UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ACATLÁN

DESARROLLO DE APLICACIONES EMPRESARIALES CON ASP. NET Y EL LENGUAJE DE PROGRAMACIÓN C#

T E S I N A

QUE PARA OBTENER EL TÍTULO DE:

LIC. EN MATEMÁTICAS APLICADAS Y COMPUTACIÓN

P R E S E N T A:

JUVENAL HERNÁNDEZ CARRILLO

ASESOR: OSCAR GABRIEL CABALLERO MARTÍNEZ

FECHA: JUNIO 2005

m. 345073

Universidad Nacional Autónoma de México
UNAM
Biblioteca Central
Dirección General de Bibliotecas de la UNAM

UNAM – Dirección General de Bibliotecas
Tesis Digitales
Restricciones de uso

DERECHOS RESERVADOS ©
PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.


U.N.A.M.
FES
ACATLÁN
JUN 3 2005
DEPTO. DE TÍTULOS
UNIDAD DE ADMINISTRACIÓN ESCOLAR

AGRADEZCO

A mis padres y hermanos por su afecto y apoyo
incondicional durante todos estos años.

A mis amigos por estar conmigo en todo momento.

A todos mis maestros por sus sabias enseñanzas,
sin las cuales no hubiera sido posible realizar
el presente trabajo ni otros proyectos personales.

Índice de contenido.

Capitulo V - Seguridad

Convenios de notación.

Para ayudar a resaltar la información clave se utilizan diferentes convenciones respecto a los tipos de letra usados para representar cada tipo de contenido.

- **El texto que señala un tema general se ha escrito usando la fuente Arial de 12 puntos de tamaño, en negrita y subrayado.**

- **El texto que señala un tema particular se ha escrito usando la fuente Arial de 12 puntos de tamaño, en negrita.**

- ***El texto que señala un subtema se ha escrito usando la fuente Arial de 11 puntos de tamaño, en negrita y cursiva.***

- *El texto que señala un concepto o término se ha escrito usando la fuente Arial de 11 puntos de tamaño, en cursiva.*

- El texto correspondiente a explicaciones se ha escrito usando la fuente Arial de 10 puntos de tamaño, como es el caso de este párrafo.

- Los fragmentos de código fuente, las referencias a textos de la interfaz del sistema operativo (nombres de archivo y directorios, texto de la línea de comandos, etc.) se han escrito usando la fuente Courier New de 8 puntos de tamaño tal y como se muestra a continuación:

```
class HolaMundo
{
        static void Main()
        {
                System.Console.WriteLine("¡Hola Mundo!");
        }
}
```

  Esta misma fuente es la que se usará desde las explicaciones cada vez que se haga referencia a algún elemento del código fuente.

- El texto que señala el titulo de una Figura o Tabla se señalara en negritas y el texto que lo explica en cursiva, ambos escritos usando la fuente Arial de 10 puntos de tamaño, como se muestra a continuación:

Figura 2.1 *Estructura del Framework .NET.*

- El texto que señala una nota al pie de página se ha escrito usando la fuente Arial de 9 puntos de tamaño.

Introducción.

En versiones tempranas del cómputo cliente-servidor, las aplicaciones de software se distribuían por medio de programas que eran instalados por separado en cada computadora personal con el fin de proporcionar una interfaz de usuario al cliente. Generalmente una actualización a la aplicación del lado del servidor requería también una actualización en todos los clientes, lo cual resultaba costoso e ineficiente.

En contraste a esto, las aplicaciones Web son distribuidas a los usuarios desde un servidor Web sobre la red, así como la World Wide Web o una Intranet, a través de una serie de páginas Web en un formato estándar de documento HTML como interfaz de usuario. Las aplicaciones Web emplean capacidades de cómputo y tecnología de Internet directamente para realizar una tarea específica que cumple las necesidades del usuario y en beneficio del usuario.

Cada página Web individual es técnicamente un documento estático. Pero cuando se ligan una secuencia de páginas proporciona una experiencia interactiva; la entrada del usuario se ingresa a través de formularios Web dentro de elementos incrustados en la página para ser procesadas por el servidor y regresar una respuesta. Durante la sesión, el Web browser interpreta y despliega las páginas, que actúan como un cliente universal para cualquier aplicación Web. Así, la habilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software en cientos de computadoras cliente es otra de las razones por lo cual son importantes y populares.

El objetivo de las aplicaciones Web es ofrecer un servicio a cada uno de los usuarios que se conectan a las mismas. Las aplicaciones empresariales, abarcan dos grandes grupos de negocios: los que se llevan a cabo entre empresas y los que tienen por objetivo los consumidores. Los primeros se designan por B2B (business-to-business), y los segundos por B2C (business-to-consumer). Estas aplicaciones que utilizan tecnologías de Internet tienen como interfaz de comunicación con el usuario una aplicación Web.

La importancia de las aplicaciones empresariales puede resumirse en gran parte al éxito de Internet. Internet ha llegado a más gente en menos tiempo que cualquier otra tecnología anterior. La capitalización de empresas que exponen sus servicios en Internet ha incrementado sus ganancias de forma considerable. Los clientes tienen acceso a mayor información sobre productos y servicios en un mercado mucho más competitivo. Las organizaciones se desmontan para ser más competitivas en un mundo de empresas interconectadas. Por ejemplo, en una aplicación de tipo B2C la mayoría de los productos o servicios que un consumidor necesita, pueden adquirirse en Internet accediendo a un portal de comercio electrónico. En una aplicación de tipo B2B las empresas pueden negociar con sus proveedores y coordinar los suministros disminuyendo los costos de transacción.

Debido a la importancia de las aplicaciones Web surge la necesidad de utilizar tecnología para generar contenido dinámico de páginas Web. Tradicionalmente, la programación dinámica de páginas Web ha sido creada utilizando programas ejecutables del lado del servidor, para lo cual se necesitaba grandes cantidades de recursos de cómputo en el servidor Web. En respuesta a estos y otros problemas surge un producto ampliamente aceptado para el desarrollo de aplicaciones Web: ASP (*Active Server Pages*).

Desde su concepción, la tecnología ASP se volvió bastante popular debido a la facilidad con la que se obtienen aplicaciones usando VBScript; un lenguaje de programación de

secuencia de comandos similar a Visual Basic, el cuál ha sido adoptado por un gran número de desarrolladores en el mundo, incluyendo nuestro país.

Sin embargo, también acarreaba algunos problemas. Era sumamente fácil de programar, pero igualmente difícil de mantener. Fue por eso que se empezó a trabajar en una nueva tecnología que corrigiera los problemas de la versión anterior, sin sacrificar la simplicidad del desarrollo, agregando la introducción de un nuevo lenguaje de programación más eficiente y enfocado a sistemas basados en componentes. Así fue como surgió ASP.NET y C#.

El presente trabajo expone las características y ventajas que ofrece la tecnología ASP.NET, el cual va dirigido a desarrolladores de versiones previas que quieran actualizar sus conocimientos, a desarrolladores que estén pensando en migrar sus aplicaciones, a Arquitectos de software y Diseñadores de sistemas interesados en explorar los conceptos de esta tecnología para aplicarlos a una solución particular; y finalmente a estudiantes y profesores como material de apoyo y de consulta para una materia en particular, los cuales pueden tomar un rol de los mencionados anteriormente en algún momento de su vida profesional.

ASP.NET puede usar cualquier lenguaje de programación que sea compatible con la plataforma .NET, por lo que no es necesario encasillarse en aprender exclusivamente C#. Es utilizado en el presente trabajo con el fin de ejemplificar los conceptos mediante código para una mayor comprensión. Sin embargo, cuestiones más detalladas sobre el lenguaje deberán ser consultadas de otras fuentes.

Una aplicación Web típica, puede resumirse técnicamente en su forma más simple como una interfaz de usuario puesta encima de un programa que utiliza la base de datos para leer y escribir información publicada a través del servidor Web con determinados permisos y restricciones sobre lo que se puede hacer en la aplicación. Es por eso que se revisan las herramientas que proporciona ASP.NET y otras tecnologías dentro del Framework .NET para cumplir este objetivo.

Con la finalidad de que el texto sea lo más compresible para el lector, se tratará de utilizar un lenguaje simple y entendible. Sin embargo, cierta terminología requiere de conocimientos previos en áreas de sistemas computacionales debido a su carácter técnico. Además, se ha procurado que cada capítulo contenga una introducción del tema a tratar, desarrollo del tema, y un ejemplo de los conceptos traducidos en código cuando se requiera.

Por último, he tratado de avocar la información del texto a un escenario más real de trabajo como profesionista, el cual puede ser empleado como texto de estudio más allá de fines académicos.

Objetivo.

Exposición del conjunto de tecnologías con el nombre de ASP.NET como herramienta de desarrollo de aplicaciones empresariales utilizando el lenguaje de programación C#.

Introducción a ASP.NET.

Este capítulo es una introducción al entorno .NET donde se analizan las ventajas de sus lenguajes, arquitectura, filosofía, y características principales. También se justificará el porqué la necesidad de una nueva versión de ASP, y se exploran los conceptos detrás de su diseño e implementación. Esta información es de gran utilidad para desarrolladores de versiones previas de ASP que quieran migrar sus aplicaciones a ASP.NET.

ASP.NET esta diseñado para ser compatible con versiones previas de ASP. Se puede instalar ASP.NET en un servidor con Windows 2000 (o posterior) y ASP 3.0. Esto nos permite experimentar con la nueva versión, sin requerir un servidor independiente de pruebas. Podemos continuar usando aplicaciones ASP existentes, y migrarlas a ASP.NET cuando se requiera.

Si decidimos migrar aplicaciones a ASP.NET, esto nos dará poco de los beneficios que la nueva versión nos ofrece. ASP.NET tiene nuevas características que proporcionan mayor facilidad de uso, mas poder y mejor eficiencia en tiempo de ejecución, pero para tomar ventaja de ello necesitamos entender mas acerca de la manera en que trabaja ASP.NET.

La evolución de Active Server Pages.

Desde su concepción en el año de 1996, ha crecido rápidamente y ha venido a ser la mejor técnica para la programación Web en ambiente Windows (y en algunas otras plataformas utilizando otras implementaciones que acepten la misma sintaxis o similar).

El problema: desarrollar aplicaciones Web dinámicas.

Por contenido dinámico en páginas Web, nos referimos al contenido confeccionado para el usuario final en una vista particular. El contenido en la página permite la comunicación en ambas direcciones cliente-servidor. Al utilizar un formulario en una página Web, un usuario puede enviar peticiones de contenido personalizado. Las comunicaciones entre usuario y servidor incluyen algo más que solamente formularios y contenido personalizado, como las cookies, o pequeños trozos de información que deberían grabarse en la máquina del usuario para ayudar a identificarlo más tarde en la sesión y visitas posteriores al sitio Web.

La programación de cliente utilizando JavaScript u otro lenguaje de secuencia de comandos normalmente no es suficiente para crear páginas Web totalmente dinámicas. Una utilización particularmente efectiva de esto es proporcionar validación de cliente sin requerir un viaje de ida y vuelta al servidor.

Programación dinámica Web.

A mediados de los años noventa, muchas compañías estaban bajo creciente presión para bajar los costos totales de sus aplicaciones. Las aplicaciones tradicionales basadas en "cliente pesado", con bibliotecas de vínculos dinámicos (DLL) y configuración de registro, estaban convirtiéndose en una parte creciente de este costo. Muchas compañías vieron las aplicaciones Web como una manera de distribuir rápidamente aplicaciones de misión crítica a través de la empresa con un mínimo impacto en las máquinas clientes. A

continuación se describen diferentes técnicas para crear contenido dinámico con sus pros y sus contras.

Interfaz de pasarela común (CGI).

Tradicionalmente, la programación dinámica de páginas Web han sido creadas utilizando programas ejecutables del lado del servidor. Una interfase de servidor Web estandarizada llamada CGI (*Common Gateway Interface*) permite a un programa ejecutable acceder a la información y recursos para peticiones entrantes de clientes. El programa puede entonces generar toda la salida requerida para crear la página de retorno (el HTML[1], código script, texto, etc.), y enviarlo de regreso al cliente vía el servidor Web.

Para hacer la vida del desarrollador más fácil, y librarlos de crear programas ejecutables, lenguajes como Perl utilizan una aplicación que acepta archivos script basados en texto. El desarrollador simplemente escribe el script, y el servidor Web los ejecuta utilizando un interprete de Perl.

El modelo CGI es bueno porque una vez que se ejecuta el programa y se termina, se puede modificar o eliminar como cualquier otro programa; sin embargo, esta capacidad es el centro del problema del CGI. Cuando se ejecuta un programa de CGI, se carga en la memoria, y cuando finaliza el programa, se elimina completamente de la memoria. Se asocia mucho trabajo con crear y destruir procesos. Crear un proceso es una operación relativamente cara. Esta creación y destrucción de procesos para cada solicitud acarrea una escasez de recursos. Si hay cien clientes accediendo al mismo programa CGI, habrá cien instancias de ese programa en la memoria. Esto puede acabar rápidamente con los recursos de un servidor Web y causar problemas de escalabilidad.

Interfaz de programación de aplicaciones de servidor para Internet (ISAPI).

Para una mejor comprensión de ASP y de ASP.NET se requiere entender ISAPI (*Internet Server Application Programming Interface*).

Microsoft desarrolló una nueva manera de construir aplicaciones escalables con su propio servidor de Web, Internet Information Server. Esta alternativa es ISAPI, y difiere del CGI al permitir código compilado dentro de una librería de enlace dinámico (DLL) para ser ejecutada directamente por el servidor Web. Como sucede con el CGI, el código puede acceder a la información y recursos en las peticiones del cliente, y generar la salida para la página de retorno.

Uno de los primeros productos con tiempo de vida corto fue *dbWeb*, una tecnología de acceso a datos que suministraba capacidades de búsqueda, filtrado, y formateo para acceso a datos almacenados en el servidor, y para interactuar con el cliente.

Un segundo desarrollo fue el *Internet Database Conector* (IDC). Este tuvo un gran éxito con desarrolladores, no únicamente por su rapidez y eficiencia (lo contrario de dbWeb), sino porque era mucho más fácil de programar. IDC introdujo el concepto de *plantillas*, permitiendo a desarrolladores adaptar fácilmente páginas HTML existentes para usar sus características y rápidamente construir nuevas aplicaciones alrededor de este concepto.

[1] HTML es un lenguaje de marcado de hipertexto, el cuál es un sistema de escritura y despliegue de texto que permite vincular diversos documentos relacionados, comúnmente utilizado para crear páginas Web.

IDC utiliza dos archivos de texto por cada página. El primero es un script simple que define la manera en que los datos deberán ser recolectados del servidor de base de datos. En esencia, esto es solo un enunciado SQL con información adicional de configuración.

El servidor ejecuta este archivo para obtener los resultados en un resultset[2], y luego carga el archivo de plantilla. La plantilla es sólo una página Web común, que incluye HTML, texto y otros objetos, pero con uno o mas delimitadores especiales insertados. En estos delimitadores se programa la salida del texto con la información obtenida del archivo de consulta.

Los problemas con ISAPI están casi todos asociados con el desarrollo de la aplicación ISAPI. Desarrollar una aplicación ISAPI requiere de un desarrollador que esté familiarizado al menos con C++, la biblioteca de clases Microsoft Foundation (*Microsoft Foundation Class Library*, MFC) y HTML. El desarrollo de ISAPI no puede dividirse fácilmente entre el núcleo de la aplicación y los detalles de la presentación. ISAPI tiene una única y monolítica DLL, y sin proporcionar su propia secuencia de comandos a la medida, no hay una forma fácil de que el diseñador de interfaz de usuario HTML y el diseñador de la lógica de negocio interna lleven a cabo tareas independientes.

El segundo problema con el desarrollo de las aplicaciones ISAPI aparece al probar las compilaciones de la DLL. Por defecto, la aplicación ISAPI se carga en la memoria y se mantiene ahí hasta que el servicio de publicación Web se detiene. Además, depurar la DLL es un proceso complicado en un entorno de desarrollo rudimentario.

Las versiones de ASP.

En inicios de 1996, *Denali* (el nombre código de ASP) fue liberado como un producto en su versión beta 0.9. La habilidad de ejecutar código en línea dentro de una página Web fue simple y también poderosa. Con el abastecimiento de una serie de componentes que desempeñan características avanzadas, mas los notables *ActiveX Data Objects* (ADO), fue muy sencillo crear páginas Web dinámicas.

La liberación final de Active Server Pages 1.0, disponible como un agregado de IIS 3.0, fue tempranamente utilizada en plataformas Windows. La combinación de ASP con ADO, habilitó a los desarrolladores para crear fácilmente recordsets[3] accediendo a bases de datos. No hay duda que esto fue uno de los factores principales para su rápida aceptación.

En 1998, Microsoft introdujo ASP 2.0 como parte de su Windows NT4 Option Pack. La gran diferencia entre esta versión de ASP y la versión 1.0 fue la manera en que componentes externos podían ser instanciados. Con ASP 2.0 y IIS 4.0, es posible crear una aplicación ASP, y con esto correr componentes con su propio espacio de memoria (fuera de proceso). Con *Microsoft Transaction Server* (MTS[4]) se pueden construir componentes que soporten transacciones.

¿Cómo convierte ASP secuencias de comandos HTML? Por medio de ISAPI. Existe una asociación de extensión de archivo (`.asp`) con una DLL ISAPI (`asp.dll`) que se invocará

[2] Un Resultset se usa para denominar a un conjunto de resultados de datos.
[3] Un Recordset es un término dentro de la tecnología ADO equivalente a un Resultset.
[4] MTS es un conjunto de tecnologías que comparten objetos entre un gran número de clientes y garantizan transacciones.

cuando se incluya en una URL[5] un archivo con la extensión especificada. Las DLL ISAPI tienen acceso a funciones de retrollamada que les permiten obtener toda la información que necesitan para procesar peticiones.

Windows 2000, COM+ y ASP 3.0.

Windows 2000 incluye la versión 5.0 de IIS, y la versión 3.0 de ASP. En Windows 2000, Microsoft combinó MTS con el entorno de ejecución de COM[6] para crear COM+[7]. Esto proporciona nuevas características que hacen más fácil el uso de componentes, y también una plataforma de ejecución más eficiente, estable, y escalable.

IIS no tuvo un gran cambio con otras versiones, sin embargo, ahora utiliza COM+ Component Services para ofrecer un mejor ambiente para componentes que se ejecutan en él, incluyendo ejecución fuera de proceso como valor por defecto y la opción de correr cada componente en su propio proceso aislado si se requiere.

Desventajas de ASP clásico.

Limitaciones de secuencias de comandos (script). ASP no proporciona un entorno de programación verdaderamente escalable. Los lenguajes que soporta no son tipificados, lo cual ocasiona que el código sea susceptible a errores por permitir que las variables puedan almacenar cualquier tipo de dato. Otro problema, es no obligar al desarrollador a declarar las variables, las cuales pueden crearse accidentalmente por errores de captura, lo cual en programas considerablemente extensos es difícil rastrear el error.

Esto no es apropiado para los desarrolladores profesionales que crean sitios escalables y de confianza. VBScript, al ser un lenguaje de secuencia de comandos, resulta ser muy limitado. Para dar mayor funcionalidad a las aplicaciones, los desarrolladores normalmente deben añadir componentes separados, lo cual agrega una nueva capa de complejidad.

Inexistencia de estructura de aplicación. Este problema se refiere a la capacidad de mezclar y unir HTML estándar y secuencias de comandos. Más aún, el problema es la necesidad de intercalar directivas de código en HTML. Además de estropear el rendimiento[8] solicitando un cambio de contexto cada vez que se entra o sale de una sección de secuencia de comandos, este código entremezclado en HTML puro hace extremadamente difícil separar la presentación del núcleo de la aplicación. Una solución común consiste en utilizar un conjunto complejo de archivos de inclusión que permitan incluir el contenido por separado, pero el mantenimiento de múltiples archivos y compartir en forma desestructurada los detalles de presentación entre los archivos que definen el contenido y los archivos que definen la presentación es una tarea compleja.

Distribución y configuración. A causa del modo en que funcionan COM y ASP, no es posible modificar fácilmente las aplicaciones que utiliza el sitio Web. A menudo es necesario detener y reiniciar manualmente el servidor, lo que no resulta práctico en un

---

[5] URL (Universal Resource Locator) se utiliza para hacer referencia a un servicio o archivo determinado.
[6] COM es un estándar binario que establece lineamientos para la creación de objetos; como exposición de interfaces, identificador de tipo de objeto para asociar determinada instancia de objeto con una aplicación.
[7] COM+ es un conjunto de servicios que incluyen manejo de hilos, seguridad, manejo de transacciones, contenedor de objetos, manejo de colas y administración de paquetes.
[8] El rendimiento de una aplicación es la medición del volumen de transacciones y el uso de recursos que dan como resultado su tiempo de respuesta.

servidor Web de producción. Cambiar las opciones de configuración también implica tiempo. ASP.NET introduce nuevas características que permiten la modificación y reconfiguración dinámica de los sitios Web.

Limitaciones de estado. Una de las características más eficaces de ASP es su facilidad para integrar el estado de sesión. Sin embargo, el estado de sesión resulta inútil en escenarios donde un sitio Web se almacena en varios servidores Web separados. En este escenario, un cliente podría tener acceso al servidor B mientras que su información de sesión está en el servidor A donde normalmente se abandona. ASP.NET corrige este problema permitiendo que el estado se almacene en un repositorio central: ya sea un proceso separado o una base de datos a donde pueden acceder todos los servidores.

Evolución en las aplicaciones Web. La proporción de usuarios en la comunidad Web que accede el sitio a través de un "dispositivo Internet" como un teléfono celular móvil, asistente personal digital (*Personal digital assistant*, PDA), TV, consola de juegos, o cualquier otro, próximamente será más grande que el numero de usuarios que utilizan una PC y un Web Browser tradicional. Esto significa que probablemente se tendrá que estar preparado para hacer más trabajo en el servidor para adecuar las páginas y satisfacer un dispositivo específico. También se tendrá que crear la salida en un nuevo rango de formatos como el *Wireless Markup Language* (WML[9]). Además de crear WML, nuevos dispositivos Internet y aplicaciones de negocio buscarán la habilidad de enviar y recibir datos XML[10] desde aplicaciones Web. Hacer esto con ASP requiere hacerlo de forma rudimentaria e implica un gran esfuerzo. ASP.NET proporciona clases diseñadas para éste tipo de dispositivos, simplificando el desarrollo.

El cambio acelerado en la naturaleza de aplicaciones distribuidas requiere rápido desarrollo, un mayor apego a desarrollo de componentes y reutilización, mantenimiento sencillo, y el apoyo de una plataforma más robusta. Nuevos estándares como el *Simple Object Access Protocol* (SOAP[11]), y nuevos requerimientos comerciales como el intercambio de datos *Business-To-Business* (B2B[12]), requieren nuevas técnicas que sean utilizadas para generar salida de datos y comunicarse con otros sistemas. Aplicaciones Web y sitios Web también necesitan dar un servicio más robusto y escalable, el cual proporciona ASP.NET a través de monitoreo proactivo y reinicio automático de aplicaciones en caso de falla, colisión de memoria, etc.

Componentes de la plataforma .NET.

Los componentes de la plataforma .NET incluyen el Framework .NET, el .NET Building Block Services, los .NET Enterprise Servers y Visual Studio .NET. La meta de la plataforma .NET es simplificar el desarrollo Web proporcionando todas las herramientas y tecnologías necesarias para construir aplicaciones Web distribuidas.

---

[9] WML es un lenguaje de marcado basado en XML, leído e interpretado por un micronavegador instalado en un dispositivo WAP (Wireless Application Protocol es un protocolo que tiene como objetivo adaptar Internet a la telefonía celular)

[10] XML es un metalenguaje utilizado para definir estructuras de datos susceptibles de ser procesadas por aplicaciones para un intercambio electrónico de datos independiente de plataforma.

[11] SOAP es un protocolo basado en XML para el intercambio de información en un ambiente distribuido.

[12] El B2B es una modalidad de comercio electrónico en el que las operaciones comerciales se realizan entre empresas (por ejemplo, una empresa y sus proveedores)

La plataforma .NET es un conjunto de tecnologías diseñadas para transformar la Internet en una plataforma de computo distribuido totalmente escalable. La plataforma .NET soporta totalmente la existencia de infraestructura existente de Internet, incluyendo HTTP, XML, y SOAP.

Tecnologías centrales en la plataforma .NET

Framework .NET.

El Framework .NET esta basado en un nuevo lenguaje común en tiempo de ejecución que proporciona un conjunto de servicios sin importar el lenguaje de programación utilizado. Estos servicios proporcionan los cimientos para aplicaciones de cualquier tipo, por todas las capas de aplicación.

Microsoft Visual Basic, Microsoft Visual C++, y otros lenguajes de programación Microsoft han sido reforzados para tomar ventaja de estos servicios. Lenguajes de terceros que son escritos para la plataforma .NET también tienen acceso a los mismos servicios.

.Net Building Block Services.

Los .NET Building Block Services son servicios programables distribuidos disponibles en línea y fuera de línea.

Un servicio puede ser invocado en una computadora no conectada a Internet, proporcionado por un servidor local corriendo dentro de una compañía, o accedida por medio de Internet. El .NET Building Block Services puede ser utilizado desde cualquier plataforma que soporte SOAP. Los servicios incluyen identidad, notificación y mensajes, personalización, almacenamiento esquematizado, calendario, directorio, búsqueda, y entrega de software.

.NET Enterprise Servers.

Proporcionan escalabilidad[13], administración, integración dentro de las organizaciones a través de un conjunto de herramientas de software que soportan la tecnología .NET, como por ejemplo: Microsoft SQL Server, Microsoft BizTalk Server, Microsoft Commerce Server, entre otros.

Visual Studio .NET.

Visual Studio .NET proporciona un ambiente de desarrollo de alto nivel para construir aplicaciones que utilicen el Framework .NET. Esto proporciona tecnologías para simplificar la creación, despliegue, y la evolución continúa de aplicaciones Web y servicios Web escalables, confiables[14], y de alta disponibilidad[15]. Este también habilita una nueva generación de aplicaciones basadas en Windows con las nuevas características disponibles en el Framework .NET.

---

[13] Escalabilidad es la calidad sistémica de un software para soportar los requerimientos de procesamiento conforme la carga aumenta. Se caracteriza por no requerir aumento de recursos ni rediseño.
[14] La confiabilidad del software se refiere a la precisión con la que una aplicación proporciona, sin errores, los servicios que se establecieron en las especificaciones originales.
[15] Alta disponibilidad se refiere al tiempo que una aplicación está disponible para dar servicio a un cliente.

ASP.NET y la siguiente generación de Servicios Web del Framework.

La base de la estructura de ASP.NET es muy diferente de las versiones anteriores, aunque por fuera éste parece ofrecer un modelo de desarrollo muy similar. ASP.NET es casi enteramente basado en componentes y modularizado. Cada página, objeto, y elemento HTML que utilizamos puede ser un objeto componente en tiempo de ejecución.

Para ejecutarlos eficientemente, y proporcionar una solución escalable, la administración de estos objetos es un requisito necesario. El nuevo entorno de ejecución lleva a cabo esta administración automáticamente, permitiendo a ASP.NET volverse mas orientado a objetos. Esto permite a los desarrolladores construir aplicaciones más poderosas accediendo estos objetos y componentes en una forma más granular y de manera controlada.

La orientación a objetos de ASP.NET ofrece extensibilidad del ambiente en conjunto. Los desarrolladores pueden agregar y extender el ambiente, ambos creando nuevos componentes o heredando de las clases base que crearon, y sobrescribiendo su comportamiento como se requiera. Internamente, el entorno de ejecución de COM+ administra la instanciación, pooling, y asignación de los objetos automáticamente.

ASP.NET añade muchas características y mejora muchas de las capacidades de ASP clásico. ASP.NET no es meramente una mejora incrementada de ASP; es realmente un producto completamente nuevo, aunque diseñado para permitir la misma experiencia de desarrollo favorable para programadores de ASP.

Una de las mejoras más importantes que ofrece ASP.NET es la forma en la que se utiliza el código en tiempo de ejecución. Aunque los modelos de desarrollo ASP.NET y ASP son similares, las dos tecnologías difieren bastante. En vez de interpretar el código fuente cada vez que el cliente solicita una página, ASP.NET compila inmediatamente la página a lenguaje intermedio de Microsoft (*Microsoft Intermediate Language*, MSIL). Una vez que la página se ha compilado en MSIL, el compilador just-in-time (JIT) convierte el MSIL en código nativo.

La siguiente generación del Framework de servicios.

ASP.NET es actualmente parte de un nuevo Framework que ofrece soporte para todo tipo de aplicaciones. Inicialmente este Framework fue conocido como *Microsoft Next Generation Web Services* (NGWS). El NGWS Framework soporta todas las técnicas de programación del lado del servidor, como un nuevo servicio de administración de componentes, soporte para la construcción de aplicaciones ejecutables y Servicios Windows, etc.

El NGWS Framework incorpora nuevos servicios para aplicaciones escalables y distribuidas:

- Un unificado, y rico conjunto de librerías de programación.
- Un seguro, entorno de ejecución multilenguaje.
- Creación, desarrollo y mantenimiento de aplicaciones simplificadas.
- Incrementa escalabilidad para aplicaciones distribuidas.

Actualmente se ha adoptado el nombre de Framework .NET para identificar este conjunto de servicios y librerías. La Figura 1.1 muestra la estructura del Framework .NET.

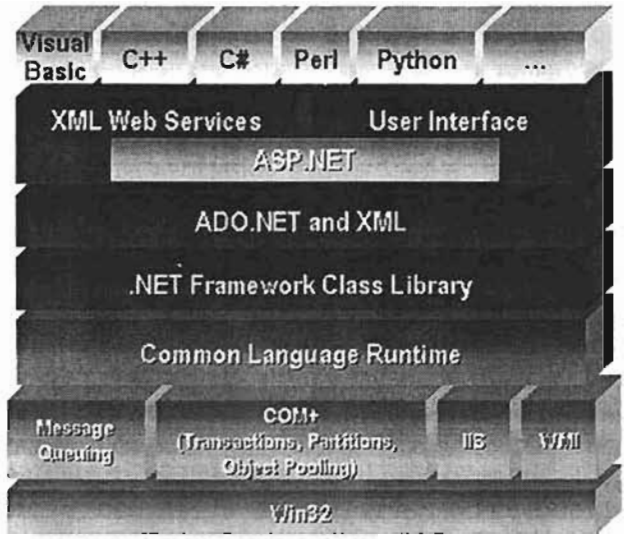

Figura 1.1 *Estructura del Framework .NET.*

Apreciación global del Framework .NET.

El Framework .NET está diseñado desde cero para permitir tanto a los desarrolladores de aplicaciones tradicionales (de escritorio o de consola), como de aplicaciones Web el construir sus aplicaciones de forma más eficiente y permitirles trabajar de forma más flexible. Una de las características más importantes del Framework .NET es que permite que código que se ha escrito en diferentes lenguajes trabaje en conjunto sin problemas.

El Framework .NET debe correr en un sistema operativo. Actualmente el Framework .NET esta construido para correr en sistemas operativos Microsoft Windows. En un futuro, podrá ser extendido para correr en otras plataformas.

Dando soporte a todo el sistema, están los *servicios del sistema*. En la implementación actual, esta base la constituyen el API Win32[16] y los servicios COM+, aunque en teoría, no en práctica, la abstracción permitiría a cualquier sistema operativo el proporcionar los servicios. Tradicionalmente, las aplicaciones han invocado las API del sistema operativo directamente.

Sobre los servicios del sistema, está la capa de *servicios de aplicación*, como Component Services, Message Queuing, Windows Internet Information Server (IIS), y Windows Management Instrumentation (WMI), que están disponibles para el desarrollador a través de librerías de clases.

Sobre los servicios de aplicación, está la capa del *entorno de ejecución de lenguaje común*. El entorno de ejecución carga y ejecuta código escrito en cualquier lenguaje creado para dicho entorno. Al código preparado para el entorno de ejecución se le conoce como *código manejado* (El código manejado se describe con detalle más adelante en este capítulo en la sección de *"Proceso de verificación de seguridad"*). El entorno de ejecución también proporciona una seguridad integrada. El entorno de ejecución proporciona

[16] API Win32 (Interfaz de programación de aplicaciones Windows).

seguridad de acceso al código que permite a los desarrolladores especificar los permisos que se requieren para ejecutar el código. Cuando se carga el software y a medida que se invocan los métodos, el entorno de ejecución puede determinar si el código tiene los permisos de acceso necesarios. Los desarrolladores pueden también definir permisos limitados, lo que implica que el código que se ha diseñado para hacer algo sencillo y no demasiado peligroso puede tener los permisos mínimos. El sistema de seguridad basado en roles que proporciona el entorno de ejecución hace posible que se fijen los valores de los permisos basándonos en el usuario bajo el cual se está ejecutando el código. Aquí podemos identificar a los servicios del garbage collector y security.

Sobre el entorno de ejecución, tenemos las *clases del Framework .NET*. Estas clases que proporciona el Framework .NET se pueden invocar desde cualquier lenguaje de programación preparado para .NET. Las clases siguen un conjunto coherente de nomenclatura y diseño, haciendo más fácil para los desarrolladores el aprenderlas rápidamente. Cubren prácticamente todas las áreas que esperaría un desarrollador, permitiendo extenderlas para crear sus propias librerías de clases.

Sobre estas clases del Framework .NET, están *ADO.NET y los datos XML*. ADO.NET es un conjunto de clases que proporcionan soporte de acceso a datos para el Framework .NET. ADO.NET está basado en ADO pero está diseñado para trabajar con XML y para trabajar en un entorno desconectado.

Sobre ADO.NET y XML, nos encontramos con soporte para dos tipos distintos de aplicaciones. Una de ellas es la aplicación cliente tradicional que utiliza formularios de Windows, una combinación que ofrecía Visual Basic y la biblioteca de clases *Microsoft Foundation Classes* (MFC[17]). El otro tipo de aplicación disponible es ASP.NET, incluyendo los Web Forms y servicios Web XML.

Los *servicios Web XML* son componentes Web programables que pueden compartirse entre aplicaciones en Internet o en una Intranet. El Framework .NET proporciona herramientas y clases para construir, probar, y distribuir servicios Web XML. Estos encabezan una nueva generación de aplicaciones que integran múltiples servicios remotos independientes de plataforma usando estándares de la industria como protocolos HTTP y SOAP. Estas aplicaciones quedan fuera del alcance del presente trabajo.

El Framework .NET soporta tres tipos de interfaces de usuario:

- Web Forms, que trabajan a través de ASP.NET, visibles en clientes Web.
- Windows Forms, los cuales corren en clientes Windows.
- Aplicaciones de consola, que también corren en clientes Windows.

Sobre ASP.NET y los formularios Web está la *especificación de lenguaje común* (CLS) y los lenguajes que la adoptan. CLS es un conjunto de reglas que un lenguaje compatible con CLS tiene que cumplir, garantizando que todos los lenguajes tienen un conjunto de características en común. El Framework .NET proporciona Visual Basic, Visual C++, Visual C#, Visual J#, y JScript. Otros fabricantes de software proporcionan lenguajes adicionales para la plataforma.

[17] Las MFC son un conjunto de librerías que envuelven parte de la API de Windows en clases de C++ para formar un framework de aplicaciones.

Beneficios del Framework .NET.

El Framework .NET fue diseñado para cumplir las siguientes metas:

- ***Basado en prácticas y estándares Web.***

  El Framework .NET soporta tecnologías existentes de Internet, incluyendo HTML (Hypertext Markup Language), XML (Extensible Markup Language), SOAP, XSLT[18] (Extensible Stylesheet Language Transformations), entre otros. En .NET, la mayoría de los estándares se integran su propio entorno de trabajo. Por ejemplo, ADO.NET (tecnología de acceso a datos) utiliza XML de forma nativa en su implementación. De forma similar, los servicios Web funcionan automáticamente a través de XML y HTTP.

- ***Diseñado a partir de modelos de aplicación unificados.***

  La funcionalidad de una clase .NET está disponible desde cualquier lenguaje compatible con .NET, por lo que el desarrollador de aplicaciones podrá utilizar el lenguaje de programación de su preferencia. Los costos asociados a la curva de aprendizaje de un nuevo lenguaje por parte del desarrollador se ven mitigados.

- ***Fácil de utilizar por desarrolladores.***

  El código del Framework .NET, está organizado en *espacios de nombre* (namespaces) y clases ordenadas de forma jerárquica. El Framework .NET proporciona un sistema común de tipos, conocido como sistema de tipos unificado, el cual es utilizado por cualquier lenguaje compatible con .NET. En el sistema de tipos unificado, todos los elementos del lenguaje son objetos. No hay tipos variant, hay únicamente un tipo cadena (string), y todos los datos de cadena son Unicode[19].

- ***Clases extensibles.***

  Se pueden acceder y extender las clases de .NET (a menos que sean selladas) a través de herencia.

- ***Modelo desconectado.***

  Mayor énfasis sobre aplicaciones distribuidas e Internet. Las tecnologías como ADO.NET se diseñaron a partir de cero para trabajar desconectadas, con acceso escalable y no precisa ningún código especial para utilizarlo en una aplicación ASP.NET.

- ***Énfasis sobre la infraestructura.***

  La filosofía de Microsoft es proporcionar la infraestructura de forma que los desarrolladores de aplicaciones únicamente necesiten escribir el código de negocio específico. Por ejemplo, el entorno de trabajo de .NET maneja

[18] XSLT es un lenguaje para transformar documentos XML en otros documentos XML.
[19] Unicode es un sistema de intercambio, procesamiento y representación de textos escritos en diversos idiomas del mundo moderno.

automáticamente archivos, colas de mensajes, bases de datos y llamadas a métodos remotos. Simplemente será necesario añadir la lógica de la organización.

- ***Capa de presentación independiente del cliente.***

  ASP.NET incluye un modelo de programación para la capa de presentación diseñado para simplificar la proliferación de sistemas cliente. Desde una perspectiva de costos, el desarrollo, la depuración, y mantenimiento son mucho más fáciles y más baratos cuando el desarrollador de la capa de presentación no es responsable de determinar que será desplegado en un dispositivo cliente.

Muchos de los beneficios del Framework .NET están asociados al entorno de ejecución de lenguaje común, el cual es el núcleo de la plataforma .NET.

Desventajas de la plataforma .NET

El Framework .NET ha sido desarrollado enteramente por una sola compañía, y nuevas innovaciones han sido pensadas principalmente en tecnologías existentes de Microsoft.

Aunque en teoría el Framework .NET puede ser extendido para correr en varias plataformas, actualmente esta construido para correr en sistemas operativos Microsoft Windows.

No es posible estudiar aún el código completo de toda la plataforma ,NET. Microsoft sólo ha abierto parte de las tecnologías .NET, como el caso del lenguaje C#, pero ha mantenido las partes clave de la plataforma sin publicar abiertamente.

La tecnología .NET es relativamente nueva, lo cual la pone en desventaja frente a otras tecnologías con mayor tiempo en el mercado. No tiene aún toda la experiencia e infraestructura que se van adquiriendo con la liberación de múltiples versiones.

Las aplicaciones Web de .NET se ejecutan únicamente en el servidor Web de Microsoft: IIS (*Internet Information Server*). No es posible por el momento utilizar un servidor de aplicaciones alterno.

Consideraciones

Aunque .NET por el momento no es neutral de plataforma, no existe actualmente un producto en el mercado que permita distribuir componentes empresariales de aplicación de una plataforma a otra sin hacer algún tipo de modificación para adaptar su migración, más aún si estos componentes dependen de distintas implementaciones del software para una plataforma en particular.

La plataforma .NET ha sido adoptada por un gran número de compañías y otras más han migrado sus aplicaciones de versiones previas de ASP y Visual Basic a la nueva versión. En casos donde la infraestructura tecnológica predominante sea Microsoft, la mejor elección será .NET, además de una transición más sencilla y menos costosa a la hora de migrar aplicaciones. De igual forma, el rendimiento de las aplicaciones se verá incrementado considerablemente.

La escalabilidad se refiere a la habilidad de agregar más carga de trabajo a la solución con un mínimo impacto en los recursos y rediseño de la aplicación. Bajo éste contexto,

habremos de considerar que una solución Microsoft sobre Windows es mucho más barata que otras implementaciones en otras plataformas debido a que las máquinas dónde se ejecuta son mucho más económicas. La elección de plataforma dependerá en mucho de las necesidades del negocio y del cliente.

Entorno de ejecución de lenguaje común (CLR).

El CLR (*Common Language Runtime*) es el motor encargado de gestionar la ejecución de las aplicaciones y a las que ofrece numerosos servicios que simplifican su desarrollo y favorecen su fiabilidad y seguridad.

La mayoría de los lenguajes modernos utilizan motores en tiempo de ejecución a través de un conjunto de librerías utilizadas por el lenguaje.

El CLR proporciona un completo conjunto de servicios como la verificación de código, la optimización y la recolección de basura y puede ejecutar código escrito en cualquier lenguaje .NET.

Las aplicaciones .NET se ejecutan dentro de un entorno administrado especial con recolección de basura y un lenguaje intermedio y comparte las características avanzadas de un conjunto de librerías de clases.

Las principales características y servicios que ofrece el CLR son:

- ***Modelo de programación consistente:*** A todos los servicios ofrecidos por el CLR se accede de la misma forma: a través de un modelo de programación orientado a objetos. Esto es una diferencia importante respecto al modo de acceso a los servicios ofrecidos por los algunos sistemas operativos actuales (por ejemplo, los de la familia Windows), en los que a algunos servicios se les accede a través de llamadas a funciones globales definidas en DLLs y a otros a través de objetos COM.

- ***Modelo de programación sencillo.*** Con el CLR desaparecen muchos elementos complejos incluidos en los sistemas operativos Windows (por ejemplo el registro de Windows).

