UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

INSTITUTO DE CIENCIAS BÁSICAS E INGENIERÍA

LICENCIATURA EN SISTEMAS COMPUTACIONALES

MONOGRAFÍA:

LA PLATAFORMA DE DESARROLLO .NET

QUE PARA OBTENER EL TÍTULO DE LICENCIATURA EN SISTEMAS COMPUTACIONALES

PRESENTA:

ARTURO HURTADO VENTURA

ASESOR:

LIC. EN COMP. LUIS ISLAS HERNÁNDEZ

PACHUCA DE SOTO, JUNIO 2007.

ÍNDICE

ÍNDICE DE TABLAS Y FIGURAS.

AGRADECIMIENTO.

A mis padres por todos sus grandes esfuerzos que realizaron y consejos, que han sido una piedra angular en mi carrera, a mis hermanas por todo su apoyo incondicional que siempre han tenido para mi, aunque mucho tiempo después el objetivo se a cumplido y por último a mi esposa junto con el futuro hijo.

Otro reconocimientos es para todos aquellos que fueron mis catedráticos en la Universidad Autónoma del Estado de Hidalgo, ya que gracias a su desempeño nos hicieron transmitieron sus conocimientos para que por medio de ellos nosotros los aplicáramos en una vida laboral y muy en especial a mi asesor el Lic. En Computación Luis Islas Hernández por todo su apoyo y a mis asesores por el tiempo dedicado a revisar este trabajo.

Objetivo.

Se dará a conocer los conceptos que se deben de tomar en cuenta para el conocimiento de la plataforma de desarrollo .NET, toman como referencia las aplicaciones del tipo consola, Windows y Servicios Web, y en relación a está última aplicación se darán mas detalle, donde se conocerá los elementos que se tienen, así como su proceso de compilación, la forma como se deben de ejecutar los programas, los diferentes lenguajes de programación que son considerados como lenguajes .NET sus características que deben de cumplir, las ventajas y desventajas que se tienen ante otras plataformas de desarrollo.

Justificación.

En vista de que los sistemas de información cada día se vuelven mas complejos y la necesidad de ser mas globales surge la necesidad de realizar herramientas que nos ayuden y donde se ocupen al máximo los recursos que se tienen en las computadoras y la plataforma de desarrollo .NET es un herramienta que nos ayuda a hacer fácilmente aplicaciones tanto tipo consola, Windows y Web, por lo que el conocer como esta compuesta, cuales son las características que tiene así como saber las diferencias que existen con otras herramientas como COM, DCOM, etc.

Para tener las bases necesarias cuando se realiza el análisis del sistema a desarrollar para saber cual de ellas es la más adecuada de acuerdo a los requerimientos con los que debe de cumplir nuestra nueva aplicación para tener un mejor funcionamiento y un producto de software final de calidad.

La plataforma de desarrollo .NET es multilenguajes con los que podemos decir cual de ellos es conviene utilizar y que pueden ejecutarse bajo cualquier plataforma de sistema operativo.

CAPÍTULO 1

ANTECEDENTES DE LA PLATAFORMA .NET.

En este capítulo daremos a conocer, un marco teórico referente al surgimiento de los lenguajes de programación y sus diferentes clasificaciones, los conceptos de una programación orientada a objetos, así como la arquitectura paralela de las computadoras, los compiladores y por último los principios de la tecnología COM de Microsoft, la base para el desarrollo y comprensión de la plataforma .NET.

1.1 Los lenguajes de programación y su clasificación.

1.1.1 Definición.

Los lenguajes de programación son estructuras simbólicas que permiten disponer de los dispositivos de una computadora [4].

Dichos lenguajes están dotados de sintaxis, palabras reservadas, semántica y una estructura general; cuyo objetivo primordial es hacer la vida más sencilla, proporcionando formas humanamente comprensibles de enviar a la computadora una secuencia de números binarios, a través de símbolos y palabras equivalentes.

1.1.2 Clasificación.

Los lenguajes de programación pueden ser clasificados de diferentes maneras. La figura 1 muestra algunas agrupaciones clásicas:

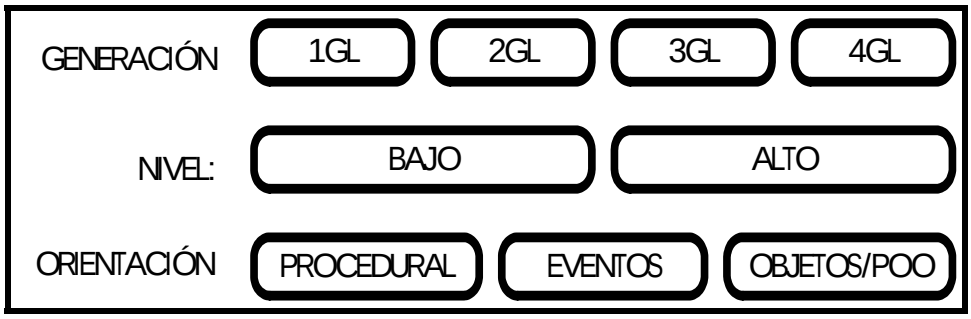


Figura 1 Clasificación de los lenguajes de programación

1.1.2.1 Clasificación en base a su generación.

La generación a la que pertenece un lenguaje indica, la capacidad de representación que tienen los símbolos que lo componen. Entre más secuencias de números binarios represente cada símbolo mayor será su generación..

a) Lenguajes de Primera Generación.

En un inicio, las instrucciones que se le proporcionaban a la computadora eran en lenguaje máquina a través de un formato de secuencia binaria, en obvia proporción de 1 a 1, es decir, a una secuencia binaria le correspondía una instrucción.

Para desarrollar en un lenguaje de primera generación era necesario conocer que representaba para la computadora cada una de las secuencias binarias que se utilizarían; por ello prácticamente solo el constructor de la misma podía programarla. Esto planteaba una enorme delimitación, ya que la capacidad humana para preparar las instrucciones, llevaba a la elaboración de programas bastante rudimentarios, por ejemplo una suma o resta. Elaborar un programa compuesto por millones de secuencias de números binarios era imposible.

Esto así, que en la primera generación se proporcionaban secuencias binarias directas, que pasaban a ejecución sin interpretación o manipulación alguna, por ejemplo:

C7 06 0000 002

Dicha instrucción se realizaba para mover el número 2 a la ubicación del registro de memoria 0000 (en hexadecimal), en un lenguaje máquina.

b) Lenguajes de Segunda Generación.

La mayor limitación de la primera generación era que las secuencias binarias eran difíciles de entender. En la segunda generación se recurrió al uso de términos mnemotécnicos, con los cuales, la proporción de instrucciones y símbolos continuaba siendo de 1 a 1, pero con la ventaja de que ya no era necesario aprender secuencias binarias, sino palabras, generalmente abreviadas.

Esto es, la herramienta de desarrollo traducía los mnemotécnicos a secuencias binarias, que pasaban a la ejecución en la computadora. En ésta generación se empleo el lenguaje ensamblador y algunos otros tales como AUTOCODER, SPS, BAL o EASYCODER.

Teniendo el mismo ejemplo, para mover el número 2 a la ubicación de la memoria, donde se considera que x es la ubicación del registro de memoria 0000, en lenguaje ensamblador se representaría con la siguiente forma:

Mov AX,2

c) Lenguajes de la Tercera Generación.

Si en la segunda generación era posible representar una secuencia binaria a través de una palabra, también debería ser posible representar varias secuencias binarias a través de una palabra. Bajo ésta premisa, nacen los lenguajes de tercera generación, como BASIC, COBOL, PASCAL, C, etcétera. Es estos la proporción de instrucciones y símbolos cambia de muchos a 1, lo que significó un avance exponencial en el uso de lenguajes. Además de la proporción, otro rasgo característico de ésta generación se dio en la

codificación, que aunque más sencilla, seguía siendo realizada por los programadores.

Es decir, en la tercera generación se programan instrucciones que la herramienta de desarrollo interpreta y traduce en una o más secuencias binarias, que pasan a ejecución en la computadora.

Con el simple hecho de ejecutar la siguiente instrucción, x = 2, se está realizando la misma operación de ejemplos anteriores.

d) Lenguajes de Cuarta Generación.

Algunos lenguajes y herramientas de desarrollo comenzaron a integrar sistemas de macroinstrucciones y generadores de código, automatizando el proceso de proporcionar a la computadora instrucciones de bajo nivel.

Las macroinstrucciones son instrucciones de nivel superior, que al ejecutarse derivan en la ejecución de instrucciones del mismo lenguaje, pero de nivel inferior.

Los generadores de código, por otro lado son herramientas que con un poco de interacción del programador generan el código capaz de desarrollar ciertas tareas rutinarias, dejando al programador la codificación de aquellas reglas específicas del negocio, que no podían ser realizadas de forma automática.

Las macroinstrucciones y los generadores de código son rasgos característicos de los lenguajes de cuarta generación. La codificación es realizada por el ser humano, pero también por el lenguaje o por la herramienta misma, esto se refiere a que es un proceso asistido.

En ésta generación se programan macroinstrucciones e instrucciones; las primeras son interpretadas y traducidas en instrucciones de menor nivel por la herramienta de desarrollo, que además las junta con aquellas que el programador haya enviado, las complementa con las generadas de forma automática, para después convertirlas en secuencias binarias, que pasan a ejecución en la computadora.

Estos lenguajes permiten generar aplicaciones de cierta complejidad con un número de líneas menor que el que se tendría si se usara uno de tercera generación. Para construir dichas aplicaciones el programador cuenta, con un conjunto de instrucciones secuenciales, además de una gran cantidad de mecanismos como son: el rellenado de formularios, la interacción con la pantalla, etc. Un ejemplo de este tipo es SQL.

1.1.2.2 Clasificación en base a su nivel.

El nivel ésta asociado al número de plataformas en las que puede ejecutarse o correr un determinado lenguaje. Como se mencionó anteriormente, las computadoras solo actúan en repuesta a secuencias binarias; y por consiguiente cada plataforma de computadoras responde a las secuencias binarias que reconoce como instrucciones.

a) Lenguajes de alto nivel.

Un lenguaje o herramienta de desarrollo es de alto nivel, siempre y cuando, exista la posibilidad de generar a partir de un mismo código fuente, secuencias binarias que sean reconocidas por varias plataformas de computadoras. Los lenguajes C, PASCAL, y COBOL, pueden servir en plataformas distintas, tales como: Windows, UNÍX y Linux, con un mayor o menor grado de portabilidad.

La portabilidad es la capacidad que tiene un lenguaje o herramienta de desarrollo para generar productos que se ejecuten en diferentes plataformas, preferentemente sin cambios en la codificación.

b) Lenguajes de bajo nivel.

Se considera de bajo nivel, cuando no existe la posibilidad de generar a partir de un mismo código fuente, secuencias binarias que sean reconocidas por varias plataformas de computadoras. Los ejemplos de bajo nivel por excelencia son el lenguaje máquina y los ensambladores.

1.1.2.3 Clasificación en base a su propósito.

El propósito está asociado a las ramas del conocimiento humano que las aplicaciones desarrolladas en los lenguajes pueden cubrir.

a) Lenguajes de propósito específico.

Son aquellos que permiten, desarrollar aplicaciones que cubren una determinada rama del conocimiento humano. Un ejemplo de este tipo, es COBOL, el cual fue pensado para aplicaciones de negocio; por lo que desarrollar, una compleja aplicación de graficación vectorial en dicho lenguaje, puede resultar si no imposible, si bastante inadecuado.

b) Lenguajes de propósito general.

Estos permiten desarrollar aplicaciones que prácticamente cubren todas las ramas del conocimiento. Un ejemplo de esto es BASIC, que igual permite desarrollar una aplicación científica, como de negocios o base de datos.

1.1.2.4 Clasificación en base a su orientación.

La orientación tiene que ver con la forma en que se estructuran internamente las instrucciones en el código fuente y los programas.

a) Programación procedural.

Llamados también de procedimientos o de paradigma imperativo, implica que las instrucciones deben de ser ejecutadas de manera secuencial, una tras otra, tal y como fueron especificadas.

Este esquema permite saltos de control, es decir, transferir el control del programa de una línea a otra (GOTO).

La reutilización del código en este esquema solo se da mediante la definición de procedimientos que pueden ser llamados a ejecución, o bien, repitiendo secuencia de instrucciones.

Este tipo de programación tiene la ventaja de ser bastante lógica (siempre y cuando no se abuse de los saltos de control), e incluso se recomienda para el aprendizaje de la programación.

Una clara desventaja se presenta en la reutilización del código, la cual resulta rudimentaria y redundante; dado que si un mismo procedimiento, aplica con pequeñas diferencias para dos circunstancias distintas, es necesaria la existencia de los dos procedimientos diferentes. Es así, que el comportamiento de los procedimientos es estricto y específico para una tarea. Otra desventaja consiste en que siendo estos secuenciales obligan a que a que la interfaz de las aplicaciones también sea secuencial.

b) Programación orientada a eventos.

La palabra evento proviene del latín *evenire*, que significa suceder, producirse; en materia de programación, indica que el uso de la interfaz del usuario provoca la ejecución de un procedimiento. Un ejemplo clásico de un evento es hacer clic en un botón de la interfaz; e usuario hace clic, luego entonces, pasa algo.

Como se puede apreciar, la programación orientada a eventos es en el fondo procedural, con la diferencia que la ejecución de procedimientos no se realiza de manera secuencial, sino aleatoria, de acuerdo a la forma en que el usuario en contacto con la interfaz provoca los eventos.

Un procedimiento de evento es la secuencia de instrucciones que se ejecutan cuando un usuario provoca el evento.

La ventaja de este tipo de programación, consiste en que, el uso de la interfaz de las aplicaciones es menos estricto, ya que el usuario puede utilizarla sin orden alguno, y ello no perjudica la secuencia de ejecución del programa. La desventaja es que en el fondo sigue siendo procedural, y por lo tanto, cada circunstancia distinta requiere su propio procedimiento.

Es importante señalar que un evento no solo puede ser causado por el usuario en su interacción con la interfaz, sino que también puede ser generado por la aplicación en interacción consigo misma.

c) Programación orientada a objetos.

Los objetos son entidades encapsuladas de código y datos, que a partir de datos de entrada que reciben, proporcionan una funcionalidad determinada.

En este tipo de programación no interesan las instrucciones y las secuencias de las mismas, ya que lo verdaderamente importante es la funcionalidad de los objetos, sus entradas y salidas. Al hacer uso de un objeto, en realidad no sabemos, si este a su vez hizo uso de otros objetos, y estos de otros que incluso se pueden encontrar en otro equipo de cómputo. De tal suerte que el orden secuencial de las instrucciones es totalmente irrelevante: no importa como o a través de que se obtienen resultados, lo importante es que se tengan.

La programación orientada a objetos es extensa, por lo que será en el siguiente tema donde se den a conocer más a detalle sus características.

1.2. Programación Orientada a Objetos

1.2.1 Definición.

El elemento fundamental de ésta programación, es como su nombre lo indica, el objeto y puede ser definido como “Un conjunto complejo de datos y programas que poseen estructura y forman parte de una organización”. Es importante recalcar, que un objeto no es un dato simple, sino que más bien contiene en su interior cierto número de componentes bien estructurados; además de que no es un ente aislado, pues forma parte de una organización jerárquica o de otro tipo.

Otra definición de objeto es “Un concepto, abstracción o cosa con un significado y limites claros en el problema en cuestión”.

1.2.2 Estructura.

Un objeto puede considerarse como una especie de cápsula dividida en tres partes:

a) Las relaciones.

Estas permiten que el objeto se inserte en la organización, y están formadas esencialmente por punteros a otros objetos. Existen dos tipos principales de relaciones:

1) Asociación.- Es una conexión entre dos clases que representan una comunicación, la cual puede tener un nombre. Dicha comunicación puede darse tanto de forma unidireccional como bidireccional (por defecto) y , la multiplicidad es el número de instancias que participan en una asociación.

2) Agregación.- Es una forma especial de asociación donde un todo se relaciona con sus partes, también se le conoce como “una parte de” o una relación de contención.

b) Las propiedades.

Son aquellas características que distinguen a un objeto determinado, de los restantes que forman parte de la misma organización y tienen valores que dependen de la propiedad de que se trate. Las propiedades de un objeto pueden ser heredadas a sus descendientes en la organización.

c) Los métodos.

Son operaciones que pueden ser realizadas sobre el objeto, normalmente están incorporadas en forma de código, de tal modo que el objeto es capaz de ejecutarlas, y también de ponerlas a disposición de sus descendientes a través de la herencia.

El verdadero impacto de la programación orientada a objetos, se refleja en el cambio de perspectivas en el desarrollo de sistemas. En ésta nueva visión, el objetivo final a largo plazo, es amortizar el costo del desarrollo de sistemas mediante la reutilización de componentes.

Ésta perspectiva modifica, tanto el ciclo de desarrollo de un sistema, como las metodologías de análisis y diseño, la administración de sistemas y en general la cultura de desarrollo de sistemas.

1.2.3 Conceptos de programación orientada a objetos.

A continuación se detallan algunos de los conceptos que pretenden facilitar la comprensión de ésta nueva perspectiva:

a) Abstracción.

Es una forma de caracterizar a los objetos, fijándose solamente en aquellas propiedades que interesan en un momento dado, e ignorando otros detalles que no son importantes. El proceso de abstracción, permite seleccionar las características relevantes dentro de un conjunto, e identificar comportamientos comunes, para definir nuevos tipos de identidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella, se puede llegar a armar un conjunto de clases que permita modelar la realidad o el problema que se quiere atacar.

b) Encapsulación.

Es la manera de ocultar los detalles de la representación e implementación de un objeto, permite abstraer al resto del mundo, de la complejidad de la implementación interna, y expone el estado del objeto, solo a través del comportamiento que se le haya definido, mediante miembros públicos presentando únicamente la interfaz disponible al usuario.

c) Modularidad

Es una agrupación de elementos lógicamente relacionados. La metodología de análisis y diseño orientada a objetos, genera módulos con un alto grado de cohesión y ligeramente acoplados entre sí.

d) Jerarquía.

Por medio de ésta, se clasifican y ordenan los componentes en distintos niveles de abstracción. El agrupamiento en jerarquías se basa en estructuras comunes o comportamiento similar.

e) Objeto y Clase.

Un objeto representa a un individuo, una unidad o entidad identificable, bien sea real o abstracto, con un rol bien definido en el dominio del problema. Se caracteriza por tener estado, comportamiento e identidad. Una clase es una especificación genérica para un conjunto de objetos que comparten una estructura y una conducta o comportamiento común. El objeto es una entidad concreta que existe en tiempo y espacio, la clase representa solamente una abstracción, la esencia del objeto.

f) Interfaz.

Es un recurso de diseño soportado por los lenguajes orientados a objetos, que permite definir el comportamiento, es decir, aunque dos clases no estén

estrechamente relacionadas entre si, son susceptibles de tener el mismo comportamiento. Se puede decir que la implementación de la interfaz, es un contrato que obliga a la clase a implementar todos los métodos definidos en la primera.

La interfaz no solo proporciona la visión externa de la clase, sino que además enfatiza la abstracción, ocultando la estructura y los secretos de su conducta. Ésta se compone principalmente de la declaración de las operaciones aplicables a las instancias de la clase.

g) Instancia.

Los objetos que se comportan de la misma manera especificada por una clase se llaman instancias de una clase.

h) Polimorfismo.

Es la propiedad que tienen los objetos de invocar genéricamente un comportamiento (método) cuya implementación es delegada al objeto correspondiente en tiempo de ejecución. Es también la habilidad de dos o más clases de objetos de responder al mismo mensaje de forma análoga haciendo uso de sus propios recursos, detalle de implementación y comportamiento.

El polimorfismo tiende a existir en las relaciones de herencia, pero no siempre es así.

i) Herencia.

Es una relación entre clases en la cual una clase comparte la estructura y comportamiento definido en otra clase [4].

Cada clase que hereda de otra posee los atributos de la clase base además de sus propios, soporta todos o algunos de los métodos de la clase base.

Existen dos tipos de herencias:

1) Herencia simple.- Una clase derivada puede heredar solo de una clase base. (Los lenguajes .NET soportan este tipo de herencia).

2) Herencia Múltiple.- Una clase derivada puede heredar de una o más clases base. (C++ es un ejemplo de lenguaje que soporta este tipo de herencia).

j) Reutilización de código.

Se refiere a la manera dinámica en que los objetos se adaptan a las circunstancias específicas de ejecución, permitiendo que un solo objeto maneje diferentes escenarios.

1.3. Arquitectura del software.

La arquitectura de un sistema constituye un amplio marco que describe su forma y estructura, sus componentes y como encajan estos juntos [10].

