CAPITULO II

Plataforma de Desarollo .Net Framework

CONTENIDO

2.1 Definición de COM+
2.2 Definición de .Net Framework
2.3 Net Framework y COM+
2.4 El Lenguaje Universal CLR
2.5 Lenguaje Intermedio de Microsoft (MSIL)
2.6 Librería de Classes Base (BCL)
2.7 Páginas dinámicas ASP frente a ASPX.Net
2.8 ADO frente ADO.Net

2.1 Definición de COM+

COM versión que combina las características basadas en servicios del Servidor de Transacciones Microsoft (**MTS** Microsoft Transaction Server), y Modelo de Objeto de Componentes Distribuido (DCOM). **COM+** proporciona administración de procesos, servicios de sincronización, modelo de transacción declarativo y agrupación de conexiones a objetos y bases de datos. Los servicios de COM+ están principalmente orientados hacia el desarrollo de la aplicación de capa intermedia y se enfoca en proporcionar confiabilidad y escalabilidad para aplicaciones distribuidas a gran escala. [WWW005]

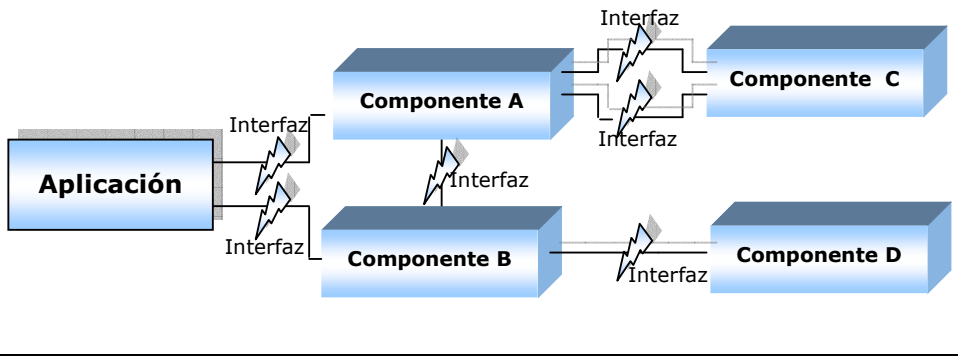

Figura 2.1 ***Entorno de componentes***

2.2 Definición .Net Framework

El .Net Framework es un entorno de desarrollo para construir, instalar y ejecutar servicios Web y otras aplicaciones. Se compone de tres partes principales: **CLR** (Common Language Runtime) que es la rutina universal, una librería de clases unificada llamada **BCL** (Librería de Clase Base), **ASP.NET** con formularios Web, servicios Web **XML** y un subsistema de acceso a datos (**ADO .NET**).

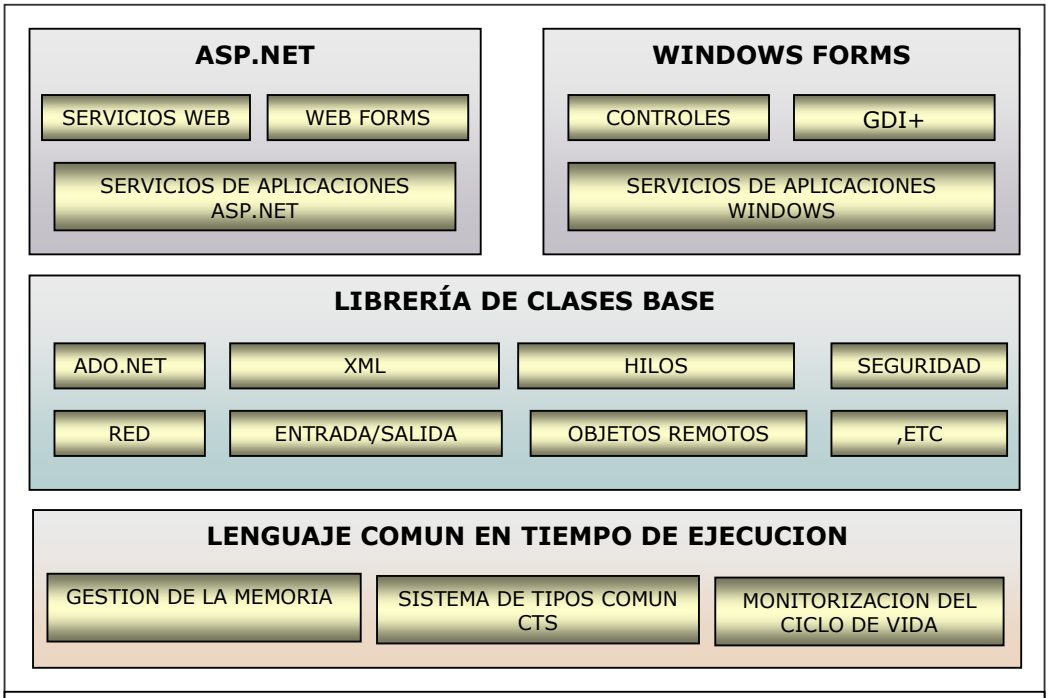

Figura 2.2 ***Esquema del Framework.Net***

2.3. Net Framework y COM+

La diferencia primordial entre el modelo COM y el modelo de .Net esta en que .net maneja un ***código administrado*** o gestionado. Mientras que COM maneja código no administrado que es un entorno no controlado, como programas en C/C++.

> Código Administrado es código en un entorno controlado, es por tanto un código que no puede causar daños.

Los modelos de programación administrados y no administrados son diferentes en varios aspectos. La siguiente tabla muestra las características que definen a cada modelo.

Función	**COM+**	**NETFRAMEWORK**
	Modelo no administrado	**Modelo administrado**
Modelo de codificación	Basado en una interfaz	Basado en objetos
Identidad	**GUIDs**	Nombre fijos
Mecanismo de manejo de errores	**HRESULT**s	Excepciones
Compatibilidad de tipo	Estándar binario	Estándar de tipo
Definición de tipo	Biblioteca de tipo	Metadatos
Versiones	Inmutable	Flexible
Tiempo de vida del objeto	Recolección de basura	Conteo de referencia

Tabla 2.1. ***Comparativa Modelo Administrado (.Netframework) vs Modelo no Administrado (COM).***

Modelos de codificación: Los objetos no administrados se comunican a través de interfaces, mientras que los objetos y las clases administradas pueden transmitir datos directamente, .NET proporciona herencia de implementación y herencia de interfaz para que pueda heredar una clase desde un lenguaje diferente.

Identidades: Los GUIDs identifican un tipo específico no administrado y no proporcionan información de ubicación sobre ese tipo. Los nombres fijos consisten en un nombre de ensamble único. Los tipos administrados se identifican a través de una combinación de su nombre de tipo y del ensamble del cual son parte.

Mecanismos de manejo de errores: Los métodos COM normalmente regresan un HRESULT, indicando que la llamada tuvo éxito o que falló. El código administrado utiliza ***EXCEPCIONES*** para transmitir las condiciones de error.

Compatibilidad de tipo: COM se basa en un estándar binario; .NET en un estándar de tipo que permite la integración entre lenguajes, seguridad de tipo y ejecución de código de alto rendimiento.

Definiciones de tipo: COM utiliza **BIBLIOTECAS DE TIPO** para almacenar información de tipo. Una biblioteca de tipo contiene tipos públicos y es opcional. En .Net, la información de tipo, conocida como **METADATOS**, y es obligatoria para todos los tipos. Los servicios de interoperabilidad proporcionan herramientas que convierten las bibliotecas de tipo en metadatos dentro de ensambles y viceversa.

Versiones: Las interfaces COM son inalterables. Si cambia una interfaz, debe renombrarla con un nuevo **GUID**. Los tipos administrados pueden evolucionar, manteniendo el mismo nombre, y a la vez crear tipos administrados que sean compatibles con versiones anteriores de ese tipo.

Tiempo de vida del objeto: Los clientes de los objetos COM administran el tiempo de vida del objeto a través de un conteo de referencia. En .NET, el CLR administra el tiempo de vida de los objetos a través de la recolección de basura.

Microsoft .NET Framework proporciona servicios de interoperabilidad con COM, para aprovechar las aplicaciones existentes, facilitar la migración y simplificar la interacción entre código administrado de .Net y no administrado de COM.

La interoperabilidad COM es un servicio bidireccional entre .NET Framework y COM, y viceversa, lo cual permite a los clientes .NET llamar a componentes COM y a los clientes COM llamar a componentes .NET Framework. Por lo tanto se generan dos modelos [WWW005]:

a). El modelo de interoperación .NET a COM

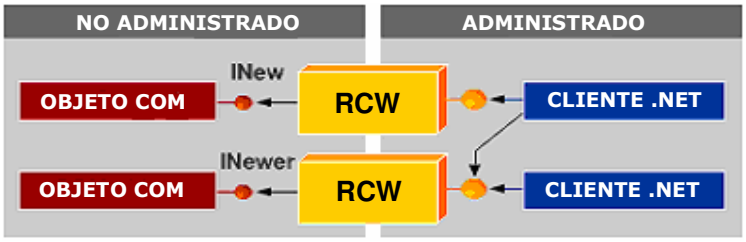

Figura 2.3 ***Modelo de interoperación .Net a COM***

Cuando un cliente .NET invoca un objeto COM, se crea una envoltura invocable de tiempo de ejecución (**RCW**) para cada objeto sin importar el número de referencias que existan en dicho objeto, de manera que los clientes .Net creen que están llamando un código .Net .RCW actúa como un proxy entre el código administrado y no administrado cuya función principal es ordenar llamadas entre un cliente .NET y un objeto COM. Cualquier número de clientes administrados pueden mantener una referencia con los objetos COM que exponen las interfaces INew y INewer.

Cada RCW mantiene una memoria caché de punteros de interfaz en el objeto COM que envuelve y libera su referencia en el objeto COM cuando ya no se necesita la RCW. Después de su liberación se realiza una recolección de basura en la RCW para nuevamente asignar memoria.

b). El modelo de interoperación COM a .NET

Cuando un cliente COM invoca un objeto .NET debe agregar la definición de clase .Net al registro del sistema de COM, el CLR crea el objeto administrado y una envoltura COM (**CCW**) sin importar el número de

clientes COM que soliciten sus servicios. Al no poder hacer referencia a un objeto .NET directamente, los clientes COM usan el CCW como un proxy para el objeto administrado y pueden mantener una referencia para la CCW que expone la interfaz INew.