- ***Eliminación del "infierno de las DLL".*** En la plataforma .NET desaparece el problema conocido como "infierno de las DLL" común en los sistemas operativos Windows que soportan una versión única de DLL. El problema consiste en sustituir versiones de DLLs por versiones más recientes y afectar aplicaciones que fueron diseñadas para ser ejecutadas usando la versión previa, dejando de funcionar si no es 100% compatible. En la plataforma .NET las versiones recientes de las DLLs pueden coexistir con versiones previas, de modo que las aplicaciones diseñadas para ejecutarse usando versiones previas podrán seguir haciéndolo aún cuando se instalen versiones más recientes. Esto simplifica la instalación y desinstalación de software. Los programas de MSIL almacenan información extra sobre sus clases y los componentes que necesitan (llamada metadatos). El CLR examina esta información y automáticamente impide que una aplicación utilice la versión errónea de un componente.

- ***Ejecución conjunta.*** El CLR también tiene la posibilidad de cargar más de una versión de un componente a la vez. En otras palabras, es posible modificar un componente varias veces y se cargará y utilizará la versión correcta de cada aplicación. Como efecto lateral, es posible instalar varias versiones del entono de trabajo de .NET, lo que implica que es capaz de adaptarse a nuevas versiones de ASP.NET sin reemplazar la versión inicial o volver a escribir las aplicaciones.

- ***Ejecución multiplataforma.*** El CLR actúa como una máquina virtual[20], encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier plataforma para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET. Microsoft ha desarrollado versiones del CLR para la mayoría de las versiones de Windows: Windows 95, Windows 98, Windows ME, Windows NT 4.0, Windows 2000, Windows 2003, Windows XP y Windows CE (que puede ser usado en CPUs que no sean de la familia x86). Por otro lado hay proyectos de terceros que están desarrollando de manera independiente versiones de distribución libre del CLR para Linux. Asimismo, dado que la arquitectura del CLR es totalmente abierta, es posible que en el futuro se diseñen versiones del mismo para otros sistemas operativos. Sin embargo, no se está completamente seguro de que .NET esté destinado a utilizarse en otros sistemas operativos y plataformas.

- ***Integración de lenguajes.*** Se puede generar código para la plataforma .NET desde cualquier lenguaje que sea compatible. La integración de lenguajes hace posible escribir una clase en C# que herede de otra escrita en Visual Basic .NET.

- ***Gestión de memoria.*** El CLR incluye un recolector de basura (garbage collector) que monitorea la memoria y evita que el desarrollador tenga que destruir los objetos que dejen de serle útiles, lo cual en ocasiones acarrea errores de programación como intentos de borrado de objetos ya eliminados, agotamiento de memoria por no eliminar objetos que no se están ocupando, o solicitud de acceso a miembros de objetos ya destruidos. Este recolector es una aplicación que se activa cuando se quiere crear algún objeto nuevo y se detecta que no queda memoria libre para hacerlo, caso en que el recolector recorre la memoria dinámica asociada a la aplicación, detecta qué objetos hay en ella que no puedan ser accedidos por el código de la aplicación, y los elimina para permitir la creación de otros.

- ***Seguridad de tipos.*** El CLR facilita la detección de errores de programación difíciles de localizar comprobando que toda conversión de tipos sea segura.

- ***Aislamiento de procesos.*** El CLR asegura que el código perteneciente a un determinado proceso no pueda acceder a código o datos pertenecientes a otro, lo cual evita errores de programación frecuentes al acceder a posiciones arbitrarias de memoria.

- ***Tratamiento de excepciones.*** En el CLR todos los errores que se produzcan durante la ejecución de una aplicación se propagan mediante excepciones. Esto es muy diferente a como se venía haciendo en los sistemas Windows, donde algunos errores se transmitían mediante códigos de error en formato Win32, otros mediante HRESULTs y otros mediante excepciones.

[20] Una máquina virtual es la encargada de gestionar la ejecución del código y de proporcionar una serie de servicios a dicho código.

El CLR permite que excepciones lanzadas desde código para .NET escrito en un lenguaje se puedan capturar en código escrito en otro lenguaje, e incluye mecanismos de depuración que pueden saltar desde código escrito para .NET en un determinado lenguaje a código escrito en cualquier otro.

- ***Soporte multihilo.*** El CLR es capaz de trabajar con aplicaciones divididas en múltiples hilos de ejecución que pueden ir evolucionando por separado en paralelo o intercalándose, según el número de procesadores de la máquina sobre la que se ejecuten. Las aplicaciones pueden lanzar nuevos hilos, destruirlos, suspenderlos por un tiempo o hasta que les llegue una notificación, enviarles notificaciones, sincronizarlos, etc.

- ***Distribución transparente.*** El CLR ofrece la infraestructura necesaria para crear objetos remotos y acceder a ellos de manera completamente transparente a su localización real, tal y como si se encontrasen en la máquina que los utiliza.

- ***Seguridad avanzada.*** El CLR proporciona mecanismos para restringir la ejecución de ciertos códigos o los permisos asignados a los mismos según su procedencia o el usuario que los ejecute. Es decir, puede no darse el mismo nivel de confianza a código procedente de Internet que a código instalado localmente o procedente de una red local; puede no darse los mismos permisos a código procedente de un determinado fabricante que a código de otro; y puede no darse los mismos permisos a un mismo código según el usuario que lo esté ejecutando o según el rol que éste desempeñe. Esto permite asegurar al administrador de un sistema que el código que se esté ejecutando no pueda poner en peligro la integridad de los archivos y la del sistema operativo.

- ***Interoperabilidad con versiones de código previas.*** El CLR incorpora los mecanismos necesarios para poder acceder desde código escrito para la plataforma .NET *(código manejado)* a código escrito para plataformas previas a .NET *(código no manejado)*. Estos mecanismos permiten el acceso a objetos COM y acceso a funciones de DLLs preexistentes (como la API Win32).

- ***Transparencia de código.*** El lenguaje MSIL es mucho más fácil de desensamblar, lo cual implica que si se distribuye un componente compilado, al resto de los desarrolladores les costará menos trabajo determinar como funciona su código. Esto no es un problema para las aplicaciones ASP.NET, que no se distribuyen si no que se mantienen sobre un servidor Web.

El proceso de ejecución manejada.

En el Framework .NET, el lenguaje común proporciona la infraestructura para un ambiente de ejecución manejada como se muestra en la Figura 1.2.

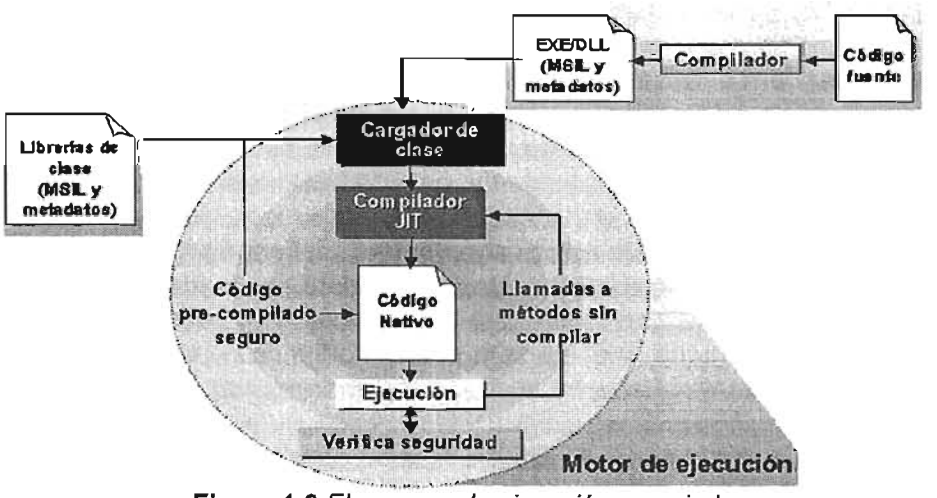

Figura 1.2 *El proceso de ejecución manejada*

Compilando el Código fuente.

Cuando se desarrolla una aplicación en el Framework .NET, podemos escribir el código fuente en cualquier lenguaje de programación y utilizar el compilador compatible para transformarlo en lenguaje intermedio. La compilación del código fuente produce un modulo manejado. El modulo manejado está contenido dentro de un archivo físico conocido como ejecutable portable (*portable executable*, PE), el cual podrá ejecutarse en diferentes sistemas operativos que tengan alguna versión del CLR.

El archivo puede contener los siguientes elementos:

- ***Lenguaje intermedio (Microsoft Intermediate Language, MSIL).***

  El compilador traduce el código fuente en MSIL, un conjunto de instrucciones independiente del CPU que puede ser eficientemente convertido a código nativo.

- ***Metadatos (Metadata).***

  Esta información describe tipos, miembros, y otras referencias, y es utilizado por el entorno de ejecución de lenguaje común en tiempo de ejecución.

- ***Recursos.***

  Por ejemplo, archivos `.bmp`, `.jpg`.

En la plataforma .NET se distinguen dos tipos de módulos manejados: ejecutables (extensión `.exe`) y librerías de enlace dinámico (extensión `.dll`). La diferencia entre estos archivos (ejecutables portables) es que los ejecutables disponen de un método especial que sirve de punto de entrada para ejecutar el código que contienen por medio de una llamada desde la línea de comando del sistema operativo.

Si el compilador genera un ejecutable o librería, entonces el compilador produce un modulo manejado conocido como ensamblado (*assembly*). Los ensamblados son las unidades fundamentales de compartición, distribución, seguridad, y versionamiento en el CLR. El CLR únicamente ejecuta código MSIL que está contenido en un ensamblado.

Ejecutando código.

Cuando un usuario ejecuta una aplicación manejada, el sistema operativo carga el motor de ejecución, el cual comienza ejecutando el código MSIL del módulo manejado. Debido a que las CPU's actuales no ejecutan las instrucciones MSIL directamente, el CLR debe primero convertir las instrucciones MSIL a código nativo.

El CLR no convierte todo el modulo MSIL en instrucciones para la CPU en tiempo de carga; sino que lo convierte a instrucciones conforme se vayan llamando las funciones. El MSIL es compilado únicamente cuando se necesite. El componente del CLR encargado de esta función es llamado compilador *Just-in-time* (JIT). La compilación del JIT conserva la memoria y ahorra tiempo durante la inicialización de la aplicación.

Metadatos (Metadata).

Los metadatos son un conjunto de tablas de datos, las cuales describen cada elemento que esta definido en un modulo. Esta información puede incluir tipos de datos y miembros con sus declaraciones e implementaciones, y referencias a otros tipos y miembros.

Los metadatos proporcionan toda la información que es requerida para la interacción con el componente de software. Los metadatos están incrustados en los archivos .exe o .dll que contienen el código MSIL. Por consiguiente, es imposible separar metadatos del código MSIL.

Uso de los metadatos:

- Localizar y cargar clases.

  Debido a que los metadatos y MSIL están incluidos en el mismo archivo, todos los tipos de información en el archivo están disponibles al CLR en tiempo de compilación. No hay necesidad de archivos de cabecera porque todos los tipos en un ensamblado particular son descritos por el manifiesto del ensamblado.

- Reforzando seguridad.

  Los metadatos pueden contener o no contener permisos requeridos para ejecutar el código. El sistema de seguridad utiliza permisos para prevenir que el código acceda recursos que no tienen autorización de acceder.

- Otros usos incluyen:
  - Resolver llamadas a métodos.
  - Preparar límites de contexto de ejecución.
  - Proporcionar capacidad de reflexión.

El Lenguaje Intermedio de Microsoft.

Todos los compiladores compatibles con la plataforma .NET no generan código nativo para una CPU particular, en su lugar generan código en lenguaje intermedio (*Microsoft Intermediate Language,* MSIL) llamado a veces código manejado; el cual se convierte en código nativo en tiempo de ejecución.

Antes de que el código MSIL pueda ser ejecutado, debe ser convertido a código nativo para una CPU específica por medio de un compilador JIT (Just In Time).

Un programa ejecutable compilado para utilizarse con el Framework .NET se divide en dos partes: la primera parte es el código MSIL (Lenguaje intermedio) que se utiliza para generar el código nativo. La segunda parte son los metadatos, información sobre el código y otros elementos que necesita el entorno de ejecución.

MSIL contiene instrucciones para muchas operaciones comunes, incluyendo instrucciones para crear e inicializar objetos, y para llamar métodos en objetos. Adicionalmente, incluye instrucciones para operaciones lógicas y aritméticas, control de flujo, acceso directo a memoria, y manejo de excepciones.

Ensamblados.

En la tecnología COM, las interfaces inmutables parecían ser la solución al problema de conflictos de DLL. En la práctica, crear nuevas versiones de un componente COM que no afectara a las aplicaciones existentes, incluso si la interfaz expuesta permanecía inmutable, implicaba que las aplicaciones que utilizaban dicho componente dejaban de funcionar. La solución .NET a este problema es el ensamblado (*assembly*).

Un ensamblado está formado por uno o más archivos que están agrupados lógicamente y se distribuyen conjuntamente. Los ensamblados pueden contener módulos manejados, archivos de recurso o de datos. También son la unidad básica de despliegue, versionamiento, seguridad y reutilización.

Los ensamblados contienen un manifiesto de ensamblado (*assembly manifest*), que son metadatos en forma de tabla con información de las características del ensamblado, en el que cada entrada es el nombre del archivo o parte del ensamblado. El manifiesto incluye los metadatos que son necesarios para especificar los requerimientos de versión, identidad de seguridad, y la información que es utilizada para definir el alcance del ensamblado y resolver referencias a recursos y clases. Debido a que los metadatos hacen al ensamblado autodescriptivo, el CLR tiene la información requerida para ejecutar cada ensamblado. Todos los archivos que componen el ensamblado deben estar listados en el manifiesto de ensamblado. La Figura 1.3 muestra la estructura de un ensamblado.

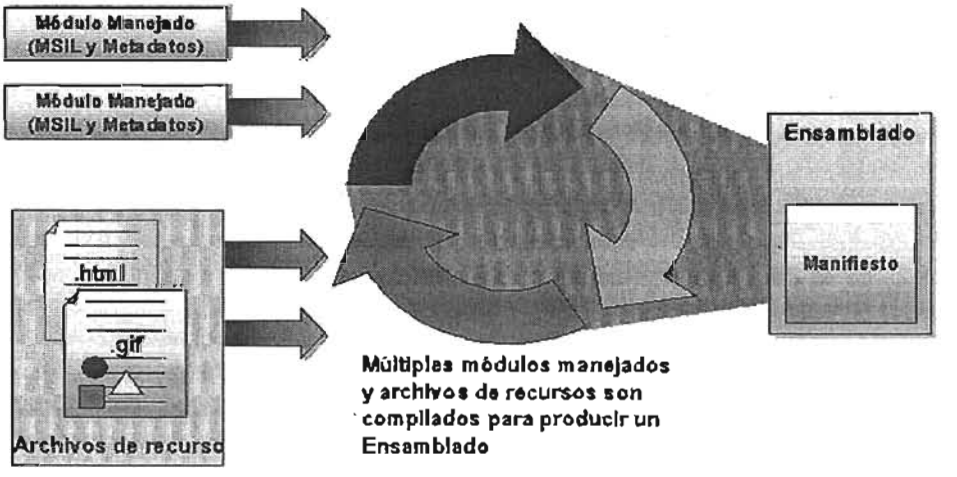

Figura 1.3 *Ensamblados*

¿En que difieren los ensamblados de los componentes COM? Los componentes COM suponen que cada máquina sólo puede tener una única versión de un componente que implementa una interfaz particular. El Framework .NET adopta la idea de ejecución simultánea. Pueden coexistir múltiples versiones de un ensamblado, y tanto el desarrollador como el administrador del sistema tienen control sobre qué ensamblado se utilizará.

Hay dos tipos de ensamblados: *ensamblados privados* y *ensamblados compartidos*. Los privados se almacenan en el mismo directorio que la aplicación que los usa y sólo pueden usarlos la misma aplicación, mientras que los compartidos se almacenan en un Caché de ensamblado global (*Global Assembly Cache*, GAC) y pueden usarlos cualquier aplicación que haya sido compilada haciendo referencia a ellos.

Los compartidos se identifican en el GAC con la combinación de identidad del ensamblado (el cual incluye su nombre en texto simple y sin extensión), número de versión, información de cultura (si se proporciona), más su llave pública, lo que permite instalar varios ensamblados con el mismo nombre y diferentes llaves públicas en el GAC. A este tipo de ensamblados se les conoce como ensamblados con nombre fuerte (*assembly with a strong name*). Esta política permite resolver los conflictos derivados al intentar instalar en una misma computadora varios ensamblados compartidos con el mismo nombre pero procedentes de distintas empresas, pues estos tendrán diferentes llaves públicas.

Para evitar problemas, en el GAC se pueden mantener múltiples versiones de un mismo ensamblado. Así, si una aplicación fue compilada usando cierta versión de un ensamblado compartido, cuando se ejecute, sólo podrá hacer uso de esa versión del ensamblado y no de otra más reciente que se encuentre instalada en el GAC. De esta forma se soluciona el problema del "infierno de las DLL" comentado anteriormente en las "características y servicios que ofrece el CLR".

En realidad es posible modificar tanto las políticas de búsqueda de ensamblados (por ejemplo, para buscar ensamblados privados fuera del directorio de la aplicación) como la política de aceptación de ensamblados compartidos (por ejemplo, para que se haga automáticamente uso de nuevas versiones que se instalen de DLLs compartidas) incluyendo en el directorio de instalación de la aplicación un archivo de configuración en formato XML con las reglas para las mismas. Este archivo ha de llamarse igual que el ejecutable de la aplicación pero con la extensión `.config`.

El compilador just-in-time (JIT).

Como se comentaba anteriormente, el código MSIL debe ser convertido en código nativo antes de poder ejecutarse. Debido a que un paso intermedio está involucrado, el CLR optimiza el proceso de compilación compilando MSIL como se vaya necesitando por medio del compilador JIT. El proceso básico se describe a continuación:

1. Cuando el CLR carga un tipo de clase, esté asocia un stub[21] para cada método de la clase.
2. Para llamadas subsecuentes al método, el stub direcciona la ejecución del programa al componente del CLR responsable de compilar el método MSIL a código nativo, conocido como compilador JIT.

[21] Stub se utiliza para designar al código auxiliar en un programa.

3. El compilador JIT compila el MSIL y el método stub es sustituido con la dirección del código nativo recién creado. Futuras llamadas al método no involucran al compilador JIT porque el código nativo simplemente se ejecuta.

El compilar una aplicación a partir del código de ensamblado MSIL impondría alguna carga en el rendimiento de la aplicación. En la práctica, la sobrecarga incurrida constituye una diferencia poco significativa. Parte de este bajo coste es ciertamente el diseño del compilador JIT, pero mucho del crédito va a la forma en que se utilizan los programas comúnmente. Generalmente, no todas las líneas de código dentro de un programa se utilizan cada vez que éste se ejecuta. Para aprovecharse de este hecho, en vez de compilar todo el código MSIL en un archivo ejecutable nativo al principio, el compilador JIT compila el código únicamente cuando se necesita, y luego coloca en la caché el código nativo compilado para que sea reutilizado.

Proceso de verificación de seguridad.

El *código manejado* es código que proporciona suficiente información para permitir que el CLR lleve a cabo las siguientes tareas:

- Dada una dirección en el código, localizar los metadatos que describen el método.
- Manejar excepciones.
- Almacenar y recuperar información de seguridad.

Para que el CLR lleve a cabo estas tareas, el código debe pasar un *proceso de verificación*, a menos que un administrador haya establecido una política que permita que el código se ejecute sin verificación. Durante el proceso de verificación, el compilador JIT examina el código MSIL y los metadatos para determinar si el código se puede clasificar como de tipo seguro. El código de *tipo seguro* es el código para el cual se puede determinar que accede sólo a posiciones de memoria que posee. Esta restricción asegura que el código opera y se ejecuta bien junto con otros programas y que el código es suficientemente seguro como para no causar corrupciones accidentales o maliciosas.

Asociado al código manejado están los datos manejados. Los *datos manejados* son datos que tienen memoria que se reserva y libera automáticamente mediante el entorno de ejecución, que utiliza un proceso llamado recolector de basura (*garbage collector*). Cuando un elemento para el que se ha reservado memoria sale fuera de ámbito, el entorno de ejecución lo libera por medio del recolector de basura.

Una consecuencia del uso de la recolección de basura es que no se puede determinar el momento y el orden de destrucción de los objetos. La liberación no determinista de objetos no es un problema a menos que el objeto tenga algún recurso persistente que no gestione el entorno de ejecución, por ejemplo, una conexión con la base de datos. Cuando un objeto posee tales recursos, la solución es proporcionar un método para liberar memoria e implementar una interfaz que se puede invocar de forma explícita para liberar los recursos.

A pesar de todos los cambios al núcleo operativo y la forma en que se ejecutan las aplicaciones, se ha tenido un gran cuidado en mantener la compatibilidad con otras versiones anteriores de Windows, COM y ASP. En la mayoría de los casos, aplicaciones existentes, componentes COM y COM+, páginas ASP, y otros scripts y ejecutables trabajan bajo el ambiente de ejecución de .NET. Alternativamente, podemos actualizarlos en el tiempo en el que los requerimientos del negocio lo demanden.

El código no manejado.

En algunos casos no es posible utilizar código manejado, por ejemplo en la utilización de apuntadores. La solución a este problema es utilizar código no manejado o inseguro, el cual no puede ser comprobado por el entorno de ejecución. Para estas situaciones, C# proporciona la palabra clave `unsafe`, el cual se indica en determinada sección del código.

Dominio de la aplicación.

Los sistemas operativos y ambientes de ejecución típicamente proporcionan alguna forma de aislamiento entre aplicaciones. Este aislamiento es necesario para asegurarse que el código corriendo en una aplicación no puede afectar otra.

Históricamente, las fronteras de procesos han sido utilizadas para aislar aplicaciones corriendo en la misma computadora. Cada aplicación es cargada en procesos separados, lo cual separa una aplicación de otra corriendo en la misma computadora.

El código manejado debe ser pasado por un proceso de verificación antes que pueda correr (a menos que un administrador haya establecido una política que permita que el código se ejecute sin verificación). El proceso de verificación determina si el código puede intentar el acceso a direcciones de memoria inválidas o realizar alguna otra acción que pueda causar que el proceso en el cual está corriendo falle u opere incorrectamente.

El código que pasa la prueba de verificación se dice que es de tipo seguro. La habilidad de verificar código como tipo seguro habilita al CLR para proporcionar un nivel de aislamiento como las fronteras de proceso, a mucho menor costo de rendimiento.

Los dominios de aplicación proporcionan una segura y versátil unidad de procesamiento que el CLR puede usar para proporcionar aislamiento entre aplicaciones. Se pueden correr varios dominios de aplicación en un proceso simple con el mismo nivel de aislamiento que existe en procesos separados, pero sin incurrir en sobrecarga adicional haciendo llamadas entre procesos o cambiar de un proceso a otro. La habilidad de correr múltiples aplicaciones dentro de un proceso simple incrementa dramáticamente la escalabilidad del servidor.

El aislamiento proporcionado por los dominios de aplicación tienen los siguientes beneficios;

- Las fallas en una aplicación no afectan otra aplicación. Debido a que el código de tipo seguro no puede causar fallas de memoria, al usar los dominios de aplicación se asegura que el código corriendo en un dominio no afecta otra aplicación en el proceso.

- Aplicaciones individuales pueden ser detenidas sin parar el proceso entero. Usando los dominios de aplicación, se habilita la opción para descargar el código corriendo de una aplicación simple.

Introducción a C#.

Origen y necesidad de un nuevo lenguaje.

C# (leído en inglés "See Sharp") es el nuevo lenguaje de propósito general diseñado por Microsoft para su plataforma .NET.

Aunque es posible escribir código para la plataforma .NET en otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programar usando C# es mucho más sencillo e intuitivo que hacerlo con cualquier otro lenguaje.

La sintaxis y estructuración de C# es muy similar a C, C++ y Java, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes y facilitar su aprendizaje para los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java, C y C++ para combinarlas en uno solo.

Características de C#.

Algunas de las características aquí señaladas tienen que ver con la plataforma .NET. Sin embargo, se comentan debido a que tienen repercusión directa sobre el lenguaje.

- El código escrito en C# es auto-contenido, lo que significa que no necesita de archivos adicionales al código fuente tales como archivos de cabecera.

- El tamaño de los tipos de datos básicos es fijo e independiente del compilador, sistema operativo o CPU para el cual se compile, para facilitar la portabilidad del código.

- ***Orientación a objetos.*** C# soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.

  En lo referente a la encapsulación es importante señalar que aparte de los típicos modificadores `public`, `private` y `protected`, C# añade un cuarto modificador llamado `internal`, que puede combinarse con `protected` para indicar que el elemento a cuya definición precede, sólo se puede acceder desde su propio ensamblado.

  Respecto a la herencia, C# sólo admite herencia simple de clases, pero permite simular herencia múltiple a través de interfaces.

- ***Orientación a componentes.*** La propia sintaxis de C# incluye elementos propios del diseño de componentes que otros lenguajes tienen que simular mediante construcciones más o menos complejas. Es decir, la sintaxis de C# permite definir cómodamente *propiedades* (similares a campos de acceso controlado), *eventos* (asociación controlada de funciones de respuesta a notificaciones), *atributos* (información sobre un tipo), y *métodos* (operaciones y funcionalidad de la clase).

- ***Gestión automática de memoria.*** Como ya se comentó, todo lenguaje de .NET tiene a su disposición el recolector de basura del CLR. Esto tiene el efecto en el lenguaje de que no es necesario incluir instrucciones de destrucción de objetos. Sin embargo, la destrucción de los objetos a través del recolector de basura es no determinista y sólo se realiza cuando éste se activa –ya sea por falta de memoria, finalización de la aplicación o solicitud explícita en el código fuente-. C# proporciona destructores y el método `Finalize` para ejecutar código antes de reciclar la memoria y un mecanismo de liberación de recursos determinista (diferentes a memoria) a través de la implementación del método `Dispose`.

- ***Seguridad de tipos.*** C# incluye mecanismos que permiten asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que evita que se produzcan errores difíciles de detectar por acceso a memoria no perteneciente al tipo. Para ello se toman las siguientes medidas:

    - Sólo se admiten conversiones entre tipos compatibles. Esto es, entre un tipo y antecesores suyos, entre tipos para los que explícitamente se haya definido un operador de conversión, y entre un tipo y un tipo hijo suyo; en el que un objeto del primero almacene una referencia del segundo (*downcasting*).

    - Se comprueba que todo acceso a elementos de un arreglo se realice con índices que se encuentren dentro del rango del mismo.

    - Pueden definirse métodos que admitan un número de parámetros de determinado tipo, y siempre se comprueba que los valores que se les pasen en cada llamada sean de los tipos apropiados.

- ***Instrucciones seguras.*** Para evitar errores muy comunes, en C# se han impuesto una serie de restricciones en el uso de las instrucciones de control más comunes. Por ejemplo, para toda condición se utiliza una expresión condicional y no aritmética, con lo que se evitan errores de confusión del operador de igualdad (==) con el de asignación (=); y todos los casos dentro de una instrucción `switch` han de terminar en un `break` o `goto` que indique cuál es la siguiente acción a realizar, lo que evita la ejecución accidental de casos y facilita su reordenación.

- ***Sistema de tipos unificado.*** En C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada `System.Object`, por lo que dispondrán de todos los miembros definidos en ésta clase base.

  Esto también es aplicable a los tipos de datos básicos. Además, para conseguir que ello no tenga una repercusión negativa en rendimiento, se ha incluido un mecanismo transparente con el que se consigue que sólo sean tratados como objetos cuando la situación lo requiera.

  El hecho de que todos los tipos del lenguaje deriven de una clase común facilita enormemente el diseño de colecciones genéricas que puedan almacenar objetos de cualquier tipo.

- ***Sobrecarga de operadores.*** Nos permite crear un método que se invocará cuando se utilice determinado operador. Esto ayuda a facilitar la legibilidad del código y conseguir que los nuevos tipos de datos básicos que se definan a través

de las estructuras estén al mismo nivel de los tipos básicos predefinidos en el lenguaje, C# permite redefinir el significado de la mayoría de los operadores.

La sobrecarga de operadores se hace de manera inteligente, de modo que a partir de una única definición de los operadores ++ y -- el compilador puede deducir automáticamente como ejecutarlos de manera prefija y postfija; y definiendo operadores simples (como +), el compilador deduce cómo aplicar su versión de asignación compuesta (+=). Además, para asegurar la consistencia, el compilador vigila que los operadores con opuesto (operadores de comparación) siempre se redefinan por parejas (por ejemplo, si se redefine ==, también hay que redefinir I=).

- ***Eficiente.*** En principio, en C# todo el código incluye numerosas restricciones para garantizar su seguridad y no permite el uso de apuntadores. Sin embargo, es posible saltarse dichas restricciones manipulando objetos a través de apuntadores. Para ello basta marcar regiones de código como inseguras (modificador `unsafe`) y podrán usarse en ellas apuntadores, lo que puede resultar vital para situaciones donde se necesite una eficiencia y velocidad procesamiento muy grandes.

- ***Compatible.*** Para facilitar la migración de desarrolladores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java, sino que permite incluir directamente en código C# fragmentos de código escrito en estos lenguajes. El CLR también ofrece, a través de los llamados *Platform Invocation Services* (PInvoke), la posibilidad de acceder a código nativo escrito como funciones no orientadas a objetos tales como las DLLs de la API Win32. Nótese que la capacidad de usar apuntadores en código inseguro permite que se pueda acceder con facilidad a este tipo de funciones, ya que éstas muchas veces esperan recibir o devuelven apuntadores.

  También es posible acceder desde código escrito en C# a objetos COM. Para facilitar esto, el *Framework .NET SDK* incluye las herramientas llamadas `tlbimp` y `regasm` mediante las que es posible generar automáticamente clases proxy que permitan, respectivamente, usar objetos COM desde .NET como si se tratara de objetos .NET y registrar objetos .NET para su uso desde COM.

  Finalmente, también se da la posibilidad de usar controles ActiveX[22] desde código .NET y viceversa. Para lo primero se utiliza la utilidad `aximp`, mientras que para lo segundo se usa la ya mencionada `regasm`.

[22] ActiveX es una tecnología compuesta de elementos que realizan una tarea especializada haciendo uso de la tecnología de vinculación e incrustación de objetos (OLE), la cual permite transferir información de un formato de archivo a otro con posibilidad de edición. Un control ActiveX es una pieza modular de software que efectúa una tarea específica y puede ser reutilizado por programas que tengan la capacidad de contener controles ActiveX como Internet Explorer y aplicaciones Visual Basic.

Fundamentos de la Programación Orientada a Objetos.

En éste capítulo se revisarán algunos conceptos y terminología de Programación Orientada a Objetos (POO). Más adelante se mostrará como crear y organizar objetos en espacios de nombres y dar a conocer los objetos fundamentales que componen una aplicación Web.

Definición de clase.

- Es una categorización de objetos similares, con los mismos atributos y valores asignados posiblemente distintos, que definen su estado, con el mismo conjunto de métodos que definen su comportamiento. Representa un molde, esquema o patrón que define la forma de sus objetos. O bien, como la estructura estática que define los estados y comportamiento que van a tener los objetos.

- Es una categoría o grupo de cosas que tienen atributos y comportamiento similares.

Las clases no están restringidas a clasificar objetos concretos (como carros); también pueden ser utilizadas para abstraer conceptos (como el tiempo).

Definición de objeto.

- Es una abstracción conceptual del mundo real que se puede traducir a un lenguaje de programación orientado a objetos.

- Es una instancia de una clase.

Los objetos exhiben:

- ***Identidad.*** Características que distinguen un objeto de otros objetos de la misma clase.

- ***Comportamiento.*** Características que definen las acciones que pueden realizar los objetos y que determinan su clasificación.

- ***Estado.*** Los objetos almacenan información para conocer lo que éste representa y definir su comportamiento. Un buen diseño de objetos mantiene su estado inaccesible.

La identidad del objeto se puede interpretar como la referencia. El estado de objeto es una lista de variables conocidas como sus atributos, cuyos valores representan el estado que caracteriza al objeto. El comportamiento es una lista de procedimientos, funciones u operaciones conocidos como métodos, que un objeto puede ejecutar a solicitud de otros objetos.

Definición de Abstracción.

- La abstracción denota las características esenciales de un objeto que lo distingue de otras clases de objetos proveyéndolo de fronteras conceptuales de definición muy claras y dentro de una perspectiva funcional.

Una buena abstracción logra hacer énfasis en los detalles más significativos o relevantes de la solución y discrimina cualquier otra característica menos importante.

El Sistema Común de Tipos.

Cada variable tiene un tipo que determina que valores pueden ser almacenados en la variable. C# es un lenguaje de tipo seguro, lo que significa que el compilador garantiza que los valores almacenados en variables son siempre del tipo apropiado.

El CLR incluye un sistema común de tipos (CTS) que define un conjunto de tipos para definir las variables. Éste es el modelo que define las reglas que se siguen para declarar, usar y manejar tipos. El CTS establece un Framework que habilita la integración entre lenguajes .NET y seguridad de tipos.

Los tipos de datos básicos (por ejemplo, tipos primitivos) son realmente objetos creados a partir de la biblioteca de clases. Sin embargo, los tipos de datos básicos funcionan como tipos valor mientras que los objetos se comportan como tipos referencia.

Tipos valor.

Una variable de tipo valor contiene la información que se asigne a ella (como números o cadenas de texto). Estas instancias puede ser libremente copiadas y existir en los objetos como variables locales o atributos. Las operaciones sobre estos tipos de variables no pueden afectar otras.

Tipos referencia.

Por otra parte, las variables tipo referencia almacenan una referencia que apunta a una ubicación en memoria donde se almacena el objeto, no al objeto en si mismo. El objeto está en un área separada de memoria. Estas instancias no pueden ser copiadas, solamente su referencia. Las operaciones sobre estos tipos de variables pueden afectar otras. Hay dos casos en los que se observa que las variables de tipo referencia actúan de forma diferente que las variables de tipo valor: en las operaciones de asignación y comparación.

Operaciones de asignación.

Al asignar una variable de tipo de dato básico a otra variable de tipo de dato básico, se copia el contenido de la variable.

```
int EnteroA = 0, EnteroB = 1;
EnteroA = EnteroB;
/* EnteroA ahora tiene una copia del contenido de EnteroB.
    En memoria hay dos enteros duplicados.
    Ambas variables tienen el valor 1.
*/
```

Los objetos funcionan de forma ligeramente diferente. La copia de todo el contenido de objeto podría ralentizar la aplicación, particularmente si se realizaran varias asignaciones. Con objetos, la opción por defecto es simplemente copiar la referencia en una operación de asignación.

```
System.Object ObjetoA = null;
System.Object ObjetoB = new System.Object();
ObjetoA = ObjetoB;
/* ObjetoA y ObjetoB apuntan ahora al mismo objeto.
   Hay un solo objeto y dos formas de acceder a él
*/
```

Comprobación de igualdad.

Se hace una distinción similar cuando se comparan dos variables. Al comparar dos variables simples se compara el contenido.

```
if (EnteroA==EnteroB)
{
  //Esto es Verdadero siempre que los enteros tengan el mismo contenido.
}
```

Al comparar variables objeto se comprueba si son la misma instancia. Se comprueba si apuntan al mismo objeto en memoria, no si coincide su contenido.

```
if (ObjetoA==ObjetoB)
{
  /* Es Verdadero si tanto ObjetoA como ObjetoB apuntan a la misma dirección.
     Es Falso si apuntan a lugares separados, aunque sean objetos idénticos.
  */
}
```

Atributos, métodos, propiedades y constructores.

Atributos.

Un atributo es un valor o dato almacenado en una variable dentro de una clase.

Propiedades.

Son valores que pueden ser almacenados o extraídos de una clase. Al igual que los atributos y métodos, las propiedades se consideran miembros de un objeto. Son una manera conveniente de exponer valores o datos a objetos externos. Algunas propiedades pueden ser de sólo lectura, mientras que otras pueden modificarse mediante el código que utiliza el objeto.

Las propiedades son diferentes de los atributos porque el acceso a la propiedades esta controlado a través de métodos de acceso conocidos como `get` y `set`. Asimismo, las propiedades no necesariamente se almacenan en una variable; pueden ser calculadas al vuelo o almacenadas en múltiples variables.

Creación de propiedades.

Los procedimientos de propiedad permiten que los objetos sean manipulados a través de código, esto es, proporcionan acceso a las variables internas. Estos consisten de dos partes: un procedimiento que permite que la clase usuario recupere los datos y otro que permita que la clase usuario modifique los datos. Los procedimientos de propiedad son similares a cualquier otro tipo de procedimiento en cuanto a que pueden contener tanto código como sea necesario.

Métodos.

Permiten realizar una acción determinada con un objeto, modificando significativamente su comportamiento.

Constructores.

Si se diseña una clase que requiere código especial de inicialización, se puede utilizar un constructor para inicializar la clase propiamente cuando los objetos son creados. Un constructor es un método con el mismo nombre que la clase. El constructor se ejecuta cuando una nueva instancia de la clase es creada. Si no se crea un constructor, .NET proporciona un constructor por defecto. Sin embargo, si se crea un constructor, el código cliente debe utilizarlo. Esta restricción resulta útil porque se pueden crear constructores que requieren parámetros específicos. Si la llamada intenta crear un objeto sin especificar valores válidos para todos estos parámetros, es posible lanzar un error y denegar la creación del objeto.

Definición de espacios de nombres y ensamblados.

Los espacios de nombres (*namespaces*) representan la organización principal utilizada para agrupar todos los tipos diferentes de la biblioteca de clase. Sin espacios de nombres, estos tipos se agruparían en una lista inmensa y desorganizada.

Los espacios de nombre son una manera de organizar código. Proporcionan una protección para evitar conflictos de nombres, a veces llamados colisiones de namespaces. Esta protección es especialmente necesaria en proyectos extensos donde es fácil que dos piezas de código accidentalmente tengan el mismo nombre. Organizando el código en namespaces, se reduce la posibilidad de estos conflictos. Para crear un namespace en C#:

```
// Nombre del proyecto: Ventas.
namespace Utilerias
{
  public class Cadenas
  {
    // Convierte a mayúsculas
    public static string Mayusculas(string Texto)
    {
      return Texto.ToUpper();
    }
  }
}
```

Debido a que los namespaces son una herramienta de organización de código que puede utilizarse donde sea, son públicos por definición. Cuándo se utiliza en el código, la referencia toma la forma:

```
NombreDelProyecto.NameSpace.NombreDelModulo.NombreDelMiembro
```

Los lenguajes soportados por el Framework .NET utilizan los mismos tipos de datos escalares y convenciones de pase de parámetros por defecto.

Los namespaces de `System` ofrecen una interfaz orientada a objetos a las funciones de Windows.

Podemos utilizar código desde un namespace en nuestras aplicaciones de dos maneras:

- Utilizar el nombre cualificado completo del miembro. El siguiente código llama la función `Mayusculas` del namespace `Ventas.Utilerias`:
  ```
  string cadena1 = Ventas.Utilerias.Cadenas.Mayusculas("minusculas");
  ```

- Agregar un enunciado `using` al comienzo de la clase o módulo. Este enunciado ofrece un camino corto al miembro, no se tiene que utilizar el nombre cualificado en el código:
  ```
  using Ventas.Utilerias;

  string cadena1 = Cadenas.Mayusculas("minusculas");
  ```

Los namespaces utilizan la notación de punto para especificar jerarquía. Considerar la siguiente declaración de namespace:

```
// Nombre del proyecto: Ventas.
namespace Utilerias
{
  namespace Tipos
  {
    public class Numeros
    {

    }
  }
}
```

Esta declaración es lo mismo que:

```
// Nombre del proyecto: Ventas.
namespace Utilerias.Tipos
{
  public class Numeros
  {

  }
}
```

Referencias.

Se agregan referencias al proyecto para utilizar namespaces fuera del proyecto actual. Utilizar el enunciado `using` para tener un camino corto al namespace. Este proporciona una manera abreviada de hacer referencia al namespace en el código.

Para agregar la referencia al proyecto, desde Visual Studio .NET:

1. Desde el menú Project, escoger `Add Reference`. Visual Studio .NET despliega la caja de diálogo `Add Reference`, como se muestra en la Figura 2.1.

2. Seleccionar la referencia que se vaya a agregar desde las listas de .NET, COM, o Projects. Click en OK.

3. Visual Studio agrega la referencia a la ventana de Project.

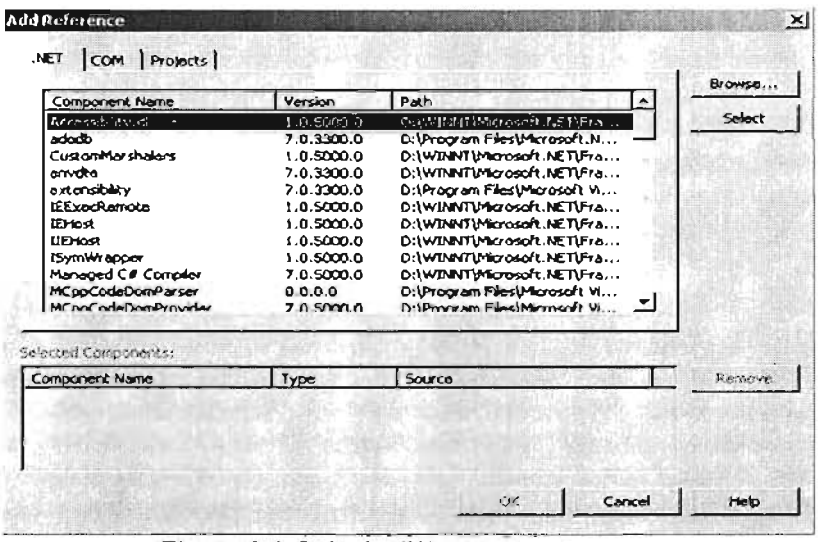

Figura 2.1 *Caja de diálogo* `Add Reference`.

Ensamblados.