1.3.1 Arquitectura Maestro / Esclavo

También conocida como Amo / Esclavo consiste en que toda la funcionalidad de la aplicación se realiza en un lado del servidor (Maestro, Amo). Los equipos más conocidos que trabajan bajo este esquema son los equipos Mainframe, la figura 2 muestra su esquema general de ésta arquitectura.

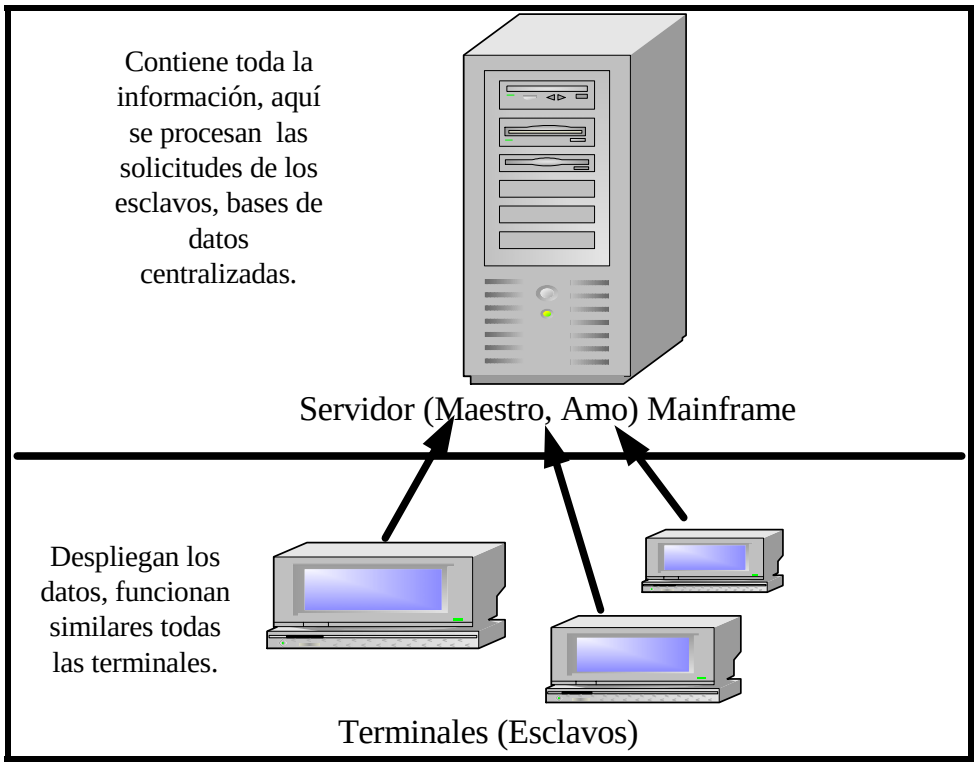


Figura 2. Arquitectura Maestro/Esclavo.

1.3.1.1 Características.

- Los sistemas basados en ésta arquitectura son desarrollados en módulos, donde cada módulo o programa es independiente, y espera recibir o enviar parámetros de entrada o salida.

- La seguridad de la información se encuentra centralizada en una sola base de datos que es única y resguardada en el Maestro.

- Es en el maestro donde se corren las aplicaciones, se guardan grandes volúmenes de datos, se realizan muchas transacciones en tiempo real; mientras que el esclavo se encarga únicamente de desplegar los datos enviados por el Maestro.

- Dado que todos los esclavos funcionan idénticamente se les llama terminales tontas. Un ejemplo donde se utiliza ésta arquitectura es en los cajeros automáticos.

1.3.2 Arquitectura Cliente / Servidor

Es aquella en la cual el usuario final (***cliente***), tiene el acceso y procesa los datos desde una máquina remota (***servidor***). En este ambiente, el servidor maneja no solo la distribución de datos, sino que también rige la forma en que estos son accedidos y manipulados por el cliente. La aplicación cliente realmente solo sirve como un medio para presentar o extraer datos.

1.3.2.1 Elementos de la arquitectura Cliente / Servidor.

a) El Cliente

La aplicación cliente proporciona la interfaz que el usuario necesita, para manipular datos del servidor desde el cliente. La manera en que el cliente interactúa con el servidor es mediante la petición de servicios. Un servicio típico podría ser, adicionar un cliente, agregar una orden o imprimir un reporte, así el cliente simplemente hace la solicitud, y proporciona los datos necesarios; mientras que el servidor es el responsable de procesarla.

b) El Servidor

El servidor proporciona los servicios al cliente. Es imprescindible esperar a que este último haga una solicitud y entonces procesarla. Un servidor debe ser capaz de procesar múltiples solicitudes desde múltiples clientes, además de

priorizar estas solicitudes. Es muy probable que el servidor corra continuamente para permitir un acceso constante a estos servicios.

c) Reglas del Negocio

Son los procedimientos que regulan como los clientes tienen acceso a los datos del servidor. La decisión acerca de donde se quiere que existan estas regla, o el porque se quieran dividir entre el servidor y el cliente, depende de varios factores, algunos de los cuales son:

1. Seguridad de los Datos. Este factor se considera, cuando se requiere proporcionar acceso limitado a varias partes de los datos, o a varias tareas que pueden ser realizadas en los datos, lo que se logra con los privilegios de acceso a usuarios para objetos de Bases de Datos, como son las vistas y procedimientos almacenados.

2. Integridad de los Datos. A menos que se tengan las reglas necesarias para proteger los datos, es posible que estos puedan tener errores; un camino para evitarlo, es limitar el tipo de operaciones que pueden ser realizadas a los datos desde procedimientos almacenados, otro camino consiste en colocar la mayor parte de la lógica del negocio en el servidor o en la capa media.

3. Control Centralizado de Datos. Otro beneficio de tener la lógica del negocio en el servidor, o en otra capa, se refleja en que se pueden implementar actualizaciones en la lógica del negocio sin afectar la operación de la aplicación cliente. Si agregamos código adicional en algún procedimiento almacenado, este cambio es transparente para el cliente.

4. Distribución de Trabajo. Colocando las reglas del negocio en el servidor o por separado en varias capas medias, se pueden realizar más fácilmente las tareas, dividiendo las responsabilidades en las áreas específicas y conservando con ello la integridad y seguridad de los datos.

Una consideración a tomar en cuenta en las aplicaciones cliente / servidor, es que son muy costosas. Estos costos incluyen el software para redes, el sistema operativo servidor, el servidor de base de datos y hardware capaz de manejar el software.

La arquitectura cliente / servidor es uno de los mecanismos clásicos más extendidos para la construcción de aplicaciones distribuidas.

1.3.2.2 Características de la arquitectura Cliente/Servidor.

- Permite el acceso limitado a los datos, al seleccionar solo los procesos del negocio del cual son responsables.

- Proporciona acceso a los datos para tomar decisiones eficientemente.
- Aumenta el control centralizado para garantizar la integridad de datos.
- Hace cumplir las reglas de integridad de datos en la base de datos.
- Mejora la división del trabajo entre el cliente y el servidor.
- Es capaz de usar la habilidad en el manejo de la integridad de datos proporcionada por muchos manejadores de base de datos.
- Reduce el tráfico en la red, dado que el subconjunto de datos es regresado por el cliente.

1.3.2.3 Modelos de la arquitectura Cliente /Servidor.

Los sistemas cliente / servidor se encuentran divididos en modelos tales como:

a) Modelo de Dos Capas.

Son aquellos sistemas en los que la aplicación cliente se comunica directamente con el servidor sin intermediarios. Se utilizan en pequeños entornos y están limitados a un número de usuarios, generalmente menos de cincuenta.

Un problema muy común con este tipo de modelos se presenta cuando, tras un diseño inicial del sistema para unos pocos usuarios, se intenta escalar el sistema añadiendo más usuarios. De este modo solo se consigue tener un servidor cargado.

b) Modelo de Tres Capas.

En estos modelos se introduce un servidor intermedio o agente entre el cliente y el servidor final. Este agente se puede utilizar de diferentes formas, dependiendo del escenario concreto con el que se enfrente aunque, en general, desempeñara muchas funciones.

El agente intermedio limita el número de conexiones concurrentes contra el servidor para evitar sobrecarga, balancea la carga de peticiones contra un número de servidores finales, y con ello da servicio a una gran cantidad de usuarios, o bien, implementa servicios de seguridad comunes a todas las aplicaciones.

Aquí el cliente tiene los datos de la interfaz del usuario, la base de datos remota del servidor es en donde los datos residen. La aplicación del cliente hace los requerimientos de acceso o modifica los datos a través de una

aplicación del servidor o de un corredor de datos remoto, el corredor de datos remoto típicamente es el lugar en donde las reglas de negocio existen. Para la distribución el cliente, el servidor y las reglas de negocio son máquinas separadas. Los diseñadores pueden efectivamente optimizar el acceso de datos y mantener la integridad de los datos para otras aplicaciones en todo el sistema.

c) Modelo de N Capas o Multicapas y sus ventajas.

A partir del modelo de tres capas ya es considerado de n capas, lo que más comúnmente tenemos es la de cuatro, la capa que se agrega es la que surge de separar de la capa de reglas del negocio la del “Acceso de Datos” .

La capa de acceso de datos nos brinda la ventaja de aislar definitivamente nuestra lógica, ya que desde el manejo de la conexión hasta la ejecución de una consulta, la manejara la capa de acceso de datos.

De este modo ante cualquier eventual cambio, solo se deberá tocar el modulo específico, así como al momento de planear la escalabilidad de nuestro sistema, de ésta forma hemos respetado las reglas básicas de diseño por lo tanto no deberíamos afrontar grandes modificaciones.

Existen muchas razones por las que usar el desarrollo de aplicaciones bajo el modelo de n capas, algunas se detallan a continuación:

1) Abstracción total acerca del origen de datos.

Las distintas capas se especializan absolutamente en la funcionalidad que deben brindar (procesamiento en las reglas de negocios o presentación de datos en la capa cliente) sin importar cual es el origen de los datos procesados.

2) Bajo costo de desarrollo y mantenimiento de las aplicaciones.

Si bien al momento del diseño podemos observar una mayor carga de complejidad, la utilización de este modelo nos brinda un control más cercano de cada componente, así como también la posibilidad de una verdadera reutilización del código.

3) Estandarización de las reglas de negocio.

Las reglas de negocio se encuentran encapsuladas en un conjunto de rutinas comunes y pueden ser llamadas desde diversas aplicaciones sin necesidad de saber cómo ésta funciona o ha sido diseñada.

4) Mejor calidad en las aplicaciones.

Como las aplicaciones son construidas en unidades separadas, estas pueden ser probadas independientemente y con mucho más detalle, esto conduce a obtener un producto mucho más sólido.

5) Reutilización de código.

La concepción natural de un sistema desarrollado con ésta modelo, promueve la reutilización de sus componentes en varias partes del propio desarrollo y de futuros sistemas.

6) Escalabilidad.

Utilizando Servicios Transaccionales de Microsoft (MTS), donde muchos objetos pueden escalar y ser distribuidos en un ambiente transaccional de alta seguridad.

El desarrollo basado en componentes u objetos distribuidos es uno de los modelos multicapas o n capas.

1.4 Compilador.

1.4.1 Definición.

Es un programa que lee código fuente y lo traduce a lenguaje objeto. Como parte importante de este proceso de traducción el compilador informa al usuario de la presencia de errores en el programa fuente[4], la figura 3 muestra los componentes de un compilador.

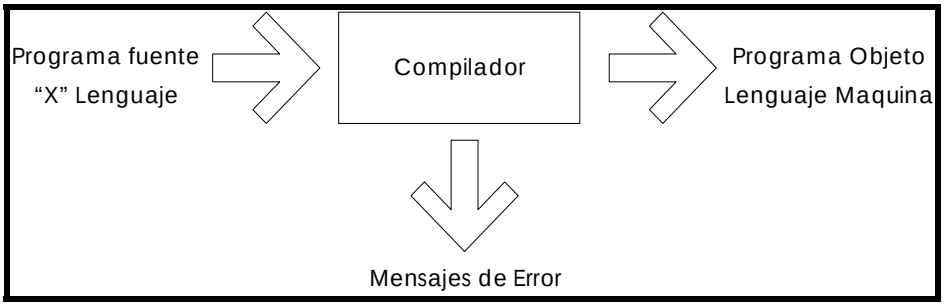


Figura 3. Compilador

Un compilador se diferencia de un intérprete en que este último acepta un lenguaje fuente y lo ejecuta; analiza cada sentencia al ejecutarlo, por lo que es muy lento.

1.4.2 Clasificación.

Los compiladores a menudo se clasifican como de una pasada, de múltiples pasadas, de carga y ejecución, de depuración o de optimación, dependiendo de cómo fueron construidos o de la función que realizan. A pesar de su complejidad las tareas básicas que debe de realizar un compilador son esencialmente las mismas, por lo que a continuación definimos algunas clasificaciones de compilador[6].

a) Compilador de una pasada.

Examina el código fuente una vez, generando el programa objeto.

b) Compilador de múltiples pasadas

Requieren pasos intermedios para producir un código en otro lenguaje, y una pasada final para producir y optimizar el código producido durante los pasos anteriores

c) Compilador de optimación

Lee un código fuente, lo analiza y descubre errores potenciales sin ejecutar el programa.

1.4.3 Partes de un compilador

El compilador consta de 2 partes el análisis y la síntesis.

a) Análisis.

Este se encarga de dividir al código fuente en los elementos que lo componen y genera una representación interna del mismo; para esto se requieren de técnicas muy especializadas, que determinan las operaciones que implican el programa fuente y las cuáles se registran en una estructura jerárquica conocida como árbol [4].

b) Síntesis.

Este se encarga de construir el programa objeto deseado, a partir de la representación intermedia y requiere también de técnicas especializadas.

1.5 La tecnología COM de Microsoft.

Microsoft lleva muchos años desarrollando su propia tecnología de componentes distribuidos, denominada Modelo de Objetos Componentes (COM).

COM es una arquitectura propietaria de Microsoft cuya implementación más extendida forma parte de los sistemas operativos Windows, donde inició su aparición a principios de los años noventa [7].

En la actualidad, ésta tecnología se utiliza de manera intensiva en la plataforma Windows. Compañías como Software AG proporcionan implementaciones de COM dentro de su línea de productos EntireX para Unix, Linux y Mainframes, Digital lo hace para la plataforma Open VMS y Microsoft para Windows y Solaris. Sin lugar a dudas, es una tecnología madura, probada y fiable.

El objetivo de COM es facilitar la integración de las partes que forman un sistema complejo y favorecer la reutilización de componentes. Una de sus mayores ventajas es que define un estándar binario para la interacción de componentes por lo que, esencialmente, es independiente de cualquier lenguaje concreto de programación.

En esta arquitectura, cada componente es un objeto que puede presentar diversas facetas o roles. De este modo, se puede añadir más funcionalidad a un componente sin que los programas existentes se vean afectados. Cada componente tiene asignado un Identificador Único Global (GUID), formado por un número de 128 bits, que se utiliza para distinguir a un componente del resto.

Las distintas facetas de un componente se implementan como interfaces separadas. Cada interfaz está formada por un conjunto de funciones o métodos relacionados. Lo más habitual es que un componente COM presente varias interfaces a las aplicaciones cliente.

La interfaz es la “cara visible” del objeto y representa la única manera de acceder a el, ya que la arquitectura no permite utilizar directamente un objeto si no es a través de sus interfaces. Cada interfaz especifica un contrato, un acuerdo entre el programador que usa el componente y el creador del mismo. El contrato consiste en una declaración de intenciones: los programadores que utilizan el componente se comprometen hacerlo correctamente para evitar así un funcionamiento erróneo o inesperado. El creador del componente, por su parte, se compromete a no modificar las interfaces del objeto, por lo que las aplicaciones cliente seguirán funcionan en el futuro aún cuando se realicen mejora en el componente. De este modo, si fuera necesario extender la funcionalidad del componente se añadirían nuevas interfaces.

Pero, si los componentes implementan múltiples interfaces, ¿cómo se sabe cuáles se implementan en primer lugar? ¿Cómo se pueden obtener las distintas caras del componente? Todas estas cuestiones las resuelve la arquitectura COM, ya que obliga a que todos los componentes implementen una interfaz base Iunknown. Ésta sencilla interfaz incluye la funcionalidad común a todos los componentes y, lo que es más interesante, incorpora una función que permite obtener el resto de las interfaces del componente.

La implementación directa de los componentes COM es una tarea compleja que requiere un conocimiento profundo de las técnicas de programación bajo Windows. La manera más habitual de programarlos es utilizando C++ y la biblioteca de plantilla Activa (ATL). No obstante, para los más atrevidos, también es posible implementar en C++ usando directamente la Interfaz de programación de aplicaciones (API)..

La creación de componentes COM desde el entorno Visual Studio con lenguajes como Visual Basic, Visual Basic Script, Jscript o C# es una tarea mucho más sencilla. Señalando que casi nunca será necesario programar componentes propios al desarrollar aplicaciones sobre la plataforma .NET.

Las interfaces de un componente COM se escriben con un lenguaje específico, bastante sencillo, llamado Lenguaje de Definición de Interfaces Microsoft (MIDL). Aunque pueda parecer exagerado e incluso molesto tener que aprender un nuevo lenguaje para la definición de interfaces, es el pequeño precio a pagar a cambio de que el sistema sea independiente de cualquier lenguaje de programación en concreto. Tras la definición de una interfaz con este lenguaje, emplearemos un compilador especial de MIDL para traducir la interfaz escrita en MIDL al lenguaje de programación que estemos utilizando para desarrollar el componente COM.

Un servidor COM puede implementarse como un archivo ejecutable (EXE) o una biblioteca dinámica (DLL). En cualquiera de los dos casos, el proceso de registro del servidor COM pasa por la generación de un identificador único GUID y por el registro del servidor con la utilidad estándar REGSRV32.EXE. La información sobre los servidores COM registrados en el sistema y el modo en que deben iniciarse está almacenada en el registro de Windows bajo la rama HKEY_CLASSES_ROOT. Para que nos hagamos una idea de lo extensamente que se usa ésta tecnología, diremos que una instalación típica de Windows incluye más de 1500 componentes COM registrados bajo ésta rama del registro.

1.5.1 Facilidades de la arquitectura COM.

Otro de los objetivos buscados con el diseño de la tecnología COM es la transparencia de localización entre clientes y componentes. Es posible que los clientes que acceden a los componentes sean aplicaciones sin ninguna relación con COM o con el resto de los componentes. A todos los efectos, los mecanismos que intervienen en la comunicación entre una aplicación y un componente son los mismos que los que se establecen entre dos componentes.

La transparencia de localización es una de las piedras angulares de COM, ya que permite que las aplicaciones se codifiquen de la misma forma, con independencia de que los componentes residan en la misma máquina o en máquinas separadas. Para conseguirlo, se ha utilizado una arquitectura basada en un broker (intermediario) de objetos. El intermediario ésta siempre presente, aunque oculto, actuando siempre que un cliente localiza o invoca los componentes del entorno.

El broker proporciona una serie de facilidades claves, tanto para la aplicaciones clientes como para los componentes:

a) Transparencia de localización.

El intermediario es responsable de localizar el componente, determina si se encuentra en la misma máquina o en otra diferente y redirige las peticiones a las que corresponda. El código del cliente es siempre idéntico, independientemente de la localización exacta del componente. El broker utiliza el registro de Windows a el servicio de directorio Microsoft Active Directory para descubrir la ubicación de los componentes.

b) Invocación Remota.

Existe un intermediario en cada máquina de la arquitectura. Los intermediarios solo se comunican entre sí y con los componentes de la misma máquina. Si la aplicación cliente y el componente remoto se encuentra en máquinas diferentes, el intermediario de la máquina cliente debe codificar (Marshall) los parámetros de la llamada antes de enviárselos al intermediario remoto, que los descodificará (unmarshall), activará el servidor COM en su máquina local y ejecutará el método correspondiente del servidor. La respuesta resultado de la ejecución será codificada nuevamente y remitida al intermediario de la máquina cliente que lo descodificará y entregará a la aplicación cliente. Cuando cliente y componente se encuentren en la misma máquina solo será necesario la intervención de un intermediario en todo el proceso y, como ventaja añadida se evita la codificación-descodificación de parámetros y la sobrecarga de la comunicación por red. De este modo, la invocación de un componente en la máquina local es siempre más rápido y eficiente.

c) Gestión de componentes.

El intermediario es el responsable de la gestión de los componentes, de su activación y desactivación, y de añadir, modificar y eliminar los servicios (componentes) disponibles.

d) Ocultación de la plataforma.

Los sistemas operativos Microsoft Windows se ejecutan mayoritariamente en la plataforma Intel, aunque también pueden hacerlo sobre estaciones Alpha. Independientemente del microprocesador concreto y de las características de la máquina donde residan los distintos servicios, los intermediarios hacen transparentes todos los detalles y consiguen que sea posible la cooperación entre componentes desplegados en distintas plataformas hardware.

