos de sistema y el notorio "infierno de las DLL". Por otra parte, el escribir esta infraestructura COM no permitía a los desarrolladores centrarse en la lógica de negocio.

Uno de los objetivos iniciales de .NET Framework fue hacer el desarrollo de COM más fácil, automatizando todo lo que es y supone actualmente COM, incluido el recuento de referencias, descripción de la interfaz y registro. En el caso

de componentes del .NET Framework, el CLR automatiza esas características; los componentes se describen a sí mismos y pueden ser por tanto instalados sin necesidad de inscribirlos en el registro de Windows.

COM+ es el nombre de COM tras combinarlo con MTS (*Microsoft Transaction Server*) y DCOM (*Distributed COM*). Proporciona un conjunto de servicios orientados principalmente al desarrollo de aplicaciones para la capa media de una aplicación (capa de lógica de negocio; las otras dos capas son la de presentación y la de datos), enfocados a proporcionar fiabilidad y escalabilidad para aplicaciones distribuidas de gran escala. Estos servicios son complementarios de los servicios de programación del .NET Framework, ya que las clases del .NET Framework proporcionan acceso directo a ellos.

El problema con COM+ (igual que con CORBA o RMI) es que no se pueden escalar a Internet. El acoplamiento entre el servicio y el consumidor del servicio es muy estrecho, lo que significa que si la implementación en un lado cambia, el otro lado falla. Para evitar esto, con .NET Framework los servicios web se acoplan de manera suelta. Técnicamente, esto se traduce en utilizar una tecnología asíncrona, basada en mensajería utilizando protocolos web y XML.

Los sistemas de mensajes envuelven las unidades fundamentales de comunicación (los mensajes) en paquetes que se autodescriben y que los propios sistemas ponen en la red, con la única suposición de que los receptores los entenderán. En cambio, con un sistema de objetos distribuidos, el emisor hace muchas suposiciones acerca del receptor, como activar la aplicación, llamar a sus interfaces, apagar la aplicación, etc.

Visual Studio

Visual Studio es un conjunto completo de herramientas de desarrollo para construir aplicaciones web, servicios web, aplicaciones Windows o de escritorio y aplicaciones para dispositivos móviles. El entorno de desarrollo integrado que ofrece esta plataforma con todas sus herramientas y con la biblioteca de clases .NET Framework es compartido en su totalidad por Visual C#, Visual Basic y Visual C++, permitiendo así crear con facilidad soluciones en las que intervengan varios lenguajes y en las que el diseño se realiza separadamente respecto a la programación.

CAPÍTULO 2

MI PRIMERA APLICACIÓN

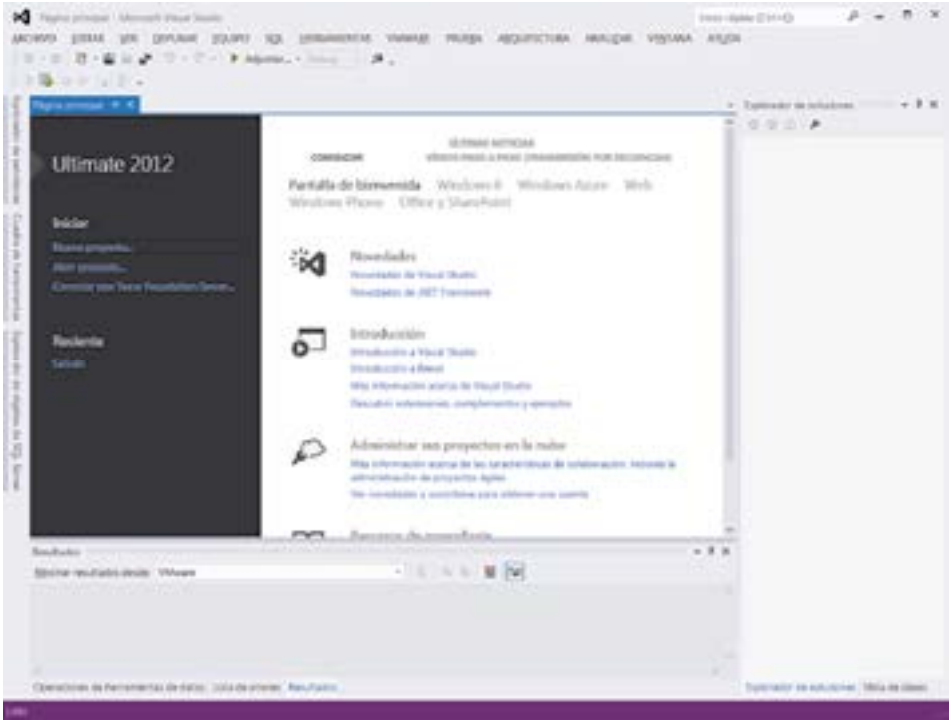
Se puede desarrollar una aplicación que muestre una interfaz gráfica utilizando como herramientas *Microsoft .NET Framework SDK* (proporciona, entre otras cosas, la biblioteca de clases .NET y el compilador de C#) y un simple editor de texto, o bien utilizando un entorno de desarrollo integrado (EDI). En el primer caso hay que escribir el código fuente línea a línea, para después, desde la línea de órdenes, compilarlo, ejecutarlo y depurarlo. Lógicamente, escribir todo el código necesario para crear la interfaz gráfica de la aplicación es una tarea repetitiva que, de poder mecanizarse, ahorraría mucho tiempo en la implementación de una aplicación y permitiría centrarse más y mejor en resolver los problemas relativos a su lógica y no a su aspecto. Justamente esto es lo que aporta *Visual Studio*, o bien, en su defecto, la versión de *Visual C# Express*.