Podríamos preguntarnos qué es lo que nos da la posibilidad de utilizar los namespaces de la biblioteca de clases en un programa .NET. Todas las clases .NET se almacenan en ensamblados, que es el equivalente .NET de los archivos `.exe` y `.dll` tradicionales. Existen dos formas de poder utilizar un ensamblado:

- Ubicándolos en el mismo directorio o en un subdirectorio de la aplicación. El CLR automáticamente examina todos estos ensamblados y hace que sus clases estén disponibles para la aplicación. No se necesitan pasos adicionales o el registro de estos componentes.

- Ubicándolos en el Caché global de ensamblado (*Global Assembly Cache*, GAC), que almacena una lista de ensamblados compartidos. No se deberá escoger esta opción en componentes propios a menos que se decida distribuirlos. Todas las clases .NET, sin embargo, forman parte del GAC, dado que no tendría sentido instalarlos por separado en cada directorio de aplicación.

Ensamblados y clases .NET.

Las clases .NET se almacenan en distintos ensamblados. Por ejemplo, los tipos básicos del espacio de nombres `System` se encuentran en el ensamblado `mscorlib.dll`. Muchos tipos ASP.NET se encuentran en el ensamblado `System.Web.dll`. Si se precompilan las páginas ASP.NET, no es necesario realizar una referencia explícita a estos ensamblados.

Programación de clases.

En C#, cualquier código está contenido en una clase. En la Tabla 2.1 se muestran algunos de los conceptos clave de programación orientada a objetos y su relación con el lenguaje de programación C#.

Concepto	Descripción en C#
Definición	Definir las clases utilizando la palabra clave `class`. Todo el código ejecutable es parte de una clase.
Acceso	Hay cinco niveles de acceso a clases y sus miembros: `public`, `protected`, `internal`, `protected internal`, y `private`.
Herencia	Las clases pueden heredar de miembros de clases base y sobrescribir o sobrecargar miembros de las clases heredadas.
Constructores y destructores	Las clases tienen constructores y destructores que son llamados cuando un objeto basado en la clase es creado o destruido. Los métodos constructores tienen el mismo nombre que su clase, y los métodos destructores utilizan el nombre de la clase precedido de una tilde (~)
Clases abstractas e interfaces	Las interfaces definen los nombres de miembros y una lista de parámetros para las clases que utilicen la interfaz. Los miembros abstractos de una clase proporcionan los elementos que serán heredados por las clases derivadas de ellos.

Tabla 2.1 *Conceptos clave de POO.*

Encapsulación.

En POO, la encapsulación es el proceso de ocultar los detalles de implementación internos de una clase empaquetando atributos y funcionalidad para prevenir que otros objetos puedan manipular sus datos o procedimientos directamente. La encapsulación también permite que el objeto que está solicitando el servicio ignore los detalles de la implementación del servicio. Debido a que se ocultan los datos y funcionalidad interna de una clase de otras clases, la encapsulación proporciona gran flexibilidad en el diseño de clases.

De acuerdo a esta definición, se propone fragmentar las abstracciones en dos partes principales; una parte constitutiva y otra parte de interfaz o servicio visible a otros objetos. De esta manera la complejidad con la que se elabora cierta abstracción queda a salvo y no preocupa a capas superiores de abstracción.

¿Porque encapsular?

Permite control:

- La utilización del objeto es solamente a través de miembros visibles.

Permite cambios:

- La utilización del objeto no se afecta si los tipos de datos privados cambian.

La encapsulación permite reducir el potencial de errores que puedan ocurrir. En un sistema constituido de objetos, éstos dependen unos de otros en diversas formas. Si uno de ellos falla y los especialistas de software tienen que modificarlo de alguna forma, el ocultar su implementación de otros objetos significará que tal vez no sea necesario modificar los demás objetos. Esto conlleva a otros escenarios donde simplemente se desea optimizar el código para que haga lo mismo pero con menor costo de rendimiento.

C# proporciona un conjunto de modificadores de acceso que varían el grado de accesibilidad a miembros de cualquier tipo.

Definición de clases y modificadores de acceso.

En C#, se usa la palabra clave `class` para definir clases. Se pueden tener una o más clases por archivo. Utilizar cualquiera de las palabras clave de acceso mostradas en la Tabla 2.2 para definir cuales otras clases pueden utilizar los miembros de la clase actual.

Palabra clave C#	Disponible para
`public`	Todos los miembros en todas las clases y ensamblados.
`internal`	Todos los miembros de cualquiera de las clases del ensamblado actual.
`protected`	Todos los miembros en la clase actual y en las clases derivadas. Pueden ser utilizadas únicamente en la definición de miembros, no para clases.
`protected internal`	Todos los miembros en la clase actual y en clases derivadas en el proyecto actual. Pueden ser utilizadas únicamente en definiciones de miembros, no para clases.
`private`	Únicamente la clase actual. Pueden ser utilizadas únicamente en la definición de miembros, no para clases.

Tabla 2.2 *Niveles de acceso para clases.*

Herencia.

En POO, herencia es un método para reutilizar miembros de clase en otras clases.

La herencia es de gran utilidad cuando múltiples clases comparten los mismos métodos y datos. Los métodos comunes y datos pueden ser abstraídos en clases separadas para que otras clases hereden de ellas. En la forma más simple de herencia, una clase hereda sus miembros de una clase base de la cual obtiene su funcionalidad, y al mismo tiempo agrega funcionalidad adicional propia.

Toda clase hereda de `System.Object`, por lo que todas las clases tienen los miembros de `System.Object`.

Herencia simple y múltiple.

El Framework .NET únicamente soporta herencia simple de tipos de clase. La herencia simple significa que una clase puede derivar de una sola clase base. La herencia múltiple significa que una clase puede derivar de dos ó más clases base. El CTS simula herencia múltiple a través de interfaces.

Otra restricción sobre herencia es que no podemos derivar una clase de tipo Web Form de una clase base Web Form. Aunque los Web Forms son definidos como clases, no se soporta derivar nuevos Web Forms de existentes Web Forms base.

La sintaxis para definir herencia en C# es:

```
public class ClaseDerivada : ClaseBase
{
}
```

C# utiliza las palabras clave descritas en la Tabla 2.3 para crear clases base y derivar nuevas clases de ellas.

Palabras clave C#	Utilizada para
`ClaseDerivada : ClaseBase`	Basar una clase en otra, heredando miembros desde la clase base.
`virtual`	Declarar que un miembro de la clase base puede ser sobrescrito en una clase derivada.
`override`	Declarar que un miembro de la clase derivada sobrescribe el miembro del mismo nombre en la clase base.
`new`	Declarar que un miembro de una clase derivada oculta el miembro del mismo nombre en la clase base.
`abstract (en una clase)`	Declarar que una clase proporciona una plantilla para clases derivadas. Este tipo de clases son llamadas clases abstractas, y no pueden ser instanciadas.
`abstract (en un miembro)`	Declarar que un miembro de una clase proporciona una plantilla para los miembros de una clase derivada. Este tipo de miembro es llamado miembro abstracto y no puede ser invocado.
`base`	Llamar al miembro de una clase base desde una clase derivada.
`this`	Llamar a un miembro de una instancia actual de una clase.
`interface`	Crear una interfaz que define los miembros que una clase debe proporcionar.
`NombreClase : NombreInterfaz`	Utilizar la definición de una interfaz en una clase.

Tabla 2.3 *Palabras clave de herencia.*

Polimorfismo.

El polimorfismo en POO permite que un método declarado en la clase base pueda ser implementado de diferentes formas en clases derivadas.

- El nombre del método reside en la clase base.
- La implementación del método reside en las clases derivadas.
- En algunos casos es conveniente no definir el cuerpo del método en las clases base que será implementado en las clases derivadas. A estos métodos sin cuerpo se les llama operaciones.

Overriding, overloading y shadowing de miembros.

Una clase derivada hereda los miembros de su clase base. Para explorar la herencia, consideremos como ejemplo la clase base `Libro` y una subclase llamada `LibroUsado`.

```
public class Libro
{
    private string _isbn;
    private string _titulo;
    private double _precio;

    public string ISBN{
      get {return _isbn;}
      set {_isbn = value;}
    }

    public string Titulo{
      get {return _titulo;}
      set {_titulo = value;}
    }

    public double Precio{
      get {return _precio;}
      set {_precio = value;}
    }
}
```

La subclase hereda de la clase base, el cual es un tipo de libro más especializado.

```
public class LibroUsado : Libro
{
  DateTime _fechaCompra;
  public DateTime FechaCompra
  {
    get {return _fechaCompra;}
    set {_fechaCompra = value;}
  }
}
```

La *firma de un método* consiste del nombre del método, el número de parámetros, el tipo de dato y modificador (`out` ó `ref`) de cada parámetro. El nombre de los parámetros y el tipo de retorno de los métodos no forman parte de la firma.

Overloading (sobrecarga) permite que una clase derivada defina métodos del mismo nombre, pero diferente lista de parámetros y tipos de parámetro que el método base. Realmente no importa el nombre de los parámetros, sino su tipo de dato. Para fines del ejemplo, agreguemos el siguiente código a la clase `Libro`:

```
private int _cantidad;

public int Cantidad{
  get {return _cantidad;}
  set {_cantidad = value;}
}

public void Comprar(){                          // (1)
  _cantidad += 1;
}

public void Comprar(int cantidad){                          // (2)
  _cantidad += cantidad;
}
```

Éste código agrega una nueva propiedad y método a la clase. El método `(1)` permite indicar que se ha comprado una nueva copia del libro. El método `(2)` permite indicar la

compra de una mayor cantidad de libros. Dado que la firma del método (1) no recibe argumentos, y la firma del método (2) recibe un parámetro entero (lo cual los hace distintos aunque tengan el mismo nombre) se dice que el método está sobrecargado. La sobrecarga es únicamente posible cuando cada versión del método tiene una firma diferente.

Notar que la subclase `LibroUsado` ha heredado las dos implementaciones del método `Comprar` porque deriva de la clase `Libro`. Supongamos que necesitamos otra versión del método `Comprar` en la subclase `LibroUsado` porque nos interesa la fecha de compra:

```
public void Comprar(DateTime fechaCompra)
{
  _fechaCompra = fechaCompra;
  base.Comprar();                    // (1)
}
```

Notar que esta nueva implementación del método `Comprar` tiene una firma diferente que las otras en la clase base. La línea de código (1) causa que la subclase llame el método `Comprar` de la clase base.

Ahora veamos como sobrescribir ó alterar métodos en la clase base. Para hacer esto existen dos formas diferentes: overriding (sobrescritura) y shadowing (ocultamiento).

El diseñador de una clase base es responsable de pensar a futuro y determinar como se usará la clase base a través de herencia. En particular, el diseñador de clases base debe escoger cuales métodos pueden ser sobrescritos ó alterados por una subclase y cuales permanecen inmutables.

Por defecto, los métodos son inmutables, y el autor de una subclase no puede alterar o sobrescribirlos haciendo overriding. Los métodos pueden ser siempre remplazados por medio de shadowing. Se prefiere overriding porque se necesita del permiso en la clase base para hacerlo; definido con la palabra clave `virtual` como parte de la declaración del método.

Aplicando esto al ejemplo, la clase `Libro` puede considerar que algunas subclases en el futuro pueden necesitar un esquema diferente de precios:

```
public virtual double Precio
{
  get {return _precio;}
  set {_precio = value;}
}
```

Con esto la subclase `LibroUsado` puede ahora implementar su propia versión de la propiedad `Precio` si se desea. Consideremos el hecho de que un libro usado generalmente tiene un precio menor que uno nuevo, así que la implementación del método `Precio` en la subclase `Libro` sería diferente:

```
public override double Precio{
  get{
    if ((DateTime.Now - _fechaCompra).Days > 365)
      return base.Precio / 2;
    else
      return base.Precio;
  }
  set{
    base.Precio = value;
  }
}
```

Notar que el uso de la palabra clave `override` en la declaración del método es requerida, pues al omitirse el nuevo método ocultará (shadowing) la implementación de la clase base.

Hasta éste punto es necesario entender la diferencia entre el tipo de dato de una variable y el tipo de dato de un objeto. Típicamente se declara una variable del tipo de objeto al cual hará referencia de la siguiente forma:

```
LibroUsado librousado = new LibroUsado();
```

Cuando se usa herencia, se puede escoger el declarar una variable del tipo de clase base del cual deriva el objeto, en lugar del tipo de objeto actual:

```
Libro libroGenerico = new LibroUsado();
```

Esto significa que se puede escribir código que trabaje con cualquier tipo de `Libro`, en lugar de una subclase específica. Esto funciona porque se conoce que todos los libros tienen las propiedades ISBN, titulo y precio. Aunque la instancia sea de un `LibroUsado`, sólo se podrá acceder a los métodos definidos en la clase base `Libro` porque es el tipo de variable que se está usando. Para acceder los métodos especializados de una subclase específica se necesita aplicar un cast al objeto `libroGenerico` para verlo temporalmente del tipo `LibroUsado`:

```
((LibroUsado)(libroGenerico)).Comprar(DateTime.Parse("2000/12/31"));
```

El resultado será la ejecución del método de la subclase `LibroUsado` aunque se trate de un tipo de variable `Libro`. Cuando un método es marcado como `override`, se convierte en un método virtual. Esto significa que, sin importar el tipo de dato de la variable que se esté utilizando, el tipo de dato del objeto dicta que implementación del método será invocado.

La siguiente tabla ilustra que implementación de método será invocado basado en el tipo de dato de la variable y el objeto:

Variable	Objeto	Método
Base	Base	Base
Base	Subclase	Subclase
Subclase	Subclase	Subclase

Ahora veamos que sucede cuando se tiene la necesidad de alterar o sobrescribir un método existente y el diseño de la clase base no proporciona el permiso. Para esto utilizamos shadowing y para entenderlo será necesario hacer una comparación contra overloading y overriding.

Overloading permite a una subclase agregar una nueva variación de un método. Aunque la firma del método sea diferente, el nuevo método puede tener el mismo nombre que uno existente en la clase base. *Shadowing*, por otro lado, reemplaza enteramente todas las variaciones del método de la clase base, dejando la subclase con una sola versión del método.

Overriding permite a una subclase alterar o remplazar un método existente con el permiso de la clase base definido en su diseño. *Shadowing* puede hacerlo sin permiso de la clase base. Overriding usa métodos virtuales, de tal forma que la implementación del método que es invocado está basado en el tipo de dato del objeto instanciado (que puede ser

derivado de una clase base), no del tipo de dato de la variable que se está utilizando. Cuando un método es ocultado (shadowing) no es virtual, y será el tipo de dato de la variable que dicte la implementación del método que será usado.

Para fines ilustrativos tomemos de nuevo el método Precio de la subclase `LibroUsado`. En la definición del método se pondrá la palabra clave `new` para indicar que se ocultará el método, aunque no es requerida por ser el comportamiento por defecto.

```
public new double Precio{
  get{
    if ((DateTime.Now - _fechaCompra).Days > 365)
      return base.Precio / 2;
    else
      return base.Precio;
  }
  set{
    base.Precio = value;
  }
}
```

Ahora consideremos la instancia de un objeto `LibroUsado` referenciado por una variable de tipo `Libro`:

```
Libro libroGenerico = new LibroUsado();
```

Al hacer el cast a `libroGenerico` para tratar de acceder los métodos especializados de la subclase `LibroUsado` no se obtendrán los resultados esperados, porque la variable tiene el tipo de dato de Libro, y los métodos que usan shadowing usan el tipo de dato de la variable, en lugar del tipo de dato del objeto para determinar que implementación será invocada.

```
((LibroUsado)( libroGenerico)).Comprar(DateTime.Parse("2000/12/31"));
```

La siguiente tabla muestra que implementación será invocada cuando se usa shadowing, basada en el tipo de dato de la variable y el objeto:

Variable	Objeto	Método
Base	Base	Base
Base	Subclase	Base
Subclase	Subclase	Subclase

Utilizando datos y métodos estáticos.

En algunos casos no tiene sentido guardar la misma información compartida que es común entre objetos creados a partir de una clase particular. Por lo que se recomienda describir datos comunes a nivel clase (descritos como estáticos), y no a nivel objeto.

Los *datos estáticos* son declarados dentro de una clase (la cual es una entidad estática en tiempo de compilación) utilizando la palabra clave `static`, los cuales existen aún si el programa no crea instancias de objetos de esa clase.

Los métodos estáticos se utilizan para encapsular datos estáticos. Estos se declaran en la misma forma que los datos estáticos y existen a nivel clase. Se controla la accesibilidad para métodos y datos estáticos a través de modificadores de acceso. Si se escogen métodos estáticos públicos y datos estáticos privados, se pueden encapsular los datos estáticos de la misma manera que se encapsulan los datos de objetos.

Un método estático existe a nivel clase y es llamado refiriéndose a la clase, no al objeto. Esto quiere decir que un método estático no puede utilizar el operador `this`, que implícitamente se refiere al objeto; por lo que solo puede acceder datos estáticos o métodos estáticos.

Interfaces.

Las *interfaces* proporcionan una plantilla que puede utilizarse para crear nuevas clases. Características importantes de las interfaces es que no proporcionan ninguna implementación de los miembros de la clase, los miembros son implícitamente públicos, no pueden ser instanciadas, y sólo pueden extender otras interfaces.

Cuando se implementa una interfaz particular en una clase, instancias de esa clase pueden ser utilizadas para cualquier argumento o variable declarada como esa interfaz. Por ejemplo, el siguiente código declara una interfaz para los objetos `Libro` creados anteriormente:

```
interface ILibro
{
  string ISBN{
    get;
    set;
  }

  string Titulo{
    get;
    set;
  }

  double Precio{
    get;
    set;
  }
}
```

Para utilizar la interfaz, implementarla en una clase como se muestra a continuación:

```
public class Libro : ILibro
{
  private string _isbn;
  private string _titulo;
  private double _precio;

  public string ISBN{
    get {return _isbn;}
    set {_isbn = value;}
  }

  public string Titulo{
    get {return _titulo;}
    set {_titulo = value;}
  }

  public double Precio{
    get {return _precio;}
    set {_precio = value;}
  }
}
```

Debido a que la clase concreta `Libro` en el ejemplo implementa la interfaz `ILibro`, todas las clases derivadas de `Libro` heredan la implementación de `ILibro`. Esto significa que objetos

del tipo `LibroUsado` (clase definida en ejemplos anteriores), los cuales derivan de `Libro`, pueden ser utilizados como argumentos o variables del tipo `ILibro`.

La clave es que todos los elementos definidos en la interfaz deben existir en cualquier clase que implemente la interfaz. Si se omite cualquier miembro, el Framework .NET genera un error en tiempo de compilación.

Clases abstractas.

Una *clase abstracta* es una clase que define una interfaz para clases derivadas. Una clase abstracta es esencialmente un contrato diciendo que todas las clases basadas en ésta, proporcionarán ciertos métodos y propiedades. No podemos crear objetos de una clase abstracta, sólo podemos derivar nuevas clases de ella.

En una jerarquía de clases típica, una operación es declarada en la clase base, y el método es implementado de diferentes maneras en distintas clases derivadas. La clase base existe solamente para introducir el nombre del método en la jerarquía. En particular, la operación de la clase base no requiere implementación. Esto hace que la clase base no sea usada como una clase regular. Más importante aún, es que no se debe permitir el crear instancias de este tipo de clases base, por lo que necesita ser marcada como abstracta. Las clases abstractas pueden extender interfaces y clases concretas.

Las clases abstractas pueden implementar miembros que llegan a ser comunes para todas las clases que derivan de ellas.

Las clases abstractas son declaradas en C# con la palabra clave `abstract` (1) al igual que métodos y propiedades. Modifiquemos la clase `Libro` utilizada anteriormente definiéndola como abstracta:

```
public abstract class Libro                          // (1)
{
  private string _isbn;
  private string _titulo;
  private double _precio;

  public string ISBN{                                // (3)
    get {return _isbn;}
    set {_isbn = value;}
  }

  public string Titulo{
    get {return _titulo;}
    set {_titulo = value;}
  }

  public virtual double Precio{
    get {return _precio;}
    set {_precio = value;}
  }

  public abstract void Comprar(DateTime fechaCompra);                    // (2)
}
```

Notar que los miembros abstractos (2) son sólo definiciones, no hay enunciados en el cuerpo del procedimiento porque estos miembros serán definidos (sobrescritos) en la clase derivada. Sin embargo, se pueden implementar miembros que no sean abstractos (3) en la misma clase.

La siguiente clase concreta demuestra como es heredada una clase abstracta. En C#, la definición de la clase (1) declara que está basada en la clase abstracta `Libro`. La palabra clave `override` (2) es requerida en cada una de las definiciones que sobrescriben miembros de la clase abstracta que han sido definidos previamente con las palabras clave `abstract` Ó `virtual`.

```
public class LibroNuevo : Libro                    // (1)
{
  private double _precio;
  private DateTime _fechaCompra;

  public override double Precio{                    // (2)
    get {return _precio;}
    set {_precio = value;}
  }

  public override void Comprar(DateTime fechaCompra){                    // (2)
    _fechaCompra = fechaCompra;
  }
}
```

Espacios de nombre en una aplicación Web.

Las definiciones de clase para objetos utilizados en aplicaciones Web residen en el namespace de `System.Web`. La Tabla 2.4 muestra su jerarquía y describe los tipos de definición de clases que podemos encontrar en cada namespace.

Namespace	Contiene clases para
`System.Web`	Los objetos `Application`, `Browser`, `Cache`, `Cookies`, `Exception`, `Request`, `Response`, `Server`, `Trace`. La mayoría de las tareas de programación utilizan estas clases. El objeto `Application` definido en el `Global.asax` esta basado en la clase `Application`.
`System.Web.SessionState`	El objeto `Session`. Utilizar estas clases para guardar y extraer elementos guardados en el estado `Session`.
`System.Web.Services`	El objeto `WebService`. Utilizar estas clases para crear y usar servicios Web XML.
`System.Web.UI`	Los objetos `Page` y `Control`. Utilizar estas clases con un Web Form para crear y controlar una aplicación de interfaz de usuario. Los Web Forms están basados en la clase `Page`.
`System.Web.UI.WebControls`	Todos los objetos de controles Web. Utilizar estas clases con los Web Forms.
`System.Web.UI.HTMLControls`	Todos los objetos de controles HTML. Utilizar estas clases con los Web Forms.
`System.Web.Caching`	El objeto `Cache`. Utilizar estas clases para controlar el caché del lado del servidor para mejorar el rendimiento de la aplicación.
`System.Web.Mail`	Los objetos `MailMessage`, `MailAttachment`, y `SmtpMail`. Utilizar estas clases para enviar mensajes mail desde la aplicación.
`System.Web.Security`	Los objetos de autenticación. Utilizar estas clases para autenticar usuarios y proporcionar seguridad en la aplicación.

Tabla 2.4 *La jerarquía de* `System.Web`.

En una aplicación Web, se trata directamente con dos tipos de objetos derivados de las clases en los namespaces Web:

- ***Objeto*** `Application`. Derivado de la clase `HttpApplication`. En la aplicación, esta definición reside en el archivo `Global.asax`.

- ***Objetos*** `Web Form`. Derivados de la clase `Page`. En la aplicación, esta definición reside en módulos Web Form.

```
public class Global : System.Web.HttpApplication
{
}
public class WebFormN : System.Web.UI.Page
{
}
```

Los objetos `Global` y `WebFormN` (que puede llamarse de cualquier manera) son los puntos de entrada que se utilizan para obtener otros objetos Web en una aplicación. Estos objetos, junto con `Request` y `Response`, son los objetos que mas comúnmente se utilizan en el código.

Utilizando el objeto `Application`.

Utilizar el objeto `Application` para configurar la aplicación y guardar la información de estado.

Cuando comienza una aplicación, esta automáticamente crea una instancia del objeto `Global` definido en `Global.asax`. Utilizar los eventos en el objeto `Global` para configurar la aplicación e inicializar a nivel aplicación las variables de estado.

La clase base `HttpApplication` proporciona las propiedades para acceder objetos subordinados como se muestra en la Tabla 2.5.

Propiedad	Utilizar para
`Application`	Guardar/Obtener elementos de datos en el estado `Application`.
`Context`	Obtener los objetos `Trace`, `Cache`, `Error`, y otros objetos para el contexto actual.
`Modules`	Acceder módulos HTTP.
`Request`	Leer texto o datos de un request y obtener objetos `Browser`, `ClientCertificates`, `Cookies`, y `File` del request actual.
`Response`	Escribir texto o datos en un response y obtener objetos `Cache`, `Cookies`, y `Output` del response actual.
`Server`	Procesar request y responses. El objeto `Server` proporciona métodos para codificar y decodificar la URL.
`Session`	Guardar elementos de datos en el estado `Session`.
`User`	Obtener información de autenticación acerca del usuario que esta haciendo el request actual. Por defecto, las aplicaciones Web permiten acceso anónimo.

Tabla 2.5 *Propiedades del objeto* `Application`.

Utilizando el objeto `Page`.

El objeto `Page` controla la interfaz de usuario de la aplicación. Cuando un usuario requiere una página de la aplicación, ASP.NET automáticamente crea una instancia del Web Form y despliega la página. Se agregan procedimientos de eventos al Web Form para controlar la interfaz de usuario y para interactuar con el usuario.

La clase base `Page` proporciona las propiedades que se utilizan más frecuentemente en los Web Forms, mostradas en la Tabla 2.6.

Propiedad	Utilizar para
`Application`	Obtener el objeto `Application` de la solicitud Web actual.
`Cache`	Controlar como los response son almacenados en caché del servidor.
`Controls`	Obtener controles de la página.
`Request`	Leer un request y obtener objetos `Browser`, `ClientCertificates`, `Cookies`, y `File`, del request actual.
`Response`	Escribir texto o datos a un response y obtener objetos `Cache`, `Cookies`, y `Output` del response actual.
`Server`	Procesar request y response. El objeto `Server` proporciona métodos para codificar y decodificar la URL.
`Session`	Guardar elementos de datos en el estado `Session`.

Tabla 2.6 *Propiedades del objeto* `Page`.

Utilizando el objeto `Request`.

El objeto `Request` contiene la información enviada por el cliente Web Browser cuando una página es solicitada por la aplicación.

El objeto `Request` proporciona las propiedades para acceder objetos subordinados descritos en la Tabla 2.7.

Propiedad	Utilizar para
`Browser`	Determinar las capacidades del Web Browser haciendo el request. Las propiedades del Web Browser proporcionan su número de versión, determina si soporta cookies, y otra información.
`ClientCertificate`	Autenticar el cliente.
`Cookies`	Obtener información del cliente en forma de cookies.
`Files`	Obtener archivos que son cargados por el cliente.
`InputStream`	Obtener el contenido del cuerpo de la entidad HTTP entrante.

Tabla 2.7 *Propiedades del objeto* `Request`.

Utilizando el objeto `Response`.

Utilizar el objeto `Response` para formar la respuesta enviada del servidor al cliente Web Browser. El objeto `Response` proporciona las propiedades para acceder los objetos subordinados descritos en la Tabla 2.8.

Propiedad	Utilizar para
`Cache`	Determinar como el servidor almacena en caché peticiones antes de que sean enviadas al cliente.
`Cookies`	Establecer el contenido de las cookies que se envían al cliente.
`Output`	Obtener o establecer los datos regresados al cliente como response.

Tabla 2.8 *Propiedades del objeto* `Response`.

Introducción a Web Forms.

En éste capítulo nos introduciremos en las aplicaciones ASP.NET. El tema central es el entendimiento de cómo funcionan estas aplicaciones desde su base, cómo configurarla y cómo desarrollar con código asociado para separar la interfaz de usuario de una página Web de la lógica de negocio. Más adelante se mostrará cómo utilizar controles de servidor para crear la interfaz de usuario, navegación entre páginas, validación de datos y mantener el estado de los datos en las páginas Web.

Cada aplicación ASP.NET individual comparte un conjunto de recursos y opciones de configuración común. Las páginas Web de otras aplicaciones ASP.NET, incluso auque estén en el mismo servidor, no comparten estos recursos. Técnicamente hablando, cada aplicación ASP.NET se ejecuta dentro de un dominio de aplicación separado. Los dominios de aplicación se aíslan en memoria, implicando que si una aplicación Web provoca un error fatal no afectará al resto de aplicaciones actualmente en ejecución. También implica que las páginas Web de una aplicación no pueden acceder a la información almacenada en memoria de otra aplicación. Cada aplicación tiene su propio conjunto de caché[23], aplicación y datos de estado de sesión.

La definición estándar de una aplicación ASP.NET la describe como una combinación de archivos, páginas, manejadores, módulos y código ejecutable que puede invocarse desde un directorio virtual (y, opcionalmente sus subdirectorios) de un servidor Web. En otras palabras, el directorio virtual es la estructura de agrupación básica que delimita una aplicación.

Los *Web Forms* son objetos de la aplicación que definen la interfaz de usuario de la aplicación Web. El texto y los controles que se ponen en un Web Form determinan que ve el usuario cuando se ejecuta la aplicación.

Las aplicaciones Web corren en un servidor y son distribuidas a clientes a través de Internet o Intranets corporativas. ASP.NET está integrado con el servidor Web Microsoft Internet Information Server (IIS), lo cual implica que las aplicaciones tengan determinada organización.

Organizando proyectos en IIS.

Una aplicación Web puede existir únicamente en una ubicación que ha sido publicada por IIS como un directorio virtual. Un *directorio virtual* es un conjunto de recursos identificados por un alias que representan una ubicación física dentro de un dispositivo de almacenamiento en un servidor.

El directorio virtual `http://localhost` ó `http://NombreComputadora` es el directorio raíz en la computadora donde se encuentra instalado el servicio de publicación Web. IIS determina la ubicación física del directorio raíz. Por defecto, IIS lo instala en una unidad de disco en la dirección `\Inetpub\wwwroot`. La organización de los proyectos es independiente de IIS, por lo que se recomienda organizar los proyectos creando directorios separados para ambiente de desarrollo, pruebas y producción.

[23] El caché es un almacenamiento temporal de datos usado para aumentar el rendimiento de la aplicación. Puede estar ubicado del lado del servidor o del lado del cliente.

Creando directorios virtuales

Utilizar IIS para crear nuevos directorios virtuales y administrarlos con Visual Studio .NET ó Visual InterDev (para compatibilidad con versiones previas de ASP), requiere de dos tareas:

- ***Crear directorios virtuales.*** Los directorios virtuales especifican donde se almacenan físicamente los proyectos de aplicación Web, así que se pueden utilizar para organizar los proyectos durante el desarrollo.

- ***Crear directorios subweb.*** Un *directorio subweb* es simplemente un directorio virtual que contiene las Extensiones de Servidor FrontPage (*FrontPage Server Extensions*) instaladas, que habilita a Visual InterDev para crear y mantener aplicaciones Web de tipo ASP clásico en ese directorio. El nombre del directorio físico y virtual deberán llamarse igual. Si el directorio físico se encuentra fuera del sitio Web, deberán copiarse los archivos que se generen a la ubicación correcta, pues por defecto se crean en el directorio del sitio Web. Se deberán otorgar permisos a los usuarios que deseen crear ó modificar archivos sobre el directorio.

Los pasos para crear un directorio virtual son:

1. Abrir la consola de Administración de Servicios Internet y dar click derecho sobre el icono `Default Web Site`, y sobre el menú `New` dar click sobre `Virtual Directory` como se muestra en la Figura 3.1.

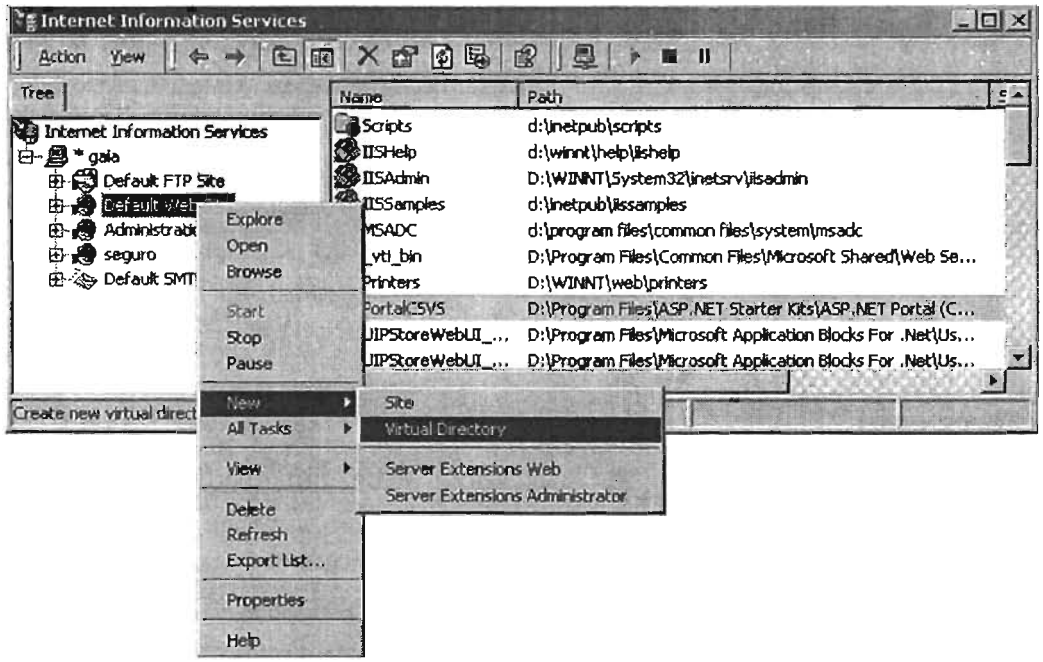

Figura 3.1 *Crear un nuevo directorio virtual.*

2. IIS comienza con el Wizard para guiar sobre la creación del nuevo directorio virtual. Dar click en `Next` para desplegar la ventana de `Virtual Directory Alias`, Como se muestra en la Figura 3.2.

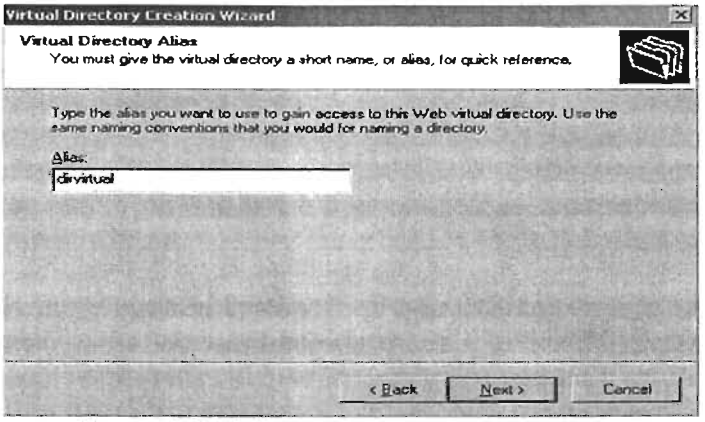

Figura 3.2 *Ventana* `Virtual Directory Alias`.

3. Escribir el nombre del alias para el directorio. El *alias* es un nombre que se usa para identificar el recurso del directorio virtual. En Visual Studio .NET, este es el mismo que se utiliza para especificar la ubicación del proyecto. Dar click en `Next` para desplegar la ventana `Web Site Content Directory`, como se muestra en la Figura 3.3.

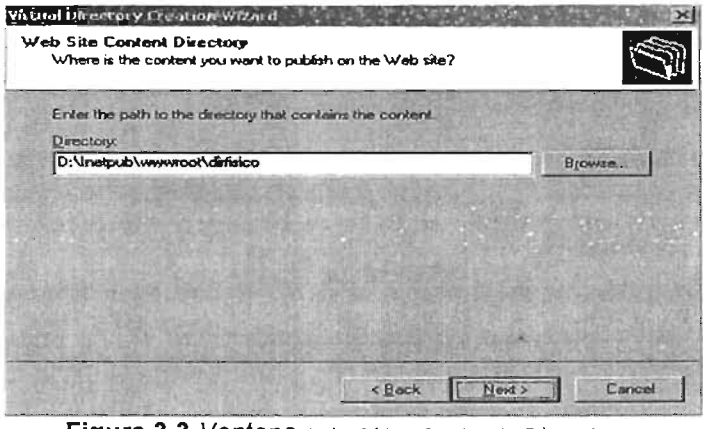

Figura 3.3 *Ventana* `Web Site Content Directory`.

4. Teclear la ubicación del directorio físico para asociarlo con el directorio virtual. Este es el directorio base donde serán creados los directorios del proyecto. Dar click en `Next` para desplegar la ventana de `Access Permissions`, como se muestra en la Figura 3.4.

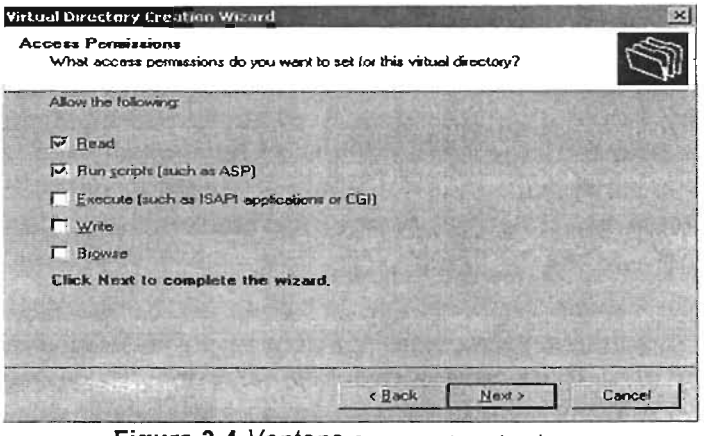

Figura 3.4 *Ventana* `Access Permissions`.

5. Conservar los permisos por defecto y dar click en `Next` para finalizar la creación del directorio virtual y el Wizard.

Creando directorios subweb.

Para agregar las Extensiones de Servidor Front Page a un directorio virtual, seguir los siguientes pasos:

1. Abrir la consola de Administración de Servicios Internet y dar click derecho sobre el icono `Default Web Site`, y sobre el menú `New` dar click sobre `Server Extensions Web`.

2. IIS muestra el Wizard `New Subweb`. Dar click en `Next` para desplegar la ventana `Subweb Name`, como se muestra en la Figura 3.5.

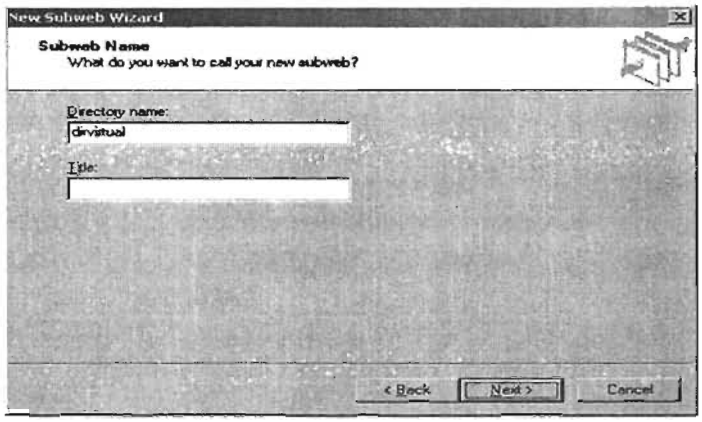

Figura 3.5 *Ventana* `Subweb Name`.

3. Teclear el nombre del directorio virtual que se haya creado previamente en la caja de texto `Directory Name`. La caja de texto `Title` puede dejarse vacía. Dar click en `Next`. El Wizard despliega la ventana `Access Control`, como se muestra en la Figura 3.6.

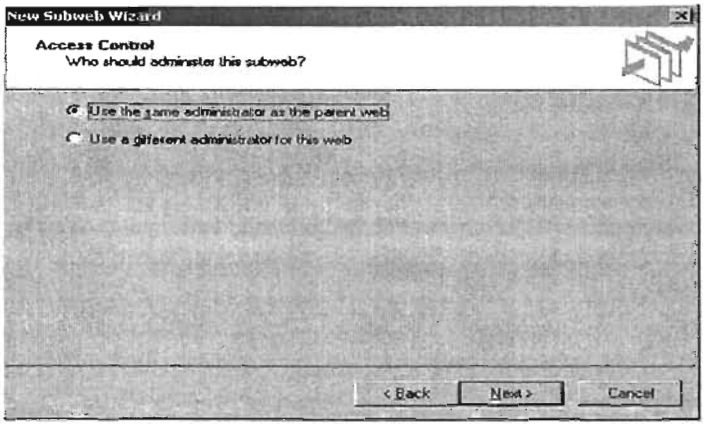

Figura 3.6 *Ventana* `Access Control`.

4. Aceptar la configuración de control de acceso por defecto dando click en `Next`, y luego click en `Finish` para crear el directorio subweb.

Creando un nuevo proyecto en el directorio virtual.

Desde Visual Studio .NET se puede crear un directorio virtual y los archivos por defecto de una aplicación Web, pero no es posible establecer la ubicación física del directorio virtual fuera del `Default Web Site`, ni nombres distintos para el directorio virtual y el directorio físico.

La ubicación que se especifica en la caja de dialogo de `New Project` toma la forma `http://NombreServidor/DirectorioVirtual` como se indica en la Figura 3.7. El nombre del servidor `localhost` indica que el servidor de publicación está corriendo en la misma computadora donde se está desarrollando, pero si no es así, se debe indicar el nombre del servidor. El nombre del `DirectorioVirtual` es el alias que identifica al directorio virtual.

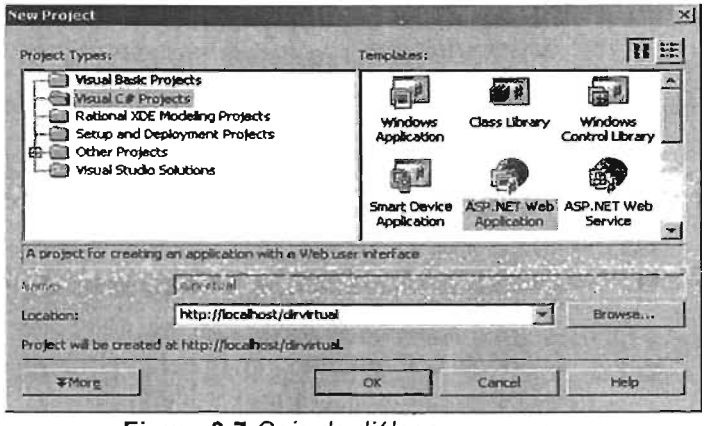

Figura 3.7 *Caja de diálogo* `New Project`.