1.5.2 Componentes ActiveX.

Una vez que la arquitectura de componentes y servicios básicos de COM se hubo extendido ampliamente, Microsoft comenzó a trabajar en una variante de sus componentes que permitieran el acceso a la funcionalidad de las aplicaciones más importantes del fabricante de software, tales como: Explorer, Outlook, Word, Excel, Access y PowerPoint.

La complejidad que presentan estas aplicaciones hacía inviable crear todos los componentes necesarios para encapsular toda la funcionalidad completa de la suite Office y Explorer. En la práctica, esto le hubiera supuesto a Microsoft tener que rediseñar sus aplicaciones con componentes y hacer públicas las interfaces de los componentes, una vía demasiado costosa y que exponía los detalles de implementación de sus programas.

Como alternativa, se ideó un tipo nuevo de componentes COM, caracterizados por las siguientes propiedades:

- Se utilizan para acceder a la funcionalidad de una aplicación compleja que se ejecuta tras el componente. Se trata de una característica distintiva ya que, normalmente, los componentes implementan toda la funcionalidad por si mismos, no “ocultan” ni sirven de intermediarios a otras aplicaciones.

- En lugar de ofrecer interfaces complejas específicas para cada aplicación, todos ellos implementan una interfaz común, IDispatch, que permite al componente recibir mensajes con órdenes específicas de la aplicación cliente. Es responsabilidad de cada componente documentar y definir claramente los mensajes concretos que va a interpretar y, a continuación, implementar el código que los ejecutará.

- Implementan interfaces adicionales específicos, IConnectionPointContainer e IConnectionPoint, que permiten la comunicación asíncrona mediante eventos entre la aplicación cliente y el componente. La aplicación cliente podrá utilizar estas interfaces para registrar funciones manejadoras que se ejecutarán en respuesta a eventos recibidos por el componente.

- Para promover la difusión de la tecnología COM, el departamento de marketing de Microsoft bautizó estos componentes con el nombre de componentes ActiveX.

- Se introduce el concepto de contenedor de componentes. El contenedor coopera con los componentes para dotarles de un contexto en el que se van a ejecutar. Un ejemplo de contenedor de componentes ActiveX es el navegador Microsoft Explorer, en el que se ejecutan los componentes descargados de la

Web. Estos componentes, adquieren la capacidad de interactuar con el resto de componentes de nuestra computadora a través del contenedor.

Los componentes ActiveX son adaptadores COM que no van a permitir acceder a aplicaciones complejas. La comunicación entre la aplicación cliente y el objeto ActiveX está basado en el envío de mensajes. Por su parte, el objeto ActiveX delegará la ejecución de los mensajes en los servicios ofrecidos por la aplicación final.

1.5.3 Componentes COM+.

Con el lanzamiento de Windows 2000, Microsoft actualizó su tecnología COM, renombrándola con la denominación COM+. El departamento de marketing de Microsoft bautizó la nueva línea de sistemas operativos para servidor bajo la marca Windows DNA (Aplicaciones Distribuidas para Internet).

Pero, ¿Qué es COM+? ¿Introduce cambios cualitativos o es una simple mejora ? Para empezar, digamos que COM+ es la suma de COM más un conjunto de servicios básicos. Este cambio es importante, ya que los servicios básicos ofrecen una funcionalidad común que suele ser necesaria para escribir cualquier componente: servicios de mensajería, colas, asíncronas, soporte a las transacciones, seguridad, sincronización, servicio de eventos y un largo etcétera. La filosofía de Microsoft se resume en utilizar COM+ como tecnología básica de Windows para todos los desarrollos.

A partir de Windows 2000, todos los sistemas operativos de Microsoft incluyen soporte para COM+ totalmente integrado. Aparece también ahora un catálogo de componentes COM+, en el que encontramos almacenada toda la información relativa a los objetos. Este catálogo se apoya en una base de datos llamada RegDB y el registro de Windows.

COM+, además de los servicios básicos ya comentados también incluye como novedad de Servicios Transaccionales de Microsoft (MTS, 3.0). Mientras que COM separa interfaces de implementación, MTS separa el comportamiento de un componente de su estado. MTS nos permite implementar componentes de manera que puedan formar parte de una transacción compleja. Además, también hace posible realizar transacciones entre componentes que residen en máquinas diferentes, para lo que utiliza un protocolo distribuido transaccional de dos fases.

La aportación más novedosa que introduce COM+ consisten que ahora los componentes COM+ se ejecutan dentro de un contener de componentes o servidor de aplicaciones. De este modo, cuando la aplicación cliente solicita un componente, el contener (dllhost.exe) intercepta la solicitud y gestiona el ciclo de vida del componente, lo construye en la primera invocación y lo destruye cuando ya no es necesario. Esto hace posible que el contenedor controle la forma segura todo el

proceso de ejecución y que el componente se beneficie de los servicios que le ofrece el contenedor.

Los contenedores han sido diseñados para la ejecución de aplicaciones COM+, paquetes formados por una colección de componentes que se ejecutan bajo el mismo proceso, dentro del mismo contenedor. La administración y configuración de cada aplicación debe realizarse de manera individualizada. En COM+ disponemos de dos tipos de aplicaciones: aplicaciones biblioteca y aplicaciones servidor. Las aplicaciones de tipo biblioteca residen en el mismo espacio de direccionamiento que la aplicación cliente, sólo pueden ser utilizadas por éste y sólo pueden acceder a algunos servicios del contenedor. Por otro lado, las aplicaciones de tipo servidor se ejecutan en un proceso separado, son accesibles para más de una aplicación cliente y utilizan con ventaja todas las facilidades que ofrece el contenedor.

CAPÍTULO 2

INICIOS DE LA PLATAFORMA .NET.

En este capítulo se darán a conocer los principios de la plataforma .NET, su definición, la forma en que se encontraban los sistemas antes de la plataforma, así como sus objetivos principales, ventajas y desventajas en el uso del desarrollo de sistemas, las diferentes versiones que ya se tienen para utilizarse y cuáles se encuentran en desarrollo.

2.1 El por que surge la plataforma .NET

Para ello se debe saber como funcionaban las anteriores, para lo cual explicaremos el escenario de cómo se trabaja en las plataformas que existían y cuales son las necesidades que surgieron para generar otra más adecuada a los requerimientos actuales en el desarrollo de aplicaciones.

2.1.1 Escenario Actual.

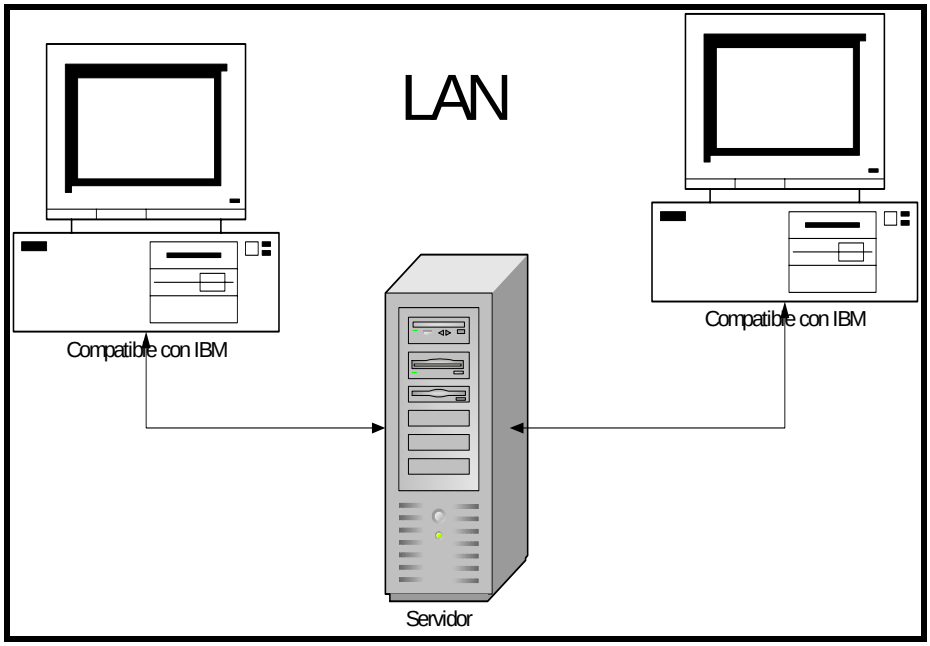


Figura 4. Esquema Cliente/Servidor con Windows DNA.

Es un esquema típico de trabajo en Cliente / Servidor basado en la tecnología COM, el flujo de trabajo de la información, en las organizaciones tenía un ámbito limitado y controlado. Para empezar los servidores y equipos estaban a poco distancia uno del otro y conectados a través de una red de área local, en si las distancias estaban determinadas por la capacidad de transmisión del tipo de cable que se utilizaba y la comunicación se daba prácticamente punto a punto. Tal como se mostró en la figura 4. Otra ventaja era que el número de equipos que conformaban la red, generalmente eran limitados y conocidos, se podía saber en todo momento cuantos equipos contenía la red y cuantos de ellos podían demandar el servicio de las aplicaciones. Otro rasgo característico es que todos los equipos pertenecían generalmente a la misma compañía lo cual permitía tener el control de la compatibilidad de los equipos tanto en Software como en Hardware.

Dentro de esta infraestructura la información viajaba en grandes paquetes de datos binarios, y no había problema de que la información fueran de esta manera, ya que al

ser compatibles todos los equipos de la red, se podían interpretar los paquetes de la misma forma, tampoco había problemas de que los paquetes binarios sean grandes ya que las redes de área local tiene un ancho de banda grande.

Otro aspecto importante se refiere en términos de disponibilidad a que las redes de área local son bastante confiables, si no falla el cableado la comunicación continua se puede garantizar, estos permite la posibilidad de estar permanentemente conectados a las base de datos de la organización sin tener riesgo de caídas en la comunicación y no ocasiona pérdida de datos.

Dentro de este escenario trabajar bajo Windows DNA era el más adecuado, para lo cual se tenían que aprender muchas herramientas y lenguajes por parte de los programadores y acostumbrarse a los caprichos impuestos por los componentes COM que obligaban a una administración compleja del registro de componentes, dar de baja y levantar servicios, interrumpir las operaciones de los sistemas.

2.1.2 Nuevo escenario.

Las aplicaciones tienden a ser globales al igual que los negocios, las máquinas a las que se necesita comunicarse pueden estar muy lejos, incluso en otro continente; por que mantener redes privadas de enlace global sería muy costoso, razón que justifica el uso de Internet como medio de comunicación.

Internet dista mucho de ser una red de área local, es ahí donde los esquemas anteriores de desarrollo pasan hacer obsoletos. De inicio la velocidad de transmisión y el ancho de banda son limitados, por lo que la eficacia de las aplicaciones que no fueron desarrollas para utilizar Internet como medio de comunicación, que antes eran rápidas ahora son lentas.

En este nuevo escenario se desconocen cuantos equipos estarán demandando el servicio de las aplicaciones, ya que estos pueden ser 1, 100 o tanto como los usuarios de Internet que existen en el mundo.

Si se considera que cada cliente exige recursos del servidor, mantener un esquema de conexión permanente con el mismo es costoso, lo que llevaría a instalar supercomputadoras que por su precio no están al alcance de las empresas. Por lo que fue necesario cambiar a esquemas desconectados en el modelo de petición-respuesta que es el más adecuado y permite la adecuada escalabilidad.

Si se trabaja en forma global, se desconoce si los equipos con lo que se necesita comunicar, poseen la misma plataforma tanto en hardware como en software. En este sentidos los paquetes de datos binarios pueden tener diferentes significados dependiendo de la plataforma, por lo cual ya no son funcionales. El problema es aun mayor pues la diversidad es tal, que incluso no se sabe si los dispositivos que consuman los servicios de las aplicaciones son de naturaleza distintos a una computadora.

Otro problema que surge en los paquetes de datos binarios y por lo que no son bien vistos por las redes que componen Internet, es debido al hecho de que son dudosos y requieren ser analizados para determinar sino se trata de un virus o un programa nocivo, desgraciadamente esto provoca que muchos casos los datos validos sean confundidos y rechazados.

Como si esto fuera poco Internet no es un medio de comunicación confiable, entre punto y punto existen grandes distancias y muchos dispositivos, que no sabe cuando se interrumpe la comunicación. Tomando eso en cuenta no puede apostar a que una base de datos pueda estar permanentemente conectada a la aplicación sin correr el riesgo de que un corte en la comunicación nos provoque una perdida de datos.

En la figura 5 se muestra con este nuevo escenario como diferentes tipos de dispositivos se conectan entre sí, sin importar las características de cada una de ellos.

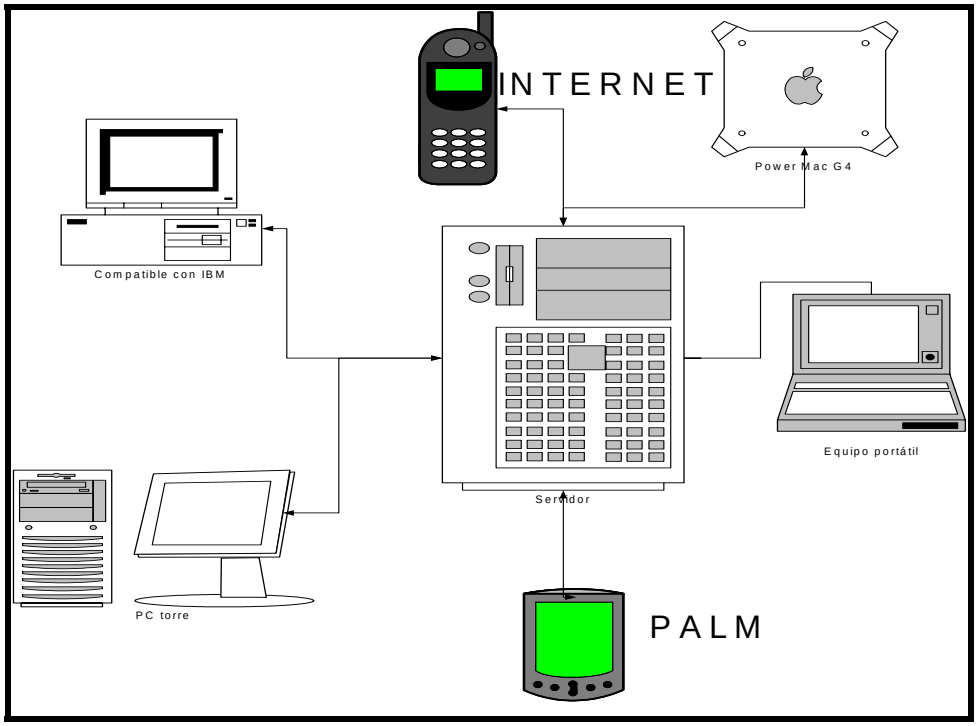


Figura 5 Esquema del nuevo tipo de aplicaciones.

2.1.3 Observaciones de los escenarios.

Las plataformas de desarrollo anteriores no están preparadas para las nuevas formas de manejo de información, e implementar aplicaciones de características globales en estas tecnologías demanda grandes esfuerzos y rinde pobres resultados, en un

escenario en donde los clientes lo han visto todo, los desarrollos limitados están en vías de extinción.

¿Que es lo mínimo que se debe de obtener de una herramienta de desarrollo?

Para las aplicaciones modernas es necesaria una plataforma de desarrollo que de manera sencilla soporte múltiples plataformas de ejecución resolviendo discrepancias de hardware y software de manera automática. Esto permitiría que las aplicaciones puedan ser ejecutadas en diferentes equipos y sistemas operativos lo que es muy recomendable en esquemas globales de trabajo.

Se espera que las aplicaciones cambien sus flujos de información de grandes paquetes binarios a pequeños bloques de información textual basados en protocolos universales como XML que además de ser auto descriptivos sean reconocidos por todos los equipos y sistemas operativos como datos y no como otro cosa. Finalmente se espera que las aplicaciones manejen base de datos en modo desconectados en donde de manera confiable se pueda usarse Internet como nuestra red empresarial todo esto obviamente sin sacrificar la velocidad y los beneficios de las plataformas de desarrollo anteriores, la solución es la plataforma .NET

2.2 Plataforma .NET.

2.2.1 Definición.

Es un conjunto de herramientas de desarrollo y lenguajes de programación, de propósito general, orientado a objetos, de tercera generación, de alto nivel, de compilación a código intermedio, que nos permiten utilizar todos los recursos disponibles en la computadora a través de una librería de clase común con la cual se pueden desarrollar aplicaciones de consola, basadas en Windows y para la Web que utilizan protocolos abiertos para la interacción entre los elementos que las componen [4] [16].

A finales del año 2001 fue presentada la plataforma .NET por Microsoft, la que de forma comercial se liberó a principios del año 2002, así como también dio a conocer que los demás productos de Microsoft utilizarían esta plataforma para su desarrollo. Esta llevaba tres años de estar desarrollando antes de su presentación [1].

La plataforma .NET es el resultado de dos proyectos. La meta del primer proyecto era mejorar el desarrollo de Microsoft Windows, tratando específicamente de mejorar el Modelo de Objetos de Componentes (COM) de Microsoft. El segundo proyecto tenía la intención de crear una plataforma para integrar el software como un servicio. El producto terminado mejora de manera importante la productividad del programador, ofrece facilidad de implementación, ejecución confiable de la aplicación e introduce un concepto totalmente nuevo para la computación: el de los servicios Web XML, aplicaciones acopladas libremente y componentes diseñados para la computación

heterogénea de hoy al comunicarse utilizando protocolos y estándares de Internet como SOAP y XML.

2.2.2 Los objetivos al desarrollar la plataforma .NET

- Proveer un ambiente de desarrollo orientado a objetos, consistente para código almacenado y ejecutado localmente, pero distribuido en Internet o ejecutado remotamente.

- Proveer un ambiente que permita simplificar la entrega de software entre versiones.

- Proveer un ambiente de ejecución seguro.

- Proveer un ambiente de ejecución que resuelva los problemas de ambientes interpretados o basados en scripts que permitan modificar el modelo de programación en el cliente y servidor que genere código conectado en el servidor como destino estándar.

- Hacer consistente la experiencia del desarrollador entre distintos tipos de aplicaciones: Windows, Web, dispositivos, consola.

- Construir todas las comunicaciones en estándares de la industria para asegurar que se integren a otros códigos, con estándares como los Web Services Interoperables, creados por un consorcio para muchas empresas, impidiendo que estas tecnologías se hagan propietarias.

2.2.3 Algunas novedades introducidas en la plataforma .NET.

- Nuevo lenguaje de programación C#. Basado en C++ y Java, pretende ser un lenguaje sencillo y potente, diseñado específicamente para la creación de componentes .NET.

- Soporte para servicios Web. SOAP es un protocolo de integración que se utiliza para interconectar aplicaciones y sistemas a través de la Web. Es independiente del lenguaje y de la plataforma, por lo que funciona igualmente bien con sistemas operativos Unix, Linux, Windows, Solaris, HP-UX, etc. El estándar SOAP permite ofrecer servicios ejecutables de manera remota a través de la Web.

- Biblioteca de componentes Win Forms y Web Forms para el desarrollo de interfaces gráficas de usuario bajo Windows y la Web, respectivamente. Lamentablemente, la biblioteca Web Forms no es utilizable desde el lenguaje C++.

- ASP.NET. Nueva versión de la tecnología de Paginas Activas de Servidor (ASP), que posibilita el acceso a los componentes como servicios Web.

- ADO.NET. Nueva versión de la tecnología de Objetos de datos ActiveX (ADO). Esta biblioteca de componentes proporciona un modelo de acceso a datos sencillo y homogéneo, con el que se puede acceder a múltiples fuentes de datos, entre otras, bases de datos relacionales, jerárquicas y documentos XML.

2.2.4 Diferentes versiones.

Actualmente existen 3 versiones de la plataforma .NET.

a) Versión 1.0: Fue liberada a principios del año 2002, e incluía la versión 1.0 del .NET Framework, la versión 2002 de Visual Studio y varios lenguajes de programación nuevos compatibles con la plataforma (como C# .NET y Visual Basic .NET)[1].
b) Versión 1.1: Fue liberada en 2003, aproximadamente un año después de su antecesora, esta versión introdujo 1.1 del .NET Framework junto con visual Studio .NET 2003, la primera versión del .NET Compact Framework y un nuevo lenguaje de programación llamado J# .NET.
c) Versión 2.0: Fue liberada a finales del año 2005, y es la primer gran renovación que sufrió la plataforma en su tiempo de vida. Con la idea de ser una evolución en lugar de una revolución, esta versión trajo consigo las versiones 2.0 del .NET Framework y del .NET Compact Framework y un nuevo Visual Studio.