MICROSOFT VISUAL STUDIO

Visual Studio permite diseñar la interfaz gráfica de una aplicación de manera visual, sin más que arrastrar con el ratón los controles que necesitemos sobre la ventana destino de los mismos. Una rejilla o unas líneas de ayuda mostradas sobre la ventana nos ayudarán a colocar estos controles y a darles el tamaño adecuado, y una página de propiedades nos facilitará la modificación de los valores de las propiedades de cada uno de los controles. Todo lo expuesto lo realizaremos sin tener que escribir ni una sola línea de código. Después, un editor de código inteligente nos ayudará a escribir el código necesario y detectará los errores sintácticos que introduzcamos, y un depurador nos ayudará a poner a punto nuestra aplicación cuando lo necesitemos.

Como ejemplo, vamos a realizar una aplicación Windows denominada *Saludo*, que presente una interfaz al usuario como la de la figura siguiente:

Para empezar, arranque Visual Studio o, en su defecto, Visual C# Express. Se visualizará una ventana como la siguiente:



¿Cuáles son los siguientes pasos para desarrollar una aplicación Windows? En general, para construir una aplicación de este tipo con Visual Studio, siga los pasos indicados a continuación:

1. Cree un nuevo proyecto (una nueva aplicación), entendiendo por proyecto un conjunto de elementos en forma de referencias, conexiones de datos, carpetas y ficheros necesarios para crear la aplicación (mientras que un proyecto normalmente contiene varios elementos, una solución puede contener varios pro-

yectos). Una vez creado el proyecto/solución, Visual Studio mostrará una página de diseño con un formulario vacío por omisión.

2. Dibuje los controles sobre el formulario. Los controles serán tomados de una caja de herramientas.

3. Defina las propiedades del formulario y de los controles.

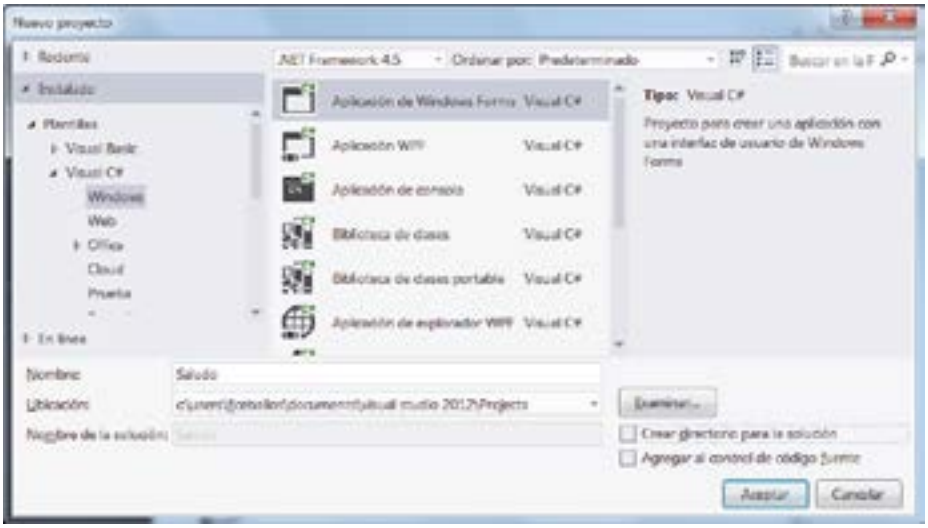
4. Escriba el código para controlar los eventos que considere de cada uno de los objetos.