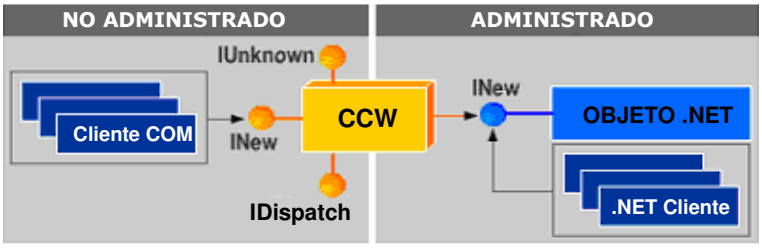

Figura 2.4 ***Modelo de interoperación COM a .NET***

El tiempo de vida de un objeto CCW, como cualquier otro objeto COM, es controlado por conteo de referencia. Cuando el conteo de referencia en la CCW llega a cero, la envoltura suelta su referencia sobre el objeto administrado. Durante el siguiente ciclo de recolección de basura se recoge un objeto administrado sin referencias restantes.

La transición entre un código administrado y no administrado tiene sus problemas en cuanto al desempeño por lo que hay que intentar reducir estas transiciones en la medida posible.

2.4 El Lenguaje Universal CLR (Common Language Runtime)

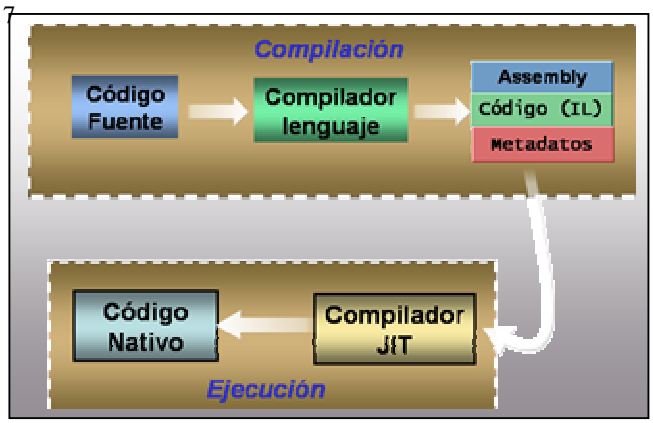

Figura 2.5 ***Common Language Runtime, proceso del código gestionado***

El lenguaje común en tiempo de ejecución, o **CLR**, es el motor de ejecución para las aplicaciones del Framework.Net. El CLR es el núcleo del Framework que desempeña el papel de una máquina virtual. Se encarga de gestionar la ejecución del código y de proporcionar servicios a dicho código.

En tiempo de ejecución, CLR es responsable de *administrar* el entorno de ejecución del código .NET, así como la asignación de memoria y administración de hilos de ejecución, reforzando las políticas de seguridad. Al automatizar estas tareas, el tiempo y proceso de desarrollo se reduce y simplifica. Por ejemplo, la recolección automatizada de memoria-basura permite las no filtraciones de memoria innecesarias

Los componentes principales de CLR se describen a continuación: [WWW001]

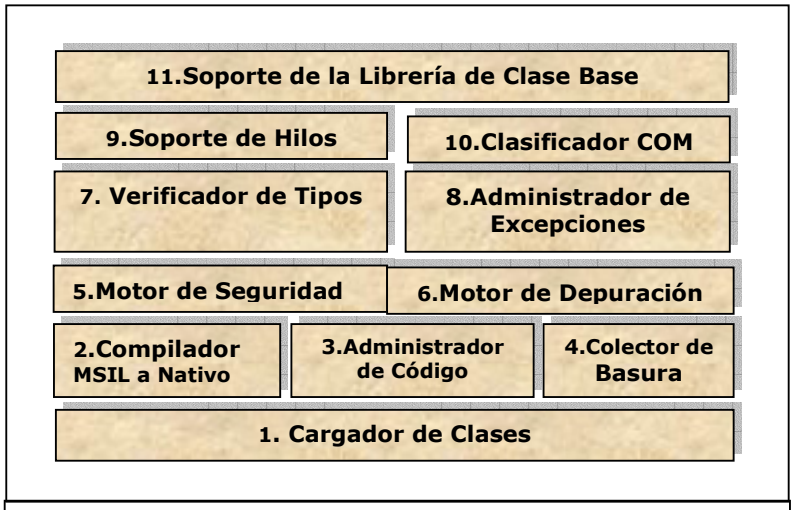

Figura 2.6 ***Componentes de la Rutina de Clases CLR***

Componente	Descripción
1. Cargador de clases	Administra metadatos, carga y diseño de las clases.
2. Microsoft Intermediate Language (MSIL) para compiladores nativos	Convierte MSIL a código nativo (Justo a tiempo (**JIT**) y Generación nativa (**NGEN**)).
3. Administrador de código	Administra la ejecución del código.
4. Recolector de basura (GC)	Proporciona administración automática del tiempo de vida de todos sus objetos. Es multiejecución y escalable.
5. Motor de seguridad	Proporciona seguridad basada en las evidencias, según el origen del código además de la identidad del código de invocación.
6. Motor de depuración	Permite depurar la aplicación y rastrear la ejecución del código.
7. Verificador de tipo	No permite transmisiones inseguras o variables no inicializadas. Se puede verificar el MSIL para garantizar la seguridad del tipo.
8. Administrador de excepciones	Proporciona manejo estructurado, donde se ha mejorado el reporte de errores.
9. Soporte para hilos	Proporciona clases e interfaces que permiten la programación **MULTIHILO**.
10. Clasificador COM	Proporciona organización desde y hacia COM.
11. Soporte para la Biblioteca de clase de .NET Framework	Integra el código con CLR que soporta la Biblioteca de clases de .NET Framework.

Tabla 2.2 ***Descripción de los componentes del CLR***

2.4.1 Características del CLR:

- ***Diseño completamente orientado a objetos*:** no existen funciones sueltas o variables.

- ***Eliminación del "conflicto de las DLLs":*** En la plataforma .NET las versiones nuevas de las DLLs pueden coexistir con las viejas, de modo que las aplicaciones diseñadas para ejecutarse usando las viejas podrán seguir usándolas tras instalación de las nuevas. Simplificando la instalación y desinstalación de software.

- ***Ejecución multiplataforma:*** para todos quienes sean compatibles con el CLR

- ***Integración de lenguajes:*** Desde cualquier lenguaje permite utilizar herencia de clases escritas en distintos lenguajes.

- ***Administración de memoria mediante un garbage collector:* (recolector de basura)** inteligente que evita que el programador tenga que tener en cuenta cuándo ha de destruir los objetos que dejen de serle útiles.

- ***Seguridad de tipos:*** El CLR facilita la detección de errores de programación difíciles de localizar comprobando antes y durante la ejecución del programa.

- ***Aislamiento de procesos:*** Desde código perteneciente a un determinado proceso no se puede acceder a código o datos pertenecientes a otro, evitando errores de programación muy frecuentes que impide que unos procesos puedan atacar a otros.

- ***Tratamiento de excepciones:*** Todos los errores que se puedan producir durante la ejecución de una aplicación se

propagan de igual manera. Esto es muy diferente a los sistemas Windows, donde ciertos errores se transmitían mediante códigos de error en formato Win32, o mediante HRESULTs.

- ***Soporte multihilo:*** trabaja 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.

- ***Distribución transparente:*** Crea objetos remotos y accede a ellos de manera transparente a su localización real, tal y como si se encontrasen en la máquina que los utiliza.

- ***Interoperabilidad con código antiguo*:** Puede acceder desde código escrito para la plataforma .NET a código escrito en versiones anteriores.

Con el objetivo de solucionar los problemas de las versiones, así como los problemas que desembocan en conflictos de DLL, el motor de tiempo de ejecución utiliza **ENSAMBLADOS*.***

Los Ensablados	Son la unidad fundamental de despliegue, control de versiones, reutilización, y permisos de seguridad de una aplicación.

Un assembly es una colección de tipos y de recursos integrados y son la unidad lógica de funcionalidad que proporciona la información que el CLR necesita sobre las implementaciones de dichos tipos. Sus objetivos son las siguientes:

- Permitir a los programadores especificar reglas de versiones entre distintos componentes de software.

- Proporcionar la infraestructura para que se cumplan las reglas de versiones.
- Proporcionar la infraestructura que permita que varias versiones de un componente se puedan ejecutar simultáneamente, lo que se conoce como ejecución simultánea.

2.4.2 Estructura de un ensamblado

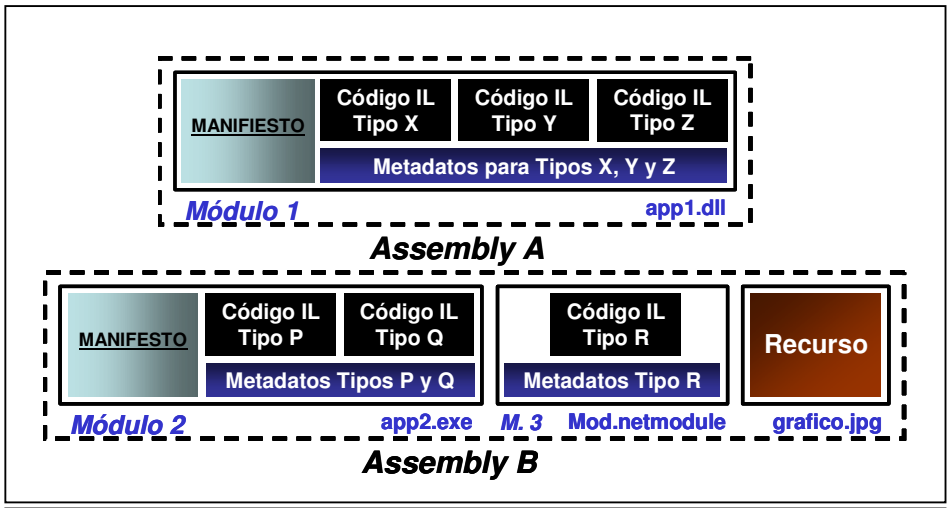

Figura 2.7 ***Estructura de un ensamblado***

En general, la estructura lógica de un ensamblado consta de cuatro elementos:

- El ***manifiesto del ensamblado***, que contiene los metadatos del ensamblado.
- Los ***metadatos*** que describen los tipos del ensamblado.
- ***El código en lenguaje intermedio (MSIL)*** que implementa los tipos.
- Un ***conjunto de recursos***.