2.2.5 Requerimientos de sistemas operativos para la plataforma .NET

Las aplicaciones .NET se pueden desarrollar sobre los sistemas operativos Windows 2000 y Windows XP y las aplicaciones de entrega final permite redistribuir libremente el .NET Framework en sistemas con estos requerimientos:

a) Cliente.

Microsoft Windows 98, 98 Second Edition, ME, NT 4.0 SP6.0a, Windows 2000, Windows XP.

b) Servidor.

Microsoft Windows 2000 SP2 Professional, Server o Advanced Server, Windows XP professional, Windows Server 2003.

Una aplicación Web basada 100% en servidor no requiere nada en el cliente excepto un explorador de Internet. Para crear aplicaciones clientes avanzados se recomienda utilizar el .NET Framework, que ofrece la mayor interactividad, capacidad de operar

de forma desconectada al servidor y el resto de beneficios de distribución automática de software.

CAPÍTULO 3

CAPAS DE LA PLATAFORMA .NET

En este capítulo se darán a conocer las capas que conforman la plataforma .NET, se indicarán los elementos que conforman a cada una de estas, así como la manera en que interactúan, cuáles son sus funciones primordiales y que papel desempeñan dentro de la plataforma .NET.

3. Capas de la plataforma .NET

Las capas que conforman la plataforma .NET son tres, la primera es la capa de lenguajes, continúa la capa del .NET Framework y por último la capa de sistemas operativos tal como se muestra en la figura 6[4].

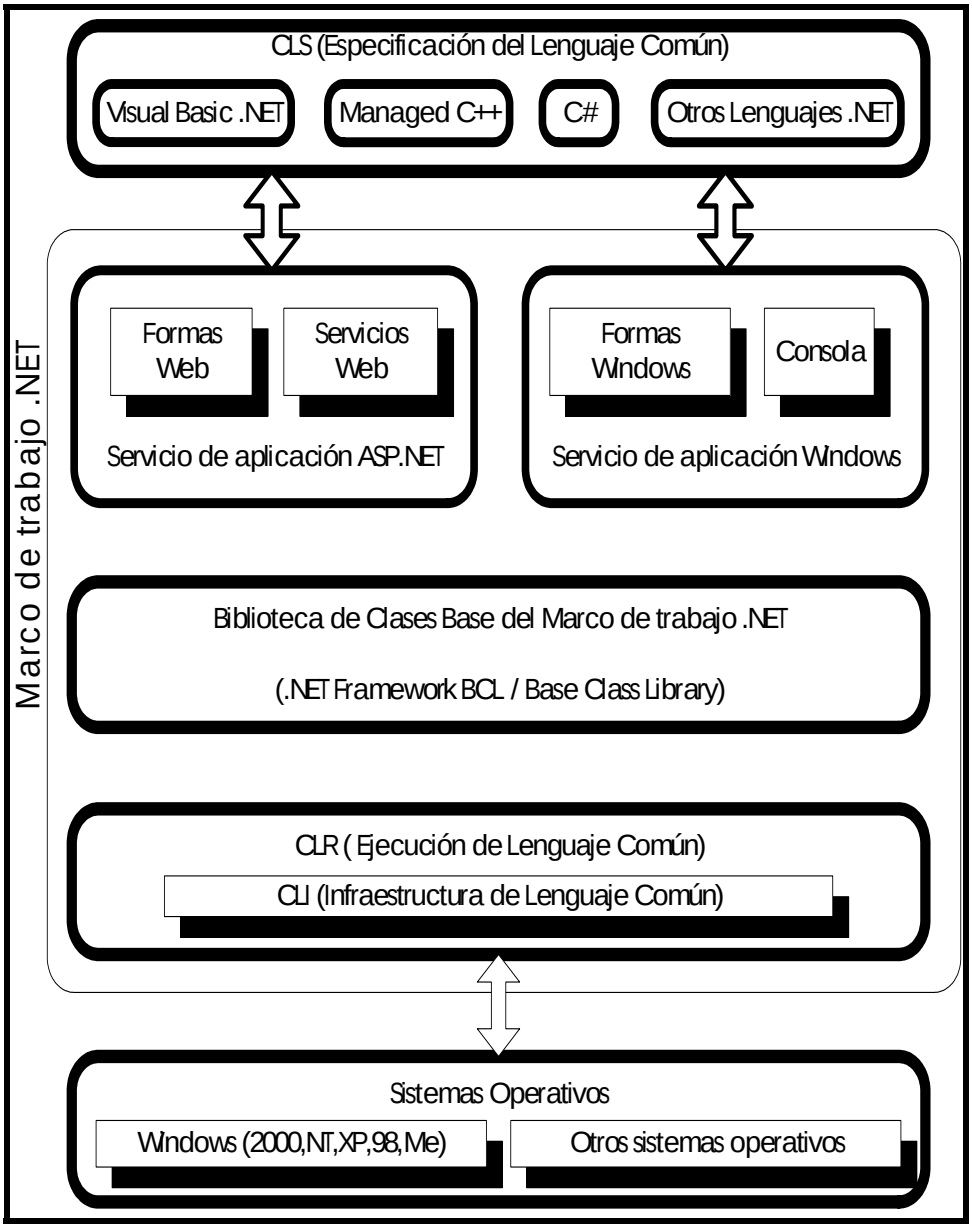


Figura 6 Capas de la plataforma .NET

3.1 Capa de Lenguajes .NET.

Es la capa compuesta por el CLS y todos los lenguajes de programación compatibles con CLS.

3.1.1 CLS (Especificación del Lenguaje Común).

CLS es un conjunto de reglas que debe de satisfacer el compilador de un lenguaje para que sea capaz de generar código MSIL compatible con el entorno .NET. Si una herramienta cumple las cuarenta y una reglas definidas por CLS, el código generado podrá utilizarse desde cualquier otra herramienta compatible con .NET

Para poder interactuar completamente con otros objetos, sea cual sea el lenguaje en que se hayan implementado, los objetos se deben exponer a los llamadores sólo aquellas características que sean comunes para todos los lenguajes con los que deben ínter operar en CLS. Las reglas de CLS definen un subconjunto del sistema de tipos común , es decir, todas las reglas aplicables al sistema de tipos común se aplican a CLS, excepto cuando se definan reglas más estrictas en CLS.

CLS ayuda a mejorar y garantizar la interoperabilidad entre lenguajes mediante la definición de un conjunto de características en las que se pueden basar los programadores y que están disponibles en una gran variedad de lenguajes.

CLS también establece los requisitos de compatibilidad con CLS; estos requisitos permiten determinar si el código administrado cumple la especificación CLS y hasta qué punto una herramienta dada admite la programación de código administrado que utilice las características de CLS.

Si un componente sólo utiliza las características de CLS en la API que expone a otro código (incluidas las clases derivadas), se garantiza que se puede obtener acceso al componente desde cualquier lenguaje de programación que admita CLS. Los componentes que cumplen las reglas de CLS y usan sólo las características incluidas en CLS se conocen como componentes compatibles con CLS.

La mayoría de los miembros definidos por los tipos de la biblioteca de clases del marco de trabado .NET son compatibles con CLS. Sin embargo, algunos tipos de la biblioteca de clases tienen uno o más miembros que no son compatibles con CLS. Estos miembros permiten el uso de características de lenguaje que no se encuentran en CLS. Los tipos y miembros que no son compatibles con CLS se identifican como tales en la documentación de referencia y, en todos los casos, existe una alternativa compatible con CLS.

CLS se diseñó de manera que fuese lo suficientemente amplio como para incluir las construcciones de lenguaje que normalmente necesitan los programadores y lo suficientemente pequeño como para que todos los lenguajes pudieran admitirlo. Además, se ha excluido de CLS cualquier construcción de lenguaje de la que no se

puede verificar rápidamente la seguridad de tipos del código, de manera que todos los lenguajes compatibles con CLS pueden generar código que es posible comprobar.

Se deberá de tener en cuanta que, en la práctica, no es posible generar código intermedio a partir de cualquier lenguaje de programación. El lenguaje de origen tendrá que satisfacer las características funcionales o filosofía del lenguaje original. El lenguaje común es un lenguaje orientado a objetos, con comprobación estricta de tipos y otras características avanzadas de la gestión automática de memoria. Algunos lenguajes de programación no se adaptan a este modelo. Para convertirlos en lenguajes .NET se tendrá que modificar tanto al lenguaje que terminaría siendo irreconocible.

3.1.2 Tipos de lenguajes .NET.

Existen dos tipos de lenguajes .NET, los consumidores y los extensores.

a) Consumidores.

Un lenguaje .NET se llama consumidor si presenta la compatibilidad mínima necesaria con CLS para utilizar los tipos de la biblioteca estándar del marco de trabajo .NET y otras bibliotecas de objetos del entorno. La finalidad de los lenguajes consumidores es poder acceder a la funcionalidad de los tipos del entorno .NET, no construir nuevos tipos de datos. Los candidatos que mejor encajan dentro de este grupo son los lenguajes interpretados o de guión como Jscript.NET.

b) Extensores.

Los lenguajes extensores deben cumplir en su totalidad la especificación del lenguaje común, lo que les permitirá acceder al entorno del objeto .NET y extenderlo con nuevas clases, tipos y componentes. Algunos ejemplos de este tipo de lenguajes son Visual C++ .NET, Visual C# y Visual Basic .NET, incluidos en el entorno.

3.1.3 Algunos lenguajes .NET.

Con este nuevo modelo de trabajo, Microsoft permite que otros fabricantes de software creen sus propias herramientas y adaptaciones de lenguajes compatibles con .NET.

Para evitar cualquier suspicacia, se ha remitido la especificación de CLS y el lenguaje C# al comité de estandarización y evolución del lenguaje Javascript.

Se ha anunciado la existencia de compiladores compatibles .NET para mas de 40 lenguajes de programación, entre los cuales están, APL, COBOL, Component Pascal, Eiffel, Fortran, Java, Pascal, Perl, Pitón, RPG, SmallTalk, entre muchos mas.

Los lenguajes .NET compilan a un código intermedio llamado MSIL (Microsoft Intermediate Language, Lenguaje intermedio de Microsoft), que no se puede ejecutar directamente en ninguna maquina.

3.1.4 Lenguaje intermedio de Microsoft (MSIL).

Cuando se compila a código administrado, el compilador convierte el código fuente en Lengua intermedio de Microsoft (MSIL), que es un conjunto de instrucciones independiente de la CPU que se pueden convertir de forma eficaz en código nativo. MSIL incluye instrucciones para cargar, almacenar, inicializar y llamar a métodos en los objetos, así como instrucciones para operaciones lógicas y aritméticas, flujo de control, acceso directo a la memoria, control de excepciones y otras operaciones.

Antes de poder ejecutar código, se debe convertir MSIL al código específico de la CPU, normalmente mediante un compilador Just-In-Time (JIT). CLR proporciona uno o varios compiladores JIT para cada arquitectura de equipo compatible, por lo que se puede compilar y ejecutar el mismo conjunto de MSIL en cualquier arquitectura compatible.

Cuando el compilador produce MSIL, también genera metadatos. Los metadatos describen los tipos que aparecen en el código, incluidas las definiciones de los tipos, las firmas de los miembros de tipos, los miembros a los que se hace referencia en el código y otros datos que el motor de tiempo de ejecución utiliza en tiempo de ejecución. El lenguaje intermedio de Microsoft (MSIL) y los metadatos se incluyen en un archivo ejecutable portable (PE), que se basa y extiende el PE de Microsoft publicado y el formato Common Object File Format (COFF) utilizado tradicionalmente para contenido ejecutable. Este formato de archivo, que contiene código MSIL o código nativo así como metadatos, permite al sistema operativo reconocer imágenes de CLR. La presencia de metadatos junto con el Lenguaje intermedio de Microsoft (MSIL) permite crear códigos auto descriptivos, con lo cual las bibliotecas de tipos y el Lenguaje de definición de interfaces (IDL) son innecesarios. El CLR localiza y extrae los metadatos del archivo cuando son necesarios durante la ejecución.

3.2 La capa del Marco de trabajo .NET.

Es la capa de .NET Framework esta compuesta por el núcleo de servicios y recursos de .NET, que incluye los compiladores, la biblioteca de clases comunes para todos los lenguajes, y los servicios que convierten lo codificado en los lenguajes en código máquina para los diversos sistemas operativos, a través del uso de código intermedio.

3.2.1 Objetivos del marco de trabajo .NET

.NET Framework es una capa informática que simplifica el desarrollo de aplicaciones en un entorno altamente distribuido como es Internet. El diseño del marco de trabajo .NET está enfocado a cumplir los objetivos siguientes:

- Proporcionar un entorno coherente de programación orientada a objetos, en el que el código de los objetos se pueda almacenar y ejecutar de forma local, ejecutar de forma local pero distribuida en Internet o ejecutar de forma remota.

- Proporcionar un entorno de ejecución de código que reduzca lo máximo posible la implementación de software y los conflictos de versiones.

- Ofrecer un entorno de ejecución de código que garantice la ejecución segura del mismo, incluso del creado por terceras personas desconocidas o que no son de plena confianza.

- Proporcionar un entorno de ejecución de código que elimine los problemas de rendimiento de los entornos en los que se utilizan secuencias de comandos o intérpretes de comandos.

- Ofrecer al programador una experiencia coherente entre tipos de aplicaciones muy diferentes, como las basadas en Windows o en el Web.

- Basar toda la comunicación en estándares del sector para asegurar que el código del marco de trabajo .NET se puede integrar con otros tipos de código.

3.2.2 Características del marco de trabajo .NET.

Algunas de las características mas importantes del marco de trabajo .NET son las siguientes:

a) Gestión de memoria.

El entorno de ejecución de .NET gestiona la memoria automática con la ayuda de un proceso recolector de basura (garbage collector). La tarea permanente del recolector de basura, abreviado GC por sus iniciales en inglés, consiste en buscar objetos que se encuentran en memoria pero ya no se utilizan, proceder a su destrucción y liberar la memoria que ocupaban[7].

Hasta ahora, el entorno de ejecución nativo de un programa escrito en C++ mantenía dos estructuras para la gestión de la memoria durante la ejecución del programa: la pila (stack) y el montón (heap). En la pila se almacenan los objetos y estructuras de datos de tipo automático o estático, así como los parámetros de invocación y el resultado de cualquier llamado a función o método. En el heap solo se almacenan los objetos construidos dinámicamente durante la ejecución del programa con el operador new.

Los objetos de tipo automático construidos en la pila se destruirán automáticamente cuando el objeto deje de utilizarse. Sin embargo, los objetos creados de manera dinámica en el heap son responsabilidad del programador que los ha construido, por lo que debe ser éste quien proceda a su destrucción con el operador delete en el momento que dejen de ser útiles. Si el programador olvida eliminar estos objetos, la memoria quedará ocupada de manera irreversible hasta que termine la ejecución del programa. Este tipo de errores son muy frecuentes y se conocen como fugas de memoria (memory leakage).

El entorno .NET añade una nueva estructura de memoria llamada montón administrado (manager heap). El motor de ejecución gestiona de manera transparente al programador este nuevo heap. El nuevo operador new de .NET nos permite construir objetos de una manera dinámica y asignarlos a este heap. Además, ya no será necesario eliminar estos objetos cuando en el programa no quede ninguna referencia al objeto, es decir, cuando se debe de utilizar, el recolector de basura destruirá el objeto y recuperará la memoria automáticamente para su uso posterior.

El recolector de basura se activara cada cierto tiempo o cuando la cantidad de memoria disponible en el heap administrado sea inferior a un determinado umbral. En este momento se buscaran todos los objetos del heap que no estén referenciados en el programa y se destruirán. El algoritmo empleado para la búsqueda de objetos candidatos a ser eliminados es de tipo generacional, basada en la evidencia empírica de que los objetos de recién creación son también son los que mas probabilidad tienen de dejar de utilizarse. Por lo cual este algoritmo analiza primero los lugares de los objetos mas jóvenes y en ultimo lugar los mas antiguos.

Además se tiene la posibilidad de eliminar los objetos de manera manual cuando el programador lo necesite.

b) Interoperabilidad.

Siempre que se introduce una nueva plataforma de programación, en este caso .NET, nos salta la duda: ¿ Que sucede con todo el código y los programas desarrollados con versiones anteriores del compilador ¿ En estos casos, la interoperabilidad con el código y componentes legados es de importancia

capital. Visual C++ .NET soporta varios mecanismos para reutilizar todo el código nativo no compatible con .NET.

La manera mas sencilla de reutilizar la funcionalidad que ya esta escrita presenta dos variantes, se dispone de una biblioteca de código DLL ya compilada y funcionando, solo será necesario utilizar los métodos contenidos en la DLL, como ya se hacia en versiones anteriores de Visual C++.

Si lo que necesita es reutilizar código escrito en C++ se está de suerte, ya que Visual C++ .NET permite mezclar en la misma aplicación código administrado (cuya gestión de memoria es automática) y código no administrado. Solo hay que copiar el código antiguo y pegarlo en el nuevo programa.

Otro caso importantísimo de interoperabilidad es el relacionado con la proliferación de componentes COM, COM+ y ActiveX. Una parte importante de la funcionalidad de Windows solo es accesible a través de componentes, por lo que se hace necesario un mecanismo para utilizarlos desde las aplicaciones .NET. La manera mas sencilla de utilizar los componentes COM desde el entorno .NET consiste en crear una clase .NET que encapsule la funcionalidad del componente. Afortunadamente, se incluye una utilidad tlbimp.exe, capaz de generar automáticamente un ensamblado a partir de un fichero con la descripción del tipo COM que quiera utilizar.

Las utilidades tlbexp.exe y regasm.exe nos, el primero es capaz de generar un fichero con la descripción del tipo COM equivalente a un ensamblado dado, la segunda nos permite registrar directamente un ensamblado como si de un componente COM se tratase.

c) Seguridad.

Existen tres capas superpuestas que gestionan la seguridad de las aplicaciones .NET.

1) Seguridad de Windows.

La seguridad de Win32 se basa en la obtención de tokens (identificadores) de acceso a determinados recursos. Una vez obtenido un identificador de acceso, y antes de poder utilizar el recurso deseado, la seguridad de Windows comprobará si nuestro token de acceso es válido consultando una Lista de Control de Acceso o ACL(Access Control List). Además, COM+ añade otro mecanismo por el que se puede configurar el proceso donde se ejecuta un componente para que adopte la identidad de un usuario con privilegios distinguidos.

2) Seguridad .NET.

El motor de ejecución de .NET verifica la seguridad del código que va a ejecutar basándose en la identidad del propio código, además de la identidad del usuario. Los criterios utilizados por .NET para determinar los permisos que otorgará al código se llaman evidencias. La URL de descarga del componente, el nombre del fabricante, el nombre completo del ensamblado, etcétera. Las evidencias del código, junto con las políticas de seguridad (configuración de la maquina, usuario y dominio), definen los permisos concretos que se concederán al código durante su ejecución.

3) Seguridad programática.

Se apoya en la política de administración de usuarios y grupos de usuarios bajo Windows. Con la ayuda de algunas clases de la biblioteca .NET nuestro programa podrá verificar el nombre y tipo de usuario que lo está ejecutando y los grupos a los que pertenece el usuario. Además, se puede aplicar seguridad de tipo declarativa, mediante la inclusión de atributos de seguridad en el código. Los atributos de seguridad especifican sobre el propio código que grupos de usuarios pueden ejecutar cada parte del código.

d) Tipos.

Todo es un tipo de dato en el entorno .NET. Así, con los lenguajes consumidores se podrá utilizar los tipos definidos en el entorno y, si se usa los lenguajes extensores, podría crear nuestros propios tipos de datos. Los tipos pueden ser de 2 clases: los de tipos valor y los tipos de referencia, dependiendo de cómo se construyan y como se gestione la memoria que ocupan.

Las clase .NET son tipos definidos por el programador que representan algunas características conocidas para los programadores de C++, como los constructores, métodos, operadores y datos miembro. Se diferencian de las clases C++ en que solo soportan herencia simple entre clases .NET, aunque si soportan la herencia múltiple de un tipo especias de clases denominadas interfaces.

Por lo demás, las clases .NET soportan la mayoría de los conceptos de la programación orientada a objetos: el poliformismo, las clases abstractas, que son clases base no utilizadas directamente y de las que solo se pueden derivar, las clases selladas, que solo pueden utilizarse y no extender, etcétera.

Las excepciones son un tipo especial de clases que se utilizan para señalizar condiciones específicas de error. Sustituyen con ventaja a los sistemas clásicos de gestión de errores basados en la devolución de códigos de error.

Los tipos referencia se construyen en el heap mientras que los tipos de valor se construyen en la pila. Normalmente, las clases se utilizan por referencia, mientras que los tipos básicos se utilizan directamente, como si fueran valores.