5. Guarde, compile y ejecute la aplicación.

6. Opcionalmente, utilice un depurador para poner a punto la aplicación.

Crear un nuevo proyecto

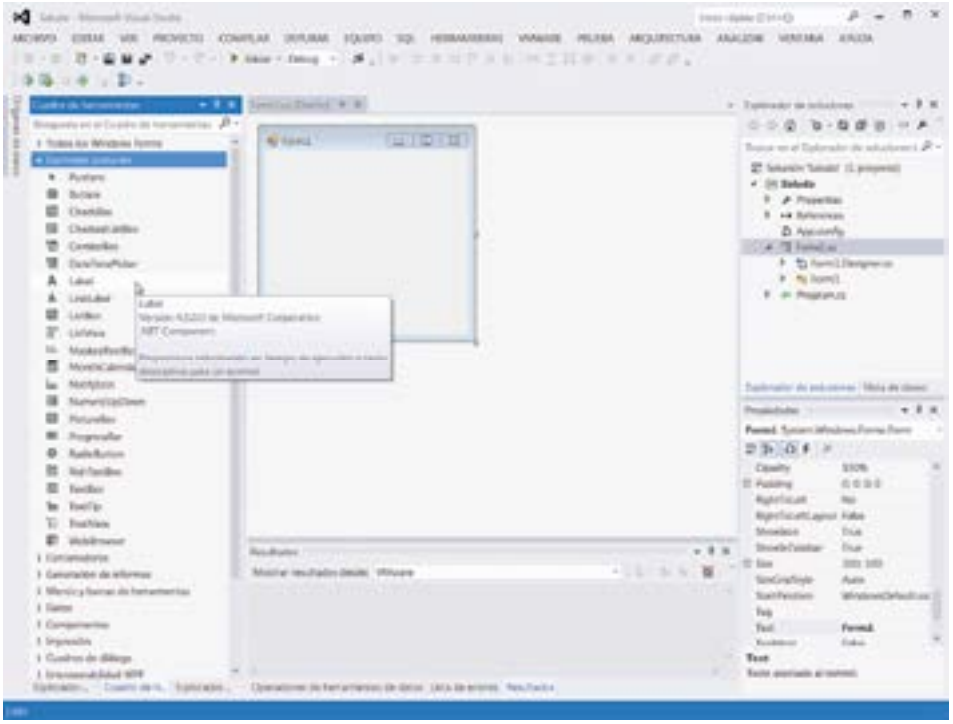
Para crear un nuevo proyecto, diríjase a la barra de menús y ejecute *Archivo > Nuevo proyecto*. En el diálogo que se visualiza, seleccione el tipo de proyecto *Visual C# > Windows*, la plantilla *Aplicación de Windows Forms*, el nombre *Saludo* y haga clic en el botón *Aceptar*:



Obsérvese que se ha elegido la carpeta donde se almacenará el proyecto. Esta tarea puede posponerse sin que afecte al desarrollo de la aplicación. Ahora bien, si desea que esta tarea se realice automáticamente en el momento de crear el proyecto, caso del autor, ejecute *Herramientas > Opciones*, seleccione la opción *Proyectos y soluciones* y marque la casilla *Guardar los proyectos nuevos al crearlos*.

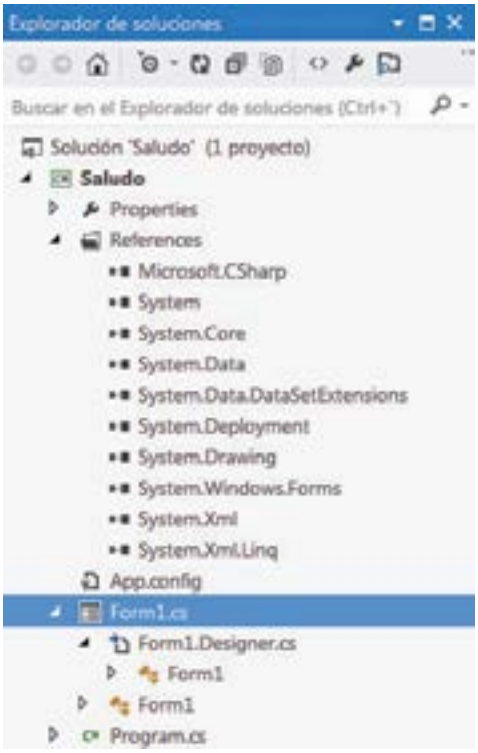
También, en la parte superior de la ventana hay una lista que le permitirá seleccionar la versión de .NET Framework que desea utilizar.