Consideraciones sobre Web Forms.

- ***Herramientas.*** Los Web Forms utilizan controles de servidor Web (Web server controls), controles de servidor HTML (HTML server controls), controles de usuario (user controls), y controles personalizados (custom controls).

- ***Interfaz de usuario.*** La apariencia de un Web Form está determinada por el Web Browser que lo despliega. La aplicación Web puede visualizarse en Internet Explorer, Netscape Navigator, o cualquier otro Web Browser que soporte HTML. Dependiendo su versión, pueden variar sus características HTML, lo cual afecta la apariencia y comportamiento de los Web Forms.

- ***Ciclo de vida.*** Los Web Forms son instanciados, enviados al Web Browser, y destruidos inmediatamente. Esto significa que todas las variables y objetos declarados en el Web Form son destruidos tan pronto se despliega. Para tratar de mantener el estado de los objetos del Web Form se necesitará guardar información especial de estado de los objetos mediante ASP.NET.

- ***Ejecución.*** Los componentes ejecutables de una aplicación Web viven en el servidor Web. De esta forma, el Web Browser es el único software instalado del lado del cliente, y toda la interfaz de usuario, y lógica de negocio corre en el servidor. Toda la comunicación entre el cliente y el servidor ocurre a través de HTML.

Archivos en un proyecto Web Form.

Visual Studio .NET genera los archivos descritos en la Tabla 3.1 para cada proyecto de aplicación Web nuevo. Adicionalmente se reconoce otra lista de archivos comunes de una aplicación Web en la Tabla 3.2, los cuales se pueden ir agregando durante el desarrollo.

Archivo	Contiene
`assemblyInfo.cs`	Todas la opciones de construcción para el proyecto, incluyendo versión, nombre de compañía, GUID[24], opciones de compilación, entre otras.
`global.asax`	Los eventos globales que ocurren en la aplicación Web, por ejemplo, cuando la aplicación comienza o termina. Se puede tener únicamente un archivo `global.asax` por proyecto y se ubica en el directorio raíz del proyecto. Este archivo se utiliza además para definir variables globales y responder a eventos globales de la aplicación.
`global.asax.cs`	El código asociado utilizado en el `global.asax`.
`web.config`	La configuración que utiliza el servidor Web cuando se procesa el proyecto. Esta configuración determina cuantos errores son reportados, que tipo de autenticación de usuario utilizar, las opciones de seguridad personalizada, administración de estado, administración de memoria, entre otros. Se puede tener únicamente un archivo `web.config` por proyecto y se ubica en el directorio raíz del proyecto.
`WebForm1.aspx`	La descripción visual de un Web Form.
`WebForm1.aspx.cs`	El código que responde a eventos del Web Form.
`WebForm1.aspx.resx`	Los recursos XML utilizados por el Web Form.
`NombreProyecto.csproj`	El archivo de proyecto listando los archivos y configuraciones utilizados en tiempo de diseño.
`NombreProyecto.csproj.webinfo`	Las preferencias Web en tiempo de diseño para el proyecto.

Tabla 3.1 *Archivos de un proyecto Web Form.*

Archivo	Descripción
`.aspx`	Un Web Form está constituido por una página Web ASP.NET (`.aspx`) y archivos de código asociados con la extensión `aspx.cs`. Los archivos `.aspx` contienen la interfaz de usuario, o la apariencia visual del Web Form.
`.ascx`	Controles de usuario Web. Los controles de usuario permiten desarrollar una pieza de la interfaz de usuario y reutilizarlo en varios Web Forms sin repetir código.
`.asmx`	Los servicios Web funcionan de forma diferente a los Web Forms, pero comparten los mismos recursos de aplicación, opciones de configuración y memoria.
`.disco` `.vsdisco`	Archivos especiales de "descripción" utilizados para ayudar a los clientes a buscar servicios Web.
`.cs`	Archivos de código asociado creado con C#. Permite separar el código de lógica de negocio de la interfaz de usuario del Web Form.
`.css`	Las definiciones de estilo para utilizar en el HTML generado por el proyecto.

Tabla 3.2 *Tipos de archivo de un proyecto Web Form.*

[24] El GUID es un identificador para un determinado tipo de objeto en determinado tiempo y espacio, utilizado por el sistema para asociar una determinada instancia de objeto con la aplicación.

Compatibilidad con ASP.

ASP.NET y ASP clásico pueden convivir en el mismo servidor. Sin embargo, hay que considerar que ninguno de los archivos ASP básicos se utiliza en ASP.NET. Si existe un directorio con archivos `.aspx` y `.asp`, realmente existen dos aplicaciones. De hecho, el proceso que administra y representa los archivos `.asp` y el servicio .NET que compila y sirve los archivos `.aspx`, son dos programas separados que no comparten información. Por lo que:

- No es posible compartir información de estado entre aplicaciones ASP clásico y ASP.NET. Los objetos `Session` y `Aplication` existentes en ASP clásico y ASP.NET, por ejemplo, están completamente separados.

- Las opciones ASP clásico y ASP.NET se configuran de forma diferente. Si se define una opción para ASP clásico, no se aplica a ASP.NET y viceversa.

Generalmente, se deberían mantener separados los archivos de ASP clásico y ASP.NET para evitar la confusión. Sin embargo si se está migrando un sitio Web grande, es posible utilizar ambos tipos de archivo de forma segura (y transferir al usuario entre ellos), siempre que no intenten compartir recursos.

El directorio `bin`.

Cada directorio de aplicación Web tiene un subdirectorio especial llamado `\bin`, el cual mantiene los ensamblados .NET utilizados por la aplicación. Cuando se construye un proyecto Web Form, Visual Studio .NET compila todo el código fuente en un ejecutable (`.dll`) y ubica el archivo en el directorio `bin`. ASP.NET lo detecta automáticamente y permite que lo utilice cualquier página de esa aplicación. Esto es más fácil que el desarrollo de componentes tradicional basado en COM, que requiere que se registre un componente antes de utilizarlo (y a menudo volverlo a registrar cuando se modifica).

Modificaciones de aplicación.

A diferencia de ASP, una característica de ASP.NET es la llamada distribución sin retoques (*zero-touch*) lo que hace posible modificar una aplicación ASP.NET sin necesidad de reiniciar el servidor.

Modificaciones de páginas.

Si se modifica un archivo de código o un Web Form, ASP.NET recompila automáticamente una versión modificada con la siguiente solicitud de cliente. Esto implica que las nuevas solicitudes del cliente siempre utilizarán la versión más reciente de la página. ASP.NET utiliza compilación automática a código nativo y mantenimiento en caché para mejorar el rendimiento.

Modificaciones de componentes.

Es posible reemplazar cualquier ensamblado del directorio `\bin` con una nueva versión, incluso aunque la actual esté en uso. El archivo nunca se bloquea. También se puede añadir o eliminar archivos de ensamblado sin problemas. ASP.NET monitoriza

continuamente el directorio \bin buscando cambios. Cuando se detecta una modificación, se crea un nuevo dominio de aplicación y se utiliza para manejar cualquier nueva solicitud. Se terminan las solicitudes existentes utilizando el antiguo dominio de aplicación, que contiene una copia en caché de todas las versiones antiguas de los componentes. Cuando se finalizan todas las solicitudes existentes, se elimina este dominio de la aplicación.

Esta característica de vuelta atrás automática, algunas veces llamada copia sombra, permite modificar sin complicaciones un sitio Web que necesita estar continuamente activo.

Modificación de configuración.

Con ASP clásico, configurar una aplicación Web no era una tarea fácil. Era necesario crear una secuencia de comandos que modificase la metabase de IIS o utilizar el administrador de IIS. Una vez realizado el cambio, normalmente debía detenerse e iniciarse el servicio Web de IIS. Algunas veces incluso era necesario reiniciar el servidor antes de que la modificación surtiera efecto.

Estas tareas ya no son necesarias con ASP.NET, quien administra su propia configuración independientemente de IIS. La configuración de una aplicación Web se define utilizando un archivo web.config, del cual se hablará más adelante en la sección *"Configuración de ASP.NET"* de éste capítulo. El archivo web.config almacena la información en un texto plano en formato XML. Si se modifica alguna opción de web.config, se realiza el mismo mecanismo de recuperación que ocurre en la modificación de componentes. Se finalizan las solicitudes existentes con las configuraciones originales y se sirven las nuevas solicitudes con un nuevo dominio de aplicación que utiliza las nuevas opciones de web.config.

Código asociado.

Con el código asociado se crea un archivo de código separado (un archivo .cs en C#) correspondiente a cada archivo .aspx. El archivo .aspx contiene la lógica de la interfaz de usuario, que es la serie de código HTML y las etiquetas ASP.NET que crean los controles de la página. El archivo .cs contiene únicamente código con la lógica de negocio.

La directiva Page.

Al principio del archivo .aspx, se utiliza una directiva especial Page que identifica el archivo de código asociado correspondiente.

```
<%@ Page Language="c#" Codebehind="WebForm1.aspx.cs" Inherits="WebForm1" %>
```

Esta directiva define el lenguaje empleado (C#) en todos los bloques de código encerrados entre <% %>; identifica el archivo de clase que contiene todo el código con lógica de negocio de la página (WebForm1.aspx.cs); e identifica la clase de código asociada para que herede de la página (WebForm1), es decir, una clase derivada de la clase System.Web.UI.Page. Como el archivo .aspx y el código fuente asociado están en un archivo de texto plano. ASP.NET lo compila automáticamente en la siguiente solicitud Web.

Las directivas se procesan antes de ejecutar cualquier código ASP.NET y configuran varias opciones que influyen en cómo el compilador ASP.NET procesa el archivo. Configurar el lenguaje predeterminado y el archivo de código asociado son sólo dos posibilidades. También se pueden añadir atributos que configuren las opciones de mantenimiento de estado.

La herencia en Web Forms.

Hay tres estados involucrados como se muestra en la Figura 3.8

Figura 3.8 *Herencia en Web Forms.*

1. Primero, la clase `Page` de la biblioteca de clases .NET. Esta clase define la funcionalidad básica que permite que una página Web mantenga otros controles, los represente como HTML y proporciona acceso a los objetos ASP tradicionales, como `Request`, `Response` y `Session`.

2. Segundo, la clase asociada (archivo con extensión `.cs`). Esta clase hereda de la clase `Page` para adquirir el conjunto básico de funciones de la página Web de ASP.NET.

3. Por último, la página (archivo con extensión `.aspx`) hereda el código del Web Form clase recién creado. Esto permite combinar la interfaz de usuario con el código que lo soporta.

Ventajas del código asociado.

Las ventajas del código asociado aparecen al desarrollar aplicaciones complejas, multicapa. Algunas de las razones para utilizar código asociado son:

- ***Mejor organización.*** Una de las desventajas de ASP clásico es su mezcla de secuencias de comandos y etiquetas de formato HTML. Esto da como resultado una maraña de código que es difícil de depurar, imposible de reutilizar e incluso más difícil para otros programadores de comprender. Con código asociado, el código está encapsulado en una clase separada. Es más fácil de leer, más fácil de aislar y permite reutilizar código. El único modo de hacer esto mismo con ASP clásico es crear componentes COM separados.

- ***Separación de la interfaz de usuario.*** Otro de los problemas asociado con la programación de ASP clásico es la mezcla de formato y contenido (la información mostrada al usuario) con la lógica de programación. Esto orilla a los desarrolladores a integrar el diseño gráfico de un sitio Web, considerando que un diseñador gráfico profesional Web difícilmente entiende de programación. A

menudo se tiene que readaptar el código a los cambios de formato. Con el código asociado, el archivo `.aspx` se puede manipular, pulir y perfeccionar sin afectar la lógica de programación.

- ***Capacidad de utilizar editores de código avanzados.*** Utilizar la programación con código asociado se beneficia del uso de herramientas de desarrollo como Visual Studio .NET, las cuales pueden verificar automáticamente la sintaxis de los archivos de código asociado, proporciona finalización de sentencias *IntelliSense* y permite diseñar el Web Form `.aspx` correspondiente. Esto no implica que no podamos utilizar un editor de sólo texto para desarrollar aplicaciones, pero en un ambiente profesional de desarrollo, son recomendables estas herramientas.

Modos de codificar Web Forms.

Existen tres modos diferentes de programar Web Forms, como se describen en la Tabla 3.3.

Tipo de desarrollo	Se crean	Se distribuyen (al servidor web)
Código en línea tradicional	Archivos `.aspx` que contienen código y presentación de la interfaz de usuario.	Archivos `.aspx`
Código asociado	Archivos `.aspx` con la interfaz de usuario y archivos `.cs` con código.	Archivos `.aspx` y `.cs`
Código asociado compilado	Archivos `.aspx` con la interfaz de usuario y archivos `.cs` con código.	Archivos `.aspx` y los archivos `.dll` compilados (que van en el directorio `\bin`)

Tabla 3.3 *Tipos de desarrollo Web Forms.*

El archivo de aplicación `Global.asax`.

El archivo `global.asax` permite escribir código de aplicación global. El aspecto del archivo `global.asax` es similar a un archivo `.aspx` normal, excepto por el hecho de no poder contener etiquetas HTML o ASP.NET. En su lugar, contiene código de tratamiento de eventos que reacciona a los eventos de aplicación o sesión.

Cada aplicación ASP.NET puede tener un archivo `global.asax` en el directorio virtual apropiado. ASP.NET lo reconoce y lo utiliza automáticamente.

El archivo `global.asax` también admite el desarrollo con código asociado (`global.asax.cs`). Hereda de la clase especial `System.Web.HttpApplication` y utiliza una directiva `Application` (dentro de `global.asax`).

Eventos en el ciclo de vida de una aplicación Web.

Una aplicación Web vive mientras tenga sesiones activas. La vida de una aplicación Web comienza cuando el Web Browser solicita una página perteneciente a la aplicación (como se muestra en la Figura 3.9). En ese punto, el servidor Web toma acción, llamando el

ejecutable (`.dll`) que responde a esa petición. El ejecutable crea una instancia del Web Form solicitado, genera el HTML para responder la solicitud, y envía la respuesta al Web Browser. Este mismo destruye la instancia del Web Form.

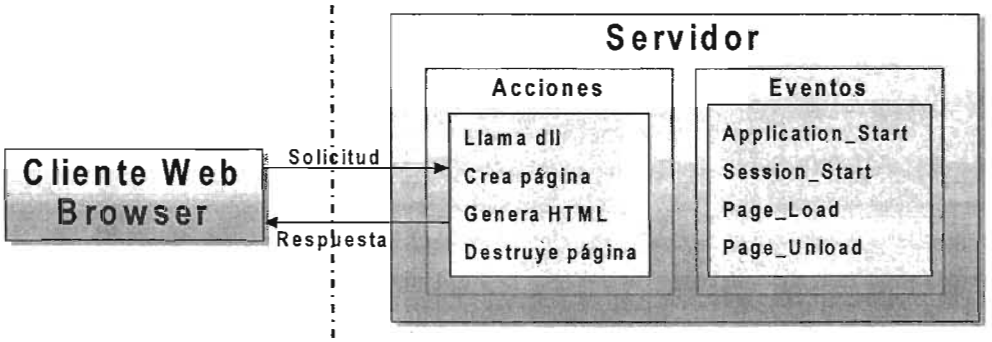

Figura 3.9 *Ciclo de vida de una aplicación Web.*

Cuando el Web Browser tiene el HTML generado, el usuario puede interactuar con la página a través de cajas de texto, botones de selección de opciones, entre otros. Algunos elementos de la página pueden disparar eventos que hacen que el Web Browser envíe los datos de la página de vuelta al servidor para que sean procesados. Cuando el servidor recibe la información, crea una nueva instancia del Web Form, y procesa cualquier evento que haya ocurrido (ver Figura 3.10). Tan pronto termina el servidor, se envía el resultado de vuelta en formato HTML al Web Browser y se destruye la instancia del Web Form.

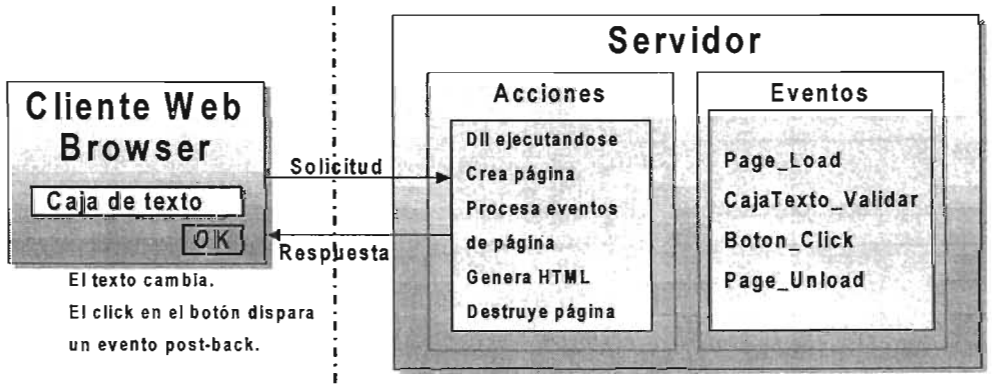

Figura 3.10 *Procesamiento de eventos.*

Cuando el usuario termina de utilizar la aplicación y cierra el Web Browser o se marcha a otro sitio Web, la sesión de usuario termina (ver Figura 3.11). Si no hay otras sesiones de otros usuarios, la aplicación termina. Esto no siempre pasa enseguida. ASP.NET administra la memoria usando el garbage collection, esto es, que el servidor rastrea periódicamente las referencias entre objetos. Cuando el servidor encuentra un objeto que ya no se está utilizando, lo desecha y recupera la memoria. Esto significa que no se sabe exactamente cuando ocurre el evento `Application_End`.

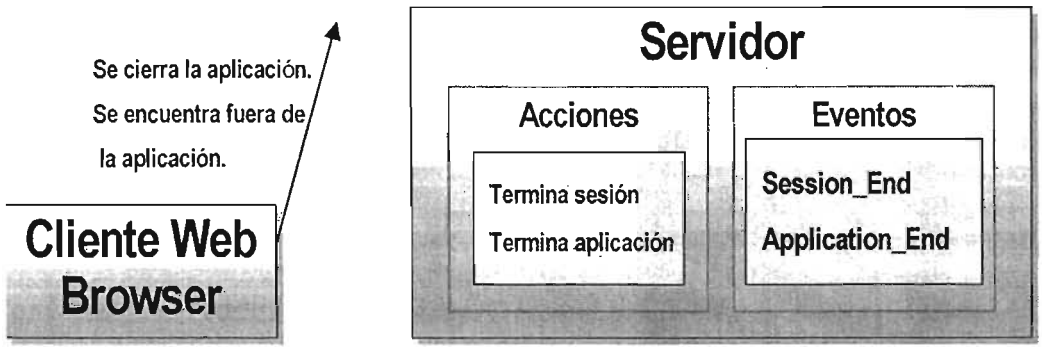

Figura 3.11 *Fin de aplicación Web.*

Eventos de aplicación y sesión.

Se puede escribir código para responder a eventos de aplicación y sesión en el archivo `global.asax`. Utilizar los *eventos de aplicación* para inicializar objetos y datos que se quieran tener disponibles para todas las sesiones actuales de la aplicación Web. Utilizar los *eventos de sesión* para inicializar datos que se quieran guardar a través de sesiones individuales, pero que no se quieran compartir entre sesiones. La Tabla 3.4 lista cada uno de los manejadores de eventos de la aplicación en el `global.asax` y describe cuando ocurren.

Manejador de evento	Ocurre cuando
`Application_Start`	Se recibe la solicitud de una página de la aplicación Web por cualquier usuario. No ocurre en las siguientes solicitudes. Este evento normalmente se utiliza para crear o mantener en caché alguna información inicial que se utilizará con frecuencia.
`Application_End`	Cuando se cierra la aplicación, normalmente cuando se reinicia el servidor. Aquí se pone el código de liberación de recursos.
`Application_BeginRequest`	Se hace una solicitud que recibe la aplicación, justo antes de ejecutar el código de la página.
`Application_EndRequest`	Se hace una solicitud que recibe la aplicación, justo después de ejecutar el código de la página.
`Session_Start`	Un nuevo usuario visita una página de la aplicación.
`Session_End`	Un usuario deja la aplicación, ya sea cerrando el Web Browser o por vencimiento de sesión.
`Application_Error`	Ocurre cuando se encuentra un error no manejado.

Tabla 3.4 *Manejadores de eventos en la aplicación.*

En Web Forms, una sesión es una instacia única del Web Browser. Un usuario puede tener múltiples instancias del Web Browser corriendo en la computadora. Si cada instancia visita la aplicación Web, cada instancia tiene una sesión única.

Para ver como ocurren los eventos de aplicación y sesión, considerar el siguiente código dentro del archivo `global.asax` en un proyecto Web Forms:

```
// De Global.asax
protected void Application_Start(Object sender, EventArgs e)
{
  // Crea las variables de estado aplicación.
  Application["ContApp"] = 0;
  Application["ContSess"] = 0;
```

```
  // Graba el comienzo de aplicación.
  Application["ContApp"] = (int)Application["ContApp"] + 1;
}

protected void Session_Start(Object sender, EventArgs e)
{
  // Graba el número de sesiones.
  Application["ContSess"] = (int)Application["ContSess"] + 1;
  // Despliega el contador de aplicaciones.
  Response.Write("Numero de aplicaciones: " +
    Application["ContApp"] + "<br>");
  // Despliega el contador de sesiones.
  Response.Write("Numero de sesiones: " +
  Application["ContSess"] + "<br>");
}
```

Para demostrar los eventos basta con solicitar cualquier Web Form de la aplicación, desde una instancia del Web Browser. Cada nueva instancia del Web Browser incrementa el contador de sesión, mientras que el contador de aplicación permanece en 1.

Es importante resaltar que los objetos `Session` y `Response` no están disponibles en el evento `Application_Start`. Para utilizar estos objetos, se tiene que esperar hasta que su evento de creación ocurra.

Eventos de Web Forms.

Se utilizan los eventos de Web Forms para procesar y mantener datos usados en una página Web, para responder a datos ligados, y para manejar excepciones. Los eventos que se muestran en la Tabla 3.5 ocurren en el orden que se muestran. El evento `Page_Load` es el evento más utilizado al codificar.

Manejador de evento	Ocurre cuando
`Page_Init`	Los controles de servidor del Web Form son cargados e inicializados. Este es el primer paso en el ciclo de vida de un Web Form.
`Page_Load`	Los controles de servidor son cargados en el objeto `Page`. La información de la variable `ViewState` está disponible hasta este punto, por lo que en este evento es donde se pone código para cambiar la configuración de controles o despliegue de texto en la página.
`Page_PreRender`	Los controles de servidor están por dar su salida al objeto `Page`. Cualquier cambio al `ViewState` del control de servidor puede ser guardado durante este evento.
`Page_Unload`	La página es descargada de memoria.
`Page_DataBinding`	Un control de la página se liga a un origen de datos.
`Page_Disposed`	El objeto `Page` es liberado de memoria. Este es el último evento en el ciclo de vida de un objeto `Page`.

Tabla 3.5 *Eventos de un Web Form.*

Se puede utilizar el evento `Page_Load` en conjunto con la propiedad `IsPostback` del objeto `Page` para inicializar datos la primera vez que el usuario visita el Web Form. Esto es similar al evento `Session_Start`; sin embargo, esto ocurre a nivel página más que nivel aplicación.

Configuración de ASP.NET.

Todo servidor Web tiene un único archivo `machine.config` que define las opciones predeterminadas. Normalmente, este archivo no se edita manualmente. Contiene una mezcla compleja de opciones específicas de la computadora. En su lugar, se puede crear un archivo `web.config` para aplicaciones propias que contenga todas las opciones especiales.

El archivo `machine.config` está ubicado en el directorio donde reside la instalación del Framework .NET, por ejemplo: `C:\WINNT\Microsoft.NET\Framework\v1.1.4322\ machine.config`.

El archivo `web.config` está ubicado en el directorio físico donde reside la aplicación ASP.NET, por ejemplo: `C:\Inetpub\wwwroot\dirfisico`.

El archivo `web.config` tiene varias ventajas con respecto a la configuración de ASP clásico:

- **Nunca se bloquea.** Es posible modificar las opciones de `web.config` en cualquier momento y ASP.NET tratará suavemente la transición a un nuevo dominio de aplicación.
- **Es fácil de acceder y replicar.** Con los derechos apropiados se puede modificar un archivo `web.config` desde una computadora remota, también se puede copiar el mismo archivo y utilizarlo para aplicar la misma configuración en otra aplicación u otro servidor Web que ejecute la misma aplicación en un escenario de granja Web.
- **Es fácil de editar y comprender.** Las opciones del archivo `web.config` son legibles y puede modificarse manualmente.

El archivo `web.config`.

El archivo `web.config` utiliza un formato especial XML. Todo está anidado en un elemento raíz `<configuration>`, que contiene un elemento `<system.web>`. Dentro del elemento `<system.web>` se encuentran elementos separados para cada aspecto de la configuración.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <!-- La sección de configuración_ul18 ?iene aquí. -->
  </system.web>
</configuration>
```

Se pueden incluir tantas secciones de configuración como se deseen. Al crear una aplicación ASP.NET desde Visual Studio .NET se crea un archivo `web.config` con un esqueleto básico mostrando las secciones más significativas. Se añaden comentarios adicionales en cada sección que describen el propósito de las opciones.

Configuración anidada.

ASP.NET utiliza un sistema de configuración multinivel que permite utilizar opciones diferentes para diferentes secciones de la aplicación. Para utilizar esta técnica es necesario crear subdirectorios adicionales dentro del directorio virtual. Estos subdirectorios pueden contener su propio archivo `web.config` con opciones adicionales.

Los subdirectorios también heredan las opciones de `web.config` del directorio padre. Por ejemplo, considere la solicitud Web `http://localhost/dirvirtualpadre/dirhijo/pagina.aspx`, que accede a un archivo dentro del directorio `dirhijo`, que es un subdirectorio del directorio virtual de la aplicación `dirvirtualpadre`. Supongamos que el directorio físico correspondiente es `C:\Inetpub\wwwroot\dirfisico`.

La solicitud Web de `pagina.aspx` puede adquirir opciones de configuración de tres archivos diferentes como se muestra en la Figura 3.12.

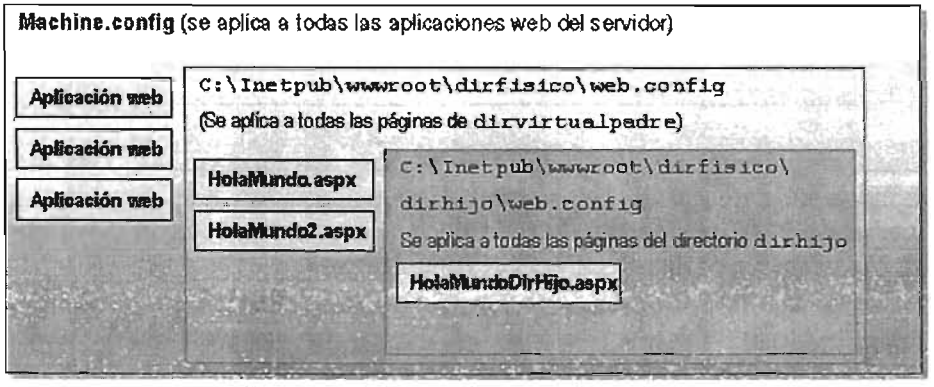

Figura 3.12 *Herencia de configuración.*

Se aplicará a la página `pagina.aspx` cualquier opción de `machine.config` o `C:\Inetpub\wwwroot\dirfisico\web.config` que no sobrescriba explícitamente en el archivo `C:\Inetpub\wwwroot\dirfisico\dirhijo\web.config`. De este modo los subdirectorios pueden definir un conjunto menor de opciones que se diferencien del resto de la aplicación Web. Una razón para utilizar múltiples directorios en una aplicación es aplicar diferentes configuraciones de seguridad. Los archivos que se necesiten asegurar se podrían ubicar en un directorio especial con un archivo `web.config` que defina opciones de seguridad más restrictivas.

Manejo de procesos de aplicaciones Web.

Las aplicaciones Web usan IIS para manejar sus procesos en el servidor. IIS determina que páginas Web son parte de una aplicación particular, como se ejecutan las DLLs de la aplicación en el servidor, y como se determina el fin de una aplicación.

Estableciendo las fronteras de la aplicación.

IIS define una aplicación Web como cualquier archivo que se ejecuta dentro de un conjunto de directorios en el sitio Web. Las fronteras de una aplicación Web son determinadas por la estructura de sus directorios. Las fronteras de la aplicación comienzan en el directorio que contiene la página de inicio de la aplicación y termina hasta el último directorio subordinado. Las aplicaciones Web pueden estar anidadas, y se identifican como un directorio virtual dentro del sitio Web o dentro de otro directorio virtual.

Manejando los procesos.

Establecer las fronteras de una aplicación es importante, porque IIS usa estas fronteras para determinar como se ejecuta la aplicación en el servidor.

- ***En-proceso (In-process) con IIS*** (`Inetinfo.exe`). Ésta opción incrementa el rendimiento, pero no ofrece protección. Si una aplicación falla, puede corromper memoria y afectar `Inetinfo.exe`, así como otras aplicaciones corriendo en-proceso.

- ***Agrupado (Pooled) con otros procesos de aplicaciones Web en*** `DLLHost.exe`. Esta es la opción por defecto y proporciona un balance entre protección y rendimiento. Si una aplicación falla, puede afectar otras aplicaciones en el pool, pero no afectar `Inetinfo.exe`.

- ***Aislado (Isolated) en su propia instancia de*** `DLLHost.exe`. Las aplicaciones aisladas son protegidas de afectar o ser afectadas por problemas en otras aplicaciones. Sin embargo, las llamadas a otras aplicaciones deben cruzar fronteras de proceso, y esto afecta el rendimiento.

Los controles de servidor ASP.NET.

Los *controles de servidor* ASP.NET son componentes que se ejecutan en el servidor encapsulando la interfaz de usuario y funcionalidad relacionada.

En el desarrollo con ASP clásico, los desarrolladores tenían que conocer en gran medida el lenguaje HTML para programar páginas Web dinámicas. Cumplir con la apariencia visual y la lógica de negocio se vuelve una tarea compleja cuando aún se tiene que preocupar por el armado de cadenas de texto que componen las etiquetas HTML. Estas aplicaciones normalmente necesitaban grandes modificaciones de código cuando la interfaz de usuario del sitio Web cambiaba.

En ASP.NET se puede utilizar un modelo de nivel superior de controles Web del lado del servidor. Estos controles se crean y configuran como objetos y automáticamente proporcionan sus propios resultados HTML. Incluso mantienen su estado y disparan eventos para realizar una acción determinada mediante código en el servidor. Esto quiere decir que la entrada de datos del usuario se guarda automáticamente cuando la página se envía de vuelta del Web Browser al servidor.

A diferencia de los controles de servidor ASP.NET, los elementos HTML corren únicamente en el Web Browser del cliente y no tienen acción en el servidor. Además no conservan estado, por lo que regresan a su estado inicial cuando se regresa la página del servidor al cliente.

En ASP.NET aún se puede utilizar `Response.Write` para el armado manual de etiquetas HTML, o utilizar los controles de servidor:

Los controles de servidor tienen el atributo `runat="server"`. Esto significa que la lógica del control se ejecuta en el servidor y no en el Web Browser.

Cuando una página es generada para un Web Browser, los controles de servidor Web determinan que Web Browser está solicitando la página y entonces entrega el HTML apropiado.

- ***Controles de servidor HTML.*** Son los equivalentes de servidor a los elementos estándar HTML.

- ***Controles de servidor Web.*** Son similares a los controles HTML, pero proporcionan un modelo de objetos más completo, con una mayor variedad de propiedades de estilo y formato, más eventos. Estos controles son más abstractos que los controles de servidor HTML, en el sentido de que no necesariamente reflejan sintaxis HTML sino que tienen un propósito especial, por ejemplo un objeto calendario, el cual puede estar compuesto de varios elementos HTML.

La Tabla 3.6 lista los controles de servidor y controles HTML por tarea de programación.

Tarea	Controles de servidor Web	Controles de servidor HTML
Desplegar texto	Label, TextBox, Literal	Label, Text Field, Text Area, Password Field
Desplegar tablas	Table, DataGrid	Table
Seleccionar de una lista	DropDownList, ListBox, DataList, Repeater	ListBox, Dropdown
Ejecutar comandos	Button, LinkButton, ImageButton	Button, ResetButton, SubmitButton
Establecer valores Boolean	CheckBox, CheckBoxList, RadioButton, RadioButtonList	Checkbox, RadioButton
Desplegar imagenes	Image, ImageButton	Image
Navegación	Hyperlink	Ninguno, utilizar tags <a> en texto.
Agrupar controles	Panel, Placeholder	Flow Layout, Grid Layout
Trabajar con fechas	Calendar	Ninguno
Desplegar anuncios	AdRotator	Ninguno
Desplegar reglas horizontales	Literal	Horizontal Rule
Obtener nombres de archivo del cliente	Ninguno	File Field
Almacenar datos en la página	Proporcionada por administración de estado	Input Hidden
Validar datos	RequiredFieldValidator, CompareValidator, RangeValidator, RegularExpressionValidator, CustomValidator, ValidationSummary	Ninguno (utilizar script a nivel página)

Tabla 3.6 Controles por tarea de programación.

Seleccionando el control apropiado.

Cuando se crean paginas ASP.NET, se tiene la opción de utilizar controles de servidor HTML y controles de servidor Web en la misma página. La mejor práctica es no utilizar los controles de servidor HTML, aunque eso depende del escenario en el que se este desarrollando.

Hagamos una comparación de características comunes de ambos tipos de controles en la Tabla 3.7.

Característica	Controles de servidor Web	Controles de servidor HTML
Eventos de servidor	Control de eventos específicos en el servidor.	Pueden disparar únicamente eventos a nivel página (post-back).
Administración de estado	Los datos introducidos en un control se mantienen entre solicitudes.	Los datos no se mantienen, deben ser guardados y restaurados utilizando scripts a nivel página. Son útiles cuando los datos son refrescados en cada solicitud.
Adaptación	Automáticamente detecta el Web Browser y adapta su visualización como sea apropiado.	No hay adaptación automática, se debe detectar el Web Browser en código. Se tiene control completo sobre la apariencia.
Propiedades	El Framework .NET proporciona un conjunto de propiedades para cada control. Las propiedades nos permiten cambiar la apariencia de los controles y comportamiento.	Únicamente atributos HTML.
Modelo de objetos	Si se necesita un modelo de programación más cercano al estilo de Visual Basic. Se tiene la posibilidad de usar POO, identificar los controles, y separar la lógica de la interfaz de usuario. Con los controles de servidor Web se pueden anidar controles y detectar eventos a nivel contenedor.	Se prefiere el modelo de objetos HTML. Los controles de servidor HTML tienen casi la misma sintaxis de elementos HTML estándar y pueden tener funcionalidad del lado del servidor.
Migración	Se está desarrollando una nueva aplicación.	Se están migrando las páginas de ASP clásico a ASP.NET, lo cual requiere de menos modificaciones. Debido a que los controles de servidor HTML mapean exactamente a elementos HTML estándar, no se corren riesgos de errores de sustitución o problemas de formato al remplazar controles.
Código del lado del cliente	Genera automáticamente el código para que todos los eventos y procesos se ejecuten del lado del servidor.	El control necesita de código del lado del cliente (sript) y de lado del servidor.
Ancho de banda[25]	El ancho de banda no está limitado y los ciclos de solicitudes-respuestas de controles de servidor Web no causan problemas de ancho de banda.	El ancho de banda es limitado y se necesita hacer una gran cantidad de proceso del lado del cliente para reducir el uso de ancho de banda.

Tabla 3.7 *Server controls vs. HTML controls.*

[25] El ancho de banda es la capacidad máxima de datos que pueden pasar por un canal de comunicación en un momento dado medido normalmente en segundos.

Utilizando controles.

Los controles definen la apariencia de la forma y proporcionan una manera para obtener información y realizar tareas del usuario.

Posicionando controles en el Web Form.

Grid Layout. Este es el posicionamiento por defecto. Los controles son ubicados exactamente donde se dibujan y tienen una posición absoluta en la página. Utilizarlo si los controles no se mezclan con largas cantidades de texto.

Flow Layout. Este posiciona los controles relativamente a otros elementos en la página. Si se agregan elementos en tiempo de ejecución, los controles que se agregan después del nuevo elemento se mueven abajo. Utilizarlo para aplicaciones estilo documento, en las cuales el texto y controles se mezclan.

Trabajando con texto.

Una manera para desplegar texto de solo lectura en una página, es escribiendo directamente al objeto `Response`. cómo `Response.Write("Texto");` se puede utilizar un control `Label` ó `TextBox` estableciendo su propiedad `ReadOnly` a `True`; entre otros objetos.

Trabajando con tablas y listas.

El texto desplegado en `Labels` y `Text Boxes` es colocado en un bloque sencillo. Para colocar texto en renglones y columnas, se necesita utilizar un control de tabla o lista descritos en la Tabla 3.8. Utilizar los controles `ListBox`, `DropDownList`, y `Table` para tablas y listas dinámicas simples. Utilizar el `DataGrid`, `DataList`, y `Repeater` para tablas complejas y listas que contienen otros controles o cargan datos.

Control	Utilizado para
`ListBox`	Despliega texto de solo lectura en formato de lista con barra de desplazamiento.
`DropDownList`	Despliega texto de solo lectura en formato de lista desplegable.
`Table`	Despliega texto y controles en columnas y renglones. Los controles de Tabla nos permiten construir dinámicamente tablas en código utilizando colecciones `TableRows` y `TableCells`.
`DataGrid`	Despliega texto y controles en columnas y renglones utilizando una plantilla para controlar la apariencia. Los controles `DataGrid` incluyen en su construcción capacidades de formateo, ordenamiento, y paginación.
`DataList`	Despliega renglones de texto y controles utilizando una plantilla para controlar la apariencia. Los controles de `DataList` incluyen en su construcción capacidades de formateo y selección.
`Repeater`	Despliega renglones de otros controles utilizando una plantilla para controlar la apariencia. Los controles de `Repeater` no incluyen en su construcción capacidades que se encuentran en los controles de `DataGrid` y `DataList`.

Tabla 3.8 *Controles de Lista y Tabla.*

Ejecutar comandos.

Los controles `Button`, `LinkButton`, e `ImageButton` disparan eventos de post-back, los cuales comienzan una solicitud desde el Web Browser al servidor para procesar los eventos y datos de la página. El evento `Click` de un control botón es procesado después de cualquier validación o eventos detectados en una página.

Obteniendo y estableciendo valores.

Utilizar los controles `RadioButton`, `RadioButtonList`, `CheckBox`, o `CheckBoxList` para obtener valores `Boolean` del usuario. Adicionalmente se pueden agrupar para selecciones múltiples o discriminación de opciones.

Desplegando gráficas y anuncios.

Hay muchas maneras para desplegar gráficas en un Web Form:

- ***Como fondo.*** Utilizar la propiedad `Background` del Web Form para desplegar una imagen en la página completa. Utilizar la propiedad `BackImageUrl` para desplegar una imagen de fondo en una región de la página, en lugar de la página completa.

- ***Como primer plano.*** Utilizar el control `Image` para desplegar imágenes en primer plano.

- ***Como botón.*** Utilizar el control de `ImageButton` para desplegar imágenes que respondan a los eventos de usuario.

- ***Como un anuncio.*** Utilizar el control `AdRotator` para desplegar imágenes de una lista de anuncios. Los anuncios mostrados en el control `AdRotator` contienen hiperligas al sitio Web del anunciante.

Agrupando controles.

Ubicar controles en un grupo cuando se quiera manipular una región particular del Web Form en código. Por ejemplo, se quiere permitir a los usuarios que se firmen a la aplicación en una región de la página, y entonces ocultar o deshabilitar esa región una vez que se haya firmado.

Validación de datos.

Uno de los pasos más importantes en la obtención de datos de un usuario es asegurarse que los datos son válidos. La validación se determina por ciertos criterios: ¿El usuario introdujo algo?, ¿Es la entrada del tipo de dato correcto?, ¿Se encuentran los datos dentro del rango requerido?

ASP.NET proporciona controles de validación para ayudar a verificar entradas de datos en los Web Forms contra los criterios definidos antes de que los datos sean aceptados.

Es pertinente aclarar que las validaciones de aplicaciones Web deben hacerse en parte del lado del servidor y otras del lado del cliente, así se evitarán vueltas innecesarias de la página y por motivos de seguridad. Adicionalmente hay que notificar los errores que se hayan detectado de la validación. El hacer rutinas de validación propias es posible pero gasta una cantidad considerable de esfuerzo en escribir el código.

ASP.NET proporciona un entorno reutilizable de controles de validación que administran los detalles comprobando campos e informando de los errores automáticamente. Estos controles se pueden utilizar desde el cliente con DHTML[26] y código JavaScript.

Los controles de validación.

ASP.NET proporciona cinco controles diferentes de validación que se muestran en la Tabla 3.9:

Clase de Control	Descripción
`RequiredFieldValidator`	La validación es correcta cuando el control de entrada no contiene una cadena vacía. Verifica si un control contiene datos.
`RangeValidator`	La validación es correcta cuando el control de entrada contiene un valor dentro de un intervalo numérico, alfabético o temporal especificado.
`ComprareValidator`	La validación es correcta si el control contiene un valor que corresponde con el valor de otro control especificado.
`RegulaExpressionValidator`	La validación es correcta si el valor de un control de entrada corresponde con una expresión regular especificada.
`CustomValidator`	La validación la realiza una función definida por el usuario.

Tabla 3.9 *Controles de validación.*

Cada control de validación se puede vincular a un control de entrada. Además, se puede aplicar más de un control de validación al mismo control de entrada para proporcionar varios tipos de validación.

Notar que la validación siempre da un resultado correcto en los controles que no contienen ningún valor. Si se utilizan los controles de validación `RangeValidator`, `CompareValidator` o `RegularExpresionValidator`, la validación siempre dará un resultado correcto si el control de entrada está vacío dado que no existe valor que validar. Si no es el comportamiento deseado, se debe añadir un `RequiredFieldValidator` adicional al control.

El proceso de validación.

Se pueden utilizar los controles de validación para verificar una página automáticamente cuando el usuario la envía o manualmente mediante código.

1. El usuario recibe una página y comienza a llenar los controles de entrada.

[26] DHTML es un conjunto de características que incorporan los Web Browser para modificar la apariencia ó contenido de un documento, accedidas mediante un lenguaje de programación.

2. Al finalizar, el usuario pulsa un botón para enviar la información de la página al servidor.

3. Cada control de tipo `Button` tiene una propiedad `CausesValidation`.

   - Si el valor de esta propiedad es `False`, ASP.NET ignora los controles de validación, la página se reenvía y se ejecuta la rutina de tratamiento normalmente.

   - Si el valor de esta propiedad es `True`, ASP.NET valida automáticamente la página cuando el usuario pulsa el botón. Se realiza la validación de cada control de la página. Si cualquier control genera un error en la validación, ASP.NET devolverá la página con información de error, dependiendo de la configuración. Puede que se ejecute o no la rutina de tratamiento de evento; dependiendo de si se tiene una comprobación específica en la rutina de tratamiento de eventos de si la página es válida o no.

Basándose en esta descripción, se notará que la validación sucede automáticamente cuando se pulsan determinados botones, no cuando la página se envía por un evento de modificación (como seleccionar un nuevo valor en una lista `AutoPostBack`) o si el usuario pulsa un botón que tiene su propiedad `CausesValidation` con valor `False`.

En los Web browser que lo soporten (Internet Explorer 5 y superiores), ASP.NET añade automáticamente código para validación del lado del cliente. En este caso, cuando el usuario pulsa el botón con el valor de la propiedad `CausesValidation` igual a `True`, aparecerán los mensajes de error sin que sea necesario reenviar la página al servidor.

Aún si la página se valida correctamente en el lado del cliente, ASP.NET la volverá a validar cuando se reciba en el servidor. Esto se hace con fines de seguridad, ya que es fácil evitar la validación del lado del cliente.

Sugerencia de pasos al utilizar controles de validación.

1. Dibujar un control de validación en un Web Form y establecer la propiedad de `ControlToValidate` con el nombre del control a validar. Si se utiliza el control `CompareValidator`, se necesitará especificar también la propiedad `ControlToCompare`.

2. Establecer la propiedad `ErrorMessage` del control con el mensaje de error que se quiera desplegar si los datos en el control no son válidos.

3. Establecer la propiedad `Text` del control si se quiere indicar brevemente donde ocurrió el error dentro del Web Form y despliega la descripción larga del mensaje en un control `ValidationSummary`.

4. Dibujar un control `ValidationSummary` en el Web Form para desplegar los mensajes de error desde los controles de validación en un solo lugar.

5. Proporcionar un control que dispare eventos post-back. Aunque ocurra una validación del lado del cliente, la validación no comienza hasta que sucede un evento post-back.

Validación manual.

Para realizar tipos complejos de validación no proporcionados por los controles de validación estándar, es necesario hacer validaciones manuales. Esto permite tomar en consideración otro tipo de información o crear un mensaje de error especializado que lleve asociado otros controles (como imágenes o botones).

Existen tres formas de crear validaciones manuales:

- Utilizar código propio para verificar los valores. En este caso, no se utilizará ninguno de los controles de validación de ASP.NET.

- Inhabilitar la propiedad `EnableClientScript` de cada control de validación. Esto permite que se envíen páginas incorrectas, para después decidir lo que se hará en caso de errores de validación.

- Añadir un botón con la propiedad `CausesValidation` igual a `False`. Cuando se pulse este botón, validar manualmente la página llamando al método `Page.Validate`. Después examinar la propiedad `IsValid` y decidir que hacer.

Navegación entre Web Forms.

Ligar información es la esencia del Web. En aplicaciones Web Forms, las hiperligas y los métodos de navegación son lo que hace posible ligar múltiples Web Forms en una aplicación. ASP.NET proporciona varias maneras de navegar entre páginas en la aplicación, y cada una de estas técnicas surte diferentes efectos en términos de cómo se despliega la página y como se intercambian los datos entre páginas. La Tabla 3.10 muestra cinco maneras de navegar entre páginas.

Método de navegación	Utilizado para
`Control Hyperlink`	Navegar a otra página.
`Método Response.Redirect`	Navegar a otra página desde código. Este es equivalente a hacer click en una hiperliga.
`Método Server.Transfer`	Finalizar el Web Form actual y comenzar la ejecución de uno nuevo. Este método trabaja únicamente cuando se navega a páginas de tipo Web Forms (`.aspx`).
`Método Server.Execute`	Comenzar la ejecución de un nuevo Web Form mientras permanece desplegándose la forma actual. Los contenidos de ambos Web Forms son combinados. Este método trabaja únicamente cuando se navegan páginas de tipo Web Forms.
`Método Window.Open`	Despliega la página en una nueva ventana del Web Browser en el cliente.

Tabla 3.10 *Navegación entre páginas.*

Utilizando hiperligas y redirección.

Los controles de servidor `Hyperlink` responden a eventos click del usuario desplegando la página especificada en la propiedad `NavigateURL`. El control `Hyperlink` no expone ningún

evento de usuario del lado del servidor; si se quiere interceptar un evento click en el código, utilizar el control `LinkButton` o el control de servidor `ImageButton`.

Para navegar desde un `LinkButton` o `ImageButton`, utilizar el método `Response.Redirect` como se muestra a continuación:

```
private void LinkButton1_Click(object sender, System.EventArgs e)
{
  // Despliega la página siguiente.
  Response.Redirect("SiguientePágina.aspx");
}
```

Utilizando el método `Transfer`.

Es muy similar a ejecutar una hiperliga o utilizar el método `Redirect`, con una diferencia: `Transfer` puede retener alguna información del origen de la página entre solicitudes. Estableciendo el argumento `preserveForm` del método `Transfer` a `true` hace que la información de `QueryString`, `ViewState`, y procedimientos de evento estén disponibles en el Web Form destino.

Para utilizar el método `Transfer` con `preserveForm` establecido a `true`, se debe establecer primero el atributo `EnableViewStateMac` en la directiva de página del Web Form a `false`.

```
<%@ Page language="c#" EnableViewStateMac="false"
 Codebehind="WebForm1.aspx.cs" AutoEventWireup="false"
 Inherits="proyecto.cap4.WebForm1" %>
```

El siguiente método de evento de un `ImageButton` muestra como se pasa la información entre Web Forms con el método `Transfer`:

```
private void LinkButton1_Click(object sender, System.EventArgs e)
{
  // Transfiere a otra forma, reteniendo ViewState.
  Server.Transfer("Webform2.aspx", true);
}
```

Utilizar el objeto `Request` con el método `Form` para extraer la información de `ViewState` del Web Form origen. El siguiente código despliega el contenido de un control `TextBox` al realizarse el evento click del `ImageButton` en el código anterior de `Webform1.aspx` en `Webform2.aspx`.

```
private void Page_Load(object sender, System.EventArgs e)
{
  System.Collections.Specialized.NameValueCollection colForm;
  // Obtener datos de la forma Web origen.
  colForm = Request.Form;
  // Desplegar el valor del Webform1.TextBox1.
  Response.Write("TextBox1.Text: " + colForm["TextBox1"]);
}
```

Utilizando el método `Execute`.

Utilizar el método `Execute` del objeto `Server` para procesar un segundo Web Form sin irse del primer Web Form. Esto permite redireccionar los resultados de una Web Form a una región en la página actual. Así como sucede con el método `Transfer`, `Execute` requiere que el atributo `EnableViewstateMac` del Web Form se establezca a `false` para deshabilitar la encriptación de `ViewState`.

Por ejemplo, el siguiente código ejecuta el Web Form `Tabla.aspx` y lo despliega en un control `Literal` en la página actual.

```
private void Button1_Click(object sender, System.EventArgs e)
{
  System.IO.StringWriter swr = new System.IO.StringWriter();
  // Ejecuta un Web form, almacena los resultados.
  Server.Execute("Tabla.aspx", swr);
  // Despliega los resultados en un control literal.
  Literal1.Text = "<h2>Tabla</h2>" + swr.ToString();
}
```

Contenido HTML de la página `Tabla.aspx`:

```
<asp:Table id="Table1" style="Z-INDEX: 101; LEFT: 198px; POSITION: absolute; TOP: 304px"
runat="server"
    Width="217px" Height="65px" BorderStyle="None">
    <asp:TableRow>
        <asp:TableCell Text="Ren0Col0"></asp:TableCell>
        <asp:TableCell Text="Ren0Col1"></asp:TableCell>
    </asp:TableRow>
    <asp:TableRow>
        <asp:TableCell Text="Ren1Col0"></asp:TableCell>
        <asp:TableCell Text="Ren1Col1"></asp:TableCell>
    </asp:TableRow>
</asp:Table>
```

El segundo argumento del método `Execute` es opcional. Si se omite, el resultado se escribe al final de la página actual en un nuevo tag `<html>`. El contenido de ambas páginas es desplegado una vez que los controles de servidor de ambas páginas puedan responder a eventos de usuario.

Cuando se combinen Web Forms utilizando el método `Execute`, hay que estar consientes de que cualquier evento post-back que ocurre en el segundo Web Form dejará en blanco el primer Web Form. Por ésta razón, el combinar Web Forms es útil cuando el segundo Web Form no contiene controles que disparen eventos post-back.

Desplegando una página en una nueva ventana del Web Browser.

Para abrir una nueva ventana del Web Browser, utilizar el método `Open` del objeto `Window` del lado del cliente. Esto se puede hacer únicamente como parte del script del cliente porque la nueva ventana es creada en el cliente. Sin embargo, hay manera para controlar el contenido y la apariencia de la nueva ventana del Web Browser desde el servidor.

El método `Window.Open` abrirá la forma que se especifique en el evento `onclick` del Botón HTML:

```
<INPUT onclick="window.open('webform2.aspx')" type="submit" value="Nueva Ventana">
```

Para utilizar una variable como target URL, reemplazar `webform2.aspx` con un data tag:

```
<INPUT onclick="window.open('<%# urlTarget %>')" type="submit" value="Nueva Ventana">
```

Para actualizar el target URL desde código de servidor, utilizar una variable pública y el método `DataBind`. El siguiente evento `Page_Load` establece el target URL y lo actualiza con el método `DataBind` cuando se carga la página:

```
public string urlTarget;
private void Page_Load(object sender, System.EventArgs e)
{
  urlTarget = "webform2.aspx";
  Page.DataBind();
}
```

Debido a que el método `Window.Open` toma diferentes argumentos para controlar los diferentes aspectos de una ventana del Web Browser, se querrá crear una clase que maneje todas las configuraciones.

Mantener el estado de los datos en un Web Form.

La conexión que se establece entre un usuario (computadora cliente) y un servidor Web es llamada *sesión.*

El *mantenimiento de estado* es el proceso por el cual se mantiene la misma información a través de múltiples solicitudes para la misma o diferentes páginas Web.

Debido a que los Web Forms tienen tiempo de vida muy corto (no mantienen estado), ASP.NET procura pasos especiales para conservar los datos introducidos en los controles del Web Form. Los datos introducidos en los controles son enviados con cada solicitud y restaurados a controles en el evento `Page_Init`. Los datos en estos controles están disponibles en el evento `Page_Load`.

Los datos que conserva ASP.NET entre solicitudes se guardan en una propiedad del Web Form llamada `viewState` o en la cadena de consulta (*Query string*). Estas opciones están disponibles únicamente durante el ciclo de solicitud/respuesta de un Web Form. Para hacer que los datos introducidos en el Web Form estén disponibles para otros Web Forms en una aplicación, se necesitan almacenar los datos en una variable de estado en los objetos `Application` o `Session`; alternativamente en cookies del Web Browser.

Niveles de estado.

Los Web Forms son destruidos y recreados cada vez que un cliente Web Browser hace una solicitud al servidor Web. Por esta característica, las variables declaradas dentro del Web Form no retienen sus valores después de que es desplegada la pagina. Para resolver este problema, ASP.NET proporciona dos tipos y sus diferentes opciones para mantener el estado y retener los datos entre solicitudes: del lado del servidor (*server-side*) y del lado del cliente (*client-side*).

Las opciones para mantener el estado *del lado del cliente* utilizan los recursos del Web Browser para almacenar información de estado:

- ***Query strings.*** Utilizarlos para pasar información entre solicitudes de páginas Web como parte de la dirección URL. Son visibles al usuario, así que no deberán contener información segura como passwords.

- ***Cookies.*** Utilizarlos para almacenar pequeñas cantidades de información en un cliente Web Browser. Los clientes pueden rechazar cookies, así que el código tiene que anticipar esta posibilidad.

- ***View state.*** ASP.NET almacena elementos agregados a la página en la propiedad `ViewState` como campos ocultos (*hidden*).

Las opciones para mantener el estado *del lado del servidor* utilizan los recursos del servidor para almacenar información de estado y tienen mayor seguridad:

- ***Session state.*** Utilizar las variables de estado `Session` para almacenar elementos que se quieran guardar para la sesión actual.

- ***Application state.*** Utilizar las variables de estado `Application` para almacenar elementos que se quieran tener disponibles para todos los usuarios de la aplicación. Se puede pensar en estado de aplicación como datos globales para múltiples usuarios. Todas las sesiones pueden leer o escribir estas variables.

- ***Servidor de estado (Microsoft SQL Server).*** Utilizar la base de datos SQL Server para almacenar grandes cantidades de información del usuario. Además puede usarse en conjunto con objetos de `Session` y cookies.

- ***El objeto*** `Cache`***.*** Utilizarlo para manejar el estado a nivel aplicación.

Utilizando Query Strings.

Nos permiten enviar información adicional entre direcciones. En HTML, estos aparecen después del signo de interrogación en una hiperliga, como se muestra a continuación:

```
<A HREF= "WebForm1.aspx?Variable=Valor">Ejemplo de Query string.</A>
```

Para enviar un query string en código, agregarlo a la dirección del método `Redirect`. El siguiente evento `Click` es equivalente al HTML anterior:

```
private void Button1_Click(object sender, System.EventArgs e)
{
  // Desplegar de nuevo está página con el QueryString
  Response.Redirect("Webform1.aspx?Variable=Valor");
}
```

Para extraer un query string en código, utilizar el método `Query String` del objeto `Response`. El siguiente código despliega el valor del elemento `Variable` del query string creado en los ejemplos anteriores:

```
private void Page_Load(object sender, System.EventArgs e)
{
  // Despliega el valor del query string.
  Response.Write(Request.QueryString["Variable"]);
}
```

Utilizando Cookies.

Una cookie es una cantidad pequeña de datos que es almacenada físicamente en un archivo de texto o en la memoria del Web Browser de la computadora del usuario. Cuando el Web Browser solicita una página, este envía la información de la cookie al servidor, el cual está autorizado para leer la cookie y extraer su valor. Todas las cookies

contienen la información del dominio que utilizó la cookie, por lo que se pueden tener varias cookies pertenecientes al mismo dominio.

Existen dos tipos de cookies:

- ***Temporales.*** también llamadas de sesión o no persistentes, existen únicamente en la memoria del Web Browser. Cuando se cierra el Web Browser, cualquier cookie temporal se pierde.

- ***Persistentes.*** Tienen un periodo de expiración definitivo. Cuando un Web Browser solicita una página que crea una cookie persistente, el Web Browser guarda la cookie al disco duro de la computadora del usuario.

Los sitios Web utilizan frecuentemente cookies para almacenar las preferencias de los usuarios u otra información que es específica del cliente. Debido a que las cookies pueden ser rechazadas, es importante verificar si el Web Browser las soporta antes de intentar crearlas.

Las cookies son poco seguras, debido a que pueden ser alteradas por el usuario en su computadora, causando que una aplicación falle si es dependiente de su información. Tampoco se garantiza el tiempo de expiración porque pueden ser eliminadas por el usuario en cualquier momento. También tiene restricciones en cuanto a tamaño de archivo, no más de 4 kilobytes (KB).

El siguiente código verifica si un Web Browser permite cookies, y si las permite, guarda las preferencias del usuario.

```
private void Page_Load(object sender, System.EventArgs e)
{
  // Corre la primera vez que la página es desplegada.
  if(!IsPostBack)
    // Si el browser soporta cookies.
    if(Request.Browser.Cookies)
    {
      // Crea una cookie.
      HttpCookie cookUPrefs = new HttpCookie("UPrefs");
      cookUPrefs.Value = "English";
      // Agrega la cookie.
      Response.Cookies.Add(cookUPrefs);
    }
}
```

El siguiente código verifica si existe una cookie, y la obtiene si esta se encuentra disponible:

```
private void Page_Load(object sender, System.EventArgs e)
{
  // Corre la primera vez que esta página es desplegada.
  if(!IsPostBack)
    // Si el browser soporta cookies.
    if(Request.Browser.Cookies)
      // Checa si la cookie UPrefs existe.
      if(Request.Cookies["UPrefs"] != null)
        // Guarda el valor de la cookie.
        Session["Lang"] = Request.Cookies["UPrefs"].Value;
}
```

Utilizando ViewState.

Útil para guardar datos en un campo oculto (*hidden*) en una página. Debido a que `ViewState` almacena datos en la página, esto está limitado a elementos que pueden ser serializados. Si se requiere almacenar elementos más complejos en `ViewState`, se deben convertir los elementos a tipo string.

Por ejemplo, el siguiente código agrega el texto de una caja de texto a celdas en una tabla en la página. Debido a que no se pueden almacenar objetos directamente en `ViewState`, se necesitan almacenar los strings en el método `Button1_Click`, y entonces crear los controles del renglón de tabla a partir de los strings:

```
private void Button1_ServerClick(object sender, System.EventArgs e)
{
  ViewState.Add(ViewState.Count.ToString(), TextBox1.Value);
}

private void Page_Load(object sender, System.EventArgs e)
{
  if (IsPostBack)
    // Para cada elemento en el ViewState
    foreach(StateItem staItem in ViewState.Values)
    {
      HtmlTableRow NuevoRenglon = new HtmlTableRow();
      HtmlTableCell NuevaCelda = new HtmlTableCell();
      // Establece el texto de la celda.
      NuevaCelda.InnerText = staItem.Value.ToString();
      // Agrega la celda al renglón.
      NuevoRenglon.Cells.Add(NuevaCelda);
      // Agrega el renglón a la tabla.
      Table1.Rows.Add(NuevoRenglon);
    }
}
```

ASP.NET codifica los datos ocultos que están almacenados en la página así que no es legible a los usuarios.

Utilizando estados de aplicación y sesión.

Utilizar estados de `Application` y `Session` para almacenar datos que se quieran guardar mientras dure la sesión o la aplicación, respectivamente. Se puede almacenar cualquier tipo de datos, incluyendo objetos. Sin embargo, se deben considerar los siguientes puntos antes de utilizarlos.

- Las variables de estado `Application` y `Session` son creadas al vuelo, sin nombre de variable o verificación de tipo. Se debe limitar los puntos de acceso a estas variables.

- Mantener el estado `Session` afecta el rendimiento. Se debe apagar a niveles de aplicación y página.

- Las variables de estado `Application` están disponibles a lo largo del proceso actual, pero no entre procesos. Si una aplicación es escalada para correr en múltiples servidores o en múltiples procesadores dentro de un servidor, cada proceso tiene su propio estado `Application`.

- Las fronteras de la aplicación establecidas por IIS determinan el alcance del estado `Application`.

ESTA TESIS NO SALE
DE LA BIBLIOTECA

En el siguiente código se guarda el número de clicks que se hacen a un botón dentro de un Web Form en una variable de estado `Session`:

```
// De Global.asax
protected void Session_Start(Object sender, EventArgs e)
{
  // Inicializa la variable de estado session llamada Clicks.
  Session["Clicks"] = 0;
}

// De Web Form cod3.1_VarEstSesion.aspx
private void Button1_Click(object sender, System.EventArgs e)
{
  // Incrementa el contador de click.
  Session["Clicks"] = (int)Session["Clicks"] + 1;
  // Despliega el número de clicks.
  Response.Write("Numero de clicks: " + Session["Clicks"] + "<br>");
}
```

Se puede almacenar cualquier tipo de dato en una variable de estado, inclusive objetos. Debido a que las variables de estado son datos globales, se necesitan estrategias de desarrollo para trabajar con ellas en las aplicaciones.

Las variables de estado deben ser inicializadas en C# antes de que se realicen operaciones con ellas. Por ejemplo, en el Código anterior se necesita asignar un valor a la variable de estado `Clicks` antes de realizar el cast[27] `(int)Session["Clicks"] + 1`. De otra forma, se recibirá un error en tiempo de ejecución.

Estructurando el acceso a variables de estado.

Las variables de estado `Application` y `Session` son muy útiles y asimismo propensas a introducir errores en el código si no son utilizadas de forma estructurada. La mejor manera para hacer esto es declarar una variable a nivel página para cada elemento que se necesite y extraer el valor de estado `Application` o `Session` en el `Page_Load` y guardar las variables nivel página a su estado anterior en el evento `Page_Unload`.

El siguiente código demuestra el acceso estructurado a variables de estado:

```
string mstrUname = "";

private void Page_Load(object sender, System.EventArgs e)
{
    // Checa si existe la variable de estado.
    if(Application["Uname"] != null)
        // Obtiene la variable de estado.
        mstrUname = Application["Uname"].ToString();
    // Establece la variable
    mstrUname = "NombreDeUsuario";
    // Utiliza la variable.
    Response.Write(mstrUname);
}

private void Page_UnLoad(object sender, System.EventArgs e)
{
    // Guarda la variable de estado de vuelta.
    Application["Uname"] = mstrUname;
}
```

[27] Cast se refiere a la conversión de un tipo de dato u objeto a otro, siempre y cuando sean compatibles.

Apagar el estado de `Session`.

ASP.NET mantiene el estado `Session` para cada página en la aplicación por defecto. Si la página no requiere información de estado a nivel sesión, se puede apagar para lograr mejor rendimiento.

Para apagar el estado `Session` de un Web Form:

- De la ventana `Properties` en el Web Form, establecer `EnableSessionState` a `False`.

Para apagar el estado `Session` en la aplicación completa:

- En el archivo `Web.config`, establecer el tag `<sessionstate mode="false">`.

Identificando y rastreando una sesión.

Cada sesión activa de una aplicación Web es identificada y rastreada utilizando un string llamado `SessionID` (identificador de sesión) que contiene únicamente caracteres ASCII que son permitidos en las URLs. Los `SessionID` son transferidos entre solicitudes cliente-servidor, mediante una cookie o un query string (en el caso que no soporte cookies el cliente y se indique en la configuración de la aplicación).

Controles personalizados.

Son en gran medida una forma de reutilizar código correspondiente a la interfaz de usuario. Por ejemplo, no se necesitará volver a escribir las etiquetas del control y configurar de nuevo la página de inicialización de código en varios lugares.

ASP.NET proporciona dos herramientas para reutilizar partes de código:

- ***Controles de usuario.*** permiten utilizar de nuevo parte de una página, colocándolas en un archivo `.ascx` especial. ASP.NET también permite que estos controles de usuario proporcionen métodos y propiedades y que configuren sus controles contenidos automáticamente.

- ***Controles personalizados o derivados.*** Permiten crear un nuevo control heredado de una clase de control ASP.NET.

Notar que los controles personalizados son una extensión de los controles de servidor ya revisados anteriormente en este capítulo, y se mencionan únicamente con carácter informativo fuera del alcance del presente trabajo.

Introducción a ADO.NET.

ADO.NET es la tecnología dentro del Framework .NET que las aplicaciones ASP.NET utilizan para comunicarse con almacenes de datos. En este capítulo se analizará la tecnología ADO.NET para acceder y modificar datos desde aplicaciones ASP.NET, entendiendo su arquitectura y componentes que la forman. Esta tecnología es parte esencial de las aplicaciones ASP.NET, usada para desplegar datos en Web Forms y para realizar las actualizaciones pertinentes a los datos con las modificaciones hechas por el usuario.

Prácticamente cualquier pieza de software trabaja con datos. De hecho, una aplicación Internet típica no es más que una interfaz de usuario puesta encima de un programa que utiliza la base de datos para leer y escribir información publicada a través del servidor Web.

Almacenamiento de datos.

El almacenamiento de datos es un método para guardar elementos específicos que juntos constituyen una unidad de información. Los elementos de datos individuales por si mismos son de poco valor; cuando se ponen en contexto con otros elementos de datos se vuelven recursos valiosos. La tabla 4.1 describe diferentes métodos de almacenamiento de datos soportados por ADO.NET.

Tipo	Características	Ejemplos
Sin estructura	Datos que no tienen orden lógico.	Texto plano sin formato.
Estructurado, no jerárquico	Los datos son separados en unidades, pero las unidades están organizadas estrictamente por su orden.	Archivos de valores separados por coma (*Comma separated Value*, CSV), archivos separados por tabulador, Hojas de Excel.
Jerárquico	Los datos son organizados en tres estructuras, con nodos que contienen otros nodos.	Documentos de datos XML.
Base de datos relacional	Los datos están organizados en tablas, con columnas conteniendo un tipo especifico de dato y renglones conteniendo un registro. Las tablas pueden ser relacionadas con columnas con tipos de datos similares.	Microsoft SQL Server, Oracle.
Base de datos orientada a objetos	Los datos son organizados como objetos	Objetivity/DB

Tabla 4.1 *Métodos de almacenamiento de datos.*

Ambiente conectado.

Por mucho tiempo en la historia de las computadoras, este fue el único ambiente con el cual se trabajaba.

Un *ambiente conectado* es uno en el cual un usuario o una aplicación es constantemente conectada a un origen de datos.

Las ventajas que presenta este escenario son:

- Es fácil de mantener un escenario seguro.
- La concurrencia es fácil de controlar.
- Los datos suelen estar más actualizados que en otros escenarios.

Las desventajas que presenta este escenario son:

- Debe tener una conexión constante en la red.
- Escalabilidad.

Ambiente desconectado.

Con el advenimiento de Internet y el incremento de uso de dispositivos móviles, los escenarios de trabajo desconectados han llegado a ser comunes. Las computadoras portátiles como laptop, notebook y handheld permiten utilizar aplicaciones cuando se encuentran desconectadas del servidor o de la base de datos.

En muchas situaciones, la gente no trabaja enteramente en un ambiente conectado o desconectado, sino en una combinación de los dos.

Un *ambiente desconectado* es uno en el cual un usuario o una aplicación no se encuentran conectados constantemente a un origen de datos. Los usuarios que utilizan dispositivos móviles generalmente trabajan en un ambiente desconectado, donde toman un subconjunto de datos para trabajarlos desconectados del origen, y después reflejar los cambios de vuelta en el almacén de datos.

Las ventajas que presenta este escenario son:

- Se puede trabajar desconectado el tiempo que sea necesario y conectarse al origen de datos en cualquier momento para procesar una solicitud.
- Otros usuarios pueden utilizar la conexión.
- Un ambiente desconectado mejora la escalabilidad y rendimiento de las aplicaciones.

Las desventajas que presenta este escenario son:

- Los datos no están siempre actualizados.
- Los conflictos de cambios pueden ocurrir y deben ser resueltos.

Modelos de aplicación de acceso a datos.

Los modelos de acceso a datos han evolucionado junto con las computadoras. Como el numero de usuarios y la cantidad de datos se ha incrementado, los modelos de acceso a datos han evolucionado desde un solo usuario en una sola aplicación a múltiples usuarios en Internet.

Dentro de un modelo de acceso a datos, una capa es un nivel lógico en el cual residen los componentes lógicos de una aplicación. Las capas pueden estar en una o mas

computadoras o capas físicas. El número de capas se refiere al número de niveles, no al numero de computadoras físicas, en el cual están divididos los servicios. Estos niveles típicamente incluyen lo siguiente:

- ***Capa cliente***, también conocida como la capa de presentación o capa de servicios de usuario. Esta capa contiene la interfaz de usuario.

- ***Capa lógica de negocio***, la cual contiene la lógica que interactúa con el origen de datos. Esta capa "intermedia" contiene la parte de la aplicación que interactúa con los datos; por ejemplo, crear una conexión al origen de datos. La capa de lógica de negocio es a menudo implementada en todas la capas; por ejemplo, como procedimientos almacenados en el almacén de datos, como clases dentro de una aplicación de servidor, o como código dentro de la aplicación cliente.

- ***Capa de servicios de datos***, la cual contiene los datos que la lógica de negocio utiliza en las aplicaciones.

- ***Capa de interoperabilidad***, la cual contiene la lógica que permite la interacción entre aplicaciones en diferentes sistemas operativos, o diferentes tipos de datos. Por ejemplo, los servicios Web XML que pueden trabajar en cualquier sistema operativo.

La mayor ventaja de agregar capas es la habilidad de escalar aplicaciones. Cada capa adicional permite agregar mas usuarios y aislar un nivel de lógica de aplicación sin requerir cambios a otras capas.

Por ejemplo, en una aplicación de una capa, un cambio a cualquier nivel de lógica requiere que la aplicación completa sea recompilada y redistribuida.

La Tabla 4.2 compara diferentes tipos de modelo de acceso a datos.

Modelo	Descripción	Ventajas	Desventajas
1-capa, o monolítica	Este modelo típicamente involucra un solo usuario y todas las capas en una sola computadora. Por ejemplo una aplicación estilo Clipper[28].	Debido a que todo está en un solo lugar, todos los componentes son fácilmente accesibles.	La actualización del programa requiere que el código sea modificado, recompilado, y redistribuido para todos los usuarios.
2-capas, cliente/servidor	La capa de usuario y la capa de lógica de negocio reside en una capa, los servicios de datos en otra. Este modelo típicamente involucra dos o más computadoras. Por ejemplo, una aplicación donde la lógica de negocio está en dos capas: parte de la lógica en la aplicación cliente, y otra parte en procedimientos almacenados en la capa de datos.	Proporciona alguna separación de funciones.	Difícil de escalar porque el cliente es un "cliente pesado" que contiene las capas de presentación y lógica de negocio. Problemas de distribución y mantenimiento.
3-capas	Cada servicio está en una capa separada. La lógica de negocio se mueve a una nueva "capa intermedia".	Buena separación de funciones. La capa cliente es un "cliente ligero" que contiene únicamente la lógica del cliente, o la capa de presentación.	Más compleja para manejar seguridad, aún no es tan flexible como un modelo n-capas.
n-capas	Una base de datos nivel empresarial donde varios clientes acceden una sola aplicación de servidor. Nuevas capas pueden ser agregadas como ocurra una nueva necesidad de lógica.	Permiten diferentes aplicaciones en diferentes sistemas operativos para interactuar con usuarios y datos.	Problemas de seguridad. Las llamadas a procedimientos remotos (RPC[29]) no pasan a través de firewalls[30].
n-capas con interfaz Web	Los servicios son distribuidos entre Internet e Intranet, con capa adicional y servidores adicionales dedicados a la red.	Cero costos de distribución al cliente. Las únicas actualizaciones son a los servidores de aplicación Web. HTTP puede pasar a través de firewalls.	Problemas de seguridad.

Tabla 4.2 *Tipos de modelo de acceso a datos.*

[28] Clipper es un compilador de programas de gestión de base de datos surgido a mediados de los 80's.
[29] RPC consiste en procesos que hacen llamadas a procedimientos con interfaz conocida que residen en una máquina distinta a la que están ejecutándose para transferencia de datos.
[30] Un Firewall es un sistema o grupo de sistemas que controlan el acceso de datos entre dos o más redes.

Consideraciones sobre el desarrollo de aplicaciones Web.

El acceso a una base de datos en una aplicación Web es un escenario completamente diferente al de acceder a una base de datos desde un programa cliente/servidor. Las aplicaciones Web conllevan un nuevo conjunto de consideraciones y problemas potenciales.

Problemas de escala.

Una aplicación Web puede utilizarse potencialmente por cientos de usuarios simultáneos. Esto significa que el uso de la memoria del servidor o los recursos limitados como las conexiones a base de datos no deben tomarse a la ligera. Si se diseña una aplicación ASP que adquiere una conexión a la base de datos y la mantiene unos pocos minutos, el resto de los usuarios puede observar un retraso significativo o incluso se les puede bloquear por completo el acceso a la base de datos. Y si no se han revisado los problemas de concurrencia cuidadosamente, pueden dificultar el acceso de los usuarios a datos consistentes.

Todos estos problemas también están en el desarrollo de base de datos tradicional cliente/servidor. La diferencia está en que es menos probable que tengan un efecto negativo porque la carga típica (esencialmente el número de usuarios simultáneos) es bastante menor. En una aplicación Web, las prácticas de base de datos que podrían influir negativamente al rendimiento en una aplicación cliente/servidor se multiplican rápidamente y provocan problemas significativos.

Problemas de estado.

HTTP es un protocolo sin estado. Esto significa que no se mantienen las conexiones entre un Web browser y el servidor Web. Cuando un usuario solicita una página en una aplicación ASP.NET, se abre una conexión, se procesa el código, se devuelve una página HTML e inmediatamente se libera la conexión. Mientras que los usuarios pueden tener la sensación de que están interactuando con una aplicación continuamente en ejecución, realmente reciben una página estática.

Problemas de ambiente desconectado.

Las aplicaciones Web son desconectadas. El acceso desconectado facilita a los usuarios realizar modificaciones que pueden crear datos inconsistentes, un problema que no se detecta hasta que las modificaciones se aplican al origen de datos. El acceso de datos desconectado también requiere especial consideración porque los cambios no se aplican en el orden que se introdujeron. Este diseño puede provocar problemas cuando se añaden datos relacionales.

Arquitectura de ADO.NET.

ADO.NET es el siguiente paso en la evolución de los Microsoft ActiveX Data Objects (ADO). Esto no comparte el mismo modelo de programación, pero comparte mucho de la funcionalidad ADO.

ADO.NET es un conjunto de clases para trabajar con datos.

Un número creciente de aplicaciones utilizan código XML para codificar datos que son pasados sobre conexiones de red. ADO.NET proporciona un modelo de programación que incorpora características de XML y ADO.NET dentro del Framework .NET.

ADO.NET proporciona las siguientes ventajas sobre otros componentes y modelos de acceso a datos.

- ***Interoperabilidad.*** ADO.NET usa XML como el formato para transmitir datos desde un origen de datos a una copia local de datos en memoria.

- ***Mantenimiento.*** Cuando un número creciente de usuarios trabaja con una aplicación, puede acabar con los recursos del servidor. Al utilizar aplicaciones n-capas, podemos cubrir parte de la lógica de aplicaciones en capas adicionales. La arquitectura de ADO.NET utiliza cachés locales de memoria para mantener copias de datos, proporcionando a capas adicionales el acceso rápido a la información.

- ***Programación.*** El modelo de programación ADO.NET usa datos fuertemente tipados. Esto hace al código mas conciso y fácil de escribir.

- ***Rendimiento.*** ADO.NET ayuda a evitar costosas conversiones de tipos de datos al utilizar datos fuertemente tipados.

- ***Escalabilidad.*** El modelo de programación ADO.NET alienta a los programadores a conservar los recursos del sistema para una aplicación que se ejecuta sobre Web. Debido a que los datos son retenidos en cachés locales de memoria, no hay necesidad de mantener candados en la base de datos o tener conexiones activas por periodos extendidos.

El acceso en ADO.NET es estandarizado para ser independiente del origen de datos -una vez que se haya establecido conexión a la base de datos-; se utilizará un conjunto consistente de objetos, propiedades, y métodos, sin importar el tipo de base de datos que se esté utilizando.

Hay tres capas para acceder datos en ADO.NET:

- ***El almacén de datos físico.*** Este puede ser una base de datos OLE, una base de datos SQL, o un archivo XML.

- ***El proveedor de datos.*** Este consiste de un objeto `Connection` y objetos para ejecutar comandos que crean la representación en memoria de los datos.

- ***El*** `DataSet`***.*** Esta es una representación en memoria de las tablas y relaciones que trabajamos con la aplicación.

La capa de proveedor de datos proporciona abstracción entre el almacén de datos físico y el `DataSet` que trabajamos en código. Después de crear el `DataSet`, no importa de donde viene o donde está almacenado. Esta arquitectura se refiere como "desconectada" porque el `DataSet` es independiente del almacén de datos.

Namespaces relacionados a datos.

ADO.NET proporciona sus objetos, propiedades, y métodos a través de los namespaces descritos en la Tabla 4.3, los cuales residen físicamente en el ensamblado `System.Data.dll`. Estos namespaces proporcionan características equivalentes para bases de datos SQL y OLE, respectivamente.

Namespace	Proporciona
`System.Data`	Clases, tipos, y servicios para crear `DataSets` y sus objetos subordinados.
`System.Data.Common`	Clases de utilidad e interfaces que son heredados e implementados por proveedores de datos .NET; como `System.Data.SqlClient` y `System.Data.OleDb` quienes proporcionan versiones específicas para los proveedores de datos SQL Server y OLE DB.
`System.Data.OleDb`	Clases y tipos para acceder bases de datos OLE DB. Estas clases admiten todos los proveedores OLE DB, excepto el controlador OLE DB de orígenes de datos ODBC.
`System.Data.SqlClient`	Clases para acceder bases de datos SQL Server.
`System.Data.SqlTypes`	Clases y estructuras para tipos de datos nativos de SQL Server. Estas clases proporcionan las mismas propiedades que sus homónimos en el espacio de nombres `System.Data.OleDb`.
`System.Xml`	Clases, interfaces, y enumeraciones que proporcionan estándares con soporte para procesar XML.

Tabla 4.3 *Namespaces de ADO.NET.*

El modelo de objetos de ADO.NET.

El modelo de objetos de ADO.NET consiste de dos partes:

- Clases de datos.
- Clases de proveedores de datos .NET.

Las clases de datos (`DataSet`) permiten almacenar y administrar datos en un caché desconectado. El `DataSet` es independiente de cualquier origen de datos, así que sus características están disponibles para todas las aplicaciones, sin importar de donde es el origen de los datos.

Las clases de proveedores de datos son específicos de un origen de datos. Por consiguiente, los proveedores de datos .NET deben ser escritos específicamente para un origen de datos, y trabajarán únicamente con ese origen de datos. Las clases de un proveedor de datos .NET tiene la habilidad de conectarse a un origen de datos, extraer datos, y realizar actualizaciones. El modelo de objetos ADO.NET incluye las siguientes clases de proveedores de datos:

- Proveedor de datos SQL Server .NET.
- Proveedor de datos OLE DB .NET.
- Otros proveedores de datos .NET.

La Figura 4.1 muestra el modelo de objetos ADO.NET:

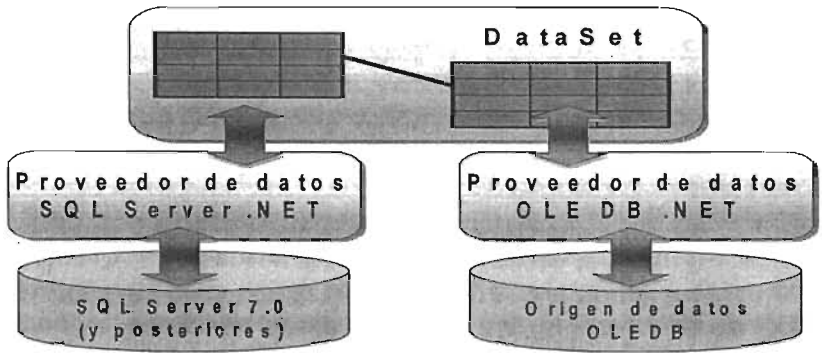

Figura 4.1 *Modelo de objetos ADO.NET.*

Los proveedores de datos SQL Server .NET y OLE DB .NET funcionan de forma similar; la única diferencia es que las clases diseñadas para SQL Server están optimizadas; proporcionando un mejor rendimiento al eliminar las capas adicionales de OLEDB/COM y conectarse a la interfaz de Flujo de datos tabular (*Tabular Data Stream*, TDS) directamente. Tanto los tipos OLE DB y SQL Server derivan de las mismas clases básicas de `System.Data.Common`. También proporcionan la misma interfaz (métodos y propiedades). sólo los detalles internos son diferentes. Por ejemplo tanto `SqlCommand` como `OleDbCommand` permiten ejecutar una sentencia SQL o un procedimiento almacenado contra un origen de datos. El proveedor de datos OLE DB .NET proporcionan acceso a SQL Server 6.5 y versiones previas; además de otras bases de datos como Oracle, Sybase y DB2, entre otras.

Otros proveedores de datos están disponibles para otros orígenes de datos. Por ejemplo el proveedor de datos ODBC (*Open Database Conectivity*) .NET.

Las clases de datos (`DataSet`).

Las clases de datos `DataSet` permiten almacenar una copia local, desconectada. No almacenan una conexión a un origen de datos. No importa el tipo de origen de datos que se utilice, objetos como `DataSet` y `DataRelation` son genéricos. El modelo de objetos se muestra en la Figura 4.2.

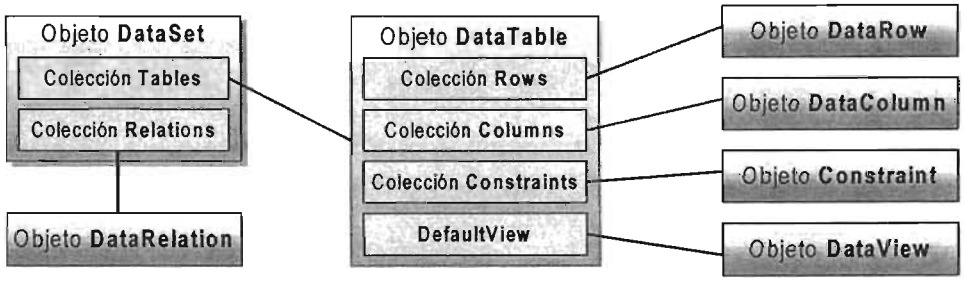

Figura 4.2 *Los objetos del modelo de datos.*

DataSet.

Es el núcleo de ADO.NET. La clase `DataSet` almacena información desconectada recogida de una base de datos y permite manipularla como un único objeto empaquetado. La clase

`DataSet` no tiene conexión directa con un origen de datos. De hecho, se puede crear un `DataSet` mediante código sin ninguna relación con una base de datos.

Los objetos `DataSet` contienen dos tipos principales de objetos: `DataTable` (proporcionados en la propiedad `Tables`) y `DataRelation` (proporcionados en la propiedad `Relations`).

Clases de proveedores de datos .NET.

Por sí mismos, los objetos de datos son de poca utilidad. Se pueden añadir tablas, renglones y datos manualmente, pero en la mayoría de los casos la información necesaria está en un origen de datos como una base de datos relacional. Para acceder a esta información, extraerla e insertarla en los objetos de datos apropiados, se necesitan un conjunto de objetos que permitan la interacción con el origen de datos.

El objetivo de los objetos de origen de datos es crear una conexión e introducir la información en un `DataSet` o un `DataReader`. La interacción completa de estos objetos depende del escenario que se utilice en la aplicación: conectado ó desconectado.

Usando clases ADO.NET en un escenario conectado.

Los proveedores de datos .NET proporcionan las clases para trabajar en un ambiente conectado y han sido diseñados para ser ligeros; esto quiere decir que crean una capa mínima entre el código y el origen de datos, para incrementar el rendimiento sin sacrificar funcionalidad.

Cuando se trabajen bases de datos en código, será necesario agregar algunas de las siguientes declaraciones al principio del código:

```
using System.Data
// Para conexiones a base de datos SQL, agregar:
using System.Data.SqlClient
// Para conexiones a base de datos OLE DB, agregar:
using System.Data.OleDb
```

ADO.NET expone un modelo común de objetos para proveedores de datos .NET. En el proveedor de datos SQL Server, los nombres de las clases son nombradas comenzando con el prefijo `Sql`. Por ejemplo, la clase de conexión es nombrada `SqlConnection`.

En el proveedor de datos OLE DB .NET, los nombres de las clases son nombradas comenzando con el prefijo `OleDb`. Por ejemplo, la clase conexión es nombrada `OleDbConnection`.

Otros proveedores de datos .NET pueden ser escritos con otros prefijos, por lo que se identificarán sus clases con el prefijo Xxx.

Conectándose a un origen de datos.

Cuando se conecta a un origen de datos, primero se debe escoger un proveedor de datos .NET. El proveedor de datos incluye las clases que habilitan para conectarse al origen de datos, leer datos, modificar y manipular datos, y actualizar el origen de datos.

Cuando se construye una aplicación que accede datos utilizando ADO.NET, normalmente se tiene que conectar a bases de datos seguras. Para esto, información de seguridad como usuario y password debe ser pasada a la base de datos antes de que sea hecha una conexión. La seguridad depende de cada base de datos.

La cadena de conexión.

Para definir una conexión se necesita la cadena de conexión. La propiedad `ConnectionString` de un objeto `Connection` proporciona la información que define una conexión al almacén de datos usando una cadena de parámetros.

La Tabla 4.4 describe varios parámetros comunes de cadenas de conexión. La tabla contiene únicamente una lista parcial de valores: no se necesitan todos estos para establecer una conexión.

Parámetro	Descripción
`Provider`	La propiedad usada para establecer o regresar el nombre del proveedor para la conexión, usada únicamente para objetos `OleDbConnection`.
`Connection Timeout` o `Connect Timeout`	La longitud de tiempo en segundos para esperar por una conexión al servidor antes de terminar el intento y generar una excepción.
`Inicial Catalog`	El nombre de la base de datos.
`Data Source`	El nombre de la instancia o servicio.
`Password`	La contraseña para conectarse al servidor.
`User ID`	La cuenta login ó identificador de usuario para conectarse al servidor.
`Persist security Info`	Cuando se establece a `false`, información sensitiva de seguridad, como el password, no se regresa como parte de la conexión si ésta se encuentra abierta. Estableciendo está propiedad a `true` puede ser un riesgo de seguridad. `false` es el valor por defecto.

Tabla 4.4 *Parámetros comunes de cadenas de conexión.*

Notar que únicamente se puede establecer la propiedad `ConnectionString` cuando la conexión está cerrada. Para restablecer una cadena de conexión se deberá cerrar y reabrir la conexión.

El siguiente código en C# es un ejemplo de cómo establecer la cadena de conexión para una base de datos SQL Server 2000 usando un objeto `SqlConnection`:

Atributo	Valor
Producto	Microsoft SQL Server 2000
Nombre de instancia	GAIA\NETSDK
Nombre de base de datos	Northwind
Username	sa
Password	pwd
Timeout	1 minuto

```
System.Data.SqlClient.SqlConnection cnNorthwind = new System.Data.SqlClient.SqlConnection();
cnNorthwind.ConnectionString = "User Id=sa; Password=pwd;Data Source=GAIA\\NETSDK;Initial
catalog=Northwind;Connection Timeout=60;";
```

Abriendo y cerrando una conexión.

Se pueden abrir y cerrar conexiones de manera implícita llamando los métodos en un objeto que use la conexión, o explícitamente, llamando los métodos `Open` y `Close`. del objeto `XxxConnection`,

El método `Open` usa la información de la propiedad `ConnectionString` para contactar el origen de datos y establecer una conexión. El método `Close` cierra la conexión.

Cerrar las conexiones es esencial, porque muchos de los orígenes de datos soportan únicamente un número limitado de conexiones abiertas las cuales ocupan recursos del sistema.

Se deben cerrar siempre las conexiones cuando se hayan terminado de usar. Para hacer esto, se pueden utilizar los métodos `Close` o `Dispose` del objeto conexión. Las conexiones no son cerradas implícitamente cuando el objeto conexión cae fuera de alcance o es reclamado por el recolector de basura.

El método `Close` hace rollback de cualquier transacción pendiente y cierra la conexión, o libera la conexión del pool (concentrador) de conexiones si está habilitado. Una aplicación puede llamar el método `Close` mas de una vez.

Si se trabaja con el objeto `DataAdapter`, no se tiene que abrir o cerrar explícitamente una conexión. Cuando se llama un método de este objeto (por ejemplo, los métodos `Fill` o `Update`), el método verifica si la conexión está abierta. Si no, el `DataAdapter` abre la conexión, realiza su lógica, y cierra la conexión.

Si se está llenando múltiples tablas en un `DataSet` de la misma base de datos, se pueden tener múltiples objetos `DataAdapter`, uno por cada tabla, pero sólo una conexión. En este caso se abre la conexión, se llaman los métodos `Fill` de los múltiples objetos `DataAdapter`, y explícitamente se cierra la conexión.

Cuando se cierra una conexión, el flujo a y desde el origen de datos se cierra, pero los recursos no manejados usados por el objeto conexión no se han liberado. Si el pool de conexiones está habilitado, la conexión es liberada del pool. Los objetos `XxxConnection` tienen un método `Dispose` para liberar los recursos no manejados. Llamando el método `Dispose`, remueve la conexión del pool de conexiones.

El siguiente código de ejemplo muestra como crear un objeto `SqlConnection`, abriendo la conexión con el método `Open`, después cerrando y liberando los recursos usados por la conexión llamando el método `Dispose` y finalmente estableciendo el objeto a `null`.

```
// Declarar e instanciar un nuevo objeto SqlConnection.
System.Data.SqlClient.SqlConnection cnNorthwind = new System.Data.SqlClient.SqlConnection();

// Establecer la propiedad ConnectionString.
cnNorthwind.ConnectionString =
  "User Id=sa;" +
  "Password=pwd;" +
  "Data Source=GAIA\\NETSDK;" +
  "Initial catalog=Northwind;" +
  "Connection Timeout=60;";

// Abre la conexión.
cnNorthwind.Open();

// Cierra la conexión, lo cual libera la conexión del pool de conexiones.
cnNorthwind.Close();
```

```
// Deshacerse de la conexión, lo cual remueve la conexión del pool de
// conexionesm liberando recursos del sistema.
cnNorthwind.Dispose();

// Libera la memoria tomada por el objeto SqlConnection (la memoria no
// será reclamada hasta que el recolector de basura se ejecute.
cnNorthwind = null;
```

Pool de conexiones.

Cada vez que se establece una conexión al origen de datos, la memoria y ciclos del procesador son usados considerablemente. Debido a que las aplicaciones requieren a menudo múltiples conexiones con múltiples usuarios, conectarse al origen de datos puede costar rendimiento a la aplicación y escasear los recursos del servidor. Al concentrar las conexiones, se pueden mantener disponibles para reutilizar, lo cual aumenta la escalabilidad y rendimiento de la aplicación.

La concentración de conexiones es el proceso de guardar conexiones activas de manera que sean reutilizadas eficientemente. Las conexiones con cadena de conexión idéntica son concentradas y reutilizadas sin necesidad de restablecer la conexión.

Un pool de conexiones es creado cuando una o más conexiones comparten una única cadena de conexión, y su funcionalidad es solicitada. Si cualquiera de los parámetros de una nueva cadena de conexión no es idéntica a una anterior, la nueva conexión será puesta en su propio pool.

Construyendo objetos de tipo `Command`.

En muchas situaciones, se puede diseñar la estrategia de acceso a datos usando comandos para comunicarse directamente con el origen de datos (típicamente una base de datos). Estas situaciones incluyen:

- Realizar consultas sobre datos cuando la aplicación solo accede a los datos de solo lectura.
- Diseñar el acceso a datos en una aplicación Web que sólo requiera entregar datos, como desplegar los resultados de una búsqueda.
- Ejecutar una consulta que regrese un valor simple, así como un cálculo o el resultado de una función agregada.
- Creando y modificando estructuras de bases de datos, así como tablas y procedimientos almacenados.

Los objetos `Command` permiten acceder datos directamente en la base de datos en un ambiente conectado.

Se puede usar un objeto `Command` para realizar las siguientes tareas:

- Ejecutar enunciados SELECT que regresen un valor simple, así como un contador de renglones ó un valor calculado.

- Ejecutar enunciados SELECT que regresen renglones. Esta es una manera eficiente para cargar grandes volúmenes de datos de sólo lectura en un control Web.

- Ejecutar enunciados DDL para crear, editar, y remover tablas, procedimientos almacenados, y otras estructuras de bases de datos.

- Ejecutar enunciados DCL para otorgar o quitar permisos en la base de datos.

- Ejecutar enunciados para obtener información de catálogos de la base de datos.

Las propiedades de un objeto `Command` contienen toda la información necesaria para ejecutar un enunciado contra la base de datos. Esta información incluye:

- ***(Nombre).*** El nombre del objeto `Command`. Usar este nombre en el código, para referirse al objeto `Command`.

- ***Connection.*** El objeto `Command` contiene una referencia al objeto `Connection`, el cual se usa para comunicarse con la base de datos.

- ***CommandType.*** Puede ser de tipo: `Text` (Texto SQL), `StoredProcedure` (nombre del procedimiento almacenado), `TableDirect` (nombres de las tablas que serán accedidas, especificar en la propiedad `CommandText`. Sólo aplica a proveedores de datos OLE DB .NET).

- ***CommandText.*** El objeto `Command` incluye el texto de un enunciado SQL o el nombre de un procedimiento almacenado a ejecutar.

- ***Parameters.*** El objeto `Command` puede incluir cero o más parámetros. Se especifican parámetros de entrada, parámetros de salida, y valores de retorno. Configurando estos parámetros en tiempo de desarrollo se asegura que los comandos se ejecutarán eficientemente en tiempo de ejecución. No habrá necesidad de un viaje extra de regreso al servidor, para determinar los tipos de datos de los parámetros.

Creando parámetros para un objeto `Command`.

Para definir un parámetro programáticamente:

1. Crear un nuevo objeto `XxxParameter`.
2. Establecer la propiedades del objeto `Parameter`. La Tabla 4.5 describe las propiedades más comúnmente utilizadas.
3. Llamar el método `Add` en la colección `Parameters` del objeto `Command`. Si el comando llama un procedimiento almacenado que regrese un valor de resultado, se debe agregar el parámetro `ParameterDirection.ReturnValue` antes de cualquier otro parámetro. El orden en que se agreguen los parámetros es indistinto.

Propiedad	Descripción
`ParameterName`	El nombre del parámetro.
`DbType`	El tipo de dato del parámetro.
`Size`	El máximo tamaño en bytes del dato en el parámetro.
`Direction`	Un valor especificado por la enumeración `ParameterDirection`. Usar uno de los siguientes valores: • `ParameterDirection.Input` (valor por defecto) • `ParameterDirection.InputOutput` • `ParameterDirection.Output` • `ParameterDirection.ReturnValue`

Tabla 4.5 *Propiedades comunes del objeto* `Parameter`.

Ejecutando un comando que regresa un valor escalar.

Ocasionalmente, se querrá ejecutar un comando de base de datos o funciones que regresen un valor simple (valor escalar).

El método `ExecuteScalar` del objeto `Command` permite ejecutar un enunciado SQL o procedimiento almacenado que regrese un resultado escalar.

Para usar el método `ExecuteScalar`:

1. Crear un objeto `Command` y establecer las propiedades y parámetros si es necesario.
2. Escribir código para abrir la conexión a la base de datos.
3. Escribir código para llamar el método `ExecuteScalar` del objeto `Command`. Asignar el valor regresado a la variable en el tipo de dato apropiado.
4. Escribir código para cerrar la conexión a la base de datos.

Como extraer valores de salida y de retorno.

Los enunciados SQL y procedimientos almacenados frecuentemente pasan valores de retorno a la aplicación que los llama. Se puede hacer esto asignando un valor a un parámetro de salida. Adicionalmente, los procedimientos almacenados pueden también especificar un valor de retorno distinto.

Para obtener parámetros de salida desde un comando:

1. Configurar la colección `Parameters` del objeto `Command`. Para cada parámetro de salida, definir un objeto `Parameter` con la propiedad de `Direction` establecida a `ParameterDirection.Output`. Si un parámetro es usado para recibir y enviar valores, establecer la propiedad `Direction` a `ParameterDirection.InputOutput`.
2. Asegurarse que los tipos de datos de cada parámetro coincidan con el tipo de dato esperado en el procedimiento almacenado.
3. Después de ejecutar el procedimiento, leer la propiedad `Value` del parámetro que ha sido pasado de retorno.

Para obtener un valor de retorno de un procedimiento almacenado:

1. Configurar la colección `Parameters` para el procedimiento almacenado. El primer parámetro en la colección debe tener una propiedad `Direction` establecida como `ParameterDirection.ReturnValue`.

2. Asegurarse que el tipo de dato de este parámetro coincide con el tipo de datos que se regresa desde el procedimiento almacenado. Notar que enunciados `INSERT`, `UPDATE`, y `DELETE` regresan un valor entero, el cual indica el número de registros afectados por el enunciado.
3. Después de ejecutar el procedimiento, leer la propiedad `Value` del parámetro que se ha regresado.

Considerar el siguiente procedimiento almacenado para el ejemplo:

```
CREATE PROCEDURE dbo.CuentaProductosEnCategoria
  ( @CatID int,
    @CatNom nvarchar(15) OUTPUT
  )
/* Procedimiento almacenado con un parámetro de entrada llamado @CatID,
   y un parámetro de salida llamado @CatNom, y un valor de retorno.
*/
AS
  SET NOCOUNT ON
  DECLARE @ContProd int

  SELECT @CatNom = Categories.CategoryName,
         @ContProd = COUNT(Products.ProductID)
    FROM Categories INNER JOIN Products
      ON Categories.CategoryID = Products.CategoryID
   WHERE (Categories.CategoryID = @CatID)
GROUP BY Categories.CategoryName

  RETURN @ContProd
```

El siguiente código en C#, asume que el objeto `Connection` ha sido previamente configurado:

```
// Parámetro @CatID, crea objeto XxxParameter y establece sus propiedades.
System.Data.SqlClient.SqlParameter ParCatID = new System.Data.SqlClient.SqlParameter();
ParCatID.ParameterName = "@CatID";
ParCatID.SqlDbType = SqlDbType.Int;
ParCatID.Size = 4;
ParCatID.Direction = ParameterDirection.Input;
ParCatID.Value = 1; // Identificador correspondiente a la categoría Beverages

// Para obtener parámetros de salida desde un comando.
// Parámetro @CatNom, crea objeto XxxParameter y establece sus propiedades.
System.Data.SqlClient.SqlParameter ParCatNom = new System.Data.SqlClient.SqlParameter();
ParCatNom.ParameterName = "@CatNom";
ParCatNom.SqlDbType = SqlDbType.NVarChar;
ParCatNom.Size = 15;
ParCatNom.Direction = ParameterDirection.InputOutput;
ParCatNom.Value = "";

// Para obtener un valor de retorno de un procedimiento almacenado.
// Parámetro @@ContProd, crea objeto XxxParameter y establece sus propiedades.
System.Data.SqlClient.SqlParameter ParContProd = new System.Data.SqlClient.SqlParameter();
ParContProd.ParameterName = "@ContProd";
ParContProd.SqlDbType = SqlDbType.Int;
ParContProd.Size = 4;
ParContProd.Direction = ParameterDirection.ReturnValue;

// Crea objeto XxxCommand, y establece sus propiedades.
System.Data.SqlClient.SqlCommand CmdProd = new System.Data.SqlClient.SqlCommand();
CmdProd.Connection = cnNorthwind;
CmdProd.CommandType = CommandType.StoredProcedure;
CmdProd.CommandText = "CuentaProductosEnCategoria";

// Agrega los parámetros al comando.
CmdProd.Parameters.Add(ParCatID);
CmdProd.Parameters.Add(ParCatNom);
CmdProd.Parameters.Add(ParContProd);

// Abre la conexión.
```

```
cnNorthwind.Open();

// Ejecuta el procedimiento almacenado.
CmdProd.ExecuteScalar();

// Cierra la conexión, lo cual libera la conexión del pool de conexiones.
cnNorthwind.Close();

// Escribe los resultados.
Response.Write("Nombre de Categoría: " +
               CmdProd.Parameters["@CatNom"].Value +
               " Cantidad de productos en categoría: " +
               CmdProd.Parameters["@ContProd"].Value);
```

Ejecutando comandos que regresan renglones.

El `DataReader` es un cursor rápido de sólo hacia delante que itera sobre los renglones resultantes de la ejecución de un objeto `XxxCommand`.

Se puede usar un objeto `Command` y ejecutar el método `ExecuteReader` para regresar un `DataReader`. Se puede ejecutar cualquier enunciado `SELECT` o un procedimiento almacenado que contenga un enunciado `SELECT`.

El `DataReader` proporciona métodos fuertemente tipados, para obtener el valor de una columna específica en el renglón actual. También se pueden obtener metadatos sobre los renglones, así como el nombre de la columna y el tipo de dato de la columna.

Cuando se procesa un Resultset con un `DataReader`, la conexión asociada se mantiene ocupada hasta que se cierra el `DataReader`. Por esta razón, se deberá cerrar el `DataReader` tan pronto como se termine de procesar el Resultset.

Propiedades y métodos del `DataReader`.

El `XxxDataReader` contiene propiedades y métodos para procesar un Resultset extraído por un objeto `Command`. Estas propiedades y métodos permiten:

- Iterar a través del Resultset, un renglón a la vez (método `Read`).
- Obtener el valor de una columna específica (métodos `GetXxx`, por ejemplo `GetString`), o todas las columnas, en el renglón actual (método `GetValues`).
- Verificar si una columna contiene un valor nulo (método `IsDbNull`).
- Obtener los metadatos de una columna, como el nombre (método `GetName`), posición ordinaria (método `GetOrdinal`), y tipo de dato.

Para iterar a través del Resultset, llamar el método `Read` en el Objeto `DataReader`. El método `Read` se mueve al siguiente renglón en el Resultset, usando el objeto `Connection` asociado.

El método `Read` regresa `false` cuando no hay más registros que leer. En este punto, se deberá llamar el método `Close` del `DataReader` y liberar el objeto `Connection`.

Existen diferentes maneras de extraer valores de columnas del renglón actual:

- La propiedad `Item` obtiene el valor de una columna con un nombre específico o posición ordinaria. Debido a que el valor es regresado en su formato nativo, se necesita convertir el valor antes de que pueda utilizarse en el código.

- El `DataReader` tiene métodos fuertemente tipados, como `GetDateTime`, `GetDouble`, `GetInt`. Usar estos métodos cuando se conozca los tipos de datos en el Recordset, para minimizar la cantidad de conversión de tipos requeridos en el código.

- El método `GetValues` regresa un arreglo de objetos conteniendo todos los valores de las columnas del renglón actual. Esto puede ser más eficiente que extraer cada columna individualmente.

Para usar el objeto `DataReader` para procesar renglones:

1. Crear un objeto `Command` y establecer las propiedades y parámetros si es necesario.
2. Declarar una variable `XxxDataReader`, dependiendo que proveedor de datos se está usando.
3. Escribir código para abrir la conexión a la base de datos.
4. Llamar el método `ExecuteReader` del objeto `Command`, incluyendo la opción para cerrar la conexión inmediatamente después de que el `DataReader` es cerrado. Asignar el valor de retorno a la variable `DataReader`.
5. Iterar a través del `DataReader` usando el método `Read`, hasta que el método regrese `false`.
6. Cerrar el `DataReader`.

Ejecutando múltiples enunciados SQL.

Un procedimiento almacenado puede contener cualquier número de enunciados SQL. Esto permite tener un grupo de tareas relacionadas en el mismo procedimiento almacenado, para encapsular las reglas de negocio y mejorar el rendimiento en tiempo de ejecución.

Cuando se ejecuta un procedimiento almacenado, se puede usar un `DataReader` para extraer las siguientes piezas de información:

- El Resultset regresado por cada enunciado `SELECT` en el procedimiento almacenado.
- El número total de registros afectados por el procedimiento almacenado.

Para procesar un comando que contiene múltiples enunciados SQL:

1. Crear un objeto `Connection` y un objeto `Command`, configurando el comando que se desee ejecutar.
2. Abrir la conexión a la base de datos.
3. Llamar el método `ExecuteReader` del objeto `Command`, y asignar el valor de retorno a una variable `DataReader`.
4. Usar el objeto `DataReader` para iterar a través de renglones en el primer `Resultset`.
5. Si hay múltiples enunciados SELECT, llamar el método `NextResult` en el `DataReader`, para avanzar al siguiente `Resultset`, y entonces repetir el paso 4.
6. Cerrar el `DataReader` y la conexión a la base de datos.
7. Usar la propiedad `RecordsAffected` en el `DataReader`, para encontrar el número total de registros cambiados, insertados o borrados durante la ejecución del objeto `Command`.

Para el ejemplo, supongamos que se decide descontinuar todos los productos que cuesten mas de $50. Todos los demás productos continuarán disponibles.

Para esto, se tiene el siguiente procedimiento almacenado que regresa dos Resultsets. El primer Resultset (SELECT) contiene los productos descontinuados. El segundo Resultset (SELECT) contiene los productos disponibles. Notar que el primer enunciado UPDATE descontinúa los productos que cuestan más de $50 y el segundo enunciado UPDATE asegura que todos los demás productos permanezcan disponibles:

```
CREATE PROCEDURE dbo.AjustarDisponibilidadDeProducto
AS
  UPDATE Products SET Discontinued = 1 WHERE UnitPrice > 50
  SELECT ProductName FROM Products WHERE Discontinued = 1
  UPDATE Products SET Discontinued = 0 WHERE UnitPrice <= 50
  SELECT ProductName FROM Products WHERE Discontinued = 0
  RETURN
```

El siguiente código en C#, asume que el objeto `Connection` y el objeto `Command` han sido previamente configurados:

```
// Abre la conexión.
cnNorthwind.Open();

// No hay sobrecarga, se crea con la ejecución del comando.
System.Data.SqlClient.SqlDataReader rdrProd = CmdProd.ExecuteReader();

// Itera a través de cada ResultSet.
do
{
  // Total de registros afectados (suma RecordSets).
  Response.Write("Total de Registros afectados: " + rdrProd.RecordsAffected);
  // Itera los elementos del ResultSet, mostrando los datos.
  while (rdrProd.Read())
  {
    for(int i=0; i<rdrProd.FieldCount; i++)
    {
      //Obtiene el valor de la columna en su formato nativo.
      Response.Write(rdrProd.GetValue(i));
    }
  }
} while (rdrProd.NextResult());

// Cierra el XxxDataReader.
rdrProd.Close();

// Cierra la conexión, lo cual libera la conexión del pool de conexiones.
cnNorthwind.Close();
```

Ejecutando comandos que no regresan renglones.

El método `ExecuteNonQuery` permite ejecutar un comando sin recuperar ninguna información, excepto el número de renglones afectados. Es ideal para ejecutar enunciados de Lenguaje de Definición de Datos (*Data Definition Language*, DDL; CREATE, ALTER, DROP) para crear y manejar estructuras de bases de datos, así como tablas, vistas, y disparadores (triggers). También se puede usar en enunciados de Lenguaje de Control de Datos (*Data Control Language*, DCL; GRANT, DENY, REVOKE) para configurar cuestiones de seguridad de una base de datos. Finalmente, se puede usar enunciados de Lenguaje de Manipulación de Datos (*Data Manipulation Language*, DML; INSERT, UPDATE, DELETE) para modificar datos en una base de datos.

Para ejecutar un comando DDL o DCL:

1. Crear un objeto `Connection` y un objeto `Command`, y configurar estos objetos para el enunciado que se quiera ejecutar.
2. Abrir la conexión a la base de datos.
3. Llamar el método `ExecuteNonQuery` del objeto `Command`.
4. Cerrar la conexión a la base de datos.

Para ejecutar un enunciado DML:

1. Crear un objeto `Connection` y un objeto `Command`, y configurar estos objetos para el enunciado DML que se desea llamar.
2. Abrir la conexión a la base de datos.
3. Llamar el método `ExecuteNonQuery` del objeto `Command`. Asignar el valor de retorno a una variable entera, para indicar el número de renglones afectados por el enunciado DML.
4. Cerrar la conexión a la base de datos.

```
CREATE PROCEDURE dbo.InsertaRegion
( @RegionDesc nchar(50) )
AS
  INSERT INTO Region(RegionDescription)
       VALUES (@RegionDesc)
  RETURN @@IDENTITY
```

El siguiente código en C#, asume que el objeto `Connection` ha sido previamente configurado; y la propiedad `Identity` debe estar activada en el diseño de la tabla `Region`.

```
// Parámetro @CatID, crea objeto XxxParameter y establece sus propiedades.
System.Data.SqlClient.SqlParameter ParRegionDesc = new System.Data.SqlClient.SqlParameter();
ParRegionDesc.ParameterName = "@RegionDesc";
ParRegionDesc.SqlDbType = SqlDbType.NChar;
ParRegionDesc.Size = 50;
ParRegionDesc.Direction = ParameterDirection.Input;
ParRegionDesc.Value = "Este"; // Nuevo registro.

// Para obtener un valor de retorno de un procedimiento almacenado.
// Parámetro @ValRetorno, crea objeto XxxParameter y establece sus propiedades.
System.Data.SqlClient.SqlParameter ParIdentity = new System.Data.SqlClient.SqlParameter();
ParIdentity.ParameterName = "@ValRetorno";
ParIdentity.SqlDbType = SqlDbType.Int;
ParIdentity.Size = 4;
ParIdentity.Direction = ParameterDirection.ReturnValue;

// Crea objeto XxxCommand, y establece sus propiedades.
System.Data.SqlClient.SqlCommand CmdRegion = new System.Data.SqlClient.SqlCommand();
CmdRegion.Connection = cnNorthwind;
CmdRegion.CommandType = CommandType.StoredProcedure;
CmdRegion.CommandText = "InsertaRegion";

// Agrega los parámetros al comando.
CmdRegion.Parameters.Add(ParRegionDesc);
CmdRegion.Parameters.Add(ParIdentity);

// Abre la conexión.
cnNorthwind.Open();

// Ejecuta el procedimiento almacenado.
System.Int32 iRenAfect = (System.Int32) CmdRegion.ExecuteNonQuery();

// Cierra la conexión, lo cual libera la conexión del pool de conexiones.
cnNorthwind.Close();

// Escribe los resultados.
Response.Write(iRenAfect +
  " renglones afectados.  Identificador asignado: " +
  CmdRegion.Parameters["@ValRetorno"].Value);
```

Usando transacciones.

Una transacción es una simple unidad de trabajo. Si una transacción es exitosa, todas las modificaciones a los datos son hechas (commit) y se hacen parte permanente de la base de datos. Si la transacción encuentra errores debe cancelarse (rollback), y todos los cambios a los datos dentro del alcance de la transacción son eliminados.

Las transacciones pueden ser manejadas en la capa de base de datos usando enunciados SQL. ADO.NET permite manejar transacciones en una aplicación para el Framework .NET en una capa intermedia.

Los objetos `XxxConnection` tienen un método `BeginTransaction`, el cual regresa un objeto `XxxTransaction`. A su vez estos objetos tienen métodos llamados `Commit` y `Rollback` para manejar la transacción en la aplicación.

Para realizar una transacción usando ADO.NET:

1. Llamar el método `BeginTransaction` del objeto `Connection`. Asignar el valor de retorno a una variable `XxxTransaction`.
2. Para todos los comandos que se quieran ejecutar dentro de esta transacción, establecer la propiedad `Transaction` refiriéndose al objeto `Transaction`.
3. Ejecutar los objetos `Command` requeridos.
4. Si los comandos se completan satisfactoriamente, llamar el método `Commit` en el objeto `Transaction`. Si cualquier problema ocurre, llamar el método `Rollback` para deshacer los cambios.

Niveles de aislamiento.

Los niveles de aislamiento especifican el comportamiento del bloqueo de la transacción para una conexión. esto es apropiado para prevenir problemas de concurrencia cuando múltiples transacciones acceden los mismos datos.

- En un extremo, se puede permitir que las transacciones tengan accesos sin impedimentos a la base de datos. Esto minimiza el tiempo de espera para los enunciados en las transacciones, pero incrementa el riesgo de corrupción de datos debido a acceso concurrente.

- En el otro extremo, se pueden especificar transacciones que sean completamente aisladas de otras. Las transacciones son ejecutadas serialmente, una tras otra.

Si varias transacciones acceden los mismos datos al mismo tiempo, los siguientes problemas de concurrencia pueden ocurrir.

- ***Lecturas sucias (Dirty reads).*** Ocurre cuando una transacción selecciona un renglón que esta siendo actualizado por otra transacción en ese momento. La transacción original está leyendo datos que no han sido comprometidos todavía, y los datos pueden ser cambiados por otra transacción.

- ***Lecturas no repetibles (Non-repeatable reads).*** Ocurre cuando una transacción lee datos comprometidos una vez, después los lee de nuevo y obtiene un valor distinto. Esto sucede si otra transacción tiene los datos actualizados entre las dos operaciones de lectura.

- ***Lecturas fantasmas (Phantom reads).*** Ocurre cuando una transacción lee datos que están siendo eliminados en ese momento por otra transacción. Si la transacción original lee los datos de nuevo, este no verá los renglones eliminados.

Los objetos `XxxTransaction` tienen una propiedad llamada `IsolationLevel`. Establecer esta propiedad cuando se llame `BeginTransaction` en el objeto `Connection`.

La Tabla 4.6 describe los niveles de aislamiento permitidos.

Nivel de aislamiento	Descripción
`ReadUncommitted`	El aislamiento de transacción es únicamente para prevenir lectura de datos corruptos. Pueden ocurrir lecturas sucias, lecturas no repetibles y lecturas fantasma.
`ReadCommited`	Los candados compartidos son retenidos mientras son leídos los datos, para prevenir lecturas sucias. Sin embargo, los datos pueden ser cambiados antes del fin de la transacción, causando lecturas no repetibles o lecturas fantasma. este es el nivel de aislamiento por defecto.
`RepeatableRead`	Todos los datos usados en una consulta están bloqueados. Esto previene que otros usuarios actualicen los datos, y por consiguiente previene lecturas no repetibles. Sin embargo, lecturas fantasma pueden ocurrir.
`Serializable`	Las transacciones son completamente aisladas una de otra. Esto previene lecturas sucias, lecturas no repetibles, y lecturas fantasma.
`Unspecified`	Un diferente nivel de aislamiento que uno especificado está siendo usado, pero el nivel no puede ser determinado.

Tabla 4.6 *Niveles de aislamiento.*

El siguiente ejemplo muestra como empezar una transacción utilizando el nivel de aislamiento `Serializable`. Esto asegura protección máxima contra errores de concurrencia, a expensas de rendimiento en tiempo de ejecución. La transacción se ejecutará exitosamente si las tablas `Region` y `Territories` no tienen ninguna restricción de integridad (ejemplo, si existe una llave foránea en `Territories` que haga referencia a una llave primaria en `Region`) comprometiendo los datos (`Commit`); en caso contrario se dará marcha atrás cualquier cambio sobre los datos (`Rollback`), inclusive si hay cualquier otro error.

```
// Abre la conexión.
cnNorthwind.Open();

// No hay sobrecarga, se crea con la ejecución del método.
System.Data.SqlClient.SqlTransaction trans =
cnNorthwind.BeginTransaction(IsolationLevel.Serializable);

// Crea objeto XxxCommand, y establece sus propiedades.
System.Data.SqlClient.SqlCommand CmdDel = new System.Data.SqlClient.SqlCommand();
CmdDel.Connection = cnNorthwind;
CmdDel.CommandType = CommandType.Text;
CmdDel.Transaction = trans;

try
{
  // Ejecuta el comando. Primer comando.
  CmdDel.CommandText = "DELETE Region";
  System.Int32 iRenAfect = (System.Int32) CmdDel.ExecuteNonQuery();

  // Segundo comando.
  CmdDel.CommandText = "DELETE Territories";
```

```
  iRenAfect = iRenAfect + (System.Int32) CmdDel.ExecuteNonQuery();

  // Compromete los datos.
  trans.Commit();

  // Escribe los resultados.
  Response.Write(iRenAfect + " renglones afectados.");
}catch(System.Exception ex){
  // Falla transacción.
  trans.Rollback();

  // Escribe la descripción del error.
  Response.Write("Error en la transacción: " + ex.Message);
}
finally
{
  // Cierra la conexión, lo cual libera la conexión del pool de conexiones.
  cnNorthwind.Close();
}
```

Usando clases ADO.NET en un escenario desconectado.

En un ambiente actual de negocio, muchas corporaciones almacenan sus datos en bases de datos relacionales. Los objetos `DataAdapter` son diseñados para extraer subconjuntos de datos de estas bases de datos y almacenarlos de manera desconectada en un `DataSet`.

La Figura 4.3 nos muestra un ejemplo de arquitectura desconectada.

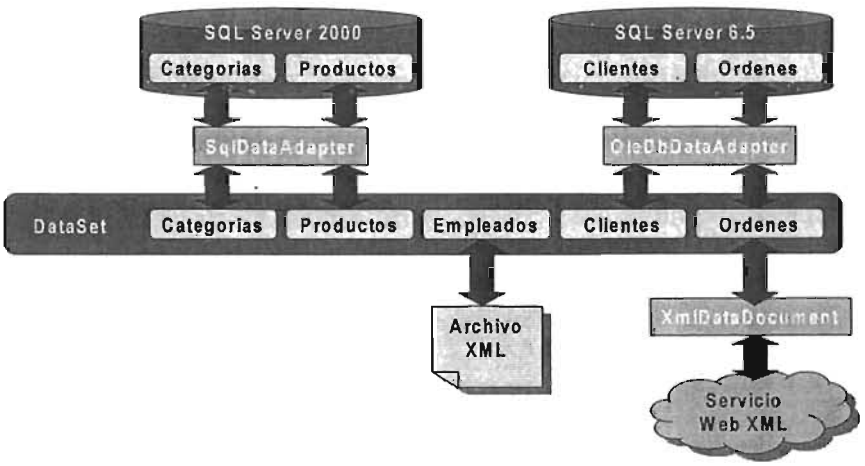

Figura 4.3 *Arquitectura desconectada.*

Cada vez se incrementa la cantidad de corporaciones que almacenan o intercambian datos utilizando formato XML. Los objetos `DataSet` pueden ser guardados como una combinación de XSD[31] (XML Schema Definition) para la estructura y XML para los datos. Se puede necesitar la conversión de otros orígenes de datos en un esquema que el objeto `DataSet` reconozca usando de intermediario un objeto `XMLDataDocument`.

[31] XSD expresa vocabulario compartido y reglas hechas por la gente que da significado a la definición de estructura, contenido y semántica de documentos XML.

Construyendo DataSets y DataTables.

En ADO.NET los objetos DataSet, DataTable, y DataColumn permiten representar datos en un caché local y proporcionan un modelo de programación relacional de datos independiente de su origen.

La conexión al origen de datos no necesita estar activa en una aplicación para ver y manipular datos en un DataSet. Esta arquitectura desconectada proporciona gran escalabilidad al utilizar los recursos del servidor de base de datos sólo cuando se necesite leer o escribir al origen de datos.

Los DataSet almacenan datos de manera similar a como se guardan los datos en una base de datos relacional con un modelo de objetos jerárquico de tablas, renglones, y columnas. Adicionalmente, se pueden definir restricciones y relaciones para los datos en un DataSet.

Los objetos DataTable son usados para representar las tablas en un DataSet. Un DataTable representa una tabla de datos relacional en memoria. Los datos son locales a la aplicación .NET en la cual residen, pero extraídos de un origen de datos existente. Un DataTable esta compuesto de DataColumns.

Un DataColumn es el bloque de construcción para crear el esquema de un DataTable. Cada DataColumn tiene una propiedad DataType que determina el tipo de dato que cada DataColumn contiene. Por ejemplo, se puede restringir el tipo de dato a enteros, cadenas, o decimales. Debido a que los datos contenidos en el DataTable son típicamente empatados de vuelta con el origen de datos, deben coincidir los tipos de datos.

La clase DataSet tiene una propiedad Tables que obtiene una colección de objetos DataTable en el DataSet, y una propiedad Relations que obtiene una colección de objetos DataRelation en el DataSet.

Un objeto DataTable contiene varias colecciones que describen los datos en la tabla y mantienen los datos en memoria. La Tabla 4.7 describe las colecciones más importantes.

Nombre de colección	Tipo de objeto en la colección	Descripción de objetos en la colección
Columns	DataColumn	Contiene metadatos acerca de una columna en la tabla, así como el nombre de la columna, tipo de dato, y si los renglones pueden contener valores nulos en esta columna.
Rows	DataRow	Contiene un renglón de datos en la tabla. Un objeto DataRow también mantiene los datos originales en el renglón, antes que cualquier cambio sea hecho por la aplicación.
Constraints	Constraint	Representa una restricción en uno o más objetos DataColumn. Constraint es una clase abstracta. Hay dos subclases concretas: UniqueConstraint y ForeignKeyConstraint.
ChildRelations	DataRelation	Representa una relación a una columna en otra tabla dentro del DataSet. Se utilizan objetos DataRelation para crear ligas entre llaves primarias y llaves foráneas en las tablas.

Tabla 4.7 *Colecciones de un DataTable.*

Si un DataSet tiene múltiples tablas, algunas de las tablas pueden contener datos relacionados. Se crean objetos DataRelation para describir estas relaciones. Un DataSet puede contener una colección de objetos DataRelation.

Se pueden utilizar objetos DataRelation para sacar programáticamente registros hijo relacionados a un registro padre, o un registro padre de un registro hijo.

Para crear un DataSet y un DataTable programáticamente, usar el constructor del DataSet para inicializar una nueva instancia de la clase DataSet, y usar el constructor de DataTable para inicializar una nueva instancia de la clase DataTable.

El Dataset debe tener un nombre para asegurar que su representación XML siempre tiene un nombre para el elemento Document, el cual es el elemento de mas alto nivel en una definición de esquema XML. Si se omite el nombre, se establece por defecto a "NewDataSet".

Se puede crear un objeto DataTable usando el constructor DataTable, o pasando argumentos al método Add de la propiedad Tables del objeto DataSet, el cual es un DataTableCollection.

Se pueden establecer parámetros para un constructor DataTable o un DataColumn al momento que el DataTable o el DataColumn es creado. Esto es recomendado porque se puede crear el DataTable o definir parámetros para este usando únicamente una línea de código.

Después de que se haya agregado un DataTable como un miembro de la colección Tables de un DataSet, no se podrá agregar este a la colección de tablas de cualquier otro DataSet. Se tendrá que usar el método Clone de un DataTable para crear un nuevo DataTable con la misma estructura (pero sin datos), o se puede usar el método Copy para crear un nuevo DataTable con la misma estructura y datos.

El siguiente ejemplo crea programáticamente un DataSet nombrado "Northwind" con una variable llamada dsNorthwind para hacer referencia al DataSet:

```
System.Data.DataSet dsNorthwind = new DataSet("Northwind");
```

El siguiente ejemplo crea una instancia de un objeto DataTable y le asigna el nombre de "Clientes":

```
System.Data.DataTable dtClientes = new System.Data.DataTable("Clientes");
```

No se requiere proporcionar un valor para la propiedad TableName cuando se crea un DataTable. Se puede especificar la propiedad TableName más adelante, o se puede dejar vacía. Sin embargo, cuando se agrega una tabla sin un valor para el parámetro TableName al DataSet, se da un valor incremental por defecto de "TableN" comenzando con "Table1" para la primera tabla sin nombre.

Cuando se crea un DataTable, este no tiene un esquema. Para definir el esquema de la tabla, se debe crear y agregar objetos DataColumn a la colección Columns.

Se crean objetos DataColumn dentro de una tabla usando el constructor DataColumn, o llamando el método Add de la propiedad Columns de la tabla. El método Add aceptará los argumentos opcionales ColumnName, DataType, y Expression para crear un nuevo DataColumn

como miembro de la colección, o aceptará un objeto `DataColumn` existente para agregarlo a la colección.

El siguiente ejemplo agrega una columna al `DataTable`, que no acepta nulos y sus valores deben ser únicos:

```
System.Data.DataColumn colClienteID = dtClientes.Columns.Add("ClienteID", typeof(Int32));
colClienteID.AllowDBNull = false;
colClienteID.Unique = true;
```

Creando constraints.

Una tabla de base de datos comúnmente tiene una columna, o grupo de columnas, que identifican de manera única cada renglón de la tabla. Esta columna que identifica columnas, o agrupa columnas, es llamada llave primaria (*Primary Key*, PK).

La propiedad `PrimaryKey` de un `DataTable` recibe como valor un arreglo de uno o más objetos `DataColumn`, como se muestra en el siguiente ejemplo:

```
System.Data.DataColumn[] arrCol = new System.Data.DataColumn[1];
arrCol[0] = dtClientes.Columns["ClienteID"];
dtClientes.PrimaryKey = arrCol;
```

Una base de datos relacional debe garantizar la integridad de los datos para asegurar la calidad de los datos. Una manera de mantener la integridad en ADO.NET es agregando constraints a las tablas dentro del `DataSet`. Un *constraint* es una regla automática aplicada a una columna, o a columnas relacionadas, que determinan que acciones deberán tomar lugar cuando el valor de un registro es modificado.

Hay dos tipos de constraints en ADO.NET, el constraint de llave foránea (*Foreign Key*, FK) y el constraint único (*Unique*). cuando se agrega un objeto `DataRelation`, el cual crea una relación entre dos o más tablas, ambos constraints pueden ser creados automáticamente.

El objeto `UniqueConstrain`, puede ser asignado a cualquier columna o arreglo de columnas en el `DataTable`. Asegura que todos los datos en la columna especificada sean únicos por renglón. Se puede crear un constraint único para una columna simple estableciendo la propiedad `Unique` de la columna a `true`.

Se puede también crear un constraint unique para una columna o arreglos de columnas usando el constructor `UniqueConstraint` y pasar el objeto `UniqueConstraint` al método `Add` de la propiedad `Constraints` de las tablas (el cual es un `ConstraintCollection`). También se puede usar el método `Add` para agregar objetos constraint existentes a la colección `Constraints`.

Adicionalmente, definiendo una columna o columnas como llave primaria para una tabla automáticamente se creará un constraint único para la columna(s) especificada.

Un objeto `UniqueConstraint` dispara una excepción cuando se intenta establecer un valor no único en una columna.

El siguiente ejemplo crea un objeto `UniqueConstraint` para una columna existente en automático:

```
dsNorthwind.Tables["Cientes"].Columns["ProductName"].Unique = true;
```

Definiendo relaciones de datos.

Un constraint de llave foránea restringe la acción realizada en tablas relacionadas cuando los datos en una tabla son actualizados o eliminados. Por ejemplo, si un valor en un renglón de una tabla esta actualizado o eliminado, y el mismo valor es también usado en una o más tablas relacionadas, se pude utilizar la propiedad `DeleteRule` o `UpdateRule` de un `ForeignKeyConstraint` para determinar que sucede en las tablas relacionadas.

Un `ForeignKeyConstraint` puede restringir, así como propagar, cambios en las columnas relacionadas. Dependiendo lo establecido en las propiedades del `ForeignKeyConstraint` de una columna, y si la propiedad `EnforceConstraints` del `DataSet` es `true` (valor por defecto), se puede por ejemplo, que un renglón padre no pueda ser eliminado si éste tiene algún renglón hijo, estableciendo la propiedad `DeleteRule` de un `ForeignKeyConstraint` en `None` para arrojar una excepción si sucede.

Cuando se crea un `ForeignKeyConstraint`, se pueden pasar los valores `DeleteRule` y `UpdateRule` al constructor como argumentos, o se pueden establecer como propiedades.

El siguiente ejemplo muestra como crear un `ForeignKeyConstraint` en la columna `ClienteID` de la tabla `Ordenes`. El valor `DeleteRule` se establece para prevenir que se elimine cualquier cliente que tenga ordenes existentes.