A continuación en la Tabla 1 se mencionan los tipos básicos de .NET, indicándose entre paréntesis su tamaño de almacenamiento en bits:

Nombre	Cantidad de Bits
Boolean	8
Char	16
Byte	8
Sbyte	8
Int16	16
UInt16	16
Int32	32
Uint32	32
Int64	64
Uint64	64
Single	32
Double	64
Decimal	64

Tabla1 Tipos de variables en .NET.

e) Atributos.

Los atributos son una de las características mas importantes del entorno .NET. Un atributo es una etiqueta entre corchetes que se añade en el código fuente. Se trata de un mecanismo básico para extender la funcionalidad de los lenguajes y entorno .NET.

Algunos atributos son manejados directamente por el compilador; sin embargo, la interpretación de otros atributos corre a cargo de bibliotecas específicas (DLLs), que proporcionan funcionalidad extendida. Por último, otros atributos se incluyen como metainformación en el código MSIL compilado, por lo que se pueden leer desde el código.

Las técnicas de programación basada en atributos se llaman programación declarativa. El programador, en lugar de escribir un trozo de código que resuelve un problema específico, añade ciertos atributos conocidos por el

compilador. Este los detecta y genera el código necesario en sustitución de los atributos.

De este modo, se pueden definir cierta funcionalidad sin escribir ni una sola línea de código, añadiendo algunos atributos en lugares específicos. Se pueden desarrollar nuestras propias bibliotecas de atributos a medida de aumentar la productividad y ahorrar código.

f) Interfaces.

Las interfaces son un potente mecanismo de la programación orientada a objetos que se utiliza para especificar la funcionalidad ofrecida por un tipo. Una interfaz no contiene almacenamiento, tan solo define un colección de métodos o funciones que debe de satisfacer el tipo que implemente la interfaz. Las interfaces tampoco contienen código, son simples marcadores de la funcionalidad esperada de un tipo concreto.

Los tipos .NET solo pueden derivar (extender la funcionalidad) de un tipo base. Sin embargo, lo que si pueden hacer es implementar mas de una interfaz al mismo tiempo.

Las interfaces son el mecanismo ideal para separar la especificación de un tipo (qué debe hacer) y su implementación (Como lo hace).

g) Delegados.

Un delegado .NET es una clase que encapsula una o varias funciones a las que se invocará indirectamente a través de un delegado.

Cuando se construye un delegado, se debe indicar el objeto y el método al que se tiene que invocar. También es posible encadenar llamadas a diferentes métodos de distintos objetos.

Un delegado simple es aquel que invoca a un único método, mientras los múltiples son aquellos que ejecutan mas de una acción de forma encadenada.

h) Eventos.

Los eventos son metainformación del entorno .NET. Un evento es un mensaje que se genera ante determinados estímulos y que, en muchos casos, es externo al propio programa que lo recibe. Como ejemplo se puede considerar el evento que se genera al hacer clic con el ratón sobre una ventana.

Los delegados y los eventos están íntimamente relacionados, de hecho, la gestión de eventos en la plataforma .NET se realiza invocando al delegado

asociado del evento. Se dice que una clase genera eventos si contiene algún delegado marcado.

Gracias a los delegados, se pueden separar la implementación de la clase que genera el evento del resto de clases interesadas en recibir una notificación cuando se genere un evento. Los eventos se usan de manera generalizada en toda biblioteca de clases .NET y, de manera especial, en la biblioteca Windows Forms.

3.2.3 Tipos de aplicaciones permitidas en el marco de trabajo .NET

El marco de trabajo .NET facilita la construcción, despliegue y ejecución de diversos tipos de aplicaciones [4].

a) Aplicaciones de consola.

Las aplicaciones de consola son aquellas aplicaciones textuales que reciben y muestran datos en el intérprete de comandos.

b) Aplicaciones Windows.

Las aplicaciones Windows son aquellas aplicaciones que tienen interfaz grafica basada en Windows. Existen aplicaciones cliente, tanto para ordenadores como para otros dispositivos (organizadores personales, consolas de videojuegos, sintonizadores de televisión por cable, teléfonos celulares,…). Estas aplicaciones se ejecutan directamente sobre el entorno .NET, que a su vez se apoya en el sistema operativo de la maquina cliente.

c) Aplicaciones Web.

Las aplicaciones Web son aquellas aplicaciones que utilizan para su ejecución un navegador Web como Internet Explorer. El crecimiento de este tipo de aplicaciones se encuentra en pleno auge debido a sus innumerables ventajas.

Pueden dar servicio a múltiples usuarios concurrentes, no necesitan una instalación especifica y se puede tener acceder a ellas desde cualquier maquina que tenga acceso a Internet.

Estas aplicaciones se ejecutan bajo la supervisión del motor de ejecución .NET dentro del servidor de páginas de Microsoft, IIS (Internet Information Server, Servidor de Información Internet).

d) Servicios Web.

Los servicios Web constituyen una tecnología de integración que permite la interconexión de sistemas complejos a través de Internet. Un servicio Web es

una pequeña aplicación que ofrece una funcionalidad remota accesible para otras aplicaciones cliente a través de la red.

3.2.4 Elementos que contiene el marco de trabajo .NET.

3.2.4.1 Capa de Servicios.

Se compone de aquellos servicios que permiten la intercomunicación entre los programas desarrollados en un lenguaje .NET, el resto de los elementos del marco de trabajo .NET , y los componentes del sistema operativo.

Se tienen dos servicios principales:

a) **Servicios de aplicaciones Windows.**

   En este servicio se apoyan a las aplicaciones en su dialogo con .NET Framework y el sistema operativo, aquí es donde se da servicio a las aplicaciones de consola y aplicaciones Windows.

b) **Servicios de aplicaciones ASP.NET.**

   En este servicio se apoya a las aplicaciones en su dialogo con .NET Framework con el sistema operativo, y además con el motor de servicios Web, que en plataforma Microsoft se trata de Internet Information Server (IIS), aquí es donde se da servicio a las aplicaciones Web y Servicios Web.

3.2.4.2 Biblioteca de clases base (BCL).

Se trata de más de 6000 clases, distribuidas en más de 80 archivos de librería DLL, que el usuario puede utilizar en sus programas, aportando funcionalidad intrínseca en el lenguaje. Prácticamente el desarrollador solo se tendrá que preocupar por codificar los procesos de negocio [4][7].

Durante el proceso de instalación del .NET, los ensamblados con las bibliotecas estándar se registran automáticamente en la GAC (Global Assembly Cache, Cache de Ensamblados Globales), que es el lugar donde residen los ensamblados.

En la biblioteca de clases del framework contiene las clases básicas necesarias para construir cualquier aplicación. En ellas se incluye funcionalidad estándar para la gestión de la entrada/salida, el manejo de cadenas de texto, la gestión de la seguridad, las comunicaciones en la red, la programación concurrente y la creación de interfaces gráficas de usuario.

Las clases agrupadas bajo el nombre ADO.NET soportan el almacenamiento y la gestión de información de manera persistente e incluyen clases para la manipulación de fuentes de datos estructuradas SQL (Structured Query Language, Lenguaje de

Consulta Estructurado) a través de interfaces estándar SQL. De este modo, se podrá acceder a la información almacenada en cualquier manejador de base de datos, como Oracle, DB2, Sybase y SQL Server. También se incluyen clases, para la manipulación de documentos XML, con soporte a búsquedas y transformación.

Las clases ASP.NET ofrecen una serie de funcionalidades listas para usar en el desarrollo de aplicaciones y servicios Web.

Las bibliotecas de clases Windows Forms y Web Forms se utilizan, para la creación de interfaces de usuario de aplicaciones Windows y aplicaciones Web respectivamente.

Todas las bibliotecas de clases del marco de trabajo .NET ofrecen interfaces consistentes para el desarrollo de aplicaciones en cualquiera de los lenguajes soportados. La excepción a la regla es Web Forms, no utilizable directamente desde C++.

Las bibliotecas de clases .NET proporcionan una funcionalidad global equivalente a todas las interfaces de programación del API Win32 completo. Estas clases se han rediseñado haciendo especial énfasis en su carácter orientado a objetos y resuelven muchos de los problemas comunes que aparecían al trabajar con el API Win32.

a) Espacios de nombre (namespace).

Es un esquema lógico que permite agrupar clases y tipos relacionados a través de un nombre; en términos generales es una referencia lógica a las librerías utilizadas por un programa. Con los espacios de nombre es como se hace referencia a la biblioteca de clases común [4].

3.2.4.3 Motor de ejecución del Lenguaje Común (CLR).

El motor de ejecución del lenguaje común es la pieza mas importante de la plataforma .NET y aquella encargada de ejecutar los programas compilados en código intermedio MSIL.

CLR agrupa compiladores de línea de comando que permite la creación de código intermedio, libre de ambigüedades, al que se le da el nombre de ensamblado; contiene además los compiladores JIT (Compilador justo en tiempo), que se encarga de generar código máquina a partir de los ensamblados. CLR se encarga de la gestión de errores, uso de recursos, y dialogo con el sistema operativo en tiempo de ejecución.

El CLR se puede considerar como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, como la administración de memoria, la administración de subprocesos y la interacción remota, al tiempo que aplica una seguridad estricta a los tipos y otras formas de especificación del código que garantizan su seguridad y solidez.

De hecho, el concepto de administración de código es un principio básico del motor de ejecución. El código destinado al motor de ejecución se denomina código administrado, a diferencia del resto de código, que se conoce como código no administrado.

El marco de trabajo .NET puede alojarse en componentes no administrados que cargan CLR en sus procesos e inician la ejecución de código administrado, con lo que se crea un entorno de software en el que se pueden utilizar características administradas y no administradas. En el marco de trabajo .NET no sólo se ofrecen varios hosts de motor de ejecución, sino que también se admite el desarrollo de estos hosts por parte de terceros.

CLR facilita el diseño de los componentes y de las aplicaciones cuyos objetos interactúan entre lenguajes distintos. Los objetos escritos en lenguajes diferentes pueden comunicarse entre sí, lo que permite integrar sus comportamientos de forma precisa. Por ejemplo, puede definir una clase y, a continuación, utilizar un lenguaje diferente para derivar una clase de la clase original o llamar a un método de la clase original. También se puede pasar al método de una clase una instancia de una clase escrita en un lenguaje diferente. Esta integración entre lenguajes diferentes es posible porque los compiladores y las herramientas de lenguajes orientados al motor de tiempo de ejecución utilizan un sistema de tipos común definido por el motor de ejecución, y los lenguajes siguen las reglas de tiempo de ejecución para definir nuevos tipos, así como para crear, utilizar, almacenar y enlazar tipos.

a) Características de CLR.

- CLR administra la memoria, ejecución de subprocesos, ejecución de código, comprobación de la seguridad del código, compilación y demás servicios del sistema. Estas características son intrínsecas del código administrado que se ejecuta en CLR.

- Con respecto a la seguridad, los componentes administrados reciben grados de confianza diferentes, en función de una serie de factores entre los que se incluye su origen (como Internet, red empresarial o equipo local). Esto significa que un componente administrado puede ser capaz o no de realizar operaciones de acceso a archivos, operaciones de acceso al Registro y otras funciones delicadas, incluso si se está utilizando en la misma aplicación activa.

- El motor de ejecución impone seguridad en el acceso al código. Por ejemplo, los usuarios pueden confiar en que un archivo ejecutable incrustado en una página Web puede reproducir una animación en la pantalla o entonar una canción, pero no puede tener acceso a sus datos personales, sistema de archivos o red. Por ello, las características de seguridad del motor de

ejecución permiten que el software legítimo implementado en Internet sea excepcionalmente variado.

- CLR impone la solidez del código mediante la implementación de una infraestructura estricta de comprobación de tipos y código denominado CTS. CTS garantiza que todo el código administrado es autodescriptivo. Los diferentes compiladores de lenguajes de Microsoft y de terceros generan código administrado que se ajusta a CTS. Esto significa que el código administrado puede usar otros tipos e instancias administrados, al tiempo que se aplica inflexiblemente la fidelidad y seguridad de los tipos.

- El entorno administrado del motor de ejecución elimina muchos problemas de software comunes. Por ejemplo, el motor de tiempo de ejecución controla automáticamente la disposición de los objetos, administra las referencias a éstos y los libera cuando ya no se utilizan. Esta administración automática de la memoria soluciona los dos errores más comunes de las aplicaciones: la pérdida de memoria y las referencias no válidas a la memoria.

- El motor de ejecución aumenta la productividad del programador. Por ejemplo, los programadores pueden crear aplicaciones en el lenguaje que prefieran y seguir sacando todo el provecho del motor de tiempo de ejecución, la biblioteca de clases y los componentes escritos en otros lenguajes por otros colegas. El proveedor de un compilador puede elegir destinarlo al motor de tiempo de ejecución. Los compiladores de lenguajes que se destinan a .NET Framework hacen que las características de .NET Framework estén disponibles para el código existente escrito en dicho lenguaje, lo que facilita enormemente el proceso de migración de las aplicaciones existentes.

- Aunque el motor de ejecución está diseñado para el software del futuro, también es compatible con el software actual y el software antiguo. La interoperabilidad entre el código administrado y no administrado permite que los programadores continúen utilizando los componentes COM y las DLL que necesiten.

- El motor de ejecución está diseñado para mejorar el rendimiento. Aunque CLR proporciona muchos servicios estándar del motor de ejecución, el código administrado nunca se interpreta. Una característica denominada compilación JIT permite ejecutar todo el código administrado en el lenguaje máquina nativo del sistema en el que se ejecuta. Mientras tanto, el administrador de memoria evita que la memoria se pueda fragmentar y aumenta la zona de referencia de la memoria para mejorar aún más el rendimiento.

- Por último, el motor de tiempo de ejecución se puede alojar en aplicaciones de servidor de gran rendimiento, como Microsoft® SQL Server™ e IIS. Esta infraestructura permite utilizar código administrado para escribir lógica empresarial, al tiempo que se disfruta del superior rendimiento de los mejores

servidores empresariales del sector que puedan alojar el motor de tiempo de ejecución

b) Ensamblados.

Los ensamblados componen la unidad fundamental de implementación, control de versiones, reutilización, ámbito de activación y permisos de seguridad en una aplicación basada en .NET.

Los ensamblados adoptan la forma de un archivo ejecutable (.exe) o un archivo de biblioteca de vínculos dinámicos (.dll), y constituyen unidades de creación de .NET Framework.

Proporcionan a CLR la información que necesita para estar al corriente de las implementaciones de tipos.

Un ensamblado puede entenderse como una colección de tipos y recursos que forman una unidad lógica de funcionalidad y que se generan para trabajar conjuntamente.

c) Manifiesto del ensamblado.

Todos los ensamblados contienen un manifiesto del ensamblado. Éste es similar a una tabla de contenido, y contiene la siguiente información:

- La identidad del ensamblado (nombre y versión).
- Una tabla de archivos que describen al resto de archivos que componen el ensamblado, incluidos, por ejemplo, otros ensamblados creados por el usuario de los que dependa el archivo .exe o .dll, e incluso archivos de mapa de bits o archivos Léame.
- Una lista de referencias de ensamblado, es decir, una lista de todas las dependencias externas, archivos .dll u otros archivos necesarios para la aplicación que otros usuarios hayan podido crear. Las referencias de ensamblado contienen referencias a objetos globales y privados. Los objetos globales residen en la caché de ensamblados global, un área disponible para otras aplicaciones, parecida al directorio System32. Los objetos privados deben encontrarse en un directorio del mismo nivel o inferior al directorio de instalación de la aplicación.

d) Funciones de los ensamblados.

Los ensamblados son una parte fundamental de la programación con .NET Framework. Un ensamblado realiza las funciones siguientes:

1) **Contiene el código que ejecuta CLR.**

El código del lenguaje intermedio de Microsoft (MSIL) de un archivo ejecutable portable (PE) no se ejecuta si no tiene asociado un manifiesto de ensamblado. Recuerde que cada ensamblado sólo puede tener un punto de entrada (es decir, **DllMain**, **WinMain** o **Main**).

2) **Crea un límite de seguridad.**

Un ensamblado es la unidad en la que se solicitan y conceden los permisos. Para obtener más información acerca de los límites de seguridad en lo que respecta a los ensamblados.

3) **Crea un límite de tipos.**

La identidad de cada tipo incluye el nombre del ensamblado en que reside. Por ello, un tipo MyType cargado en el ámbito de un ensamblado no es igual que un tipo denominado MyType cargado en el ámbito de otro ensamblado.

4) **Crea un límite de ámbito de referencia.**

El manifiesto del ensamblado contiene los metadatos del ensamblado que se utilizan para resolver tipos y satisfacer las solicitudes de recursos. Especifica los tipos y recursos que se exponen fuera del ensamblado. El manifiesto también enumera otros ensamblados de los que depende.

5) **Forma un límite de versión.**

El ensamblado es la unidad más pequeña de CLR; todos los tipos y recursos del mismo ensamblado pertenecen a la misma versión. El manifiesto del ensamblado describe las dependencias de versión que se especifiquen para los ensamblados dependientes.

6) **Crea una unidad de implementación.**

Cuando se inicia una aplicación, sólo deben estar presentes los ensamblados a los que llama la aplicación inicialmente. Los demás ensamblados, como los recursos de localización o los ensamblados que contengan clases de utilidad, se pueden recuperar a petición.

De este modo, se puede mantener la simplicidad y transparencia de las aplicaciones la primera vez que se descargan.

Es la unidad que permite la ejecución simultánea. La ejecución simultánea es la capacidad de almacenar y ejecutar múltiples versiones de una aplicación o

componente en el mismo equipo. Esto significa que se pueden tener varias versiones del motor de tiempo de ejecución y varias versiones de las aplicaciones y los componentes que utilicen una versión del motor de tiempo de ejecución, simultáneamente y en el mismo equipo.

La ejecución simultánea ofrece un mayor control sobre las versiones de un componente a las que se enlaza una aplicación, y sobre la versión del motor de tiempo de ejecución que utiliza una aplicación.

El hecho de que se admita el almacenamiento y la ejecución simultáneos de distintas versiones del mismo ensamblado es una parte integral de la creación de nombres seguros incluida en la infraestructura del motor de tiempo de ejecución.

Debido a que el número de versión del ensamblado con nombre seguro forma parte de su identidad, el motor de tiempo de ejecución puede almacenar múltiples versiones del mismo ensamblado en la caché de ensamblados global y cargar esos ensamblados en tiempo de ejecución.

Aunque el motor de tiempo de ejecución permite crear aplicaciones simultáneas, la ejecución simultánea no es automática.

e) Tipos de ensamblados.

Los ensamblados pueden ser estáticos o dinámicos.

1) **Los ensamblados estáticos**.

Estos pueden incluir tipos de .NET Framework (interfaces y clases), así como recursos para el ensamblado (mapas de bits, archivos JPEG, archivos de recursos, etc.). Los ensamblados estáticos se almacenan en el disco, en archivos ejecutables portables PE. También se puede utilizar .NET Framework para crear ensamblados dinámicos.

2) **Los ensamblados dinámicos.**

Son los que 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.

f) Como se utilizan los ensamblados.

Para utilizar un ensamblado, debe agregarle una referencia. A continuación, utilice la instrucción **Imports** para elegir el espacio de nombres de los elementos que desea utilizar Una vez que se ha hecho referencia a un ensamblado y se ha importado, todas las clases, propiedades, métodos y otros miembros que permiten el acceso de sus

espacios de nombres estarán disponibles para la aplicación, como si su código formara parte del archivo de código fuente. Un ensamblado individual puede contener varios espacios de nombres, y cada espacio de nombres puede contener una agrupación de elementos distinta, incluidos otros espacios de nombres.

g) Compilación Just In Time (JIT, Justo a Tiempo) de .NET Framework.

Para poder ejecutar el lenguaje intermedio de Microsoft (MSIL), primero se debe convertir éste, mediante un compilador Just-In-Time (JIT) de .NET Framework, a código nativo justo en el momento de su ejecución, que es el código específico de la CPU que se ejecuta en la misma arquitectura de equipo que el compilador JIT. CLR proporciona un compilador JIT para cada arquitectura de CPU compatible, por lo que los programadores pueden escribir un conjunto de MSIL que se puede compilar mediante un compilador JIT y ejecutar en equipos con diferentes arquitecturas. No obstante, el código administrado sólo se ejecutará en un determinado sistema operativo si llama a las API nativas específicas de la plataforma o a una biblioteca de clases específica de la plataforma.