Después de crear una nueva aplicación Windows, el entorno de desarrollo Visual Studio mostrará un formulario, *Form1*, en el diseñador. También pondrá a nuestra disposición una caja de herramientas con una gran cantidad de controles listos para ser incluidos en un formulario.



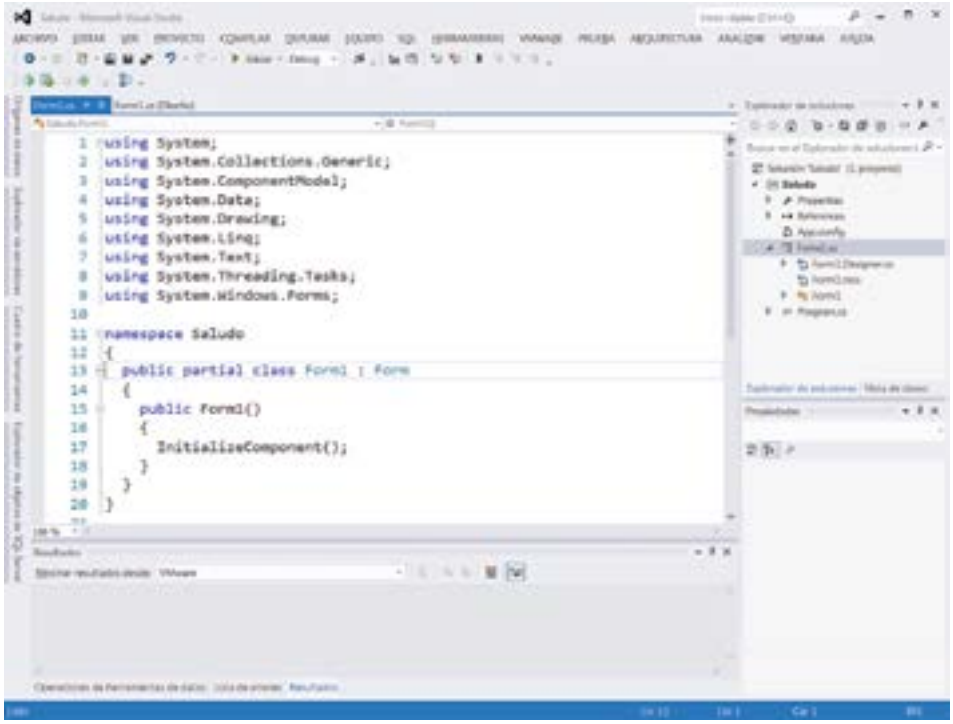
Otra característica interesante de este entorno de desarrollo es la ayuda dinámica que facilita. Se trata de un sistema de ayuda sensible al contexto; esto es, automáticamente se mostrará la ayuda relacionada con el elemento seleccionado o ayuda para completar el código mientras lo escribimos. Por ejemplo, observe en la ventana de *Propiedades*, en la esquina inferior derecha, la ayuda relativa a la propiedad seleccionada.

En la esquina superior derecha también se localiza otra ventana con varias páginas: explorador de soluciones, vista de clases, etc.; en la figura siguiente vemos el *Explorador de soluciones*:



El explorador de soluciones muestra el nombre de la solución (una solución engloba uno o más proyectos), el nombre del proyecto (un proyecto administra los ficheros que componen la aplicación) y el de todos los formularios y módulos; en nuestro caso, observamos un formulario, denominado *Form1*, descrito por los ficheros de código *Form1.cs* y *Form1.Designer.cs*; el primero es el utilizado por el programador para escribir el código y el segundo, el utilizado por el diseñador de formularios. También se observa un nodo *References* que agrupa las referencias a las bibliotecas de clases de objetos que utilizará la aplicación en curso; podemos añadir nuevas referencias a otras bibliotecas haciendo clic con el botón secundario del ratón sobre ese nodo.