```
// Toman las referencias de las tablas.
dtClientes = dsNorthwind.Tables["Cliente"];
dtOrdenes = dsNorthwind.Tables["Ordenes"];

// Crea el ForeignKeyConstraint
System.Data.ForeignKeyConstraint fkOrdenesClientes =
  new System.Data.ForeignKeyConstraint("FK_OrdenesDelCliente",
  dtClientes.Columns["ClienteID"],
  dtOrdenes.Columns["ClienteId"]);

// Define la regla.
fkOrdenesClientes.DeleteRule = Rule.None;
dtOrdenes.Constraints.Add(fkOrdenesClientes);
```

El objeto `DataRealation`.

Un objeto `DataRelation` define la relación entre dos tablas. Típicamente, dos tablas son ligadas a través de un campo sencillo que contiene el mismo tipo de dato y valores similares.

El objeto `DataRelation` proporciona registros hijos si se está en un renglón padre, y un registro padre si se está trabajando con un registro hijo.

Aunque un `Dataset` contiene tablas y columnas en una estructura relacional similares a una base de datos, el `DataSet` no tiene la habilidad para relacionar tablas. Sin embargo, se pueden crear objetos `DataRelation` que establezcan una relación entre una tabla padre (maestro) y una tabla hija (detalle) basados en una llave común.

Para crear un objeto `DataRelation`, se debe usar el constructor `DataRelation` o el método `Add` de la colección `Relations` de un `DataSet` como se muestra a continuación:

```
dsNorthwind.Relations.Add(
  "FK_OrdenesDelCliente",
  dtClientes.Columns["ClienteID"],
  dtOrdenes.Columns["ClienteID"],
  true); // Crea un ForeignKeyConstraint también.
```

Modificando datos en un DataTable.

Después de crear un DataTable en un DataSet, se pueden realizar las mismas actividades que se hacen sobre una tabla en una base de datos como agregar, editar, eliminar datos, crear vistas, monitorear errores y eventos, y consultar datos. Cuando se modifican datos en un DataTable, se puede verificar si los cambios son correctos, y determinar programáticamente si se aceptan o rechazan los cambios.

Insertando un nuevo renglón en un DataTable.

Para agregar un nuevo renglón al DataTable, declarar una nueva variable de tipo DataRow. Un nuevo objeto DataRow es regresado cuando se llama el método NewRow del DataTable. El DataTable crea un objeto DataRow basado en la estructura de la tabla, como esté definida en el DataColumnCollection.

```
System.Data.DataRow drNuevoEmpleado = dtEmpleados.NewRow();
```

Después de agregar un nuevo renglón al DataTable, se puede manipular el nuevo renglón usando un índice o el nombre de la columna:

```
drNuevoEmpleado["EmpleadoID"] = 1;
drNuevoEmpleado["Nombre"] = "Juan";
```

Después de que los datos son insertados en un nuevo renglón, el método Add es usado para agregar el renglón al DataRowCollection.

```
dtEmpleados.Rows.Add(drNuevoEmpleado);
```

Modificando y eliminando datos en una tabla.

La clase DataRow es usada para manipular registros individuales. Esta clase proporciona tres métodos para suspender y reactivar el estado del renglón mientras se edita: BeginEdit, EndEdit y CancelEdit.

Se llama a BeginEdit para suspender cualquier evento o excepción mientas se editan datos. Se usan los elementos de la colección para especificar los nombres de las columnas de los datos que se quieran modificar y los nuevos valores. Se usa EndEdit para reactivar cualquier evento o excepción.

El siguiente ejemplo muestra como usar el método BeginEdit, los elementos de la colección, y el método EndEdit:

```
System.Data.DataRow drEmpleado = dtEmpleados.Rows[1];
drEmpleado.BeginEdit();
drEmpleado["EmpleadoID"] = 1;
drEmpleado["Nombre"] = "Sara";
drEmpleado.EndEdit();
```

Existen dos métodos para eliminar un objeto DataRow de un objeto DataTable: el método Remove, que elimina un DataRow del objeto DataRowCollection, y el método Delete del objeto DataRow, que sólo marca el renglón para eliminarse.

```
System.Data.DataRow drEmpleado = dtEmpleados.Rows[1];
// Usando el método Remove.
dtEmpleados.Rows.Remove(drEmpleado);
// Usando el método Delete.
drEmpleado.Delete();
```

El objeto `DataView`.

Un objeto `DataView` es similar a una vista en un manejador de base de datos. Este es un objeto que presenta un subconjunto de datos de un `DataTable`. Un objeto `DataView` proporciona una vista filtrada y ordenada de los contenidos de una tabla. Esto capacidad permite tener controles ligados al mismo `DataTable`, pero mostrando diferentes versiones de los datos.

El siguiente es un ejemplo de cómo crear un `DataView` programáticamente:

```
System.Data.DataView dvEmpleados =
  new System.Data.DataView(dsNorthwind.Tables["Empleados"]);
dvEmpleados.Sort = "Nombre";
dvEmpleados.RowFilter = "EmpleadoID = 1";
grdEmpleados.DataSource = dvEmpleados;
```

Construyendo `DataSets` de orígenes de datos existentes.

En el ambiente de .NET, los datos pueden moverse desde un origen de datos central a un `DataSet` local. Para mover los datos, debe haber un puente desde el origen de datos hacia el `DataSet`. Este puente es el `DataAdapter`.

El objeto `DataAdapter`.

El objeto `DataAdapter` sirve como un puente entre un `DataSet` y un origen de datos para extraer y guardar datos. La clase `DataAdapter` representa un conjunto de comandos de base de datos y una conexión a la base de datos que se usa para llenar un `DataSet` y actualizar el origen de datos. La Figura 4.4 muestra el modelo de objetos del `DataAdapter` que se usan para poblar el `DataSet` y enviar las actualizaciones de vuelta al origen de datos.

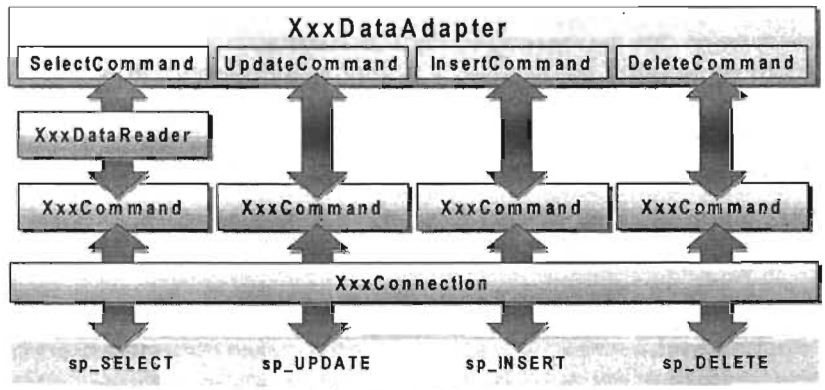

Figura 4.4 *Modelo de objetos `XxxDataAdapter`.*

Un ejemplo común de esto ocurre cuando una aplicación lee datos de una base de datos a través de un `DataSet`, para después escribir cambios dentro del `DataSet` y se apliquen de vuelta a la base de datos. Sin embargo, un `DataAdapter` puede extraer y actualizar datos desde cualquier origen.

Propiedades y métodos del DataAdapter.

Los objetos DataAdapter se usan para especificar que acciones se realizarán en los registros del origen de datos a través de sus propiedades. Estas propiedades son objetos instanciados de la clase XxxCommand:

- SelectCommand. Extrae renglones de un origen de datos.
- InsertCommand. Escribe renglones insertados del DataSet al origen de datos.
- UpdateCommand. Escribe renglones modificados del DataSet al origen de datos.
- DeleteCommand. Elimina renglones en el origen de datos.

Los métodos del DataAdapter se usan para llenar un DataSet o transmitir cambios en un una tabla del DataSet a la tabla correspondiente del almacén de datos. Estos métodos incluyen:

- Fill. Usar éste método para agregar o refrescar renglones de datos de un origen de datos y ponerlos en una tabla DataSet. Este método usa un enunciado SELECT especificado en la propiedad SelectCommand.
- Update. Usar este método para transmitir cambios de una tabla DataSet al origen de datos correspondiente. Este método llama los comandos INSERT, UPDATE O DELETE correspondientes para cada renglón especificado en un DataTable del DataSet.

Poblando un DataSet usando un DataAdapter.

Usar el método Fill en el DataAdapter, especificando la tabla del DataSet que se desea llenar, como se muestra a continuación:

```
varRenAfectados = objetoDataAdapter.Fill(objetoDataSet, "NombreTabla");
```

Éste método implícitamente ejecuta la consulta SQL o el procedimiento almacenado en la propiedad SelectCommand del DataAdapter. Los resultados de la consulta son usados para definir la estructura de la tabla del DataSet, y poblar la tabla con datos.

Se pueden usar múltiples DataAdapters para llenar un DataSet. Cada DataAdapter llena una tabla separada en el DataSet. Debido a que cada DataAdapter mapea a una tabla simple del DataSet, se puede controlar el orden en el cual se quieren escribir las actualizaciones de vuelta a la base de datos. Esto ayuda a preservar integridad referencial entre tablas relacionadas en la base de datos.

La forma más eficiente de llenar un DataSet es definir explícitamente el esquema antes de llenarlo. Esto significa que los DataTables, DataColumns, y DataRelations son de antemano conocidos antes de que sea llenado el DataSet. Esto no siempre es necesario, porque si no es conocido el esquema en tiempo de diseño, el esquema del DataSet podrá ser creado en tiempo de ejecución, basándose en la estructura de los datos extraídos; aunque es menos eficiente que construir DataSets tipados en tiempo de diseño.

El escenario del siguiente ejemplo, extrae información de forma desconectada de una base de datos. La estructura de los datos es ya conocida, por lo que se crea un DataSet

tipado, con un esquema que conforma la estructura de los datos extraídos. Supongamos que se necesita la información de todas las categorías y sus productos clasificados, entonces serán necesarios dos `DataAdapters` para llenar y mapear a las tablas del `DataSet`. Se definen constraints para garantizar integridad de datos en ambas tablas (manejados en la capa media, independiente del origen de datos). Finalmente dos controles `DataGrid` mostrarán los datos de ambas tablas del `DataSet` correspondientes.

```
// Crea la tabla Categorías, agrega sus columnas y constraint de PK.
System.Data.DataTable dtCategorias = new System.Data.DataTable("Categorias");
System.Data.DataColumn colCategoriaID =
  new System.Data.DataColumn("CategoriaID", typeof(int));
System.Data.DataColumn colCategoriaNombre =
  new System.Data.DataColumn("CategoriaNombre", typeof(String));
dtCategorias.Columns.Add(colCategoriaID);
dtCategorias.Columns.Add(colCategoriaNombre);
dtCategorias.Constraints.Add("PK_CategoriaID", colCategoriaID, true);

// Crea la tabla Productos, agrega sus columnas y constraint de PK.
System.Data.DataTable dtProductos = new System.Data.DataTable("Productos");
System.Data.DataColumn colProductoID =
  new System.Data.DataColumn("ProductoID", typeof(int));
System.Data.DataColumn colProductoNombre =
  new System.Data.DataColumn("ProductoNombre", typeof(String));
System.Data.DataColumn colProdCatID =
  new System.Data.DataColumn("CategoriaID", typeof(int));
dtProductos.Columns.Add(colProductoID);
dtProductos.Columns.Add(colProductoNombre);
dtProductos.Columns.Add(colProdCatID);
dtProductos.Constraints.Add("PK_ProductoID", colProductoID, true);

// Crea un SqlDataAdapter y un SqlCommand para llenar la tabla padre de Categorías.
System.Data.SqlClient.SqlDataAdapter daCategorias = new
System.Data.SqlClient.SqlDataAdapter();
System.Data.SqlClient.SqlCommand cmSelect = new System.Data.SqlClient.SqlCommand("SELECT " +
  " CategoryID AS CategoriaID, CategoryName AS CategoriaNombre FROM Categories",
cnNorthwind);
daCategorias.SelectCommand = cmSelect;

System.Data.DataSet dsNorthwind = new System.Data.DataSet();
dsNorthwind.Tables.Add(dtProductos);
dsNorthwind.Tables.Add(dtCategorias);

// Abre la conexión a la Base de datos.
cnNorthwind.Open();

// Mejora rendimiento apagando los eventos y excepciones mientras se cargan datos.
dsNorthwind.Tables["Categorias"].BeginLoadData();
daCategorias.Fill(dsNorthwind, "Categorias");
dsNorthwind.Tables["Categorias"].EndLoadData();

cmSelect.CommandText = "SELECT ProductID AS ProductoID, ProductName AS ProductoNombre," +
  " CategoryID AS CategoriaID FROM Products";
dsNorthwind.Tables["Productos"].BeginLoadData();
daCategorias.Fill(dsNorthwind, "Productos");
dsNorthwind.Tables["Productos"].EndLoadData();

// Cierra la conexión.
cnNorthwind.Close();
cnNorthwind.Dispose();
cnNorthwind = null;

// Crea FK.
dsNorthwind.Relations.Add(
  "FK_CategoriaDelProducto",
  dtCategorias.Columns["CategoriaID"], // Padre.
  dtProductos.Columns["CategoriaID"], // Hijo.
  true); // Crea un ForeignKeyConstraint.

// Muestra la tabla de Categorias.
DataGrid1.DataSource = dsNorthwind.Tables["Categorias"].DefaultView;
DataGrid1.DataBind();
```

```
// Muestra la tabla de Productos.
DataGrid2.DataSource = dsNorthwind.Tables["Productos"].DefaultView;
DataGrid2.DataBind();
```

Actualizando el origen de datos con un `DataAdapter`.

Cada objeto `DataRow` en un `DataTable` tiene una propiedad `RowState` que indica si un renglón ha sido modificado (`Modified`), insertado (`Added`), eliminado (`Deleted`), o no ha tenido cambios (`Unchanged`) en el `DataSet` desde que se pobló por primera vez.

El `DataSet` mantiene dos copias de datos para cada renglón: la versión actual (`Current`) y la versión original (`Original`).

Estos estatus de renglones en el `DataSet`, son usados por los comandos del `DataAdapter` para modificar datos de vuelta en el origen de datos a través de enunciados SQL o procedimientos almacenados.

Para hacer persistentes los cambios en el origen de datos se tienen las siguientes alternativas o métodos:

- `GetChanges`. Para crear un segundo `DataSet` que contenga todos los cambios hechos al `DataSet`. Típicamente se utiliza cuando es necesario pasar los cambios a otro objeto.

- `Merge`. Para combinar los datos del segundo `DataSet` en el primer `DataSet` (con esquemas similares). Típicamente se utiliza al final de una serie de procedimientos que involucran validación de cambios, reconciliación de errores, actualización del origen de datos con los cambios, y finalmente refrescar el `DataSet` existente.

- `Update`. Llama el método del `XxxDataAdapter` con el enunciado SQL apropiado para cada cambio en los renglones del `DataTable`.

- `AcceptChanges`. Para persistir los cambios, o invocar el método `RejectChanges` para cancelar los cambios.

El siguiente ejemplo actualiza el nombre de la categoría para el primer renglón del `DataSet` y hace persistentes los datos en la base de datos. La conexión a la base de datos debió haber sido previamente configurada:

```
// Crea el DataAdapter para llenar la tabla Categorias.
System.Data.SqlClient.SqlDataAdapter daCategoria =
  new System.Data.SqlClient.SqlDataAdapter("SELECT CategoryID, CategoryName " +
  " FROM Categories", cnNorthwind);

// Establece el enunciado SQL quw se dispara con el método Fill.
daCategoria.UpdateCommand = new SqlCommand("UPDATE Categories SET " +
  "CategoryName = @CategoriaNombre WHERE CategoryID = @CategoriaID" , cnNorthwind);

// Parámetro a actualizar.
daCategoria.UpdateCommand.Parameters.Add("@CategoriaNombre", SqlDbType.NVarChar, 15,
"CategoryName");

// Toma el valor original para filtrar la actualización.
System.Data.SqlClient.SqlParameter paramCatID =
  daCategoria.UpdateCommand.Parameters.Add("@CategoriaID", SqlDbType.Int);
paramCatID.SourceColumn = "CategoryID";
paramCatID.SourceVersion = DataRowVersion.Original;
```

```
// Se cargan los datos.
System.Data.DataSet dsCat = new System.Data.DataSet();
daCategoria.Fill(dsCat, "Categories");

// Toma un renglón y establece un nuevo valor.
System.Data.DataRow drRenUp = dsCat.Tables["Categories"].Rows[0];
drRenUp["CategoryName"] = "Nueva Categoria";

// Actualiza y persiste los cambios.
int renAfectados = daCategoria.Update(dsCat, "Categories");
```

Resolución de conflictos de concurrencia.

En una aplicación desconectada se pueden experimentar conflictos con los datos cuando se trata de actualizar el origen de datos con las modificaciones. Esto sucede si otra aplicación ha modificado el origen de datos mientras la aplicación se encontraba desconectada.

Las aplicaciones desconectadas usan concurrencia optimista, donde los candados a la base de datos son liberados tan pronto como operaciones de extracción o de actualización se completen, permitiendo que otras aplicaciones puedan consultar y actualizar la base de datos concurrentemente.

Esto difiere de aplicaciones conectadas, las cuales usan concurrencia pesimista. La base de datos mantiene los candados mientras se realizan una serie de operaciones sobre los datos. Esto detiene otras aplicaciones para el acceso a objetos de la base de datos hasta que otras operaciones sean completadas, previniendo conflictos a costo de acceso denegado temporal a otras aplicaciones.

Se usa la propiedad `HasErrors` de las clases `DataSet`, `DataTable` y `DataRow` para encontrar la ubicación y naturaleza del error en los datos. Adicionalmente la clase `DataRow` tiene un método `GetColumnsInError` para obtener las columnas en error de un renglón particular.

Para resolver conflictos, escoger una de las siguientes estrategias:

- Sobrescribir los cambios en la base de datos hechos por otras aplicaciones con los datos modificados por la aplicación. Esta aproximación es efectiva para aplicaciones administrativas que necesiten forzar cambios en la base de datos.

- No forzar cambios que causen conflictos en la base de datos. Retener el conflicto localmente en el `DataSet`, de tal forma que el usuario pueda actualizar la base de datos de nuevo más tarde.

- Rechazar los cambios de los datos que causan conflicto en el `DataSet` local, y regresar los datos originales cargados de la base de datos (usar el método `RejectChanges`).

- Rechazar los cambios de los datos que causan conflicto en el `DataSet` local, y recargar los últimos datos de la base de datos (usar los métodos `Clear` y `Fill`).

Creación de aplicaciones ASP.NET seguras.

La creación de aplicaciones Web seguras implica cierta complejidad que no debe ser tomada a la ligera. Las aplicaciones Web contienen muchas piezas móviles con varios niveles de exposición en un entorno desconectado. Para conseguir que esas piezas funcionen en conjunto de forma segura, se requiere del conocimiento de varias tecnologías.

En éste capítulo se revisan las técnicas para controlar el acceso a las aplicaciones Web suministradas por ASP.NET para identificar usuarios y permitirles el acceso a la aplicación; así como los conceptos fundamentales para el entendimiento del modelo de seguridad proporcionado por ASP.NET, como autenticación, autorización, suplantación y comunicación segura. Se analizan varios escenarios de autenticación y autorización donde se sugieren técnicas que han sido probadas con buenos resultados.

La seguridad es un tema amplio y con diversas ramificaciones, pues atraviesa todas las capas de una aplicación. Parte del esfuerzo se avoca a los principios de autenticación y autorización, los cuales eliminan un alto porcentaje de vulnerabilidades. La comunicación segura es una parte esencial de la protección de la aplicación distribuida para garantizar la confidencialidad de los datos (incluidas las credenciales) que se transmiten a la aplicación y desde ella, y entre niveles de la aplicación.

El entorno conectado y la necesidad de modelos de seguridad.

Consideremos las características principales del entorno actual de aplicaciones Web, en el que los servicios Web conectan a las empresas con otras empresas y con los clientes, y en el que las aplicaciones traen consigo varios niveles de exposición, como por ejemplo, a usuarios en intranets[32], extranets[33] e Internet.

- Los servicios Web utilizan estándares tales como SOAP, Lenguaje de marcado extensible (XML) y Protocolo de transferencia de hipertexto (HTTP), pero sobre todo transmiten información potencialmente confidencial mediante texto sin formato.

- Las aplicaciones B2C[34] (*business-to-consumer*) de Internet transmiten información confidencial en el Web.

- Las aplicaciones B2B (*business-to-business*) de extranet relajan las condiciones de seguridad y permiten que las aplicaciones sean llamadas por otras aplicaciones de empresas asociadas.

- Las aplicaciones de Internet comportan riesgos si se considera el carácter confidencial de las aplicaciones de nóminas y de recursos humanos. Tales aplicaciones son especialmente vulnerables a administradores malintencionados y empleados descontentos.

[32] Intranet se refiere a la aplicación de las tecnologías Web dentro de la estructura interna de una organización.
[33] Extranet se refiere a una red colaborativa que utiliza la tecnología Web aplicada en una organización para hacer enlaces entre múltiples empresas para intercambio de información.
[34] B2C es un término utilizado para definir las relaciones comerciales y transacciones entre empresas y usuarios particulares, personas físicas de un producto o servicio.

En la actualidad, se ha creado una gran desconfianza sobre las aplicaciones que utilizan tecnologías de Internet debido a las fallas de seguridad en los sistemas tomando información confidencial de sitios de comercio y portales corporativos. En muchos de los casos estas incursiones no se deben a grandes técnicas informáticas, sino a la falta de cultura y ausencia de políticas firmes de seguridad, así como un modelo bien definido de seguridad en las aplicaciones.

Para asegurar una aplicación y ayudar a definir el modelo de seguridad, será necesario delimitar su alcance, identificar dónde y cómo es necesaria.

Conceptos básicos de Seguridad.

Los siguientes conceptos estarán presentes en cualquier discusión sobre seguridad:

- ***Autenticación.*** Proceso que determina la identidad del cliente de la aplicación y lo obliga a probar quien dice ser. Pueden figurar usuarios finales, servicios, procesos o equipos.

- ***Autorización.*** Define lo que pueden ver y hacer en la aplicación los clientes autenticados. Determina si el cliente tiene los permisos suficientes para realizar una acción.

- ***Comunicaciones seguras.*** Garantizan la privacidad y la integridad de los mensajes cuando pasan de una red a otra.

- ***Suplantación o personificación.*** Técnica utilizada por una aplicación de servidor para obtener acceso a recursos en nombre de un cliente. El contexto de seguridad del cliente se utiliza para los controles de acceso realizados por el servidor.

- ***Contexto de seguridad.*** Es un término genérico usado para referirse a la colección de configuraciones de seguridad que afectan el comportamiento relativo a la seguridad de un proceso o subproceso. Los atributos de la sesión de inicio y el testigo de acceso[35] de un proceso conforman el contexto de seguridad del proceso.

- ***Identidad.*** Hace referencia a una característica exclusiva de un usuario o servicio que lo identifica. Suele tratarse, por ejemplo, de un nombre de visualización, que a menudo adopta el formato autoridad/nombre de usuario.

Niveles lógicos de una aplicación Web .NET.

La arquitectura lógica de aplicaciones considera todos los sistemas como conjuntos de servicios en cooperación que se encuentran agrupados en los niveles siguientes:

- Servicios de usuarios.
- Servicios empresariales.
- Servicios de datos.

[35] Un testigo de acceso es una estructura de datos adjunta a cada proceso de Windows. Mantiene información del contexto de seguridad del proceso.

El valor de este punto de vista de la arquitectura lógica radica en la identificación de los tipos genéricos de servicios que siempre están presentes en cualquier sistema, para garantizar la segmentación adecuada y para impulsar la definición de interfaces entre niveles. Esta segmentación permite tomar decisiones acerca de la arquitectura y del diseño al implementar cada nivel, y crear una aplicación más fácil de mantener.

Los niveles pueden describirse tal y como se explica a continuación:

- ***Servicios de usuarios.*** se encargan de la interacción de clientes con el sistema y proporcionan un puente común a la lógica empresarial básica encapsulada por componentes del nivel de Servicios empresariales. Tradicionalmente, los Servicios de usuarios suelen asociarse a los usuarios interactivos. No obstante, también llevan a cabo el procesamiento inicial de peticiones programables de otros sistemas, en las que no participa una interfaz de usuario visible. La autenticación y la autorización suelen realizarse en el nivel de Servicios de usuarios.

- ***Servicios empresariales.*** proporcionan la funcionalidad básica del sistema y encapsulan la lógica empresarial. Son independientes del canal de entrega, de los sistemas de servidor y de los orígenes de datos. Esto ofrece la estabilidad y la flexibilidad necesarias para desarrollar el sistema de forma que admita canales y sistemas de servidor nuevos y diferentes. Normalmente, el procesamiento de una petición empresarial concreta requiere la participación de varios componentes en colaboración del nivel de Servicios empresariales.

- ***Servicios de datos.*** proporcionan acceso a datos (alojados en los límites del sistema) y a otros sistemas (de servidor) a través de interfaces genéricas que resultan sencillas de utilizar desde componentes del nivel de Servicios empresariales. Los Servicios de datos comprenden la diversidad de sistemas de servidor y orígenes de datos, y encapsulan reglas de acceso y formatos de datos específicos.

La clasificación lógica de los tipos de servicios de un sistema puede corresponderse con la posible distribución física de los componentes que implementan los servicios, aunque es relativamente independiente de la misma.

También es importante tener en cuenta que los niveles lógicos pueden identificarse en cualquier nivel de agregación. Por ejemplo, cada nodo remoto que aloja un servicio Web se compone de Servicios de usuarios (que se ocupan de las peticiones y los mensajes entrantes), Servicios empresariales y Servicios de datos.

Modelo de seguridad para aplicaciones ASP.NET.

ASP.NET suele utilizarse para implementar Servicios de usuarios. ASP.NET proporciona una arquitectura conectable que puede utilizarse para crear páginas Web.

Modos de autenticación de ASP.NET.

La autenticación de ASP.NET incluye Windows, Formularios, Passport y Ninguna.

Autenticación de Windows.

Con este modo de autenticación, ASP.NET depende de IIS para la autenticación de usuarios y la creación de un testigo de acceso de Windows para representar la identidad autenticada. IIS ofrece los siguientes mecanismos de autenticación:

- ***Autenticación básica.*** La autenticación básica requiere que el usuario proporcione credenciales en forma de nombre de usuario y contraseña para probar su identidad. Se trata de una propuesta de estándar de Internet que tanto Netscape Navigator como Microsoft Internet Explorer admiten. Las credenciales del usuario se transmiten del Web browser al servidor Web en un formato codificado no cifrado. Puesto que el servidor Web obtiene las credenciales del usuario sin cifrar, puede emitir llamadas remotas (por ejemplo, para obtener acceso a equipos y recursos remotos) con las credenciales del usuario.

  La autenticación básica debe utilizarse únicamente de forma conjunta con un canal seguro (que se suele establecer mediante SSL). De lo contrario, los nombres de usuario y las contraseñas podrían robarse fácilmente con software de supervisión de redes. Si se utiliza la autenticación básica, se deberá utilizar SSL en todas las páginas (y no sólo en páginas de inicio de sesión) porque las credenciales se transmiten en todas las peticiones posteriores.

- ***Autenticación implícita.*** La autenticación implícita, que se introdujo con IIS 5.0, es parecida a la autenticación básica, excepto que las credenciales del usuario se transmiten cifradas del Web browser al servidor Web. Como consecuencia, es más segura, aunque requiere un cliente Internet Explorer 5.0 o posterior y una configuración específica de servidor.

- ***Autenticación de Windows integrada.*** La autenticación de Windows integrada (Kerberos[36] o NTLM[37] en función de la configuración de cliente y servidor) utiliza un intercambio criptográfico con el Web browser Internet Explorer del usuario para confirmar la identidad del usuario. Sólo es compatible con Internet Explorer (no con Netscape Navigator) y, por lo tanto, suele utilizarse solamente en escenarios de Intranet, en los que se puede controlar el software de cliente. La utiliza únicamente el servidor Web cuando se deshabilita el acceso anónimo o cuando se deniega el acceso anónimo mediante permisos de sistema de archivos de Windows.

- ***Autenticación de certificados.*** La autenticación de certificados utiliza certificados de cliente para confirmar la identidad de los usuarios. El certificado de cliente se transmite del Web browser del usuario (o aplicación cliente) al servidor Web. A continuación, el servidor Web extrae la identidad del usuario del certificado. Este enfoque depende de la instalación de un certificado de cliente en el equipo del usuario y, por lo tanto, suele utilizarse casi siempre en escenarios de Intranet o Extranet en los que la población de usuarios se conoce bien y está controlada. Tras la recepción de un certificado de cliente, IIS puede asignar el certificado a una cuenta de Windows.

---

[36] Kerberos es un sistema de autenticación para redes, basado en un modelo de distribución de llaves centralizado.

[37] NTLM es un protocolo de autenticación desafío/respuesta de Windows NT. Este protocolo utiliza cifrado para la transmisión segura de contraseñas.

- ***Autenticación anónima.*** Si no se necesita autenticar los clientes (o se implementa un esquema de autenticación personalizado), se puede configurar IIS para que utilice la autenticación anónima. En este caso, el servidor Web crea un testigo de acceso de Windows para representar a todos los usuarios anónimos con la misma cuenta anónima (o de invitado). La cuenta anónima predeterminada es IUSR_NOMBREEQUIPO, en la que NOMBREEQUIPO es el nombre del equipo que se especificó durante la instalación.

Autenticación de Passport.

En este modo de autenticación, ASP.NET utiliza los servicios de autenticación centralizados de Microsoft Passport. ASP.NET proporciona una práctica funcionalidad de empaquetador expuesta por el Kit de desarrollo de software (SDK) de Microsoft Passport, que debe estar instalado en el servidor Web.

Autenticación mediante Formularios.

Este enfoque utiliza la redirección de cliente para enviar usuarios no autenticados a una página Web especifica que les permite introducir sus credenciales (normalmente el nombre de usuario y la contraseña). A continuación, se validan las credenciales y se genera y devuelve al cliente un vale de autenticación. El vale de autenticación mantiene la identidad del usuario y, opcionalmente, una lista de funciones de las que es miembro el usuario durante toda la sesión del mismo.

La autenticación mediante Formularios se utiliza en ocasiones sólo para la personalización de sitios Web. En este caso, no se necesitará escribir demasiado código personalizado, puesto que ASP.NET realiza la mayor parte del proceso de forma automática con una configuración sencilla. En los escenarios de personalización, la cookie sólo necesita contener el nombre de usuario.

La autenticación mediante Formularios envía el nombre de usuario y la contraseña al servidor Web en texto sin formato. Por lo tanto, se deberá utilizar la autenticación mediante Formularios conjuntamente con un canal protegido con SSL. Para mantener la protección de la cookie de autenticación transmitida en peticiones posteriores, se deberá considerar la posibilidad de utilizar SSL en todas las páginas de la aplicación y no sólo la página de inicio de sesión.

Ninguna.

Indica que no desea autenticar usuarios o que utiliza un protocolo de autenticación personalizado.

Guardianes y puertas.

El término *guardián* se utiliza para identificar la tecnología de la que depende una puerta. Una *puerta a* representa un punto de control de acceso (que protege un recurso) de una aplicación. Por ejemplo, un recurso podría ser una operación (representada por un método de un objeto) o un recurso de base de datos o de sistema de archivos.

Tecnologías como las que se describen a continuación, proporcionan guardianes para la autorización de acceso. Las peticiones deben pasar por distintas puertas antes de

permitírseles el acceso a la operación o al recurso solicitado. A continuación, se describen las puertas por las que deben pasar las solicitudes.

- IIS proporciona una puerta cuando se autentican usuarios (es decir, cuando se deshabilita la autenticación anónima). Los permisos Web de IIS pueden utilizarse como mecanismo de control de acceso para restringir la capacidad de acceso de los usuarios Web a carpetas y archivos específicos. A diferencia de los permisos de archivos NTFS[38], los permisos Web se aplican a todos los usuarios Web, en lugar de a usuarios o grupos individuales. Los permisos de archivos NTFS ofrecen todavía más restricciones para recursos Web tales como las páginas Web, los archivos de imagen, etc. Estas restricciones se aplican a usuarios o grupos individuales.

  IIS comprueba primero los permisos Web y después los permisos de archivos NTFS. El usuario debe estar autorizado por ambos mecanismos para poder obtener acceso al archivo o carpeta. Si se produce un error en la comprobación de permisos Web, IIS devuelve una respuesta "HTTP 403 - Acceso prohibido", mientras que si se produce un error en la comprobación de permisos NTFS, IIS devuelve "HTTP 401 - Acceso denegado".

  Este guardián es responsable de las siguientes puertas:

  - Autenticación (anónima, básica, implícita, integrada, certificados).
  - Restricciones de direcciones IP y nombres de dominio (pueden utilizarse como línea de defensa adicional, pero no se debe depender de ellas debido a la facilidad con que se pueden falsificar las direcciones IP).
  - Permisos Web.
  - Permisos NTFS.

- ASP.NET proporciona varias puertas configurables y programables. Entre ellas figuran la autorización de direcciones URL, la autorización de archivos, las peticiones de permisos de principales y las funciones de .NET.

- ADO.NET utiliza cadenas de conexión. Las credenciales pueden ser explícitas o pueden utilizar autenticación de Windows.

- Los manejadores de Bases de datos generalmente incluyen varias puertas con inicios de sesión en el servidor, inicios de sesión en la base de datos y permisos de objetos de base de datos.

En resumen, los guardianes llevan a cabo la autorización en función de la identidad del usuario o del servicio que llama a la puerta e intenta obtener acceso a un recurso específico. La ventaja del uso de varias puertas reside en un mayor nivel de seguridad gracias a las múltiples líneas de defensa que proporciona.

Al usar las distintas puertas en todos los niveles de la aplicación, se pueden filtrar usuarios que tengan acceso a los recursos de servidor. El alcance del acceso se ve

[38] NTFS es un sistema de archivos que emplea el sistema operativo para almacenar y organizar los archivos. Su principal característica es permitir o bloquear el acceso a directorios y archivos.

reducido por la presencia de puertas sucesivas que se vuelven cada vez más granulares a medida que la petición pasa por la aplicación a los recursos de servidor.

Identidades y principales.

La seguridad de .NET se compone de capas superpuestas sobre la seguridad de Windows. El concepto de seguridad centrado en los usuarios en Windows, se basa en el contexto de seguridad proporcionado por una sesión de inicio, mientras que la seguridad de .NET se basa en objetos `IPrincipal` e `IIdentity`.

En la programación de Windows, cuando se desea conocer el contexto de seguridad en el que se ejecuta el código, se consulta la identidad del propietario del proceso o del subproceso en ejecución. Con la programación de .NET, si se desea consultar el contexto de seguridad del usuario actual, se tendrá que recuperar el objeto `IPrincipal`.

El Framework .NET utiliza objetos de identidad y de principal para representar a los usuarios mientras se ejecuta código de .NET; ambos tipos de objetos constituyen la espina dorsal de la autorización basada en funciones de .NET.

El Framework .NET incluye varias implementaciones concretas de `IPrincipal` e `IIdentity`, tal y como se muestra en la Figura 5.1.

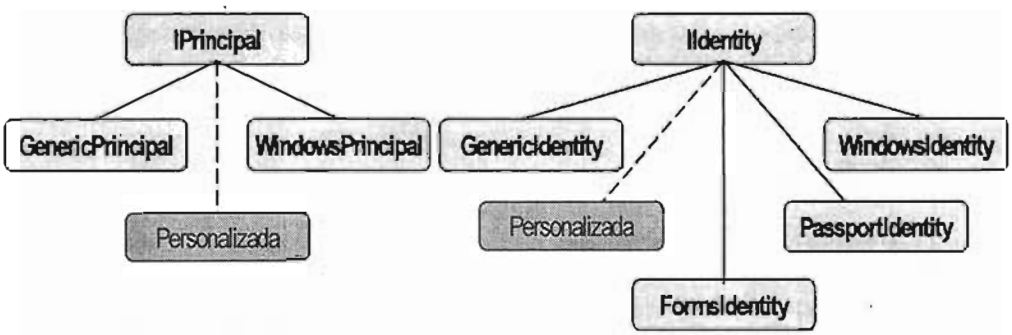

Figura 5.1 *Clases de implementación de IPrincipal e IIdentity.*

Los objetos de identidad y de principal implementan las interfaces `IIdentity` e `IPrincipal` respectivamente. Estas interfaces se definen en el espacio de nombres `System.Security.Principal`. Las interfaces comunes permiten al Framework .NET tratar a los objetos de identidad y de principal de modo polimórfico, independientemente de la información de implementación subyacente.

La interfaz `IPrincipal` permite probar la pertenencia a funciones mediante un método `IsInRole` y proporciona además acceso a un objeto `IIdentity` asociado

```
public interface IPrincipal
{
  bool IsInRole( string role );
  System.Security.Principal.IIdentity Identity {get;}
}
```

La interfaz `IIdentity` proporciona información adicional de autenticación, como el nombre y el tipo de autenticación.