Todos los ensamblados contienen un **MANIFIESTO** del ensamblado. En la tabla siguiente, se muestra la información que contiene el manifiesto del ensamblado. Los cuatro elementos (nombre del ensamblado, número de versión, referencia cultural e información sobre el nombre seguro) constituyen la identidad del ensamblado. [WWW001]

Información	Descripción
Nombre del ensamblado	Cadena de texto donde se especifica el nombre del ensamblado.
Número de versión	Número de versión principal y secundaria, y número de revisión y de versión de compilación.
Referencia cultural	Información sobre la referencia del ensamblado. Se debe utilizar sólo para designar un ensamblado como ensamblado satélite que contienen referencias a objetos globales y privados
Información sobre el nombre seguro	Clave pública del editor si el ensamblado tiene un nombre seguro.
Lista de todos los archivos del ensamblado	Un código en cada archivo que contiene el ensamblado y nombre de archivo. Todos los archivos que componen el ensamblado deben encontrarse en el mismo directorio que contiene el manifiesto de ensamblado.
Información de referencia de tipos	Información que utiliza el motor de tiempo de ejecución para asignar una referencia de tipos al archivo que contiene su declaración e implementación. Se utiliza para tipos que se exportan desde el ensamblado.
Información sobre ensamblados a los que se hace referencia	Lista de otros ensamblados a los que hace referencia estáticamente. Cada referencia tiene el nombre del ensamblado dependiente, los metadatos del ensamblado (versión, referencia cultural, sistema operativo, etc.) y la clave pública, si el ensamblado tiene un nombre seguro.

Tabla2.3 ***Descripción del manifiesto del ensamblado***

Los ensamblados reducen los conflictos de los archivos DLL y hacen que las aplicaciones sean más seguras y más fáciles de implementar. En muchos casos es posible instalar una aplicación basada en .NET simplemente copiando los archivos de la aplicación en el equipo de destino.

Clasificación de los ensamblados

Los ensamblados se pueden crear de dos tipos:

Ensamblados estáticos	Incluyen tipos de .NET Framework como (interfaces y clases), y recursos como (mapas de bits, archivos JPEG, archivos de recursos, etc.). Los ensamblados estáticos se almacenan en el disco, en archivos ejecutables portables PE.
Ensamblados dinámicos	Se ejecutan directamente desde la memoria y no se guardan en el disco antes de su ejecución. Los ensamblados dinámicos se pueden guardar en el disco una vez que se hayan ejecutado.

Tabla2.4 ***Descripción de Ensamblado Estático y Dinámico***

Dentro de este marco, cada uno de ellos puede ser:

Ensamblados Privados	Son utilizados solo por una única aplicación sin afectar al resto de aplicaciones.
Ensamblados Públicos	Residen en un único sitio físico para servir a múltiples aplicaciones.

Tabla2.5 ***Descripción de Ensamblado Privado y Ensamblado Público***

***Ensamblados privados*:** son utilizados únicamente por una aplicación, sin afectar a las demás aplicaciones instaladas en el sistema. Eliminando el problema de las DLL ya que una aplicación cuenta con su propio grupo de ensamblados. Es decir, una versión de biblioteca que utiliza las mismas propiedades y métodos podrá reemplazar a otra, copiando y describiendo el archivo por el nuevo integrante. Uno de los mayores beneficios del ensamblado es encapsular código a efecto de que el mismo pueda ser compartido por varias aplicaciones.

Los ensamblados compartidos: también brindan la característica de los ensamblados privados con respecto a la facilidad de instalación y al igual que el modelo de los componentes COM ofrecen la posibilidad de residir en único sitio físico para servir a múltiples aplicaciones. No dependen ni mantienen vinculación alguna con el archivo de registro. Pero los ensamblados compartidos ofrecen las siguientes ventajas frente a los privados:

- ***Mejor rendimiento:*** si un ensamblado ya ha sido cargado en memoria por una aplicación y otra requiere del mismo, no se verificara nuevamente su integridad.
- ***Menor Consumo de Memoria:*** el sistema operativo utilizará una única instancia en memoria para servir a todas las aplicaciones que empleen el mismo ensamblado.
- ***Facilidad de Actualización:*** visto que como el ensamblado reside en un único sitio, es más sencilla su actualización.

Sin embargo al no contar con el archivo de registro como repositorio centralizado de información. La plataforma .net cuenta con otra nueva característica llamada **Caché de Ensamblados Global** (**GAC** Global Assembly Caché) la cual es un sitio especial donde un ensamblado puede residir para que pueda ser ubicado desde todas las aplicaciones del sistema (aunque no es indispensable).[WWW006]

Las principales ventajas del almacén son:

1. ***Versiones:*** Puede albergar varias versiones de una misma biblioteca en la caché, lo que era imposible en el modelo COM.

2. ***Chequeo de integridad:*** Cuando un ensamblado es agregado a la caché, un chequeo de integridad es realizado sobre todos los archivos contenidos por

este, para asegurar que el mismo pueda ser ejecutado.

3. ***Seguridad de Archivos:*** Solo aquellos usuarios con permiso de administrador podrán eliminar archivos de caché, los instaladores deberán ser siempre ejecutados con permisos de administrador. Eliminando la posibilidad de que un usuario final pueda borrar ensamblados de una aplicación.

4. ***Múltiples nombres:*** pueden existir varios ensamblados con igual nombre físico (de archivo) sin que ello afecten a quienes lo utilizan.

Los ensamblados son unidades autodescriptivas o autocontenidas ejecutables que no dependen del archivo de registro, ni necesitan de este para su correcta ejecución.

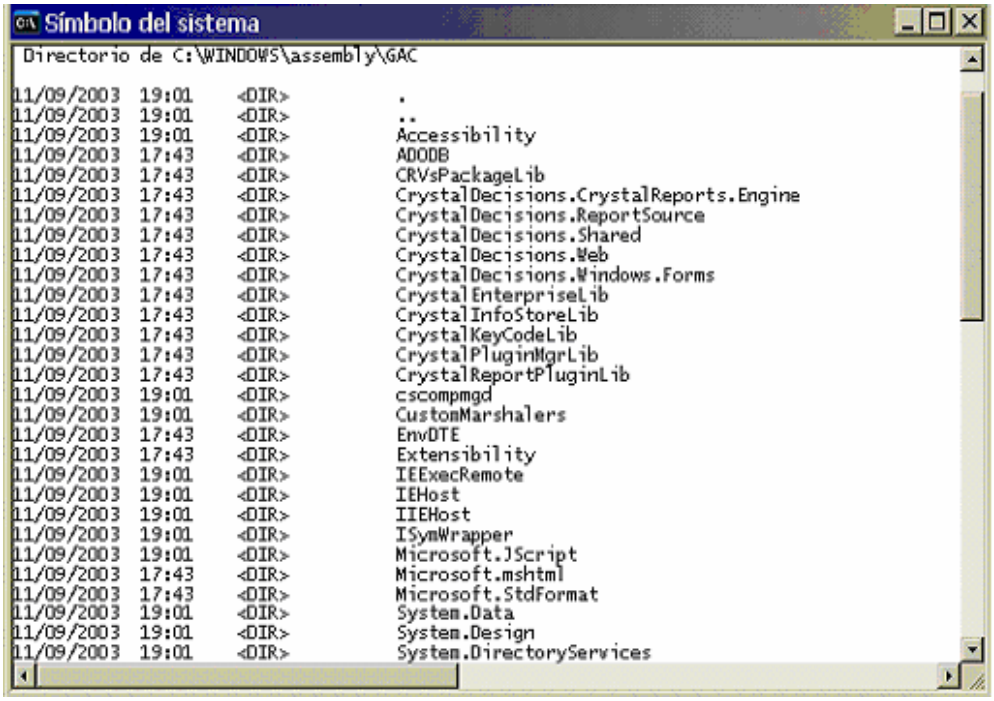

Figura 2.8 ***Caché de Ensamblados Global (GAC Global Assembly Caché) la cual es un sitio especial donde un ensamblado puede residir para que pueda ser ubicado desde todas las aplicaciones del sistema.***

El siguiente diagrama refleja la secuencia que sigue la CLR para localizar un assembly cuando una aplicación intenta cargarlo.

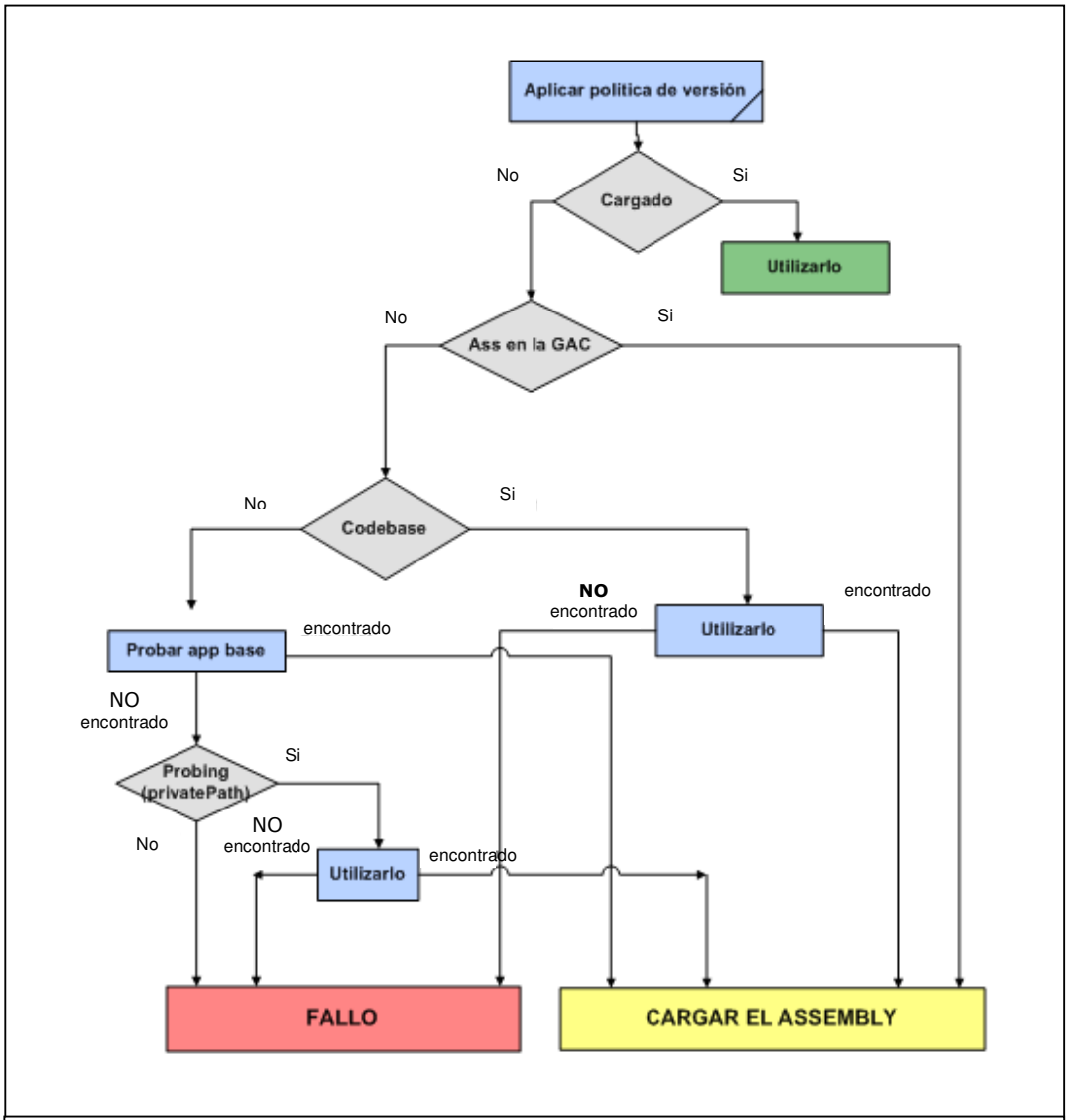

Figura 2.9 ***Algoritmo, cuando una aplicación localiza un assembly***

No existen diferencias entre un componente del modelo COM y un ensamblado ya que ambos terminan ejecutándose de forma similar desde el punto de vista del usuario. Basta con copiar el mismo a un directorio para que sus funcionalidades se hagan visibles a la aplicación.