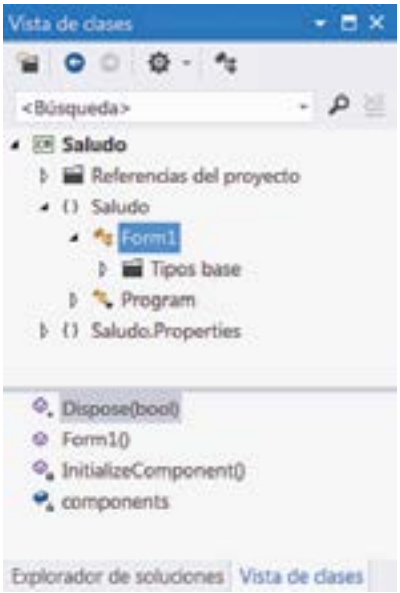
Así mismo, en su parte superior, muestra una barra de botones que permiten ver el código, mostrar todos los archivos, la ventana de propiedades, etc. Por ejemplo, si estamos viendo el diseñador de formularios y hacemos clic en el botón *Ver código*, la página de diseño será sustituida por el editor de código, como se puede observar en la figura siguiente:



Una característica digna de resaltar del editor de Visual Studio es la incorporación de bloques de código contraíbles. En la figura superior podemos ver varios de estos bloques; si hacemos clic en el nodo –, contraeremos el bloque y el nodo se convertirá en otro + que permitirá expandir de nuevo el bloque.

Otra característica del editor es la finalización y el formato de código automáticos. Por ejemplo, al escribir un método, el editor mostrará automáticamente la ayuda en línea de la palabra clave (**public**, **void**, **int**, etc.) que intenta escribir; si escribimos una sentencia **if**, exactamente igual. Puede personalizar las características del editor ejecutando *Herramientas* > *Opciones* > *Editor de texto*.

Si cuando se está visualizando el explorador de soluciones desea mostrar la vista de clases de su aplicación, ejecute la opción *Vista de clases* del menú *Ver*, o bien haga clic en la pestaña *Vista de clases* si esta está presente. Esta ventana, en su parte superior, muestra las clases que componen la aplicación, y en su parte inferior, los métodos pertenecientes a la clase seleccionada.



Expandiendo el nodo del proyecto, vemos, en primer lugar, el espacio de nombres al que pertenecen los elementos de la aplicación: *Saludo* (un espacio de nombres define un ámbito). Si ahora expandimos este otro nodo, veremos que incluye una clase, la que define el objeto *Form1*, y si expandimos a su vez este nodo, podremos observar su clase base. En la figura podemos ver seleccionada la clase *Form1*, que define un constructor, *Form1*, y los métodos *Dispose*, *InitializeComponent* y *components*.

El formulario

El formulario es el plano de fondo para los controles. Después de crear un nuevo proyecto, la página de diseño muestra uno como el de la figura siguiente. Lo que ve en la figura es el aspecto gráfico de un objeto de la clase *Form1*. Para modificar su tamaño ponga el cursor del ratón sobre alguno de los lados del cuadrado que lo rodea y arrastre en el sentido deseado.

Si ahora ejecutamos este programa, para lo cual podemos pulsar las teclas *Ctrl+F5* o elegir la orden correspondiente del menú *Depurar*, aparecerá sobre la pantalla la ventana, con el tamaño asignado, y podremos actuar sobre cualquiera de sus controles, o bien sobre las órdenes del menú de control, para minimizarla, maximizarla, moverla, ajustar su tamaño, etc. Esta es la parte que el diseñador de Visual C# realiza por nosotros y para nosotros; pruébelo. Finalmente, para cerrar la ejecución de la aplicación disponemos de varias posibilidades:

1. Hacer clic en el botón que cierra la ventana.
2. Hacer un doble clic en el icono situado a la izquierda en la barra de título de la ventana.
3. Activar el menú de control de la ventana *Form1* y ejecutar *Cerrar*.
4. Pulsar las teclas *Alt+F4*.

Dibujar los controles

En Visual C# disponemos fundamentalmente de dos tipos de objetos: *ventanas* y *controles*. Las ventanas son los objetos sobre los que se dibujan los controles como cajas de texto, botones o etiquetas, dando lugar a la interfaz gráfica que el usuario tiene que utilizar para comunicarse con la aplicación y que genéricamente denominamos “formulario”.

Para añadir un control a un formulario, utilizaremos la caja de herramientas que se muestra en la figura siguiente. Cada herramienta de la caja crea un único control. El significado de los controles más comunes se expone a continuación.

Puntero. El puntero no es un control. Se utiliza para seleccionar, mover y ajustar el tamaño de los objetos.