La compilación JIT tiene en cuenta el hecho de que durante la ejecución nunca se llamará a parte del código. En vez de utilizar tiempo y memoria para convertir todo el MSIL de un archivo ejecutable portable (PE) a código nativo, convierte el MSIL necesario durante la ejecución y almacena el código nativo resultante para que sea accesible en las llamadas posteriores. El cargador crea y asocia un código auxiliar a cada uno de los métodos del tipo cuando éste se carga. En la llamada inicial al método, el código auxiliar pasa el control al compilador JIT, el cual convierte el MSIL del método en código nativo y modifica el código auxiliar para dirigir la ejecución a la ubicación del código nativo. Las llamadas posteriores al método compilado mediante un compilador JIT pasan directamente al código nativo generado anteriormente, reduciendo el tiempo de la compilación JIT y la ejecución del código.

El motor de ejecución proporciona otro modo de compilación denominado generación de código en el momento de la instalación. El modo de generación de código en el momento de la instalación convierte el MSIL a código nativo, tal y como lo hace el compilador JIT normal, aunque convierte mayores unidades de código a la vez, almacenando el código nativo resultante para utilizarlo posteriormente al cargar y ejecutar el ensamblado. Cuando se utiliza el modo de generación de código durante la instalación, todo el ensamblado que se está instalando se convierte a código nativo, teniendo en cuenta las características de los otros ensamblados ya instalados. El archivo resultante se carga e inicia más rápidamente que si se hubiese convertido en código nativo con la opción JIT estándar.

Como parte de la compilación MSIL en código nativo, el código debe pasar un proceso de comprobación, a menos que el administrador haya establecido una directiva de seguridad que permita al código omitir esta comprobación. En esta comprobación se examina el MSIL y los metadatos para determinar si el código

garantiza la seguridad de tipos, lo que significa que el código sólo tiene acceso a aquellas ubicaciones de la memoria para las que está autorizado. La seguridad de tipos ayuda a garantizar que los objetos están aislados entre sí de manera segura y, por tanto, protegidos contra daños involuntarios o maliciosos. Además, garantiza que las restricciones de seguridad sobre el código se aplican con toda certeza.

El motor de ejecución se basa en el hecho de que se cumplan las siguientes condiciones para el código con seguridad de tipos comprobable:

- La referencia a un tipo es estrictamente compatible con el tipo al que hace referencia.

- En un objeto sólo se invocan las operaciones definidas adecuadamente.

- Una identidad es precisamente lo que dice ser.

Durante el proceso de comprobación, se examina el código MSIL para intentar confirmar que el código tiene acceso a las ubicaciones de memoria y puede llamar a los métodos sólo a través de los tipos definidos correctamente. Por ejemplo, un código no permite el acceso a los campos de un objeto si esta acción sobrecarga las ubicaciones de memoria.

Además, el proceso de comprobación examina el código para determinar si el MSIL se ha generado correctamente, ya que un MSIL incorrecto puede llevar a la infracción de las reglas en materia de seguridad de tipos. El proceso de comprobación pasa un conjunto de código con seguridad de tipos definido correctamente, y pasa de forma exclusiva código con seguridad de tipos. No obstante, algunos códigos con seguridad de tipos no pasan la comprobación debido a las limitaciones de este proceso, y algunos lenguajes no producen código con seguridad de tipos comprobable debido a su diseño. Si la directiva de seguridad requiere código con seguridad de tipos y el código no pasa la comprobación, se inicia una excepción al ejecutar el código.

3.3 Capa de Sistemas operativos.

Es la capa compuesta por los sistemas operativos y las herramientas del mismo, que están preparadas para sacar el mejor producto de los desarrollos en .NET. En este punto se agrupan productos independientes del Back End [4].

CAPÍTULO 4

SERVICIOS WEB EN LA PLATAFORMA .NET

En este capítulo se dará a conocer la forma en que se realizan los servicios Web de XML, se desarrollarán los conceptos de un servicio Web, el porqué surgen, como ayudan y cuales son los pasos a seguir para construir un servicio Web con .NET. Los servicios Web de XML constituyen uno de los principios básicos de la iniciativa .NET, ya que estos permiten un grado de interoperatividad entre aplicaciones a través de Internet que antes eran inimaginables.

4.1 Introducción

Los servicios Web son el avance de los componentes hacia Internet. Durante muchos años (incluso hoy en día), cuando se necesitaba realizar la ejecución remota de una función o procedimiento (en una arquitectura potencialmente distinta a la que invoca la ejecución) se empleaban técnicas de Llamada a procedimiento remoto (RPC). El desarrollo de nuevas tecnologías mejoró sustancialmente el panorama de la ejecución remota, como, por ejemplo, en el caso de las bibliotecas XTI. La aparición de Internet potencio, de un modo impensable hasta entonces, la interconexión de los equipos y la filosofía de trabajo basado en el paradigma cliente/servidor. Además las Interfaces de Pasarela Común (CGI), supusieron un mecanismo de integración muy popular en su momento, lo que llevó a la popularización de este tipo de servicios.

En esencia, los servicios Web están compuestos por una serie de componentes .NET que están en ejecución en maquinas donde existen conexiones con Internet. De esta forma, los servicios de estos componentes quedan expuestos hacia el exterior para que otros usuarios puedan utilizarlos. Durante años, tecnologías como COM han permitido desarrollar aplicaciones basada en componentes, cuyos servicios eran solicitados por clientes que estaban dentro de la Intranet. Sin embargo, gracias a los servicios Web, hoy en día resulta muy sencillo extender esos servicios mas allá de los límites de la Intranet y ofrecerlos a clientes externos. Como se puede ver, todas estas tecnologías buscan el mismo objetivo primordial: la integración de distintos sistemas informáticos.

Entre los puntos fuertes de esta tecnología se destacan:

- La utilización de protocolos de transporte estandarizados, no propietarios y con una difusión muy amplia, como el protocolo HTTP o el SMTP.

- La codificación de la información se realiza utilizando XML, de forma que cualquier aplicación que sea capaz de manejar información en este formato podrá solicitar servicios de forma remota, sin importar la ubicación o el tipo de arquitectura que utilice. Se podrán utilizar como parámetros de los métodos instancias de clases propias, siempre que puedan ser serializadas.

- Se mantiene un nivel de seguridad similar al alcanzado con el modelo DCOM, a pesar de utilizarse un medio de transporte público como la red Internet.

Hailstorm es el nombre de un conjunto de aplicaciones basadas en servicios Web desarrolladas por Microsoft. Su fin es ofrecer un punto único de acceso centralizado a datos personales, tales como agenda de contactos, citas, fotografías, los sitios favoritos en el navegador Web, la carpeta de Mis Documentos, etcétera. Todo ello, disponible desde Internet, y para lo cual protocolos de transporte estándares y codificación de los datos en XML, por lo que ya no es necesario plantearse el uso de

una aplicación cliente propietaria para acceder a toda esta información. Se pueden usar todos estos servicios para mantener una identidad digital en la red, accesible desde cualquier punto en lugar de almacenarla en un disco duro local. Sin duda, esta tecnología demuestra, de forma práctica, el potencial de la computación distribuida y los servicios Web. Sin embargo, las nuevas tecnologías no están exentas de ciertos problemas. En el modelo de computación tradicional en PC monopuesto, el rendimiento de las aplicaciones esta en relación directa con la capacidad del hardware del que se dispusiese. Con la computación distribuida a escala global se tiene que asumir un enfoque diferente, ya que los cuellos de botella dependen de las condiciones de tráfico de datos en Internet, la disponibilidad de los servidores y de los problemas de seguridad.

4.2 Definición de Servicio Web.

Los servicios Web son componentes software que permiten a los usuarios, usar aplicaciones de negocio que comparten datos, con otros programas modulares vía Internet. Son aplicaciones independientes de la plataforma que pueden ser fácilmente publicadas, localizadas e invocadas mediante protocolos Web estándar, como XML, SOAP, UDDI o WSDL. El objetivo final es la creación de un directorio de online de servicios Web, que pueda ser localizado de un modo sencillo y que tenga una alta fiabilidad.

Los servicios Web son la revolución informática de la nueva generación de aplicaciones que trabajan colaborativamente en las cuales el software esta distribuido en diferentes servidores.

Los servicios Web XML son los bloques de construcción de la computación distribuida en el Internet. Usted puede crear soluciones al usar los múltiples servicios de Web XML desde varias fuentes que trabajan en conjunto independientemente de dónde residan o cómo fueron implementadas.

4.3 Arquitectura.

Para poder comprender la arquitectura del sistema de acceso entre clientes y servicios Web será preciso se definan los siguientes términos:

- Canales: Son el medio utilizado entre los clientes y los servidores de los servicios Web para el envío de mensajes. En la plataforma .NET existen dos canales: HTTP y TCP; no obstante, cualquiera puede ampliar esta lista implementando sus propios protocolos de comunicación o adaptando los protocolos existentes. HTTP utiliza, de forma predeterminada, el protocolo SOAP para la codificación de la información, mientras que TCP utiliza una codificación binaria mucho más eficiente.

- Formateadores: Son los encargados de serializar los objetos de las aplicaciones, para que se puedan codificar y transmitir a través de la red en un formato independiente de la plataforma. Posteriormente, realizarán la descodificación del formato serializado y volverán a recomponer los objetos. La plataforma .NET cuenta con dos serializadores: Binary y SOAP (Ambos se encuentran en la biblioteca System::Runtime::Serialization::Formatters).

- Contexto del entorno remoto. Es una sección de la memoria del sistema en el que se almacenan los objetos que comparten propiedades comunes. Las llamadas de objetos que se encuentren en contextos diferentes, deberán realizar el salto a través de un proxy. Las clases que se pueden ligar a un contexto se denominan clases enlazadas a un contexto, y se les pueden aplicar atributos especiales de contexto. Los objetos que se pueden enlazar a un contexto deben extender la clase System::ContextBoundObject.

- Metadatos. Se utilizan para generación dinámica de las clases proxy. Estas clases con las que deseamos conectar, pero interceptarán sus llamadas para poder canalizarlas a través del canal elegido y utilizarán para ello un formateador asociado.

- Archivos de configuración. En estos archivos se almacena todo la información específica del entorno remoto, para garantizar su independencia de la implementación del código.

4.3.1 Consumo de servicios Web.

Dado que un servicio Web es, en esencia, un componente que se ejecuta en un servidor Web, resulta lógico pensar que utilizará los protocolos HTTP o TCP para comunicarse con sus clientes. Sin embargo, disponemos de tres formas de codificar la invocación de esos métodos: los métodos GET y POST del Protocolo de Transferencia de Hipertexto (HTTP) y el Protocolo Sencillo de Acceso a Objetos (SOAP).

Los métodos HTTP se utilizan habitualmente para transmitir información entre cliente Web y servidor, por lo que resultan de uso muy común hoy en día. Sin embargo, existen sutiles diferencias entre sus variantes. Mientras que el método GET añade los parámetros en la llamada URL de invocación (siendo, por tanto, visible en el cuadro de texto Dirección del navegador Web), el método POST los codifica en la cabecera HTTP de la solicitud. Así, los datos están ocultos al cliente y son menos susceptibles a modificaciones fortuitas o malintencionadas.

Sin embargo, el protocolo SOAP funciona de forma completamente diferente, ya que, aunque utiliza el mismo mecanismo de transporte, codifica la información ya no en pares nombre/valor, sino utilizando mensajes XML. Así, la información que se mueve entre el cliente y el servidor es mucho mas rica y extensible. Con este

protocolo es posible manejar estructuras de información realmente complejas y, por tanto, muy complicadas de manejar si no se dispone de las bibliotecas necesarias.

4.3.2 Descripción de los servicios (WSDL).

Cuando un servicio Web contiene una serie de servicios, resulta evidente que tiene que proporcionar información acerca de ellos a los posibles clientes que dispongan a utilizarlos. Existe un lenguaje de descripción de servicios Web basados en XML denominado Lenguaje de Descripción de Servicios Web (WSDL) que servirá para describir cada servicio. En este archivo se deberán especificar:

- Los mecanismos de transporte que acepta el servicio.
- Los tipos de datos empleados en los mensajes.
- Los métodos disponibles, así como sus parámetros de entrada y salida.

4.3.3 Descubrimiento de servicios Web.

Una vez que se han generado los servicios Web y tras su despliegue en el servidor, se debe preparar un mecanismo para que los potenciales clientes sepan se su existencia. Este mecanismo consiste en la publicación de una serie de archivos especiales en el servidor Web llamados contratos. En realidad, este paso solamente debe darse cuando se desee que los servicios Web sean de acceso público. Si no se desea que terceras partes se enteren de la existencia de determinados servicios Web, la publicación debe ser limitada.

El descubrimiento de servicios Web es una técnica destinada a que potenciales clientes busquen los servicios de un servidor determinado y obtengan una lista pormenorizada de los mismos, así como su descripción. Existen tres técnicas que podemos plantear en este contexto:

- Descubrimiento estático. Se utiliza un archivo disco que contiene una referencia a un URL con el WSDL del servicio Web.

- Descubrimiento dinámico. En lugar de utilizar una referencia explícita a la descripción del servicio Web mediante un URL, en el servidor se especifica una dirección donde se almacena un archivo de descubrimiento dinámico. Este archivo le dice al servidor que explore todos los servicios disponibles en los directorios descendientes se ese URL. Además, en el archivo se pueden incluir aquellos directorios que deseamos que no sean explorados dinámicamente (para garantizar la privacidad de los servicios).

- Uso del protocolo de Descripción, Descubrimiento e Integración Universal (UDDI). Este ofrece un servicio de directorio para servicios Web, no al nivel de un servidor en concreto. Se trata de unas páginas amarillas de los servicios Web. Su función es publicar los servicios propios, así como descubrir los servicios ofertados por otras empresas colaboradoras. El directorio UDDI es

un sistema de almacenamiento de información acerca de servicios Web que actúa de forma distribuida (de forma similar a como lo hacen los servidores DNS de Internet). En este directorio podemos tanto registrar nuestros servicios, como buscar servicios de terceros mediante un conjunto de funciones predefinidas a las que se accede con el protocolo SOAP.

La figura 7 muestra la interacción de los componentes de servicios Web.

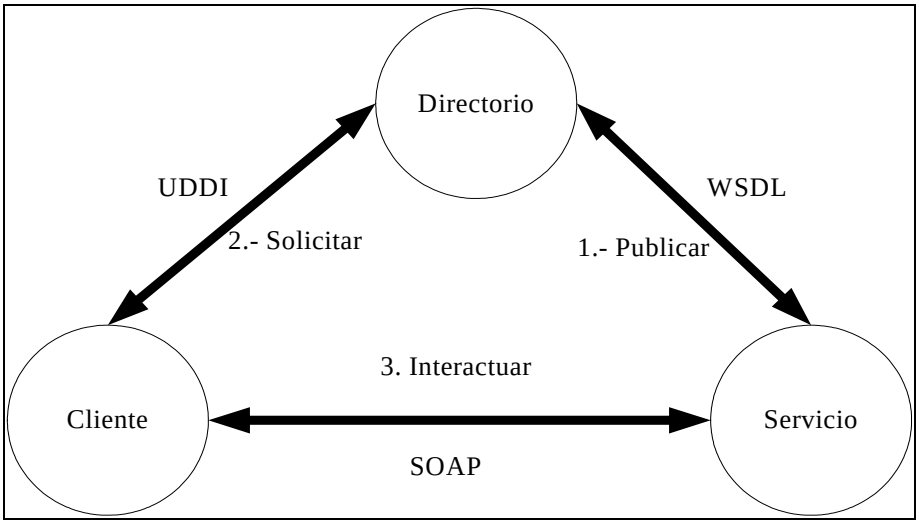


Figura 7 Interacción en un Servicio Web

4.4 Proveedor de servicios Web.

Los servidores Web están basados en la filosofía cliente/servidor, donde un programa cliente realiza una solicitud a un proveedor de servicios para que ejecute una rutina por el y le devuelva el resultado. Con el marco de trabajo .NET es posible generar aplicaciones que proporcionen servicios Web, las cuáles ofrecen su capacidad de ejecución a otros programas usando para ello un protocolo estandarizado como puede ser HTTP o SMTP (entre otros),

También podremos crear aplicaciones que consuman estos servicios. Sin embargo, hay que hacerse una reflexión profunda en este momento: ¿están diseñados los servicios Web para que se comuniquen dos aplicaciones escritas con Visual Studio .NET y utilizando código administrado? Evidentemente, no. Tal y como se dijo en la introducción, las técnicas de ejecución remotas están enfocadas a la integración de sistemas; estos sistemas se entienden de distintas naturalezas, de distintos fabricantes e incluso con distintas arquitecturas de microprocesador. Lo verdaderamente interesante es poder crear servicios que permitan que las aplicaciones Windows interactuar con procesos clientes ejecutados bajo Solaris o AS/400. Este tipo de

interconexiones son las que dan valor real a esta tecnología. Si, por añadidura, se permite conectar dos aplicaciones .NET en una Intranet mediante servicios Web , bienvenido sea, pero recalcando que hay formas mejores de conectar soluciones .NET entre si.

Para escribir servicios Web lo primero que se necesita es tener un servidor Web configurado y en ejecución. El único servidor Web compatible con .NET es el Servicio de información en Internet de Microsoft, o IIS. Por tanto, se requiere que verificar que este software funcione perfectamente. A continuación se entrará en Visual Studio .NET y se generará un nuevo proyecto de tipo Servicio Web C++ Administrado.

Para este ejemplo, se utilizará como nombre de proyecto Ejemplo.

Una vez generado se puede apreciar, que se han generado automáticamente algunos archivos. Vease cuales son y cual es su función:

Ejemplo.cpp y ejemplo.h. Estos archivos contienen la definición de una clase con un método. Se cambia el código de la clase de ejemplo por el de la clase que contenga los métodos que se quieren ofertar en el servicio Web. Un detalle importante es que la clase está contenida dentro de un espacio de nombres, ya que si no se hiciera, se tomaría el nombre del espacio de nombre por defecto (establecido en http://tempuri.org), con los problemas que supondrían los conflictos de nombre si dos métodos se llamasen igual perteneciendo a dos programas diferentes. El espacio de nombre hará únicos los métodos y garantizará su identidad.

Ejemplo.asmx. Este archivo contiene al menos una directiva del tipo <%Webservice>. Adicionalmente, el archivo asmx puede contener el código fuente del servicio Web o vincularse con el código de otro archivo. En el caso del asistente, el código fuente de la clase que contiene los métodos está en los archivos ejemplo.cpp y ejemplo.h, por lo que se tendrá que hacer una inclusión de estos archivos.

AsemblyInfo.cpp. Metainformación referente a la construcción del programa.

Ejemplo.vsdisco. El archivo de descubrimiento generado automáticamente por Visual Studio. Su uso es opcional, ya que si no es colocado en el directorio virtual de IIS, los clientes externos no podrán saber que servicios Web se ofertan en el servidor.

ReadMe.txt. Contiene unas breves instrucciones textuales acerca del proyecto recién generado.

Web.Config. Permite modificar la configuración por defecto del IIS para el directorio que contenga a este archivo, con los valores codificados en xml que se expresan en el. Es interesante porque en él se expresa el nivel de seguridad.

4.4.1 El archivo .asmx

Este es el archivo que se invocará desde el cliente, por lo tanto debe contener información precisa para que el servicio de ASP.NET sea capaz de ejecutar correctamente los servicios Web contenidos. Todo archivo .asmx comienza con un directiva <WebSevice>. En el ejemplo generado por el asistente, tiene este aspecto:

<%@ webservice class=ejemplo.class1 %>

Esta directiva puede tener varios parámetros:

Class. Indica el nombre de la clase que implementa el servicio Web.

CodeBehind. Indica que archivo contiene el código fuente de la clase. Es opcional y se utiliza cuando se requiere una compilación dinámica sin tener que declarar el código fuente en el propio archivo asmx.

Debug. Indica si la compilación debe incluir los símbolos de depuración en el archivo generado. También es opcional.

Language. Indica el lenguaje de programación en el que está expresada la clase que implementa el servicio Web. Es opcional y solo aplica cuando se desea una compilación dinámica y el código fuente está incluido dentro del archivo asmx.

Si no se utiliza la inclusión de código mediante CodeBehind, se deben declarar las clases, que contienen los métodos que conformaran el servicio Web. Para mas detalles acerca de esta declaración, consulte la siguiente sección.

Un servicio Web puede estar escrito en cualquiera de los lenguajes soportados por la arquitectura .NET. ASP.NET es capaz de manejar servicios Web cuyo código no se encuentra compilado, invocando una compilación dinámica en tiempo de ejecución. Los resultados de la compilación dinámica se guardarán en una caché de código compilado. Además, es posible invocar los métodos contenidos dentro de módulos ya compilados, con el consiguiente ahorro de tiempo y recursos en el servidor Web. Si se especifica el código fuente del servicio Web, a continuación de la cláusula <%@WebService> o bien se utiliza el parámetro CodeBehind, se estará forzando a una compilación dinámica del código en tiempo de ejecución.