2.4.3 Sistema de Tipo Común CTS (Common Type System)

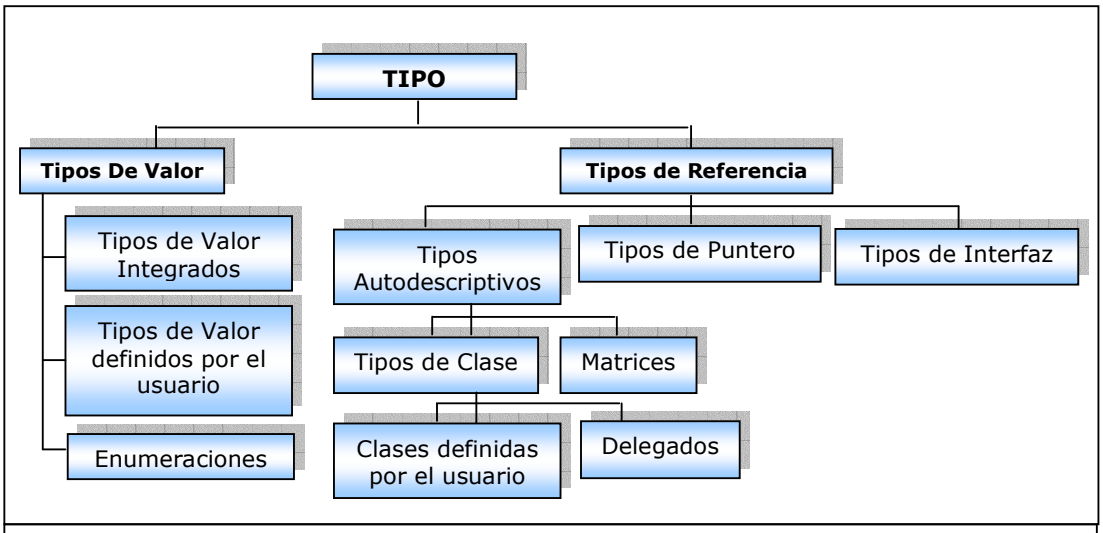

Figura 2.10 ***Clasificación general del Tipos de CTS***

El ***Sistema de tipos común*** es el modelo que define las reglas que sigue el Common Language Runtime para declarar, utilizar y administrar tipos.

El **CTS** establece un marco de trabajo que ***permite la integración entre lenguajes***, *la seguridad de tipos y ejecución de código de alto rendimiento.* Es la materia prima a partir de la cual se pueden crear bibliotecas de clases.

El sistema de tipos común realiza las funciones siguientes:

- Establece un framework que permite la integración entre lenguajes, la seguridad de tipos, y la ejecución de código con un alto rendimiento.
- Proporciona un modelo orientado a objetos que soporta la implementación de muchos lenguajes de programación.
- Define una serie de reglas que los lenguajes deben seguir para permitir la interoperabilidad de los mismos.

El CTS se divide en dos categorías generales de tipos:

TIPOS VALOR: Las instancias de los tipos Valor son almacenadas como la representación de su valor como una secuencia de bits en memoria, careciendo del concepto de identidad.

TIPOS POR VALOR	DESCRIPCIÓN	EJEMPLO
1.- Valor predefinido	Representan los tipos de datos usados por los lenguajes de programación.	Entero: *int unvalor=78;* Flotante: *float unfloat = 3.65F;* Booleano: *bool unbooleano = trae;*
2.- Valor definido	Son definidos por el usuario. Se derivan de System.ValueType.	c2 = 5;

3.- Enumeraciones	Se utilizan para escenarios de opciones múltiples, en los cuales una decisión en tiempo de ejecución se basa en un número fijo de posibilidades que se conocen en tiempo de compilación.	enum Color {
		Red, Blue,Green
		}
		public void Fill(Color color) {
		x=Color.Red
		y=Color.Blue
		z= Color.Green }

Tabla 2.6 ***Descripción de tipos por valor***

TIPOS REFERENCIA: Son almacenadas como referencias a la localización de su valor. Los tipos referencia son una combinación de una localización, su identidad, y una secuencia de bits (su valor).

TIPOS REFERENCIA	DESCRIPCIÓN	EJEMPLO
Tipos Autodescriptivos	Son de dos tipos: de Clase y Matriz. Los de clase pueden ser Definidos por el usuario, de tipo valor y Delegados	int[] tabla = {1,2,3,4}; int[][] tabla2 = {new int[] {1,2}, new int[] {3,4,5}}; int[,] tabla3 = {{1,2},{3,4,5,6}};
Tipos Puntero	Son variables que almacenan direcciones de memoria.	int * a; int*[] t; void * punteroACualquierCosa;

Tipos de Interface	Conjunto de métodos para los que no se da implementación.	public delegate void D (int x);
		interface IA{
		int PropiedadA{get;}
		void Común(int x);}
		interface IB{
		int this [int índice] {get; set;}
		void Común(int x);
		}
		interface IC: IA, IB{
		event D EventoC;
		}

Tabla 2.7 ***Descripción de tipos por referencia***

2.4.4 Especificación de Lenguaje Común CLS (Common Language Specification)

Para que el código escrito en un lenguaje sea accesible desde otros lenguajes se ha definido la ***Especificación del Lenguaje Común*** o **CLS** (Common Language Specification), que establece el conjunto mínimo de características que deben soportar los lenguajes para asegurar la interoperabilidad.

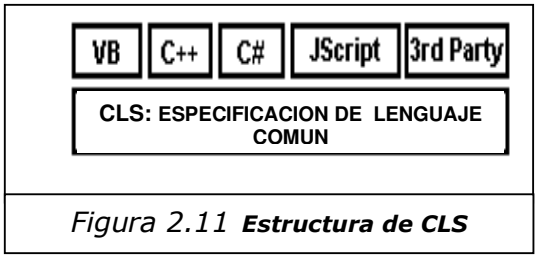

Figura 2.11 ***Estructura de CLS***

El CLS ha sido diseñado para ser lo suficientemente grande como para incluir las construcciones utilizadas comúnmente en los lenguajes, y lo suficientemente pequeño para que la mayoría de los lenguajes puedan cumplirlo.

Para que un objeto pueda interactuar con otros objetos, independientemente del lenguaje en el que hayan sido implementados, estos objetos deben exponer únicamente las características que están incluidas en el CLS. Los componentes que están adheridos a las reglas del CLS y utilizan sólo las características incluidas en el CLS se denominan como componentes conformes con CLS (CLS-compliant).

A continuación se listan algunas de reglas significativas del CLS [LIB001]:

- *Cada tipo de dato puede constar de cero o más miembros. Cada uno de estos miembros puede ser un campo, un método una propiedad o un evento.*

- *Los tipos de datos básicos admitidos son bool, char, byte, short, int, long, float, double, string y object.*

- *Se pueden definir tipos abstractos y tipos sellados. Los tipos sellados no pueden tener miembros abstractos.*

- *Las excepciones han de derivar de System.Exception, los delegados de System.Delegate, las enumeraciones de System.Enum, y los tipos por valor que no sean enumeraciones de System.ValueType.*

- *En las definiciones de atributos sólo pueden usarse enumeraciones o datos de los siguientes tipos: System.Type, string, char, bool, byte, short, int, long, float, double object.*

- *En un mismo ámbito no se pueden definir varios identificadores cuyos nombres sólo difieran en la capitalización usada. De este modo se evitan problemas al acceder a ellos usando lenguajes no sensibles a mayúsculas.*

- *Las enumeraciones no pueden implementar interfaces, y todos sus campos han de ser estáticos y del mismo tipo. El tipo de los campos de una enumeración sólo puede ser uno de estos cuatro tipos básicos: byte, short, int o long.*

2.5 Lenguaje Intermedio de Microsoft (MSIL) Microsoft Intermediate Language

Al compilar una aplicación en la plataforma .Net, el resultado no es código máquina si no un metalenguaje llamado **Lenguaje Intermedio de Microsoft** o **MSIL** (Microsoft Intermediate Languaje). El código intermedio MSIL generado por los compiladores del Framework.Net es independiente de las instrucciones de un CPU específico, y es convertido a código nativo. Posee un *formato binario* con instrucciones de bajo nivel que pueden ser solamente entendidas por el ompilador incluido en .Net llamado **JIT** (Just in time Compiler). [LIB001]

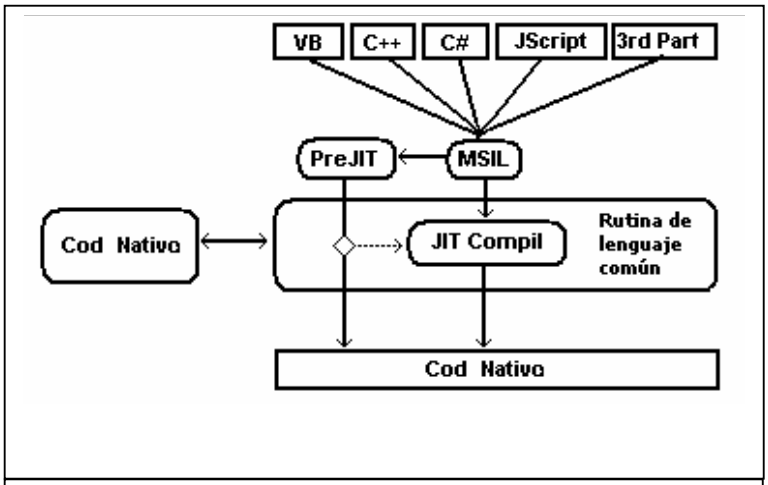

Figura 2.12 ***Proceso de compilación de MSIL***

Como parte del proceso de compilación del código MSIL, debe pasar un proceso de verificación que examine el código MSIL y los metadatos, para determinar que solo son permitidos los accesos a memoria seguros, y los objetos aislados unos de otros, es decir, que no realice conversión de punteros en forma ilegal o todo aquello que pueda causar error de protección general como fallo de memoria o acceso a recursos no autorizados, y lo transforma luego a código de máquina

teniendo en cuenta el Sistema Operativo y procesador. De esta forma se asegura de que el código final seguirá siendo siempre código de máquina y no interpretado.