```
public interface IIdentity
{
  string authenticationType {get;}
  bool IsAuthenticated {get;}
  string Name {get;}
}
```

WindowsPrincipal y *WindowsIdentity*.

La versión de .NET de un contexto de seguridad de Windows está dividida en dos clases:

- *WindowsPrincipal*. Esta clase almacena las funciones asociadas con el usuario actual de Windows. La implementación `WindowsPrincipal` trata los grupos de Windows como funciones. El método `IPrncipal.IsInRole` devuelve `true` o `false` en función de la pertenencia del usuario a grupos de Windows.

- *WindowsIdentity*. Esta clase almacena la parte de la identidad del contexto de seguridad del usuario actual y puede obtenerse mediante el método estático `WindowsIdentity.GetCurrent()`. Éste devuelve un objeto `WindowsIdentity` con una propiedad `Token` que a su vez devuelve, al testigo de acceso asociado con el subproceso en ejecución, un valor que representa un identificador de Windows.

  El método estático `WindowsIdentity.GetCurrent()` devuelve la identidad del subproceso en ejecución, que podría tratarse o no de una suplantación.

GenericPrincipal y objetos de identidad asociados.

Estas implementaciones son muy sencillas y las utilizan las aplicaciones que no usan la autenticación de Windows y cuando la aplicación no necesita representaciones complejas de un principal. Pueden crearse fácilmente mediante programación y, por lo tanto, deberá existir un cierto grado de confianza cuando una aplicación use un objeto `GenericPrincipal`.

Si se usa el método `IsInRole` de `GenericPrincipal` para tomar decisiones de autorización, se deberá confiar en la aplicación que envía el objeto `GenericPrincipal`. Esto contrasta con el uso de objetos `WindowsPrincipal`, con los que se debe confiar en que el sistema operativo proporcione un objeto `WindowsPrincipal` válido con una identidad autenticada y nombres de grupo o de función válidos.

Los tipos siguientes de objeto de identidad pueden asociarse a la clase `GenericPrincipal`:

- *FormsIdentity*. Esta clase representa a una identidad que se ha autenticado con la autenticación mediante Formularios. Contiene un `FormsAuthenticationTicket` que a su vez contiene información acerca de la sesión de autenticación del usuario.

- *PassportIdentity*. Esta clase representa a una identidad que se ha autenticado mediante la autenticación de Passport y que contiene información de perfil de Passport.

- *GenericIdentity*. Esta clase representa a un usuario lógico que no está asociado a ninguna tecnología de sistema operativo concreta y que se suele utilizar de forma conjunta con mecanismos de autenticación y autorización personalizados.

ASP.NET y `HttpContext.User`.

Normalmente, `Thread.CurrentPrincipal` se comprueba en el código de .NET antes de tomar cualquier decisión de autorización. No obstante, ASP.NET proporciona el contexto de seguridad del usuario autenticado mediante `HttpContext.User`.

Esta propiedad acepta y devuelve una interfaz `IPrincipal`. La propiedad contiene un usuario autenticado para la petición actual. ASP.NET recupera `HttpContext.User` cuando toma decisiones de autorización.

Al utilizar la autenticación de Windows, el módulo de autenticación de Windows construye de forma automática un objeto `WindowsPrincipal` y lo almacena en `HttpContext.User`. Si se utilizan otros mecanismos de autenticación, como Formularios o Passport, se deberá construir un objeto `GenericPrincipal` y almacenarlo en `HttpContext.User`.

Identidades de ASP.NET.

En cualquier punto de la ejecución de una aplicación Web ASP.NET, pueden estar presentes varias identidades durante una sola petición. Entre estas identidades, figuran:

- *`HttpContext.User`* devuelve un objeto `IPrincipal` que contiene información de seguridad para la petición Web actual. Se trata del cliente Web autenticado.

- *`WindowsIdentity.GetCurrent()`* devuelve la identidad del contexto de seguridad del subproceso de Win32 en ejecución. Esta identidad es siempre `ASPNET` de forma predeterminada, que es la cuenta predeterminada que se utiliza para ejecutar aplicaciones Web ASP.NET. No obstante, si la aplicación Web se ha configurado para la suplantación, la identidad representa al usuario autenticado (que es `IUSR_EQUIPO`, si está activada la autenticación anónima de IIS).

- *`Thread.CurrentPrincipal`* devuelve el principal del subproceso de .NET en ejecución que se superpone al subproceso de Win32.

Arquitectura de seguridad de ASP.NET.

ASP.NET actúa en combinación con IIS, el Framework .NET y los servicios de seguridad subyacentes proporcionados por el sistema operativo para ofrecer una serie de mecanismos de autenticación y autorización.

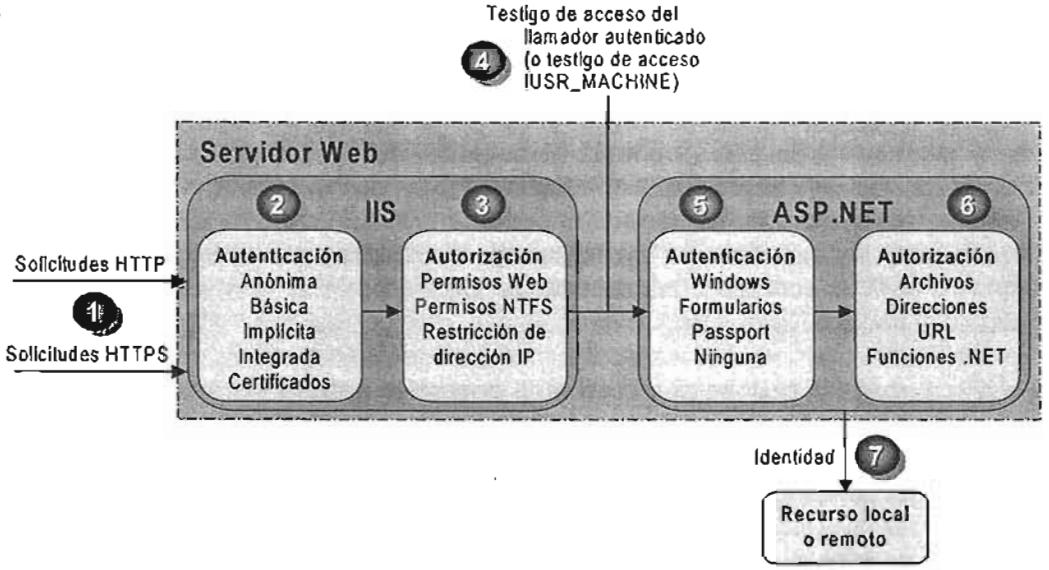

Figura 5.2 *Servicios de seguridad de ASP.NET.*

La Figura 5.2 muestra los mecanismos de autenticación y autorización que ofrecen IIS y ASP.NET. Cuando un cliente envía una solicitud Web, se produce la siguiente secuencia de eventos de autenticación y autorización:

1. Se recibe la solicitud Web HTTP(S) de la red. Puede utilizarse SSL para proteger la identidad del servidor (mediante certificados de servidor) y, a modo opcional, la identidad del cliente.

   SSL también proporciona un canal seguro para proteger los datos importantes que se transfieren del cliente al servidor (y viceversa).

2. IIS autentica al llamador mediante la autenticación básica, implícita, integrada (NTLM o Kerberos) o mediante certificados. Si el sitio, parcial o totalmente, no requiere acceso autenticado, IIS puede configurarse para la autenticación anónima. IIS crea un testigo de acceso a Windows para cada uno de los usuarios autenticados. Si se selecciona la autenticación anónima, IIS creará un testigo de acceso para la cuenta de usuario anónima de Internet (que, de manera predeterminada, es `IUSR_MACHINE`).

3. IIS autoriza al llamador para que obtenga acceso al recurso solicitado. Para autorizar el acceso se utilizan los permisos NTFS definidos por las listas ACL[39] adjuntas al recurso solicitado. IIS también puede configurarse para que acepte solicitudes únicamente de los equipos cliente con unas direcciones IP específicas.

4. IIS transfiere el testigo de acceso a Windows del llamador autenticado a ASP.NET (que puede ser el testigo de acceso del usuario de Internet anónimo si se utiliza la autenticación anónima).

5. ASP.NET autentica al llamador.

[39] Las listas ACL son recursos de información protegidos que especifican a quién se concede acceso a dichos recursos de forma precisa.

Si se configura ASP.NET para la autenticación de Windows, no se efectuará ningún otro tipo de autenticación adicional en esta fase. ASP.NET aceptará cualquier testigo que reciba de IIS.

Si se configura ASP.NET para la autenticación mediante Formularios, las credenciales que proporcione el llamador (mediante un Web Form) se autenticarán con un almacén de datos que suele ser una base de datos. Si ASP.NET se configura para la autenticación de Passport, el usuario será redirigido a un sitio Passport y será el servicio de autenticación de Passport el que se encargará de autenticar al usuario.

6. ASP.NET autoriza el acceso al recurso o la operación solicitados.

   `UrlAuthorizationModule` (un módulo HTTP proporcionado por el sistema) utiliza unas reglas de autorización configuradas en el archivo `Web.config` (en especial, en el elemento `<authorization>`) para garantizar que el llamador pueda obtener acceso al archivo o la carpeta solicitados.

   En el caso de la autenticación de Windows, `FileAuthorizationModule` (otro módulo HTTP) comprueba que el llamador disponga de los permisos necesarios para obtener acceso al recurso solicitado. El testigo de acceso del llamador se compara con la ACL de protección del recurso.

   Las funciones .NET también pueden utilizarse (mediante declaraciones o programación) para garantizar que el llamador reciba autorización para obtener acceso al recurso solicitado o para llevar a cabo la operación solicitada.

   El código de la aplicación obtiene acceso a los recursos locales o remotos mediante una determinada identidad. De forma predeterminada, ASP.NET no lleva a cabo ninguna suplantación, motivo por el que es la cuenta de proceso ASP.NET configurada la que proporciona dicha identidad.

Estrategias de autenticación y autorización.

ASP.NET proporciona una serie de mecanismos de autorización mediante declaraciones y programación que pueden utilizarse junto con una gran variedad de esquemas de autenticación. De este modo se puede desarrollar una estrategia de autorización exhaustiva y otra que pueda configurarse para ofrecer diferentes grados de granularidad: por ejemplo, por usuario o por grupo.

A continuación se resumen los esquemas de autenticación:

- Autenticación de Windows con suplantación.
- Autenticación de Windows sin suplantación.
- Autenticación de Windows con identidad fija.
- Autenticación mediante Formularios.
- Autenticación de Passport.

Autenticación de Windows con suplantación.

Los siguientes elementos de configuración muestran cómo habilitar la autenticación de Windows (IIS) y la suplantación mediante declaraciones en los archivos `Web.config` o `Machine.config`.

Es recomendable configurar la autenticación en función de cada aplicación en el archivo `Web.config` de las aplicaciones.

Con la siguiente configuración, el código de la aplicación ASP.NET suplanta al llamador autenticado por IIS.

```
<authentication mode="Windows" />
<identity impersonate="true" />
```

Seguridad configurable.

El empleo de la autenticación de Windows junto con la funcionalidad de suplantación ofrece las siguientes opciones de autorización:

- ***Listas ACL de Windows.***
- ***Recursos solicitados por los clientes.*** `FileAuthorizationModule` de ASP.NET realiza comprobaciones de acceso para los tipos de archivo solicitados que se hayan asignado a la ISAPI de ASP.NET. A fin de realizar dichas comprobaciones de acceso se utiliza el testigo de acceso del llamador original y la lista ACL adjuntos a los recursos solicitados.

  En el caso de los tipos de archivo estáticos (no asignados a ninguna extensión ISAPI), IIS realiza comprobaciones de acceso mediante el testigo de acceso del llamador y la lista ACL adjunta al archivo.

- ***Recursos a los que obtiene acceso la aplicación.*** Pueden configurarse las listas ACL de Windows de los recursos a los que obtenga acceso la aplicación (archivos, carpetas, claves de registro, etc.) en función del llamador original.

- ***Autorización mediante direcciones URL.*** La autorización mediante direcciones URL se configura en el archivo `Web.config`. Con la autenticación de Windows, los nombres de usuario adoptan la forma `NombreDominio\NombreUsuario` y las funciones se asignan una a una a los grupos de Windows.

  ```
  <authorization>
    <deny user="NombreDominio\NombreUsuario" />
    <allow roles="NombreDominio\GupoWindows" />
  </authorization>
  ```

- ***Funciones de Servicios Empresariales (COM+).*** Las funciones se almacenan en el catálogo COM+. Se pueden configurar estas funciones con ayuda de la herramienta de administración o la secuencia de comandos de Servicios de componentes.

Escenarios de uso.

Se recomienda utilizar la autenticación de Windows y la suplantación cuando se dan las siguientes condiciones:

- Los usuarios de la aplicación tienen cuentas de Windows que puede autenticar el servidor.
- Es necesario transferir el contexto de seguridad del llamador original al nivel medio o al nivel de datos de la aplicación Web para admitir la autorización de granularidad fina (por usuario).
- Se requiere transferir el contexto de seguridad del llamador original a los niveles indirectos a fin de admitir la auditoria de nivel del sistema operativo.

Entre las desventajas de la suplantación figuran:

- Escalabilidad reducida de la aplicación debido a la imposibilidad de agrupar efectivamente conexiones de bases de datos.
- Mayor esfuerzo de administración porque las listas ACL de los recursos servidor deben configurarse para cada uno de los usuarios.
- La delegación exige la autenticación Kerberos y un entorno configurado correctamente.

Autenticación de Windows sin suplantación.

Los siguientes elementos de configuración muestran cómo habilitar la autenticación de Windows (IIS) sin suplantación de forma declarativa en el archivo `web.config`.

```
<authentication mode="Windows" />
<!-- La siguiente declaración es equivalente a no tener el elemento identity -->
<identity impersonate="false" />
```

Seguridad configurable.

Tiene las mismas opciones de autorización que la autenticación de Windows con suplantación excepto la de Funciones de Servicios Empresariales (COM+).

Escenarios de uso.

Se recomienda utilizar la autenticación de Windows sin suplantación cuando se den las siguientes condiciones:

- Los usuarios de la aplicación tienen cuentas de Windows que puede autenticar el servidor.
- Se desea utilizar una identidad fija para obtener acceso a los recursos indirectos (bases de datos por ejemplo) a fin de admitir la agrupación de conexiones.

Autenticación de Windows con identidad fija.

Seguridad configurable.

El elemento `<identity>` del archivo `Web.config` es compatible con los atributos opcionales de nombre de usuario y contraseña, lo cual permite configurar una identidad fija específica para que la aplicación la suplante. El siguiente fragmento del archivo de configuración es una muestra de este enfoque.

```
<authentication mode="Windows" />
<identity impersonate="true" userName="NombreDominio\NombreUsuario"
                             password="PasswordEnTextoClaro" />
```

Escenarios de uso.

Este enfoque no es recomendable para la versión 1.0 del Framework .NET en entornos seguros por dos motivos:

- Los nombres de usuario y las contraseñas no deberían almacenarse en formato de texto sin cifrar en los archivos de configuración, en especial en los archivos de configuración que se almacenan en los directorios virtuales.

- En Windows 2000, este enfoque obliga a conceder a la cuenta de proceso ASP.NET el privilegio "Actuar como parte del sistema operativo". Esta concesión reduce la seguridad de la aplicación Web y aumenta el riesgo de la amenaza en caso de que un intruso pudiera llegar a exponer el proceso de la aplicación Web.

En el Framework .NET versión 1.1 se incluyen mejoras para este escenario en Windows 2000:

- Se cifran las credenciales.

- El inicio de sesión lo lleva a cabo el proceso IIS, de manera que ASP.NET no requiera el privilegio "Actuar como parte del sistema operativo".

Autenticación mediante Formularios.

Los siguientes elementos de configuración muestran cómo habilitar la autenticación mediante Formularios mediante declaraciones en el archivo `Web.config`.

```
<authentication mode="Forms" >
  <forms loginUrl="logon.aspx" name="CookieAutent" timeout="60" path="/">
  </forms>
</authentication>
```

Seguridad configurable.

El empleo de la autenticación mediante Formularios pone a disposición las siguientes opciones de autorización:

- ***Listas ACL de Windows.***

- ***Recursos solicitados por los clientes.*** Los recursos solicitados exigen que las listas ACL permitan el acceso de lectura a la cuenta anónima de usuario de

Internet (IIS deberá configurarse para permitir el acceso anónimo cuando se utiliza la autenticación mediante Formularios).

La autorización de archivos de ASP.NET no está disponible porque requiere la autenticación de Windows.

- ***Recursos a los que obtiene acceso la aplicación.*** Pueden configurarse las listas ACL de Windows de los recursos a los que obtenga acceso la aplicación (archivos, carpetas, claves de registro, etc.) con la identidad del proceso ASP.NET.

- ***Autorización mediante direcciones URL.*** Configurar la autorización mediante direcciones URL en el archivo `Web.config`. Al utilizar la autenticación mediante Formularios, el formato de los nombres de usuario viene determinado por el almacén de datos personalizado: un manejador de Base de datos.

  Si se utiliza un almacén de datos (manejador de Base de datos):

```
<authorization>
<deny users="?" />
  <allow users="Juan,Sara,Isabel" roles="Administrador,Vendedor" />
</authorization>
```

Escenarios de uso.

La autenticación mediante Formularios resulta específicamente idónea para las aplicaciones de Internet. Utilizar la autenticación mediante Formularios si:

- Los usuarios de la aplicación no tienen cuentas de Windows.

- Se desea que los usuarios inicien sesión en la aplicación escribiendo las credenciales en un formulario HTML.

Autenticación de Passport.

Seguridad configurable.

Los siguientes elementos de configuración muestran cómo habilitar la autenticación de Passport de forma declarativa en el archivo `Web.config`.

```
<authentication mode="Passport" />
```

Escenarios de uso.

La autenticación de Passport se utiliza en Internet cuando los usuarios de la aplicación no tienen cuentas de Windows y se desea implementar una solución de inicio de sesión único. Los usuarios que hayan iniciado una sesión previamente con una cuenta Passport en un sitio de Passport no tendrán que volver a iniciar sesión en su sitio si éste se ha configurado para la autenticación de Passport.

Configurar la seguridad.

A continuación se presentan los pasos que deben realizarse para configurar la seguridad de una aplicación Web de ASP.NET.

Configurar los parámetros de IIS.

Para configurar la seguridad de IIS se deben seguir los siguientes pasos:

1. Si se necesita, se puede instalar un certificado de servidor Web (en caso de utilizar SSL).

2. Configurar la autenticación de IIS.

3. Si se necesita, configurar la asignación de certificados de cliente (si se utiliza la autenticación mediante certificados).

4. Configurar los permisos NTFS para archivos y carpetas. Entre dichos recursos, IIS y `FileAuthorizationModule` de ASP.NET comprueban que el usuario autenticado (o la cuenta anónima de usuario de Internet) disponga de los derechos de acceso necesarios (según la configuración ACL) para obtener acceso al archivo solicitado.

Configurar los parámetros de ASP.NET.

Los parámetros de configuración del nivel de aplicación se almacenan en los archivos `Web.config`, ubicados en el directorio raíz virtual de la aplicación y, en ocasiones, también en subcarpetas adicionales (esta configuración puede suplantar algunas veces la configuración de la carpeta principal).

1. ***Configurar la autenticación.*** Debería configurarse en todos y cada uno de los archivos `Web.config` de las aplicaciones que se encuentran en el directorio raíz virtual de la aplicación (no en el archivo `Machine.config`).

```
<authentication mode="Windows|Forms|Passport|None" />
```

   - ***Configurar la suplantación.*** De manera predeterminada, las aplicaciones ASP.NET no aplican la suplantación. Las aplicaciones se ejecutan mediante la identidad del proceso configurada de ASP.NET (que suele ser `ASPNET`), utilizada para obtener acceso a los recursos.

     Para configurar la suplantación de ASP.NET, utilizar el siguiente elemento `<identity>` del archivo `Web.config` de la aplicación.

```
<identity impersonate="true" />
```

2. ***Configurar la autorización.*** La autorización mediante direcciones URL determinan si un usuario o función pueden emitir un verbo HTTP (por ejemplo, `GET`, y `POST`) para un archivo específico. Para implementar la autorización mediante direcciones URL es preciso que se lleven a cabo las siguientes tareas.

a) Agregar un elemento `<authorization>` al archivo `Web.config` ubicado en el directorio raíz virtual de la aplicación.

b) Restringir el acceso a los usuarios y las funciones mediante los atributos `allow` y `deny`. El siguiente ejemplo del archivo `Web.config` utiliza la autenticación de Windows y permite que tanto `Juan` como `Sara` obtengan acceso, pero se lo deniega al resto de los usuarios.

```
<authorization>
  <allow users=NombreDominio\Juan, NombreDominio\Sara" />
  <deny users="*" />
</authorization>
```

Importante: es preciso agregar `<deny users="?"/>` o `<deny users="*"/>` al final del elemento `<authorization>`; de lo contrario, se concederá el acceso a todas las identidades autenticadas.

Notas acerca de la autorización mediante direcciones URL.

Tener en cuenta los siguientes aspectos cuando se configure la autorización mediante direcciones URL:

- "*" hace referencia a todas las identidades.
- "?" hace referencia a las identidades no autenticadas (es decir, la identidad anónima).
- No es necesario aplicar la suplantación para que funcione la autorización mediante direcciones URL.
- La configuración de la autorización en el archivo `Web.config` suele hacer referencia a todos los archivos del directorio actual, así como a todos los subdirectorios (excepto que un subdirectorio contenga su propio archivo `Web.config` con un elemento `<authorization>`). En este caso, la configuración del subdirectorio sustituye la del directorio principal).

La autorización mediante direcciones URL sólo se aplica a los tipos de archivos a los que IIS asigna la extensión ISAPI de ASP.NET, `aspnet_isapi.dll`.

Se puede utilizar la etiqueta `<location>` para que la configuración de la autorización se aplique a un archivo o directorio específicos. El siguiente ejemplo muestra cómo aplicar la autorización a un archivo específico (`Pagina.aspx`).

```
<location path="pagina.aspx" />
  <authorization>
    <allow users=" NombreDominio\Juan, NombreDominio\Sara" />
   <deny users="*" />
  </authorization>
</location>
```

Los usuarios y las funciones de la autorización mediante direcciones URL se determinan mediante la configuración de la autenticación:

- Cuando se establece el elemento en `<authentication mode="Windows" />`, se autoriza el acceso para las cuentas de usuario y los grupos de Windows.
  Los nombres de usuario adoptan la forma "`NombreDominio\NombreUsuarioWindows`".

Los nombres de las funciones adoptan la forma "NombreDominio\NombreGrupoWindows".

- Cuando el elemento <authentication mode="Forms" /> se establece en la autenticación mediante Formularios, se autorizan los usuarios y las funciones para el objeto IPrincipal almacenado en el contexto HTTP actual. Por ejemplo, si se utilizan formularios para autenticar usuarios con una base de datos, se estará concediendo la autorización según las funciones obtenidas de la base de datos.

- Cuando el elemento del modo de autenticación se establece en Passport (<authentication mode="Passport" />), la autorización se efectúa con relación al identificador del usuario de Passport (PUID) o las funciones obtenidas de un almacén. Por ejemplo, se puede asignar un PUID a una cuenta y conjunto de funciones determinados almacenados en una base de datos.

- Cuando se establece el modo de autenticación en ninguno (<authentication mode="None" />), no se realiza ninguna autorización. "None" indica que no se desea realizar ninguna autenticación o que no se desea utilizar ninguno de los módulos de autenticación .NET sino un mecanismo personalizado propio.

  No obstante, si se utiliza la autenticación personalizada, se deberá crear un objeto IPrincipal con funciones y almacenarlo en HttpContext.User. Cuando, posteriormente, se lleve a cabo la autorización mediante direcciones URL, ésta se ejecutará con el usuario y las funciones (independientemente de cómo se obtengan) almacenadas en el objeto IPrincipal.

Nota: El elemento <authorization> funciona con el objeto IPrincipal actual almacenado en HttpContext.User y HttpContext.Request.RequestType.

Proteger la comunicación.

Muchas aplicaciones transmiten datos confidenciales en la red desde los usuarios y a los usuarios y entre nodos de aplicaciones intermedios. Entre los datos confidenciales, pueden figurar credenciales utilizadas para la autenticación o datos como números de tarjeta de crédito o detalles de transacciones bancarias. Para evitar la revelación indeseada de información y proteger los datos contra modificaciones no autorizadas durante la transmisión, deberá protegerse el canal entre los extremos de comunicación.

La comunicación segura ofrece las dos características siguientes:

- ***Privacidad.*** La privacidad se ocupa de garantizar que los datos permanezcan privados y confidenciales y no puedan verlos intrusos que utilicen software de supervisión de redes. La privacidad suele proporcionarse mediante el cifrado.

- ***Integridad.*** Los canales de comunicación segura también deben garantizar que los datos estén protegidos contra modificaciones accidentales o deliberadas (malintencionadas) durante la transmisión. La integridad suele proporcionarse mediante códigos de autenticación de mensajes (*Message Authentication Codes*, MAC).

SSL (Secure Sockets Layer).

SSL sirve para establecer un canal de comunicación cifrado entre el cliente y el servidor.

Suele utilizarse para proteger el canal entre un Web browser y el servidor Web. No obstante, también puede utilizarse para proteger mensajes de servicios Web y comunicaciones con un servidor de bases de datos (que lo soporte).

SSL asegura los datos a través de:

- ***Encriptación de datos.*** La información es encriptada usando un algoritmo y una llave de sesión. El servidor genera una llave pública que cualquier cliente puede usar. El cliente genera una llave de sesión y usa la llave pública para encriptarla antes de enviarla al servidor. Los datos son transferidos usando esta llave de sesión.

- ***Autenticación del servidor.*** Se asegura que los datos son enviados al servidor correcto y que el servidor es seguro. Para prevenir que un sitio Web suplante a otro, se usa SSL para autenticar el sitio Web. Cuando se instala SSL en el servidor Web, se debe instalar un certificado de servidor. Este certificado contiene información acerca de la organización, del sitio Web, y el emisor del certificado. Para trabajar como un identificador digital, un certificado debe ser firmado por una autoridad certificada. Una autoridad certificada actúa como una entidad de confianza que verifica la identidad de un sitio Web para sus usuarios.

- ***Integridad de datos.*** SSL se asegura de que los datos que fueron recibidos por el servidor no estén alterados de ninguna manera.

Al utilizar SSL, deberá tener en cuenta lo siguiente:

- Cuando se aplica SSL, el cliente utiliza el protocolo HTTPS (y especifica una dirección URL `https://`) y el servidor escucha en el puerto TCP 443.

- Se deberá supervisar el rendimiento de la aplicación al habilitar SSL.

- SSL utiliza funciones criptográficas complejas para cifrar y descifrar datos y, por lo tanto, afecta al rendimiento de la aplicación. La mayor disminución del rendimiento se produce durante el protocolo de enlace inicial, en el que se utiliza cifrado de claves públicas y privadas. Posteriormente (una vez generada e intercambiada una clave de sesión segura), se utiliza cifrado simétrico más rápido para cifrar los datos de la aplicación.

- Se deberá optimizar las páginas que utilizan SSL; para ello, incluir menos texto y usar gráficos más sencillos en las páginas.

- Puesto que el aumento del rendimiento asociado a SSL es mayor durante el establecimiento de la sesión, se deberá asegurar que no se agote el tiempo de espera de las conexiones.

- SSL requiere que se haya instalado un certificado de autenticación de servidor en el servidor Web (o en el servidor de bases de datos que lo requiera).

Conclusiones.

Las aplicaciones empresariales son la sangre vital que nutre a los negocios, poniendo en contacto a empleados para colaborar y compartir información y proporcionando los vínculos necesarios para establecer una comunicación efectiva con los clientes, socios y proveedores. Dependemos de estas aplicaciones y su tecnología para obtener un acceso fácil y rápido a la información necesaria con el fin de tomar decisiones informadas y mantener los niveles más altos de satisfacción del cliente.

La evolución constante de tecnologías para el desarrollo de aplicaciones empresariales ocasiona que los sistemas de cómputo vigentes se conviertan en algunos años obsoletos o incompletos. Solo hace algunos años, ASP (clásico) era una tecnología viable para crear aplicaciones Web. En la actualidad, ASP ha sufrido un cambio radical en su arquitectura y modelo de programación debido a la demanda de aplicaciones tipo Web de mayor complejidad, que solicitan mayores recursos de cómputo y mayor conocimiento en esta área. Habrá que considerar también, que ASP es una tecnología relativamente nueva.

Recurriendo a la experiencia que tuve al trabajar con la versión de ASP clásico, detecte los problemas que acarreaba su entorno de programación, principalmente la forma desorganizada del código que obligaba a utilizar técnicas de difícil mantenimiento con el fin de reutilizarlo, sin grandes logros. En aplicaciones considerablemente extensas que involucran equipos de trabajo para el desarrollo, resultaba poco eficiente y propenso a errores que hacían inestable la aplicación. A diferencia de lo que sucedía en aplicaciones de complejidad y concurrencia moderada, donde se podían obtener resultados mas o menos favorables. Lo cierto es que en su momento ASP clásico fue una excelente alternativa para desarrollar páginas Web dinámicas de manera rápida y sencilla. Sin embargo, las necesidades del negocio cambian y van en aumento para satisfacer a un número mayor de clientes, lo cual implica una evolución constante de las tecnologías, como en el caso de ASP.

Esta evolución se debe a gran medida al crecimiento de usuarios que utilizan las aplicaciones a través de dispositivos cada vez mas sofisticados que las tradicionales computadoras personales, lo cual exige una mayor complejidad en adaptar las aplicaciones para que funcionen de forma equivalente en múltiples dispositivos. La tecnología .NET se ajusta a las capacidades de ancho de banda y a las interfases del dispositivo que se esté utilizado.

Es por eso que se deberán visualizar las aplicaciones empresariales proyectadas a futuro, previniendo su capacidad de cubrir las necesidades presentes a corto, mediano y largo plazo; siempre en una constante actualización de conocimiento.

La tecnología .NET es un campo muy amplio de estudio, lo cual para desarrolladores de ASP clásico y Visual Basic (más acostumbrados a las aplicaciones de escritorio), no significará una transición sencilla. Aún mayor es el esfuerzo por aprender un lenguaje orientado a objetos y evitar las malas prácticas que proporcionaban estos entornos de programación. Sin embargo, será este el grupo de personas quienes notarán la mejora y nuevas capacidades del entorno de desarrollo. Más aún, su especialidad en la forma que se venían desarrollando aplicaciones ASP no se pierde del todo, y pueden ser de gran utilidad a la hora de migrar aplicaciones a la nueva plataforma.

La plataforma .NET hace posible que el desarrollador escriba código en el lenguaje de programación de su preferencia. Desafortunadamente podrá encontrar que el lenguaje en el cual programaba haya sufrido varias adaptaciones para trabajar en el entorno .NET. C# es el único que ha sido diseñado específicamente para ser utilizado en él, por lo que programar usando C# es mucho más sencillo e intuitivo que hacerlo con cualquier otro lenguaje. C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java, C y C++ para combinarlas en uno solo. Los desarrolladores habituados a estos lenguajes facilitarán la adopción de C#.

.NET incorpora estándares abiertos de la industria en su propia plataforma. Esto permite intercambiar información entre ambientes informáticos distintos a través de aplicaciones controladas por el usuario que cumplen con una especificación bien definida y disponible para la industria.

El cambio acelerado en la naturaleza de aplicaciones empresariales requiere rápido desarrollo, un mayor apego a desarrollo de componentes y reutilización, mantenimiento sencillo, y el apoyo de una plataforma más robusta. Por esto, será necesario identificar la arquitectura del sistema y aprovechar los servicios proporcionados por el Framework.

Una aplicación Web típica puede resumirse en su forma más simple como una interfaz de usuario puesta encima de un programa que utiliza la base de datos para leer y escribir información publicada a través del servidor Web con ciertos permisos y restricciones sobre lo que se puede hacer en la aplicación. ASP.NET proporciona las herramientas necesarias para esto, a través del Framework .NET. Tecnologías complementarias como ADO.NET se incluyen en el Framework.

ASP.NET proporciona una forma eficiente para desarrollo de la interfaz de usuario a través de sus controles de servidor dentro de Web Forms. Una característica importante de las aplicaciones desarrolladas en Visual Basic, es su facilidad para crear formularios utilizando controles predeterminados, los cuales responden a una serie de eventos fáciles de programar. Uno de los propósitos de los controles Web, es proporcionar esta misma flexibilidad de trabajo.

La evolución en las tecnologías de acceso a datos no es la excepción en el tiempo y esfuerzo que llevará al desarrollador aprender la tecnología ADO.NET, pero que está más apegada a los nuevos requerimientos que exigen las aplicaciones Web. Estas no requieren que se mantengan conexiones abiertas por tiempo indefinido, que pueden causar cuellos de botella saturando el número de conexiones abiertas que puede mantener el origen de datos y acabar rápidamente con los recursos del servidor. ADO.NET conserva los recursos del servidor reteniendo los datos en cachés locales de memoria, no hay necesidad de generar candados en la base de datos o mantener conexiones activas por periodos extendidos.

Una de las grandes preocupaciones de las empresas es la seguridad que proporcionan sus aplicaciones. Las aplicaciones Web distribuidas contienen muchas piezas móviles con varios niveles de exposición en un entorno desconectado por lo que habrán de establecerse políticas firmes y un modelo bien definido de seguridad. La seguridad atraviesa todas las capas de la aplicación, y existen bastantes tecnologías para implementarla. Sin embargo, es necesario identificar dónde y como es necesaria. Por demás, ASP.NET cuenta con las herramientas necesarias para implementar la seguridad.

Si bien es conocido que una gran cantidad de los ataques a los sistemas es por usuarios malintencionados dentro de la misma empresa; que aprovecha la debilidad de los sistemas en sus modelos de seguridad para beneficio personal, otros aprovechan los hoyos de seguridad de los productos de software, por lo que será necesario reforzar las políticas de actualización de software y una forma automatizada de hacerlo.

En lo personal considero que es necesario cubrir la demanda de conocimiento en tecnologías de cómputo actualizadas con una continua revisión a los planes de estudio de la carrera para ir a la par con las nuevas tendencias. Más aún, el número abrumador de tecnologías que aparecen en el mercado día con día; dificultan la curva de aprendizaje y la decisión a tomar si es necesario aprenderlas. Debido a esto, el estudiante de la Licenciatura de Matemáticas Aplicadas y Computación deberá mantenerse al tanto de las tendencias tecnológicas actuales referentes a su campo de trabajo, y elegir una especialidad afín a su visión como profesionista.

Habrá que mitigar el atraso tecnológico en nuestro país evaluando nuevas tecnologías desde antes de su liberación final, analizando las tendencias de la industria, y eliminando la barrera del idioma; que retrasa aún más su aprendizaje al esperar traducciones al español para su entendimiento, mayormente si se tiene que familiarizar con nueva terminología.

Espero que el presente trabajo sea de gran utilidad para el lector interesado en el estudio del conjunto de tecnologías bajo el nombre de .NET. Este cubre solo una parte importante de un extenso número de tecnologías que la componen, lo cual puede inducir al lector a extender su investigación consultando otras fuentes.

Índice alfabético.

Bibliografía.

- Anderson, Richard, Homer, Alex y Howard, Rob, et al. *A preview of Active Server Pages+.* 1ra. ed. Birmingham (UK): Wrox Press, 2000.
- Papa, John. *Professional ADO 2.5 RDS Programming with ASP 3.0.* 1ra. ed. Birmingham (UK): Wrox Press, 2000.
- McDonald, Matthew. *ASP.NET Manual de referencia.* 1ra. ed. Madrid (España): McGraw-Hill/Interamericana de España, 2002.
- *MCAD/MCSD: Developing Web Applications with Microsoft Visual Basic .NET and Microsoft Visual C# .NET.* Lugar de publicación desconocido: Microsoft Corporation, 2002.
- Dickinson, Paul y Skinner, Julian. *Professional ADO.NET Programming.* 1ra. ed. Birmingham (UK): Wrox Press, 2002.
- Walther, Stephen. *ASP.NET Unleashed.* 2da.ed. Lugar de publicación desconocido: Sams Publishing, 2003.
- *Programming with the Microsoft .Net Framework.* Lugar de publicación desconocido: Microsoft Corporation, 2002.
- *Programming with Microsoft ADO.NET.* Lugar de publicación desconocido: Microsoft Corporation, 2002.
- *Developing Microsoft ASP.NET Web Applications Using Visual Studio .NET.* Lugar de publicación desconocido: Microsoft Corporation, 2002.
- *Programming with C#.* Lugar de publicación desconocido: Microsoft Corporation, 2002.
- *Building Secure ASP.NET Applications.* Lugar de publicación desconocido: Microsoft Corporation, 2002.
- Appleman, Dan. *Desarrollo de componentes COM/ActiveX con Visual Basic 6.*1ra. ed. Madrid (España): Prentice Hall, 2000.

Documentos electrónicos.

1. Ernst, Warren. *Introducción a ActiveX.* Distrito Federal (México): Prentice Hall Hispanoamericana, 1997, p.260.

2. Sun Microsystems, Inc. [en línea]: iPlanet Application Server Programmer's Guide (Java). 2000 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://docs.sun.com/source/816-5725-10/jpggloss.htm>.

3. Nota del autor.

4. TechTarget. [en línea]: Transaction Server. 1999 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://whatis.techtarget.com/gDefinition/0,294236,sid26_gci213214,00.html>

5. Ernst, Warren. *Introducción a ActiveX.* Distrito Federal (México): Prentice Hall Hispanoamericana, 1997, p.263.

6. Appleman, Dan. *Desarrollo de componentes COM/ActiveX con Visual Basic 6.*1ra. ed. Madrid (España): Prentice Hall, 2000, p. 57.

7. Microsoft Corporation [en línea]: COM: Component Object Model Technologies. 2005 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.microsoft.com/com/default.mspx>

8. Microsoft Corporation [en línea]: Diseñar aplicaciones distribuidas con Visual Studio .NET. 2005 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://msdn.microsoft.com/library/SPA/vsent7/html/vxconPerformance.asp>

9. Universidad Católica Asunción Paraguay (Departamento de Electrónica e Informática) [en línea]: ¿Qué es WML? 2004 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.dei.uc.edu.py/tai2001/wap/info7.htm>

10. Universidad Nacional Autónoma de México (Dirección General de Servicios de Cómputo Académico) [en línea]: ¿Qué es XML? 2004 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://manuales.dgsca.unam.mx/xml/qu%E9_es.html>

11. World Wide Web Consortium [en línea]: Simple Object Access Protocol (SOAP) 1.1. 2000 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>

12. Editorial UOC [en línea]: 'La red cambia las reglas': capítulo 5 del libro Infonomia!com. 2002 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.uoc.edu/web/esp/art/uoc/cornella0402/cornella0402.html>

13. Lucasian Labs [en línea]: Introducción a la Arquitectura de Software. 2003 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.lucasian.com/edocs/LucasianLabs.Arquitectura.Software.esp.ver0.5.4.pdf>

14. Microsoft Corporation [en línea]: Diseñar aplicaciones distribuidas con Visual Studio .NET. 2005 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://msdn.microsoft.com/library/SPA/vsent7/html/vxconreliabilityoverview.asp>

15. Microsoft Corporation [en línea]: Diseñar aplicaciones distribuidas con Visual Studio .NET. 2005 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://msdn.microsoft.com/library/SPA/vsent7/html/vxoriAvailability.asp>

16. Nota del autor.

17. Wikipedia, the free encyclopedia [en línea]: Microsoft Foundation Classes. 2005 [fecha de consulta: 8 Mayo 2005]. Disponible en:
<http://en.wikipedia.org/wiki/Microsoft_Foundation_Classes>

18. World Wide Web Consortium [en línea]: XSL Transformations (XSLT). 1999 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.w3.org/TR/xslt>

19. Nicatechnologies [en línea]: Unicode. 2005 [fecha de consulta: 8 Mayo 2005]. Disponible en:
<http://www.nicatech.com.ni/u.htm>

20. Aquino, Nathalie, Frutos, Carlos. *Fundamentos de la máquina virtual Java y el entorno .NET*. Asunción (Paraguay): Universidad Católica Nuestra Señora de la Asunción, Facultad de Ciencias y Tecnología, 2002, p.16.
<http://www.jeuazarru.com/docs/Java_y_PuntoNET.pdf>

21. Nota del autor.

22. Appleman, Dan. *Desarrollo de componentes COM/ActiveX con Visual Basic 6.*1ra. ed. Madrid (España): Prentice Hall, 2000, p. 137,

23. Nota del autor.

24. Appleman, Dan. *Desarrollo de componentes COM/ActiveX con Visual Basic 6.*1ra. ed. Madrid (España): Prentice Hall, 2000, p. 57.

25. Learn the Net [en línea]: Ancho de banda. 2005 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.learnthenet.com/spanish/glossary/bandwth.htm>

26. HTML.Dyn@mico [en línea]: ¿Que es DHTML?. 2003 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://htmldyn.iwebland.com/ayuda.htm>

27. Nota del autor.

28. Nota del autor.

29. Instituto Tecnológico de la Paz [en línea]: Llamadas a procedimientos remotos (RPC). 1998. [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.itlp.edu.mx/publica/tutoriales/sistsdist2/t65.htm>

30. Network Working Group [en línea]: RFC 2828 - Internet Security Glossary. 2000. [fecha de consulta: 6 Mayo 2005]. Disponible en:
< http://www.faqs.org/rfcs/rfc2828.html>

31. World Wide Web Consortium [en línea]: XML Schema. 2001 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.w3.org/XML/Schema>

32. SearchWebServices.com [en línea]: Intranet. 2001 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci212377,00.html>

33. SearchSecurity.com [en línea]: Extranet. 2004 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://searchsecurity.techtarget.com/sDefinition/0,290660,sid14_gci212089,00.html>

34. Infonos [en línea]: ¿Qué es el B2C? 2000 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.infonos.com/faq/3.8.html>

35. Microsoft Corporation [en línea]: Crear aplicaciones ASP.NET seguras. 2004 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.microsoft.com/spanish/msdn/arquitectura/BuildSecNetApps/html/Appendix_06.asp>

36. Microsoft Corporation [en línea]: Crear aplicaciones ASP.NET seguras. 2004 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.microsoft.com/spanish/msdn/arquitectura/BuildSecNetApps/html/Appendix_06.asp>

37. Microsoft Corporation [en línea]: Crear aplicaciones ASP.NET seguras. 2004 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.microsoft.com/spanish/msdn/arquitectura/BuildSecNetApps/html/Appendix_06.asp>

38. Monografías.com [en línea]: Intoducción al Windows XP. 1997 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.monografias.com/trabajos12/algodwxp/algodwxp.shtml#NTFS>

39. Tecnológico de Monterrey [en línea]: Guía del usuario del sistema: Sistema operativo y dispositivos. 1997 [fecha de consulta: 6 Mayo 2005]. Disponible en:
<http://www.mor.itesm.mx/AIX/es_ES/a_doc_lib/aixuser/usrosdev/access_control_list.htm>