4.4.2 Implementación

Los archivos de código deben contener una clase que derive de la clase WebService. En dicha clase, además, se habrán tenido que etiquetar varios métodos. Estos serán los que conformen el servicio Web y permitan su invocación remota.

Los metadatos que podemos especificar a un método del servicio Web son los siguientes:

- BufferResponse (verdadero o falso). Indica si se debe almacenar en una memoria intermedia la respuesta del método.

- CacheDuration. Indica la duración de segundos que debe permanecer en la memoria intermedia la respuesta de un método.

- Description. Literal con una descripción textual acerca del cometido del método.

- EnableSession. (verdadero o falso). Le indica al servidor Web si la invocación está ligada a la gestión de una sesión de usuario. Dado que los servicios Web utilizan entre otros protocolos de transporte el HTTP (que no tiene un soporte nativo de sesión), ésta puede simularse mediante el uso de testigos o identificadores de sesión.

- MessageName. Los servicios Web tienen problemas ante los mecanismos de algunos lenguajes como C++ que permiten la sobrecarga de funciones y métodos (esencialmente, tener dos métodos que se llamen igual y que el compilador distingue en función de su lista de parámetros). Este atributo permite dar un nombre inequívoco a cada método para evitar este problema.

- TransactionOption. Indican si la invocación del método debe realizarse generando una nueva transacción o utilizando una existente. Por defecto, no se utilizan transacciones.

4.4.3 La especificación WSDL.

Todos servicio Web debe tener asociada una descripción que indique a los posibles clientes cuáles son los servicios, sus nombres, así como cuáles y de que tipo son sus parámetros. Esta se encontrará en un archivo formateado en XML con diversas partes. Es importante conocer el significado de estas partes, ya que la construcción de clientes en otras plataformas distintas a .NET depende en gran medida de la compresión de dicho archivo. Un archivo WSDL está constituido de los siguientes elementos:

- Types (Tipos). Define los tipos de datos que se usan en el servicio.

- Message (Mensaje).Contiene una descripción abstracta de los datos intercambiados entre cliente y servicio. Cada mensaje está compuesto por varias secciones <part>.

- Port Type (Tipo de Puerto). Contiene un conjunto abstracto de operaciones de una o más operaciones.

- Operation (Operación). Cada uno de los métodos del servicio Web. Se utilizan elementos <message> para indicar los parámetros de entrada y salida de los métodos.

- Binding.(Vinculación). Es la unión de un nombre de método, un protocolo de transporte y los tipos de mensaje de entrada y salida que se intercambian.

- Service (Servicio). Colección de puntos de terminación del servicio, por ejemplo; HTTP GET, HTTP POST, y SOAP.

- Port (Puerto). Punto de terminación del servicio, define el protocolo, la especificación del formato de los datos y la dirección de red donde se alberga el servicio Web.

4.5 Seguridad

En el mundo de los servicios Web, la seguridad procede de dos fuentes que deben complementarse. Dado que estos servicios son programas cuya entrada y salida se recibe a través de un medio de transporte como HTTP, la primera medida de seguridad se aplica al propio servicio Web. Como si de una página Web se tratara, estos servicios pueden ser públicos o privados y, gracias a la configuración del IIS, es posible delimitar quién puede acceder a ellos y quién no. Por otro lado, la seguridad también reside en los propios servicios. Mediante programación directa se puede controlar la identidad del usuario que accede al servicio, permitiendo o impidiendo la ejecución del código.

Cuando se este diseñando un servicio Web que forme parte de una plataforma de comercio electrónico B2B (Bussines to Bussines o entre empresas), se debe restringir el rango de direcciones IP con acceso a un determinado directorio virtual del servidor Web. Las únicas direcciones IP con acceso a los servicios Web deben corresponder con las que utilicen los usuarios de las empresas asociadas. Esto se logra mediante la configuración de IIS o con el uso de Firewalls.

Para modificar el nivel de seguridad mínimo que necesita un cliente al acceder a un servicio Web se recurre a la utilidad de administración de IIS y se pulsa sobre Propiedades. En la pestaña Seguridad del archivo se cambia el control de autenticación y acceso anónimo pulsando el correspondiente botón de modificar . Aparece un cuadro de dialogo similar al de la figura 9.7, donde se observa que, por defecto, está permitido el acceso anónimo a un servicio Web. Esto significa que cualquier persona que acceda el archivo .asmx podrá usar los servicio ocultos tras, sin que se verifique su identidad; es muy recomendable desactivar el acceso anónimo y limitarlo únicamente a aquellos usuarios para quienes este pensado.

La seguridad en el ámbito de la aplicación es mucho mas robusta, ya que permite llegar a niveles de implementación de seguridad que no sean estándar en un servicio Web. La forma mas sencilla de implementar un buen mecanismo de seguridad es combinar las técnicas de criptoprogramación con los esquemas basados en desafío/respuesta. Idealmente es posible añadir a todos los métodos del servicio Web un nuevo parámetro que actúe como testigo de acceso. Este testigo deberá formarse mediante la invocación de métodos especiales en el servicio Web que sean de acceso público. La forma mas habitual de generar un testigo de acceso es generar un número aleatorio que se codifica usando una clave conocida por el servicio Web y los usuarios legítimos del servicio. Para validar al cliente, se envía el numero aleatorio codificado y una transformación. Si el cliente es legítimo, sabrá descifrar el numero aleatorio y aplicarle la transformación (puede ser una simple suma aritmética o una codificación compleja). Este numero transformado actuará como testigo. Al invocar a los métodos protegidos con ese testigo, el sistema podrá determinar si se trata de un acceso legítimo o no. Visual Studio .NET cuenta con una extensa biblioteca de componentes de programación criptográfica. Lleva incorporadas implementaciones de algoritmos de seguridad de gran difusión como DES, 3DES, RC2 o RSA, así como algoritmos de hashing o cálculo de residuos como SHA o MD5, tanto para clases administradas como no administradas. Todos los algoritmos pueden aplicarse a un flujo de datos llamado CrytoStream, aunque también se pueden implementar nuevos algoritmos, con lo que la extensibilidad del modelo está asegurada.

El atributo mode del elemento authentication puede tomar diversos valores, en función del tipo de autenticación que queramos que aplique IIS:

- None. Como su nombre lo indica, no requiere mecanismo alguno de validación.

- Forms. Todas las solicitudes no autenticas pasan por un formulario de acceso.

- Windows. Utiliza los mecanismos de autenticación que se hayan configurado en IIS (proceso IIS, agrupado o aislado). El establecimiento de estos niveles puede observarse en la figura 9.7

- Passport. Las solicitudes no autenticas se desvían al servicio de validación de Microsoft.

CONCLUSIONES

Con lo tratado en este documento donde se dieron a conocer los conceptos que se deben tomar en cuenta para el desarrollo de aplicaciones del tipo consola, Windows y Web mediante la plataforma de desarrollo .NET, donde se conocieron los elementos que se tienen de ésta, así como su proceso de compilación, la forma como se deben de ejecutar los programas, los diferentes lenguajes de programación que son considerados como lenguajes .NET y las características que deben de cumplir estos, las ventajas y desventajas que se tienen ante otras plataformas de desarrollo.

Y en vista de que los sistemas de información cada día se vuelven más complejos y la necesidad de ser más globales surge la necesidad de realizar herramientas que nos ayuden al máximo para utilizar los recursos que se tienen en las computadoras y la plataforma de desarrollo .NET es un herramienta que nos ayuda a hacer fácilmente aplicaciones tanto tipo consola, Windows y Web.

Por lo que al conocer como está compuesta, cuales son las características que tiene, el porque de su surgimiento nos da la pauta para realizar una mejor elección cuando se realiza el análisis del sistema a desarrollar, para tener un mejor funcionamiento y un producto de software final de calidad. Otra de las cosas que debemos tener en cuentas son las diferencias que existen con otras herramientas como COM, DCOM, etc., para tener las bases necesarias para saber cual de ellas es la más adecuada de acuerdo a los requerimientos con los que debe de cumplir nuestra nueva aplicación.

La plataforma de desarrollo .NET es multilenguajes con los que podemos decir cual de ellos es más conviene utilizar y que pueden ejecutarse bajo cualquier plataforma de sistema operativo.

Donde una de las desventajas de .NET es el alto costo que se debe de tener tanto en software como en el hardware para poder aprovechar eficientemente todos su potencial, aunque aunado a esto Microsoft ésta proporcionando gratuitamente parte del software para el desarrollo así como sus diferentes programas de capacitación con los que cuenta, esto según mi apreciación es para que nosotros como desarrolladores tengas los elemento necesarios para justificar su costo beneficio.

Al realizar el tema de servicios Web con la plataforma .NET es el saber que es una de las grandes ventajas que se tienen es la facilidad de realizarlos, en referencia a lo que antes se manejaba y no eran tan globales, es decir, se manejaban mediante intranets y ahora ya son manejadas vía Internet.

Y todos los estándares que se han ido desarrollando alrededor de ésta con el objetivo de que sean aplicados y no se esté tratando de descubrir cosas que ya están realizadas.

Una de las ventajas más provechosas de la plataforma .NET es sus multilenguajes ya que no tenemos que estar dependiendo de un solo lenguaje en

especial y así nosotros poder decidir con cual estamos más cómodos para desarrollar software de calidad y documentación de los mismos, ya podemos manejar las versiones de cada una de ellas de una forma fácil dentro de un manifiesto en el cual conocemos cual versión tenemos trabajando y cuando de desarrollamos cuales de estas aun no se encuentran en un ambiente de trabajo productivo.

Sin duda esto último nos lleva a tener un mejor control de los proyectos a tal grado de tenerlos bien documentados ya que con los desarrollos de antes la gran mayoría no se cuenta con una documentación ni cuales fueron las versiones que se llevan de este software además de permitir incluir información de los componentes, y versión, dentro del código real. Esto hace posible que el software se instale cuando se lo soliciten automáticamente o sin la intervención del usuario, estas funciones reducen los costos de asistencia para la empresa.

Otra de las ventajas relevantes a mencionar es sin duda su nuevo lenguaje C# (C Sharp), con el cual contamos todo el potencial de un lenguaje orientado a objetos con algunas nuevas variantes como son la exclusión de punteros, C# combina las mejores ideas de lenguajes como C, C++ y Java con la mejoras de productividad de la plataforma .NET y brinda una experiencia de codificación muy productiva tanto para los nuevos programadores como para los veteranos donde se facilita la transición para los programadores de C y C++ y proporciona una manera sencilla de entenderlo para los nuevos programadores.

Así también podemos considerar la extensa biblioteca de clases .NET que nos ayudan a reutilizar el código, donde se contienen código para programar subprocesos, entrada y salida de archivos, compatibilidad para bases de datos, análisis XML y estructuras de datos como pilas y colas, con toda la funcionalidad necesaria para construir aplicaciones tipo Windows y Web listas para usarse.

Dentro del Framework .NET las ventajas que podemos considerar es la gestión automática de la memoria, la interoperabilidad con desarrollos y componentes COM ya existentes, la seguridad, los tipos de datos, los atributos, interfaces, delegados y eventos, además de un sólido sistema de control de errores a cualquier lenguaje que se integre en .NET.

La creación de aplicaciones con .NET brinda mejoras no disponibles en otros lenguajes, como la seguridad, estas medidas de seguridad pueden determinar si una aplicación puede escribir o leer un archivo de disco. Permiten insertar firmas digitales en la aplicación para asegurarse que la aplicación fue escrita por una fuente de confianza.

Con todo esto debemos de lograr aplicaciones Windows mucho más estables y con un mayor grado de seguridad y simplificar el desarrollo de aplicaciones y servicios Web que no solo funcionen en plataformas tradicionales, sino también en dispositivos móviles.

GLOSARIO

ActiveX. Versión mas reciente de la tecnología OLE de Microsoft, la cual permite a las aplicaciones comunicarse entre sí por medio de mensajes transferidos con la ayuda del sistema operativo de la computadora. La versión previa fue COM, ActiveX agrega características diseñadas para permitir la distribución de programas ejecutables, llamados controles, a través de Internet.

ADO.NET- Nueva versión de la tecnología ADO (Actives Data Objects, Objetos de Datos Actives). Esta biblioteca de componentes proporciona un modelo de acceso a datos sencillo y homogéneo, con el que podremos acceder a múltiples fuentes de datos relacionales, jerárquicas y documentos XML.

API (Application Programming Interface, Interfaz de programación de aplicaciones). Conjunto de estándares o convenciones mediante los cuales los programas pueden llamar servicios de red o sistemas operativos específicos. En servidores Web, los estándares o convenciones que permiten un hipervínculo originar una llamada a un programa externo al servidor.

ASP.NET- Nueva versión de la tecnología ASP (Active Server Pages, Paginas Activas de servidor) que posibilita el acceso a los componentes .NET, la codificación de estas paginas en cualquier lenguaje .NET y la implementación sencilla de servicios Web.

Bibliotecas de clases .NET – Es un conjunto de clases incluidas en el entorno de .NET y soportadas por la implementación Windows de .NET. Ofrecen toda la funcionalidad del API estándar de Windows (Win32) de una manera mas elegante y sencilla de utilizar.

CLI (Common Languaje Infrastructure, Infraestructura del Lenguaje Común) – Proporciona la especificación para el código ejecutable .NET y el entorno de ejecución. Incluye el CTS (tipos), CLS (lenguaje), metadatos y entorno de ejecución. Actualmente, es un estándar ratificado por el ECMA, lo que ha permitido que terceras empresas estén trabajando ya en implementaciones de entornos de ejecución .NET independientemente de Microsoft.

CLR (Common Languaje Runtime, Motor de Ejecución del Lenguaje Común) – Es el reponsable de ejecutar el código MSIL resultado de la compilación de un programa escrito en un lenguaje .NET. Su instalación es necesaria para la ejecución de un programa .NET.

CLS (Common Languaje Specification, Especificación del Lenguaje Común) – Conjunto de reglas que debe de satisfacer el compilador de un lenguaje para que sea capaz de generar código MSIL compatible con el entorno .NET. Si una herramienta cumple las cuarenta y una reglas definidas por CLS, el código generado podrá utilizarse desde cualquier otra herramienta compatible con .NET

COM (Component Objetc Model, Modelo de objetos componentes). Estándar desarrollado por Microsoft que permite a objetos intercambiar datos entre sí, incluso si tales objetos se han creado con diferentes lenguajes de programación. COM requiere que el sistema operativo de la computadora esté equipado con OLE, el cual se encuentra implementado al 100% solo en sistemas Windows de Microsoft.

CTS (Common Type Specification, Especificación de Tipos Comunes) – Es el núcleo del entorno .NET, ya que define los tipos de datos elementales del lenguaje común MSIL con el que debe de ser compatible cualquier lenguaje para que pueda compilarse a código intermedio. Los lenguajes .NET deben de permitir la traducción o conversión (automática o manual) entre los tipos de datos nativos del lenguaje y los tipos .NET definidos por el CTS.

ECMA (European Computer Manufacturers Association, Asociación de fabricantes de ordenadores Europeos) – A pesar del nombre, es un organismo europeo de estandarización de tecnologías de la información y las comunicaciones. Recientemente el ECMA ha finalizado la estandarización del lenguaje C# (ECMA-334) y del CLI (ECMA-335).

Ensamblado (assembly) – Paquete desplegable, unidad mínima resultado de la compilación de una biblioteca de clases o una aplicación .NET. El ensamblado contiene el código intermiedio MSIL de la aplicación y un conjunto de metadatos (manifiesto) con información adicional sobre la seguridad del código, origen de la aplicación, autor, etcétera.

Framework .NET – Entorno de programación diseñado para facilitar la construcción de aplicaciones web, aplicaciones cliente y servicios Web. Esta compuesto por un entorno de ejecución (runtime), la biblioteca de clases .NET y los lenguajes .NET.

GAC (Global Assembly Cache, Caché de Ensamblados Globales) – Repositorio de ensamblados comunes y globales, lugar donde se almacenan los ensamblados propios del entorno .NET como la biblioteca de clases y cualquier otro ensamblado común a todas las aplicaciones .NET.

GUI (Graphical User Interface, Interfaz gráfica de usuario). Diseño para la parte de un programa que interactúa con el usuario y que usa iconos para representar las características del programa. Los ambientes operativos Apple Macintosh y Microsoft Windows son GUIs muy populares. Luego de descubrir que la gente reconoce con mas rapidez las representaciones gráficas que las palabras o las frases que lee.

GUID (Globally Unique Identifier, Identificador Único Global). Es un número pseudo-aleatorio empleado en aplicaciones de software. Aunque no se puede garantizar GUID generado sea único, el número total de claves única (2^{128}) es tan grande que la posibilidad de que se genere un mismo número dos veces puede considerarse nula en la práctica.

JIT (Just In Time, Justo a Tiempo) – Se refiere a la tecnología de compilación "justo a tiempo" del motor de ejecución .NET, según la cual, el código MSIL se traduce a código nativo del procesador en el mismo momento de la ejecución de una aplicación .NET

Iunknown.- Es el nombre de la interfaz fundamental en el modelo de objetos componentes (COM). La especificación publicada de COM asigna que por mandato los objetos de COM deben poner como mínimo esta interfaz en ejecución .

Lenguajes .NET- Lenguajes especialmente adaptados para el desarrollo de aplicaciones para el entorno .NET. Estos lenguajes pueden ser tipo consumidor o extensores. Los primeros solo permiten utilizar las facilidades y tipos de entorno .NET mientras que con los segundos se pueden extender las clases y tipos .NET. los lenguajes .NET deben ser compatibles con el sistema de tipos comunes (CTS, Common Type System) y la Especificación del Lenguaje Común (CLS, Common Language Specification). Los lenguajes .NET soportados actualmente por Visual Studio .NET incluyen Visual C++, Visual Basic, Visual C# y Jscript.

Mainframe.- Computadora multiusuario concebida para cubrir los requisitos de computación de grandes empresas. Al principio, el término mainframe se refería al gabinete metálico que alberga a la unidad central de procesos de las primeras computadoras. El termino llego a usarse de manera general para referirse a las enormes computadoras centrales creadas en las décadas de los 50's y 60's para satisfacer las necesidades de control administrativo y de información de grandes empresas. Los mainframe mas grandes pueden utilizar miles de terminales tontas y grandes almacenamientos secundarios.

Microsoft .NET- Conjunto de tecnologías software diseñadas para construir, desplegar y ejecutar aplicaciones y servicios Web. Esta formada por tres partes distintas: herramientas de desarrollo, fundamentalmente Visual Studio .NET, productos para servidores y framework .NET.

MSIL (Microsoft Intermediate Language, Lenguaje intermedio de Microsoft)- Lenguaje intermedio al que se compilan los lenguajes .NET. Es el único lenguaje que es capaz de ejecutar el Motor de Ejecución del Lenguaje Común o CLR, la autentica "maquina" que ejecuta el código .NET.

OLE Iniciales de Vinculación e Incrustación de Objetos. Se trata de un conjunto de estándares desarrollados por Microsoft e incorporados en Windows y MacOS de Apple, que permite a los usuarios crear vínculos dinámicos entre documentos, que se actualizan automáticamente. También sirve para insertar un documento creado con una aplicación en un documento creado con otra.

Recolector de Basura (Garbage Collector) – Proceso que se ejecuta en paralelo y forma parte del motor de ejecución .NET. Su misión consiste en monitorizar continuamente los objetos creados y destruir periódicamente aquellos que ya no se utilicen (abandonados). Utiliza un algoritmo de tipo generacional.

SOAP (Simple Object Access Protocol, Protocolo de Acceso a Objetos Sencillo) – Es un Estándar del W3C (World Wide Web Consortium) que permite la ejecución de servicios remotos para el intercambio de información. Se trata de un protocolo basado en XML formado por tres partes: una especificación de formatos de mensaje de petición y respuesta, las reglas de codificación para la información y un modo de representar los procedimientos remotos.

UDDI (Universal Description, Discovery and Integration, Descripción, Descubrimiento e Integracion Universal) – Protocolo estándar que permite el acceso a directorios para recuperar la información sobre servicios Web XML existentes.

Web Forms – Es la parte de la biblioteca de clases .NET que permite generar interfases de usuario para aplicaciones Web abstrayendo muchos de los detalles del lenguaje de presentación HTML.

Windows Forms – Es la parte de la biblioteca de clases .NET que permite construir interfases graficas de usuario para las aplicaciones Windows .NET.

WSDL (Web Service Description Language, Lenguaje de Descripción de Servicios Web) – Estándar basado en XML que permite la descripción y especificación precisa en un servicio Web, esto es , su punto de acceso, formato exacto de mensaje de petición y respuesta.