2.5.1 Características de MSIL

- El código Msil ejecuta las tareas en forma segura gracias a su código gestionado.
- Posee un entorno de ejecución robusto, puede convertirse en código nativo en forma eficiente
- Seguridad inherente no causa daños.
- Desarrollo simplificado
- Fácil gestión y despliegue de aplicaciones
- Preserva inversión al desarrollador

2.5.2 Compilador JIT (Just in time).

Es un compilador bajo demanda en tiempo de ejecución. La traducción de MSIL a código nativo de la CPU es realizada por un compilador "Just In Time" o jitter, que va convirtiendo dinámicamente el código MSIL a ejecutar en código nativo según sea necesario. El proceso se muestra en la siguiente figura [WWW004]:

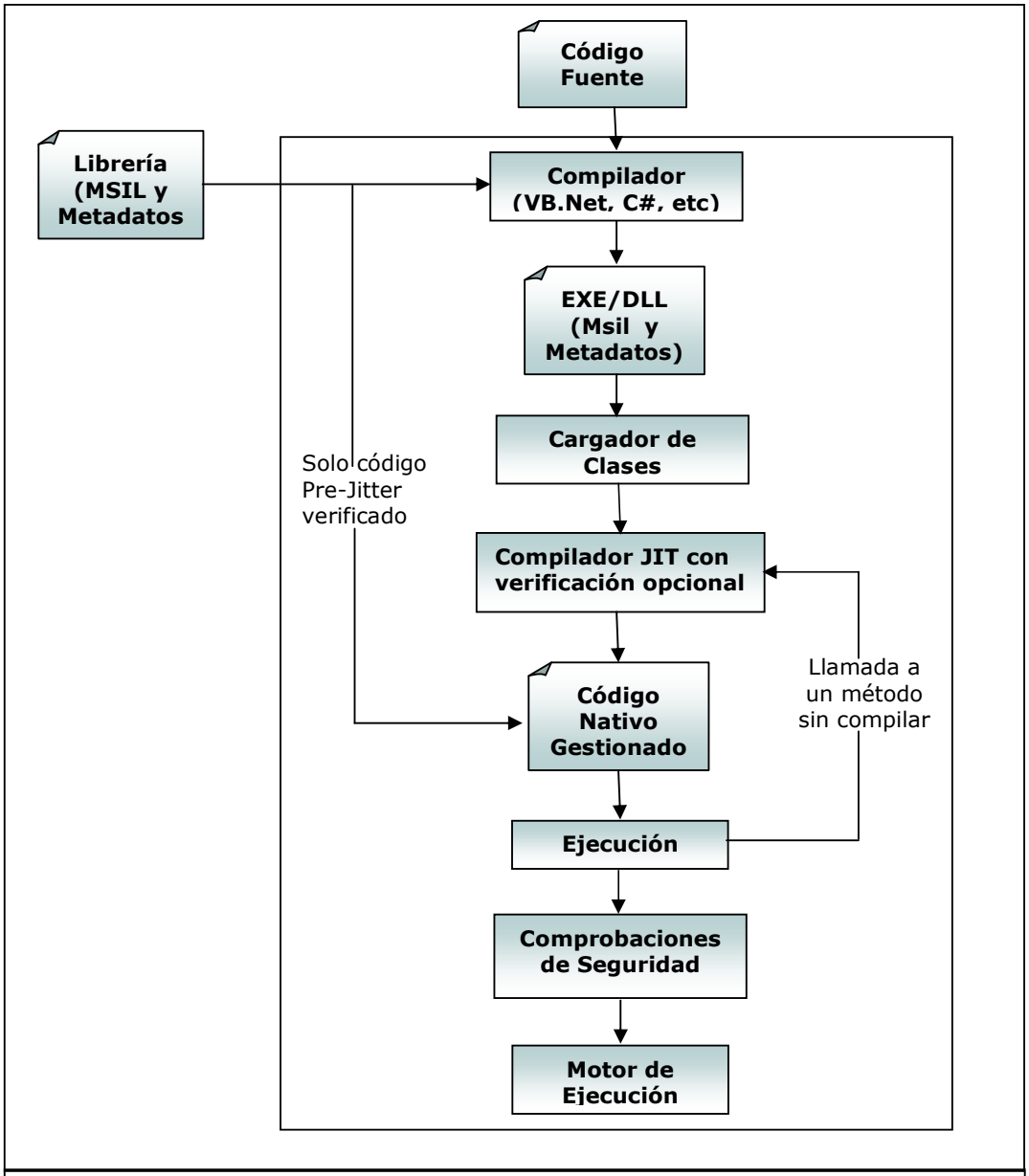

Figura 2.13 ***Proceso de compilación de MSIL a través de JIT***

JIT toma en cuenta de que no todo el código será llamado durante la ejecución, por lo que en lugar de invertir tiempo y memoria en convertir todo el código MSIL a código nativo, únicamente convierte el código que es necesario durante la ejecución, almacenándolo para futuras llamadas. El cargador crea y liga un *stub* a cada uno de los métodos de un tipo cuando este es cargado, de forma que en la

primera llamada a un método el *stub* pasa el control al compilador JIT, el cual traduce el código MSIL a código nativo y modifica el *stub* para que apunte al código nativo recién traducido, de forma que las siguientes llamadas ejecutarán directamente dicho código nativo. Durante la ejecución, el código gestionado recibe del sistema de ejecución servicios como la gestión automática de la memoria, seguridad, interoperabilidad con el código no gestionado, soporte para la depuración entre lenguajes, etc.

El compilador JIT se distribuye en tres versiones [WWW006]:

- ***Jitter normal***. se invoca cuando se ejecuta por primera vez el programa. Es el que se suele utilizar por defecto.

- ***Jitter económico.*** Funciona de forma similar al normal, diseñado para entornos con escasos recursos de memoria como dispositivos móviles. En consecuencia, el proceso de compilación se realiza de una forma mucho más rápida, y a pesar de producir un código nativo menos eficiente, la ejecución sigue siendo mucho más rápida.

- ***Prejitter.*** compila a código máquina cuando se lo esta instalando, el usuario no se da cuenta que además de instalar un programa también se lo está compilando ahorrando tiempo para la compilación dinámica.

Este esquema es mucho más eficiente, ya que el código no es interpretado, sino que es compilado la primera vez que es ejecutado. Cuando el jitter traduce el código MSIL a código nativo posee mucha mas información del entorno de ejecución que la que posee un compilador tradicional. Así, el jitter puede detectar, por ejemplo que la máquina sobre la que se encuentra es un Pentium III y generar instrucciones especiales para dicho procesador, mientras que un compilador tradicional siempre tendrá que limitarse a una máquina concreta.

2.6 Librería de Classes Base (BCL)

La Librería de Classes Base **BCL**, del Framework .NET es una librería reutilizable formada de clases, o tipos, que están altamente integrados con el framework, y que permiten acceder a los servicios ofrecidos por el CLR.

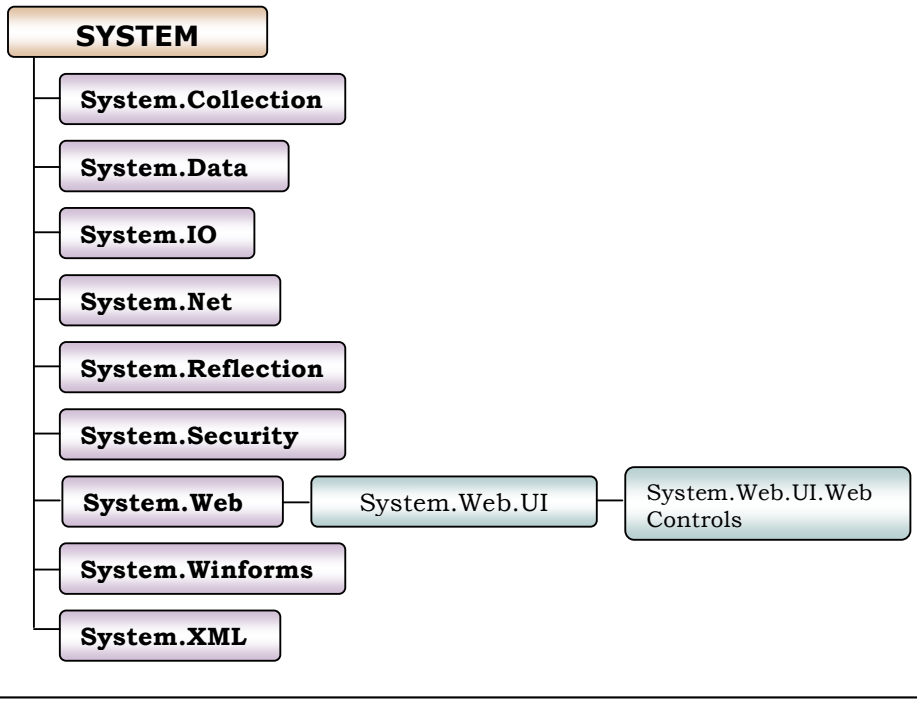

Figura 2.14 ***Ejemplo de la Estructura del Espacio de Nombre SYSTEM***

La plataforma .Net incluye clases ya compiladas como los ensamblados los cuales pueden ser utilizados libremente por el desarrollador. Debido al elevado número de clases disponibles, estas han sido agrupadas en forma jerárquica bajo una estructura llamada ***Espacio de Nombres*** la cual es un conjunto lógico de directorios utilizados para agrupar clases con funcionalidades similares y son totalmente independientes de los ensamblados que contienen a los tipos de la librería de clases. A continuación se muestra un cuadro con los espacios de nombres más utilizados de la librería de clases [WWW001]:

ESPACIO DE NOMBRES	DESCRIPCION
System	Usado para los tipos básicos, tablas, excepciones, fechas, números aleatorios, recolector de basura, entrada/salida en consola, etc.
System.Collections	Colecciones de datos de uso común como pilas, colas, listas, diccionarios, etc.
System.Data	Manipulación de bases de datos. Forman la arquitectura ADO.NET.
System.IO	Manipulación de ficheros y otros flujos de datos.
System.Net	Realización de comunicaciones en red
System.Reflection	Acceso a metadatos que acompañan a los módulos de código. Es la tarea de leer datos del manifiesto.
System.Runtime.Remoting	Acceso a objetos remotos
System.Security	Acceso a la política de seguridad que se basa el CLR
System.Web.UI.WebControls	Creación de interfaces de usuario basadas en ventanas para aplicaciones Web.
System.Winforms	Creación de interfaces de usuario basadas en ventanas para aplicaciones estándar.
System.XML	Acceso a datos en formato XML

Tabla 2.8 ***Descripción Espacio de nombres de la Librería de Clase Base BCL***

2.7 Páginas dinámicas ASP frente a ASPX.Net

La tecnología Active Server Pages (ASP) es una de las más populares y utilizadas para crear sitios Web y aplicaciones Web dinámicas. Es por ello que Microsoft ha desarrollado una nueva tecnología denominada **ASP.NET**, como parte de la estrategia .NET para el desarrollo Web, con el objetivo de resolver las limitaciones de **ASP** y posibilitar la creación de software como servicio.

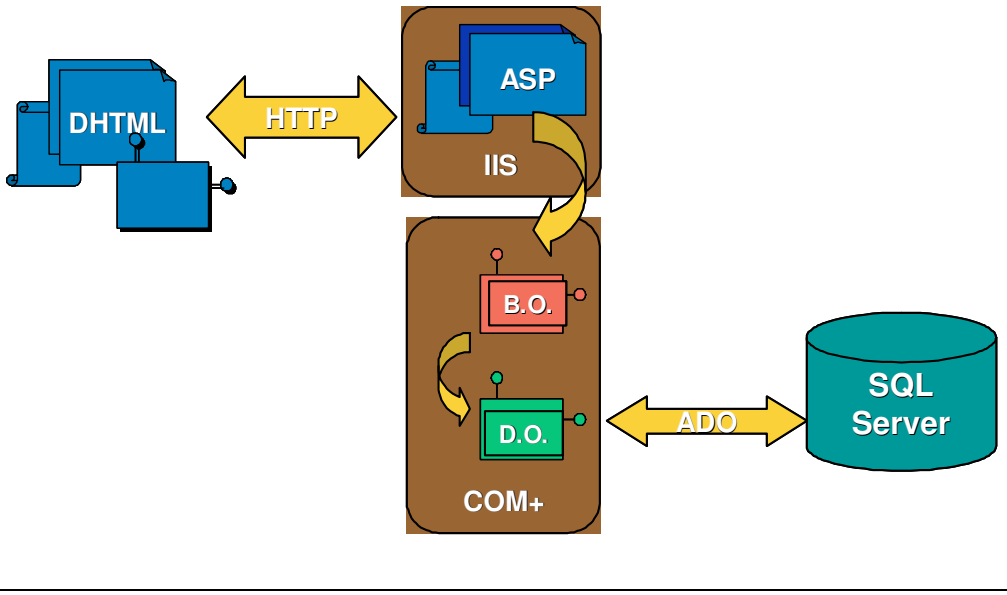

Figura 2.15 ***Modelo ASP tradicional***

ASP.NET esta formado por dos modelos de programación:

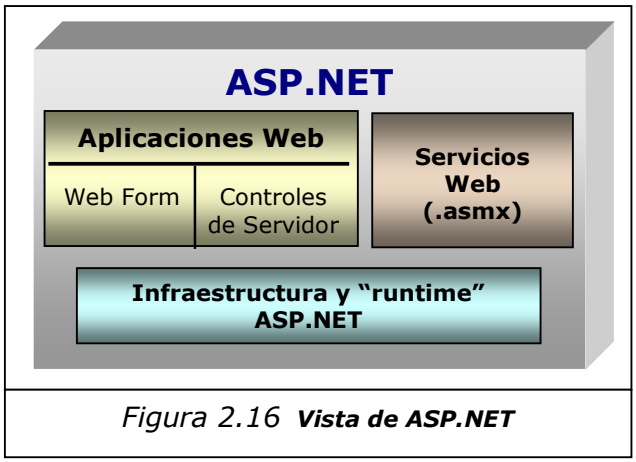

Figura 2.16 ***Vista de ASP.NET***

a.- <u>Las aplicaciones Web:</u> basadas en la generación de páginas Web, formadas por los denominados WebForms, controles servidor (server-side controls), los cuales permiten un desarrollo con mayor rapidez mediante la reutilización. WWW[008]

CONTROL	DESCRIPCION	EJEMPLO
WebForms	Utiliza la tecnología de ASP.NET para la generación dinámica de páginas Web.	*<asp:tablerow HorizontalAling=¨Center¨> <asp:TableCell text=¨1.0¨></asp:TableCell> <asp:TableRow>*
Controles de servidor o controles Web	Son encargados de la generación de interfaz de usuario de una aplicación, adaptándola a las capacidades del navegador del cliente, además del mantenimiento automático del estado del control entre los distintos accesos a una misma página durante el transcurso de la aplicación. Son más abstractos que los controles de servidor HTML debido a que no reflejan necesariamente sintaxis HTML.	Controles Predefinidos.- Tales como banners, enlaces, tablas, etc. Controles a medida.- donde el desarrollador construye sus controles a su medida de uso

Tabla 2.9 ***Descripción de las aplicaciones web***

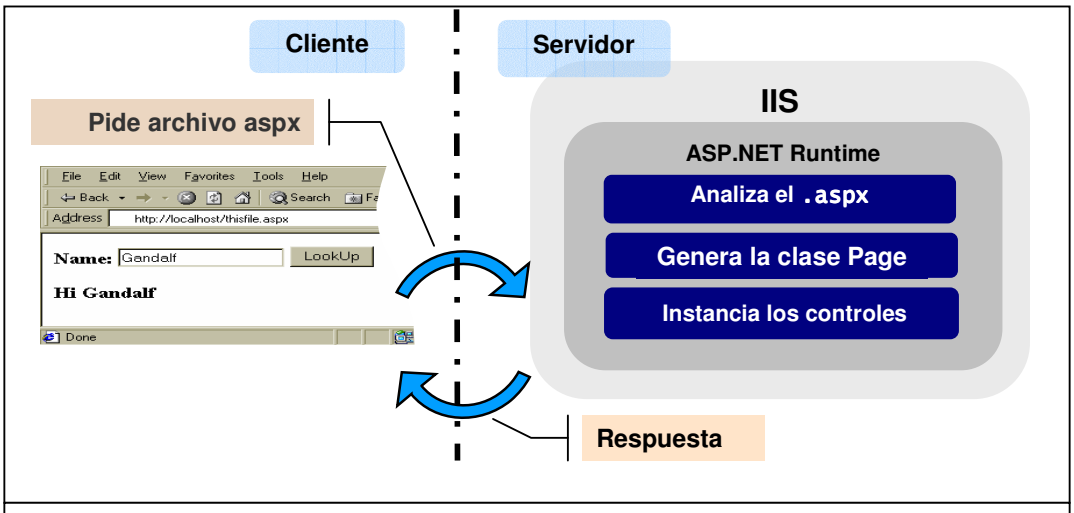

Figura 2.17 ***Web Forms de una página ASP.NET***

Cuando se requiere una página ASP.NET por primera vez, ASP.NET genera dinámicamente una clase que representa esa página. Se verifica e interpreta los contenidos de la clase derivada y la pasa al compilador (Microsoft Intermediate Language - MSIL), que es el responsable de compilar el código a un lenguaje intermedio. Tras compilar, la clase derivada se almacena en un caché de salida, de modo que las subsiguientes peticiones de la misma página se leen del caché y

se ejecutan mucho más rápido; esto permite incrementar el rendimiento tras la primera ejecución ya que se utiliza la versión compilada y el contenido de la página no tiene que ser analizada de nuevo. Incluso los lenguajes de scripting son compilados antes de su ejecución.

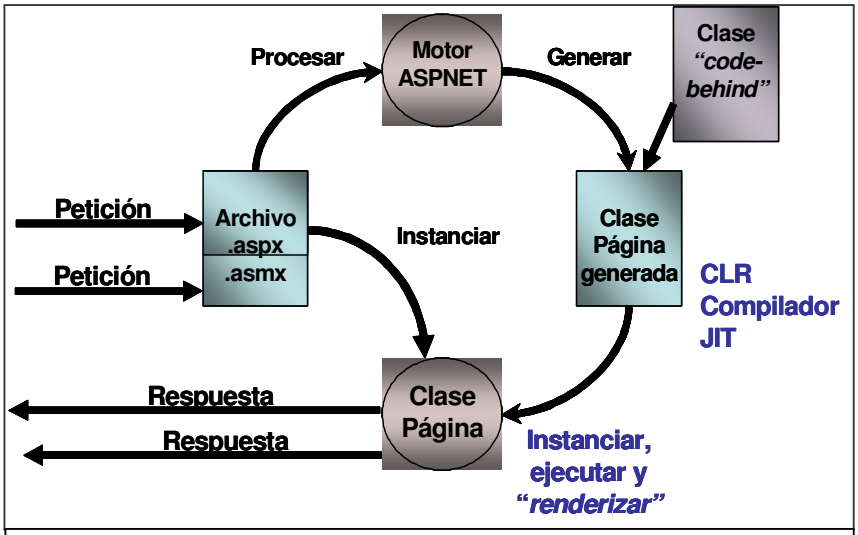

Figura 2.18 ***Descripción de una página ASP.NET***

b. <u>Los servicios Web:</u> Un servicio es una aplicación que desempeña una actividad de negocio, la cual proporciona una interfaz que puede llamarse desde otro programa, se registra y puede localizarse por medio de un servicio de registro.

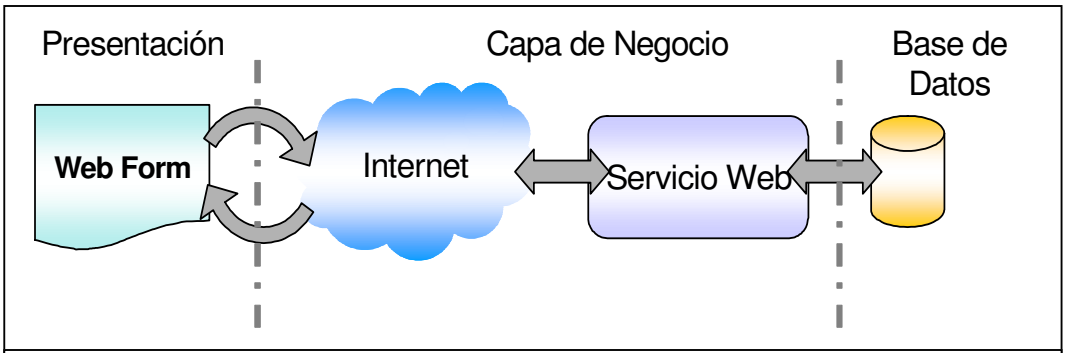

Figura 2.19 ***Estructura de un Servicio Web***

Caracterìsticas de ASP.NET

- Es un lenguaje compilado común que se ejecuta en el servidor.
- Aplica conceptos de "early binding", compilación "just-in-time", optimización de código nativa y "caching services".
- Tiene mejor rendimiento que ASP.
- Tiene un conjunto de herramientas completo y un IDE común para diseño (VisualStudio.Net).
- La .NET Framework class library, la mensajería, y las soluciones de acceso a datos son accesibles completamente desde el Web en forma transparente.
- Es independiente al lenguaje, ya que permite elegir el lenguaje que más se aplique al problema o implementar la solución con múltiples lenguajes.
- Emplea una configuración a nivel de archivos de texto, nivel jerárquico en el entorno del servidor y aplicaciones Web.
- El despliegue de una aplicación ASP.NET implica simplemente copiar los archivos necesarios al servidor.
- Se integra a la autenticación del sistema operativo Windows y permite una configuración a nivel de aplicación.

En el modelo de desarrollo Web basado en páginas ASP.NET, existen varias diferencias frente al modelo ASP que se describen a continuación:

2.7.1 Comparativa de Páginas Dinámicas Asp y Asp.Net

ASP	ASP.NET
Código desorganizado (mezcla de lenguajes). No hay una distinción entre el contenido de una página y su comportamiento.	Proporciona un modelo de desarrollo tanto para aplicaciones basadas y no basadas en el Web. Su estructura es más organizada.
La información de configuración se almacena en la metabase del servidor Web, Internet Information Server, por lo que hace difícil mover una aplicación de una máquina a otra.	La información de configuración se almacena en archivos textuales de configuración en formato XML denominados config.web, que pueden ser ubicados en los mismos directorios que los archivos de aplicación. Utiliza una arquitectura de configuración jerárquica. En ASP.NET, todos los archivos que un servidor Web necesita están ubicados bajo la carpeta raíz del sitio Web.
Es habitual escribir una gran cantidad de código para resolver necesidades sencillas.	Posee un modelo declarativo a la programación Web: los controles de servidor funcionan en una página Web solo con declarárlos. Cuando se carga la página ASP.NET, se instancian los controles listados en la página ASP y es responsabilidad del control emitir código HTML que el navegador pueda entender.
Solo utiliza lenguajes Script (VBscript y Jscript) los cuales son interpretados y no son fuertemente tipados.	Soporta múltiples lenguajes compilados (C#, VB.Net, C++.net) y fuertemente tipados, incluyendo (Scripts) que ya son compilados antes de su ejecución, lo que permite una ganancia en el rendimiento.
El único tipo de autentificación que podemos utilizar es la autentificación Windows.	Permite varios tipos de identificación y autentificación de usuario: Windows, Passport y Cookies. El servidor ejecuta el código como si el usuario estuviese presente.
Soporta un archivo declarativo global global.asax para las directivas de programa a nivel de aplicación, eventos, declaraciones de objetos globalmente accesibles y el estado de la aplicación.	Similar a ASP, ASP.NET soporta un archivo declarativo global global.asax, pero ahora este archivo soporta más de 15 eventos nuevos.
La Web, es un modelo sin estado y sin correlación entre peticiones HTTP. Esto dificulta el proceso de desarrollo de aplicaciones Web, ya que las aplicaciones generalmente necesitan mantener el estado entre varias peticiones. ASP.NET mejora y extiende los servicios de gestión de estado introducidos por ASP. El estado de aplicación, es específico de la instancia de una aplicación y no es persistente	Proporciona tres tipos de estado a las aplicaciones Web: aplicación, sesión y usuario. El estado de aplicación, como en ASP, es específico de la instancia de una aplicación y no es persistente. El estado de sesión, se almacena en un proceso separado, puede configurarse para ser almacenado en una base de datos, en un proceso o en una máquina separada. Es útil cuando se implanta una granja de servidores. El estado de usuario es similar al estado de sesión, pero no expira y es persistente.
Las aplicaciones ASP tienen una extensión .asp	Las páginas ASP.NET utilizan la extensión .aspx. se puede utilizar tanto páginas ASP como ASP.NET en un mismo sitio Web sin la obligación de reconvertirlas a páginas ASP.NET.

Tabla 2.10 ***Cuadro Comparativo entre ASP y ASP.NET***

2.8 ADO.NET frente a ADO

Microsoft ActiveX Data Objects (**ADO**) se ha estabilizado hasta tal punto de convertirse en enormemente fiable y compatible en todos los niveles. El modelo se puede aplicar fácilmente a aplicaciones de Windows y aplicaciones Web.

Sus principales características son las siguientes: [LIB001]

- Mínimo tráfico en la red.
- Número mínimo de capas entre las aplicaciones y los datos.
- Facilidad de utilización: modelo de objetos automatizado.
- Tiene en cuenta los conocimientos adquiridos por los desarrolladores sobre otras tecnologías como **DAO** **o** **RDO**.

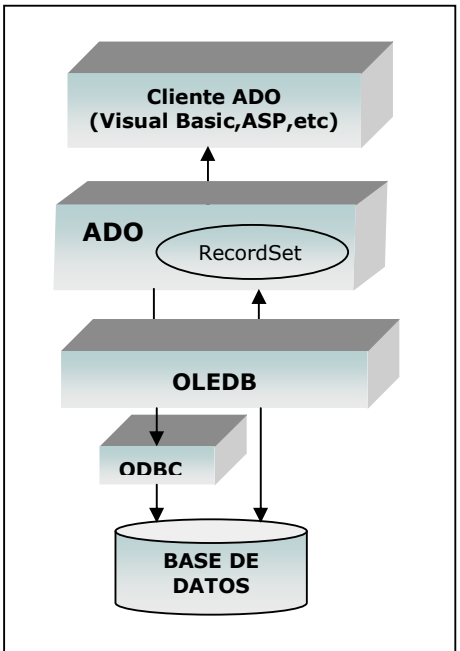

Figura 2.20 ***Arquitectura ADO***

Los proveedores **OLEDB** proporcionan un mecanismo uniforme de acceso a los datos relacionales y a los datos no relacionales. Se construyen sobre la base del modelo COM (Component Objet Model)

Los proveedores **OLEDB**, u **OLEDB PROVIDERS** son [LIB002]:

- El proveedor OLEDB para **ODBC** también llamado **KAGERA o MSDASQL**
- El proveedor OLEDB para SQL Server, llamado **SQLOLEDB**
- El proveedor OLEDB para Oracle llamado **MSDAORA**
- El proveedor OLEDB para Jet, llamado **JOLT**

2.8.1 Modelo de objetos ADO

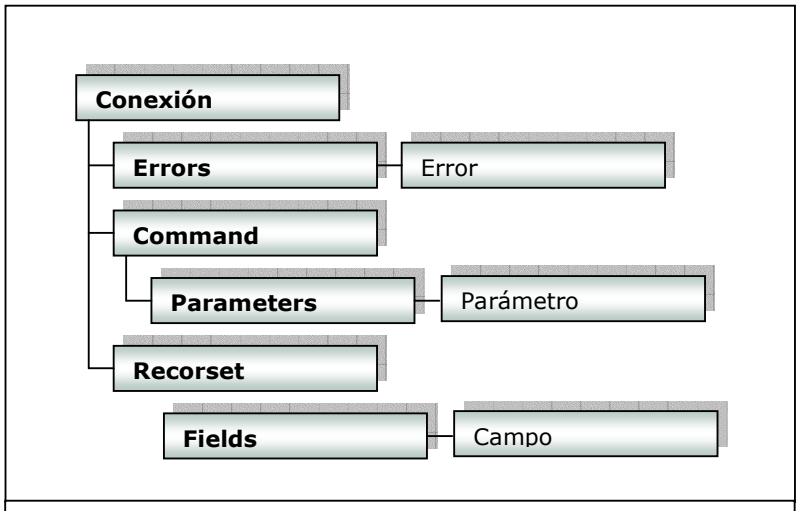

Figura.2.21 ***Arquitectura ADO. Objetos y sus relaciones***

OBJETO	USO
Objeto Connection	Permite establecer las conexiones entre el cliente y el origen de datos.
Objeto Command	Permite realizar los comandos, como las consultas SQL o las actualizaciones de una base de datos.
Objeto Recordset	Permite ver y manipular los resultados de una consulta.
Colección Parameters	Es utilizada cuando la consulta del objeto Command necesita los parámetros.
Colección Errors	La colección Errors y el objeto Error se acceden a través del objeto Connection, a no ser que se produzca un error de proveedor, por lo tanto, permite obtener información precisa sobre la causa del error.
Colección Fields	La colección Fields y el objeto Field se utilizan a través del objeto Recordset, una vez que éste contiene los datos.

Tabla 2.11 ***Descripción de los objetos del modelo ADO***

A continuación, los aspectos más débiles de ADO:

- Solo se puede gestionar una tabla o un conjunto de registros a la vez.
- No es compatible intrínsicamente con XML y no es válido para la comunicación entre distintos entornos.
- No es sencillo crear relaciones entre tablas.
- Su diseño está dirigido, esencialmente, a arquitecturas con conexión.
- La comunicación al basarse en COM, se limita a los tipos de datos que admite COM, un problema para las plataformas que no estén basadas en Windows.
- Los tipos de datos ADO se deben traducir por sus equivalentes en COM, lo que requiere gran cantidad de tiempo y de recursos del sistema.
- COM suele dar problemas a la hora de atravesar cortafuegos.

Con ADO no es posible acceder a fuentes de datos relacionales, no relacionales o de otro tipo desde un modo sin conexión. Se debe escribir distintas aplicaciones para cada nivel de acceso a datos, ya sea de Internet o de otros almacenes de

datos. En ese caso, abundan los problemas de versión, ya que las aplicaciones compatibles con ADO requieren la ubicación de las bibliotecas de ejecución adecuadas en el ordenador del cliente. Microsoft, actualiza sus bibliotecas con mejoras y, si una aplicación es compatible con una versión anterior o una nueva biblioteca se incluye con otro software, y si continúan los problemas se volverá a escribir la aplicación para la nueva DLL.

2.8.2 ADO.NET

ADO.NET es el conjunto de servicios proporcionados por la plataforma .NET para el acceso a las fuentes de datos. ADO.NET es la evolución de ADO, y para su construcción se ha tenido en mente a las aplicaciones de n-capas, cada vez más frecuentes, y a XML como su núcleo central.

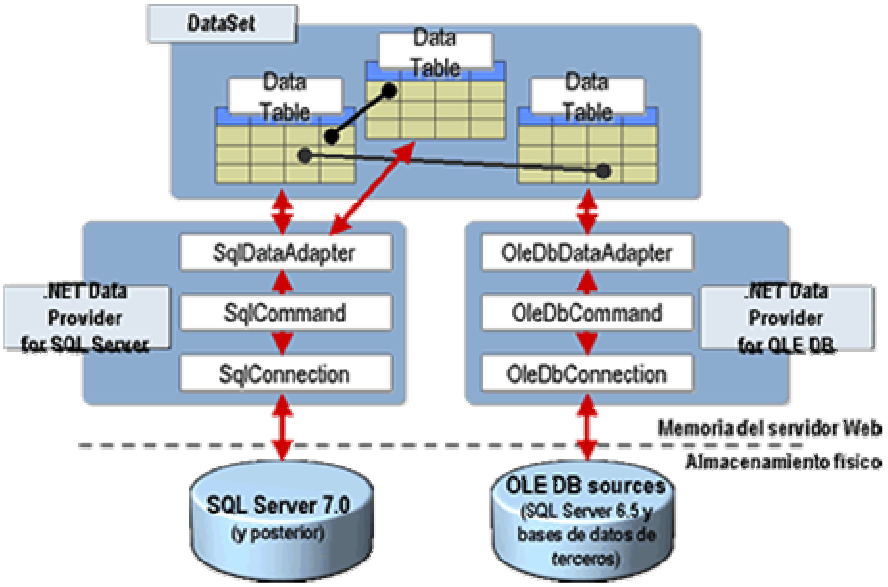

Figura.2.22 ***Arquitectura ADO.Net***

El motivo de que un servicio de acceso a datos sustituya al ADO clásico viene dado por la evolución en el desarrollo de aplicaciones, ya que cada vez son más frecuentes las aplicaciones débilmente acopladas basadas en el modelo Web, las cuales hoy utilizan XML para codificar la transmisión de sus datos. Este estilo de programación es diferente ya que las conexiones a las fuentes de datos son mantenidas durante la vida de la aplicación. El acceso a datos, ADO y OLEDB, estaba diseñado para dichos entornos conectados altamente acoplados, por lo que era necesario un nuevo modelo, el cual viene dado por ADO.NET.

El modelo de objetos de ADO.NET se divide en dos niveles:

NIVEL	DESCRIPCION
a. CONECTADO	Se comunica a la base de datos mediante el uso de los proveedores como OLEDBConnection o SQLConection.
b. DESCONECTADO	Se comunica a la base de datos mediante llamadas estándares al objeto DataSet, y luego con el Proveedor de Datos OLEDB, o directamente con SQL Server.

Tabla 2.12 ***Descripción del nivel conectado y desconectado de ADO.NET***

A continuación se describe cada uno de los niveles [LIB002]:

a.- Nivel conectado

El nivel conectado está formado por los proveedores gestionados (manager providers), que son utilizados para abrir conexiones con las bases de datos y ejecutar comandos sobre ellas. Este nivel es muy similar a la infraestructura existente en el ADO clásico.

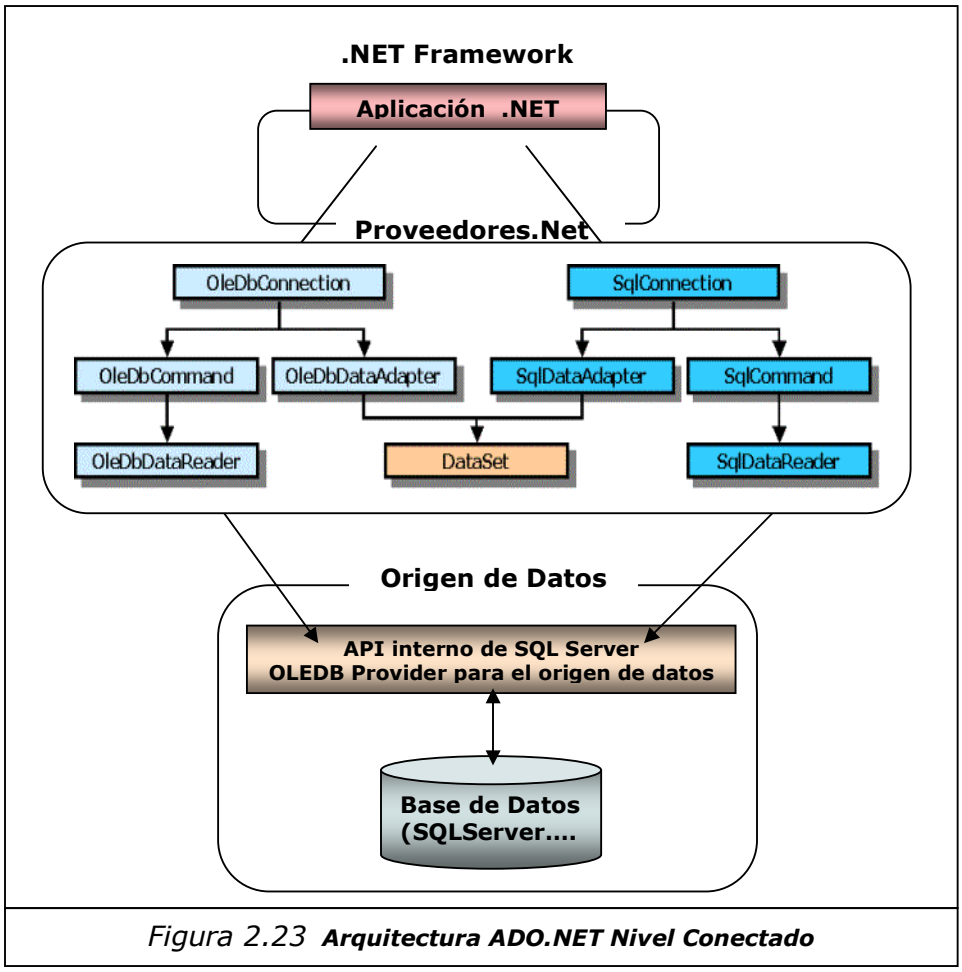

Figura 2.23 ***Arquitectura ADO.NET Nivel Conectado***

ADO.NET proporciona dos tipos de proveedores, el proveedor de **SQL,** para los servidores SQL de Microsoft, y el proveedor OLEDB, para las bases de datos tales como Oracle, Access, etc.

b.- Nivel Desconectado

El nivel desconectado lo constituyen los DataSets o conjuntos de datos, los cuales, son la representación en memoria de un conjunto de tablas relacionadas. Un DataSet proporciona un modelo de objetos formado por una serie de tablas que actúa como almacén temporal de los datos, junto con sus columnas, filas, restricciones (clave primaria,

campo no nulo, etc.), así como las relaciones entre dichas tablas. No hay una relación estricta entre un DataSet y una fuente de datos, ya que es posible crear o modificar los datos de un DataSet sin que la fuente de datos tenga constancia de ello. El DataSet almacena los datos aunque no exista conexión con la fuente de datos y es efectivo siempre que se encuentre a su alcance, pues se trata de un modelo desconectado, pudiendo explícitamente en cualquier momento actualizarse el contenido de la base de datos con el contenido del DataSet.

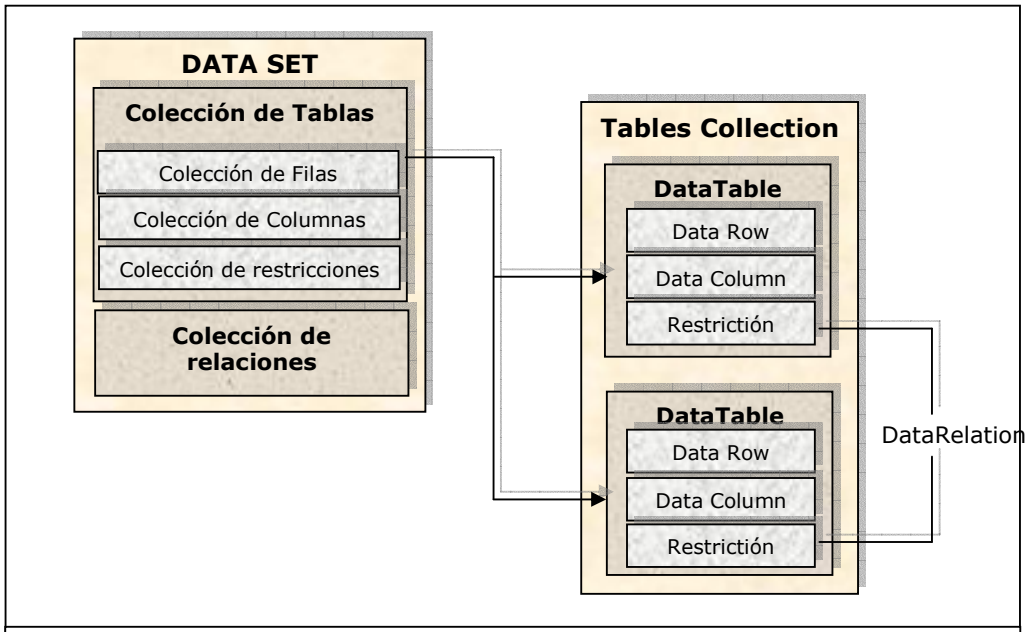

Figura 2.24 ***Estructura de un Dataset. Nivel Desconectado***

Un DataSet también puede ser cargado con los datos de una conexión de proveedor gestionado, o de una fuente de datos XML que utiliza un esquema denominado **XSD** (XML Schema Definition) para representar internamente la estructura de tablas y relaciones. La representación en XML de los DataSets los convierte en un medio ideal para transferir datos entre distintas capas de una aplicación. [LIB003]:

2.8.3 Comparación entre ADO y ADO.NET

ADO	ADO.NET
Con ADO, siempre se presenta el problema de disponer de la versión correcta de **MDAC** para acceder a los datos.	ADO.NET es compatible con cualquier versión de ADO.NET
Diseñado para acceso conectado	Diseñado para acceso desconectado
Vinculado al modelo físico de los datos	Se puede modelar la información por lógica.
El RecordSet es el contenedor central de datos.	El DataSet reemplaza al RecordSet
El RecordSet es una tabla que contiene todos los datos	El DataSet puede contener múltiples tablas
Obtener datos de más de una tabla u origen, requiere un JOIN en la base	No se requieren JOIN
Los datos son "aplanados": pierden sus relaciones y la navegación suele ser secuencial	Se preservan las relaciones: La navegación es relacional
Los tipos de datos se encuentran relacionados con tipos COM/COM+	Los tipos de datos sólo están vinculados al esquema de XML
Los datos se comparten por "marshalling COM"	No se requieren conversiones de tipos de datos.
Hay problemas para enviar información a través de firewalls (DCOM, datos binarios)	XML, como HTML, es texto plano: "Pasa las barreras"

Tabla 2.13 ***Cuadro comparativo entre ADO y ADO.NET***