XML (eXtensible Markup Language, Lenguaje Extensión basado en Marcas) Es un estándar del W3C (World Wide Web Consortium) – Es un metalenguaje que permite construir documentos estructurados de manera jerárquicas para encapsular, transmitir y compartir información.

BIBLIOGRAFÍA.

1.- Microsoft Corporation, ".NET FRAMEWORK" , Mc Graw Hill, España, 2001.

2.- Ferguson, J., Patterson, B., Beres, J., "LA BIBLIA C# ", Anaya Multimedia, España, 2003

3.- Serrano Pérez, J., "PROGRAMACIÓN CON ASP.NET", Anaya Multimedia, España, 2002.

4.- Ramírez, F., "APRENDA PRACTICANDO VISUAL BASIC .NET", Aprenda Practicando Ediciones, Primera Edición, México, 2004.

5.- Pfaffenberger, B., "DICCIONARIO DE TÉRMINOS DE COMPUTACIÓN", Prentice Hall, México, 1999.

6.- Aho, A., Sethi, R., Ullman, J., "COMPILADORES: PRINCIPIOS, TÉCNICAS Y HERAMIENTAS", Addison Wesley Iberoamericana, Segunda Edición, México,1998.

7.- Jiménez Vadillo, J., "VISUAL C++ .NET", Anaya Multimedia, España, 2003.

8.- Balena, F., "PROGRAMACIÓN AVANZADA CON VISUAL BASIC .NET", Mc Graw Hill, Primera Edición, 2003.

9.- Payne, C., "APRENDIENDO ASP.NET", Prentice Hall, Primera Edición, México, 2002.

10.- Pressman, R., "INGENIERÍA DEL SOFTWARE UN ENFOQUE PRÁCTICO", Mc Graw Hill, Quinta Edición, España, 2002.

11.- http://www.desarrolloweb.com/manuales/48/

12.- http://www.desarrolloweb.com/manuales/54/

13.- http://www.microsoft.com/spanish/msdn/comunidad/dce/curso/net/capitulo1.asp

14.- http://www.microsoft.com/spanish/msdn/comunidad/dce/curso/net/capitulo3.asp

15.- http://www.microsoft.com/spanish/msdn/comunidad/dce/curso/net/capitulo4.asp

APÉNDICE

EJEMPLO DE UN SERVICIO WEB.

Un Servicio Web es un componente de software que se comunica con otras aplicaciones codificando los mensaje en XML y enviando estos mensaje a través de protocolos estándares de Internet tales como el Hypertext Transfer Protocol (HTTP).

Por ejemplo, varias compañías están hoy en día creando Servicios Web que actúan como *front end* para aplicaciones de entrada de órdenes que están residentes internamente en un *mainframe*.

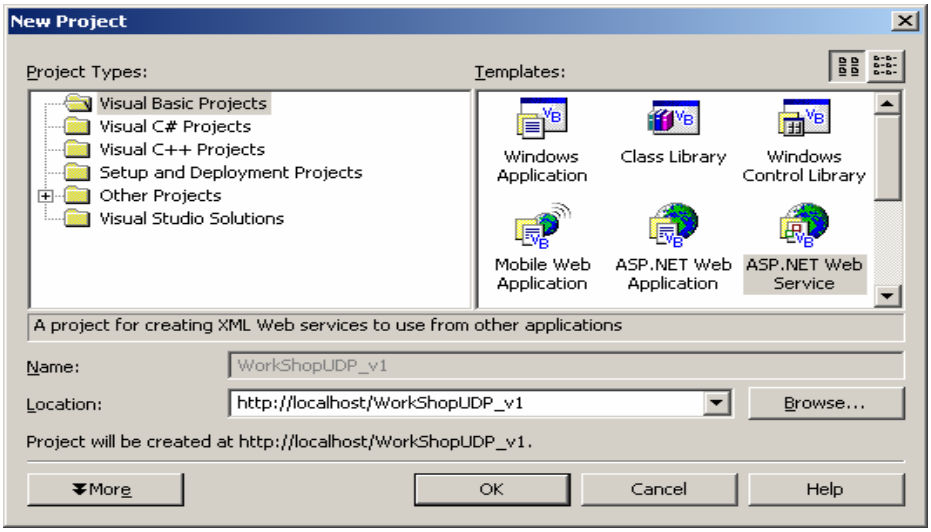
Estas compañías permiten a los sistemas de compras de sus clientes enviar órdenes de compra a través de la Internet.

Poner una capa de sobre las aplicaciones existentes es una solución muy interesante para integrar las aplicaciones desarrolladas por los diferentes departamentos y así reducir los costos de integración.

A Implementación.

A.1 Desarrollo del Servicio Web.

1.) En Visual Studio .Net, creamos un proyecto ASP.Net Servicios Web, llamado "WorkShopUDP_v1"



2.) Eliminar los comentarios (comilla simple) del método HelloWorld() de la clase - service1.
3.) Cambiamos el nombre de la "Service1" por "Saludo"

Antes:

```
Imports System.Web.Services

<Web service(Namespace := "http://tempuri.org/")> _
Public Class Service1
   Inherits System.Web.Services.Web service

#Region " Web Services Designer Generated Code "

   Public Sub New()
      MyBase.New()

      'This call is required by the Web Services Designer.
      InitializeComponent()

      'Add your own initialization code after the InitializeComponent() call

   End Sub

   'Required by the Web Services Designer
   Private components As System.ComponentModel.IContainer

   'NOTE: The following procedure is required by the Web Services Designer
   'It can be modified using the Web Services Designer.
```

```
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region

    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World.
    ' To build, uncomment the following lines then save and build the project.
    ' To test this Web service, ensure that the .asmx file is the start page
    ' and press F5.
    '
    '<WebMethod()> Public Function HelloWorld() As String
    'HelloWorld = "Hello World"
    'End Function

End Class
```

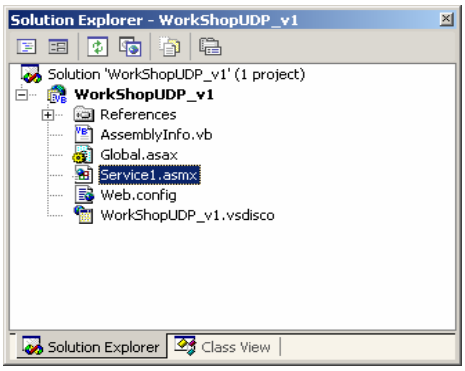
Después:

```
Imports System.Web.Services

<Web service(Namespace:="http://tempuri.org/")> _
Public Class Saludo
    Inherits System.Web.Services.Web service

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent() call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
```

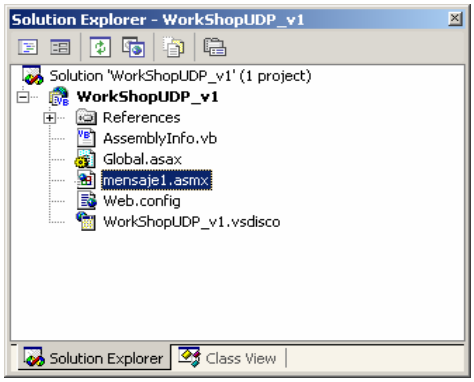
```
        'CODEGEN: This procedure is required by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region

    ' WEB SERVICE EXAMPLE
    ' The HelloWorld() example service returns the string Hello World.
    ' To build, uncomment the following lines then save and build the project.
    ' To test this Web service, ensure that the .asmx file is the start page
    ' and press F5.
    '
    <WebMethod()> Public Function HelloWorld() As String
        HelloWorld = "Hello World Marco"
    End Function

End Class
```

4.) Cambiar el nombre del archivo “Service1.asmx” a “mensaje1.asmx” a través del solution Explorer.

Antes:



Después:

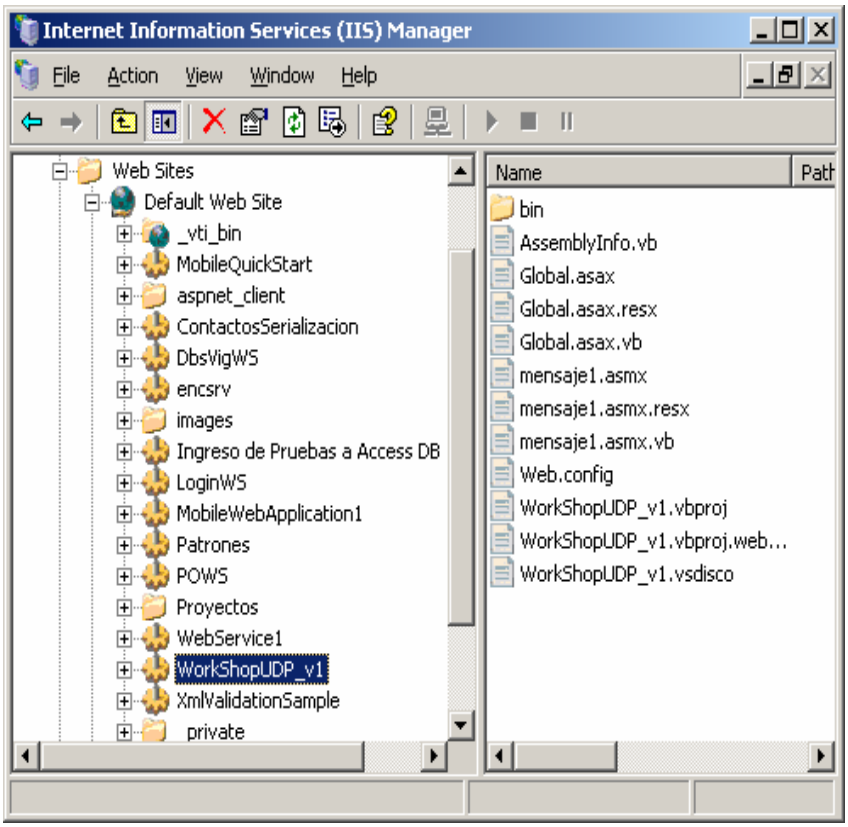


5.) Construir la solución.

Ejecutar: “Build Solution”

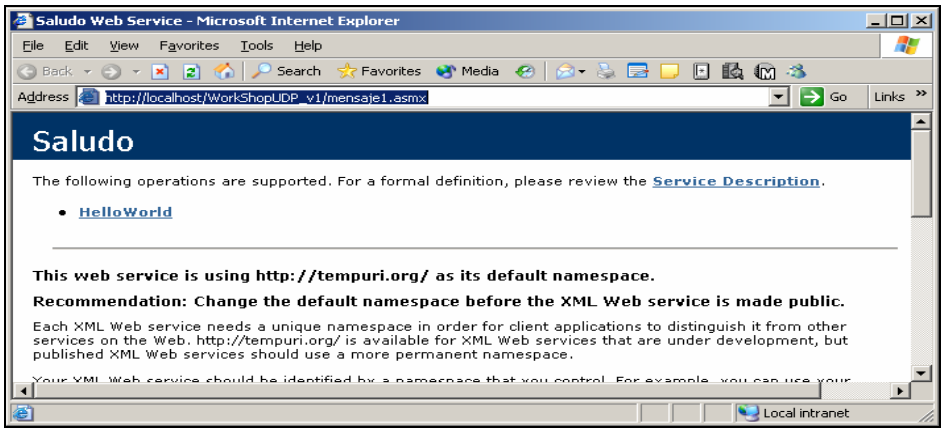
Menu: Build → “Build Solution”, o bien, “Ctrl+Shift+B”

Verificar en IIS que en “Default Web Site” está el sitio http://localhost/WorkShopUDP_v1

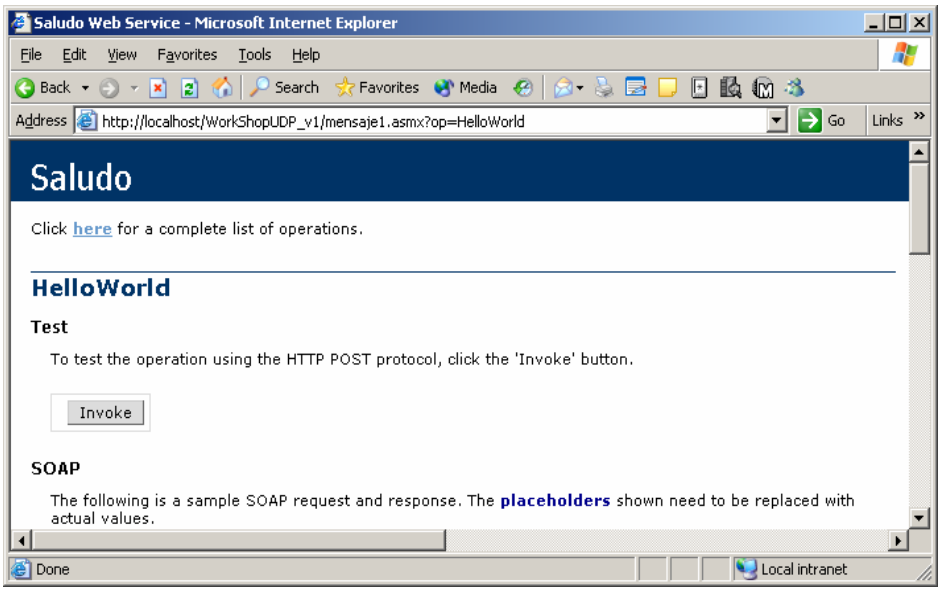


A.2 Probando el Servicio Web desde el browser.

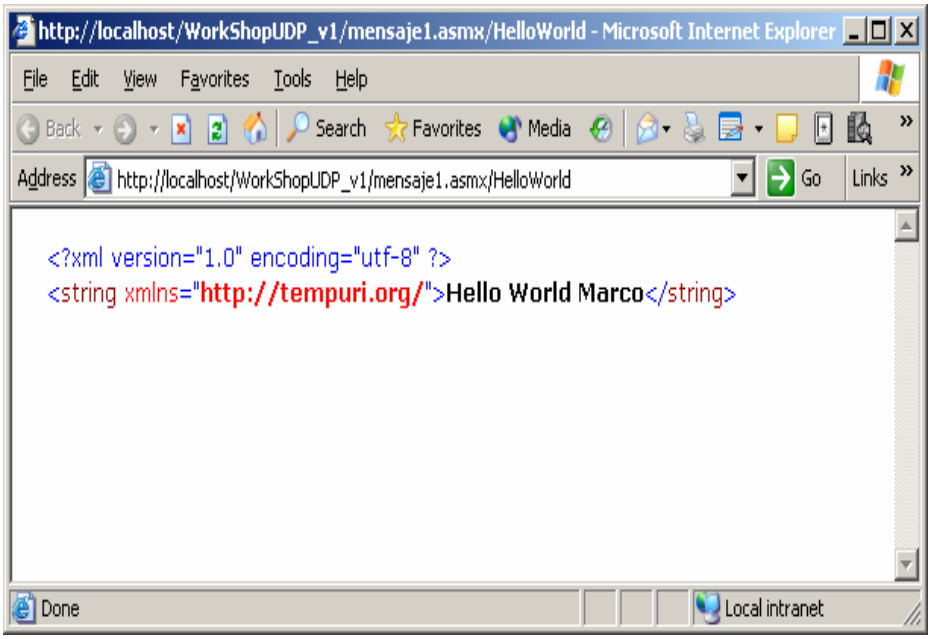
En el navegador, abrir la dirección:
http://localhost/WorkShopUDP_v1/mensaje1.asmx



Clic sobre “HelloWorld”



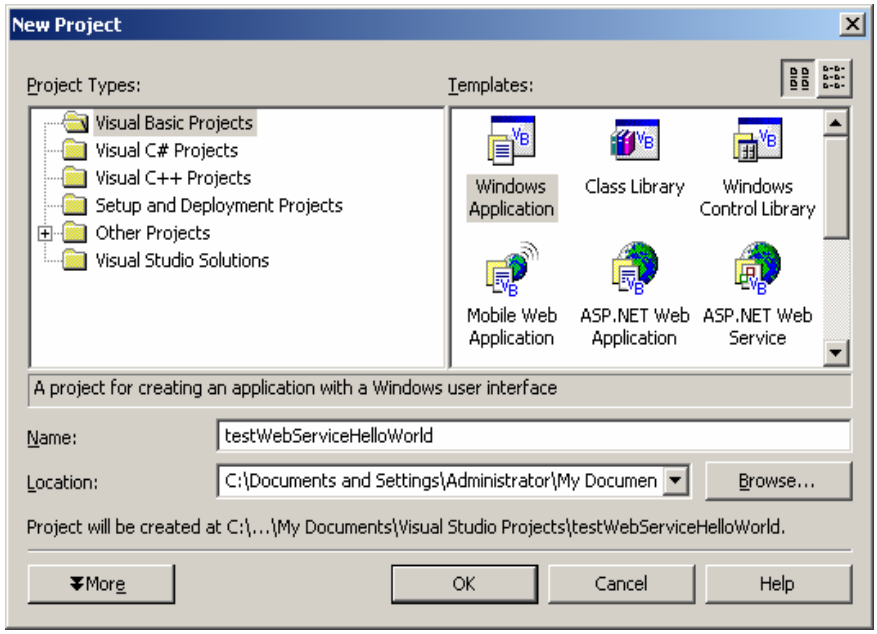
Clic en “Invoke”



A.2.1 Consumo del Servicio Web.

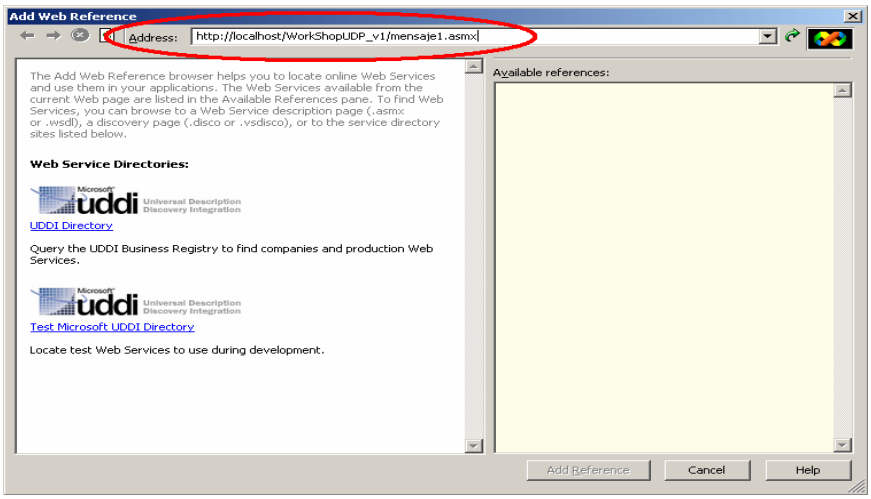
A.2.1.1 Consumo desde una Aplicación .NET Windows.

1) **Crear proyecto de Aplicación Windows llamado "testWeb serviceHelloWorld"**

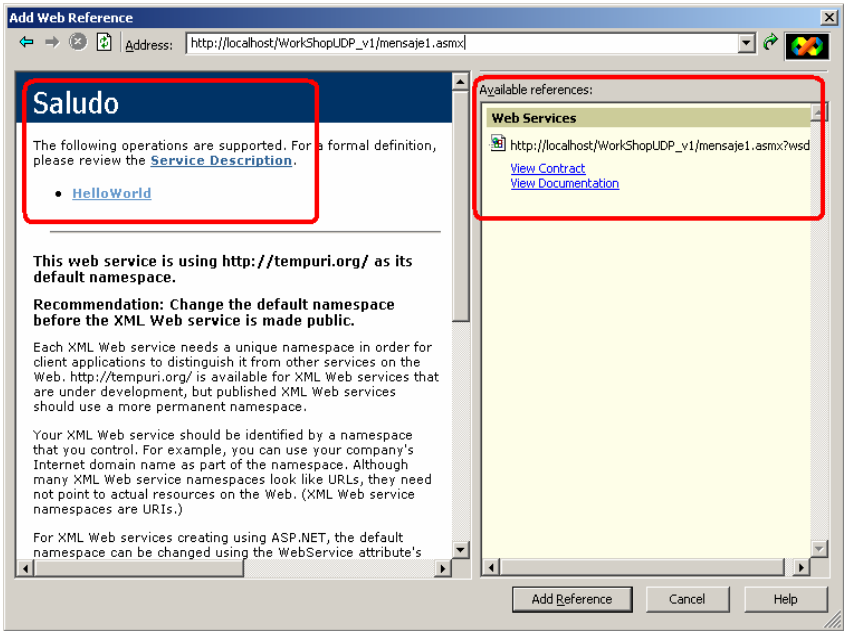


2) Agregamos referencia al proyecto Servicio Web: En el "Solution Explorer" pulsar botón derecho sobre "Reference" y "Add Web Reference"
En la barra de direcciones de la ventana, agregar la dirección del Servicio Web creado.

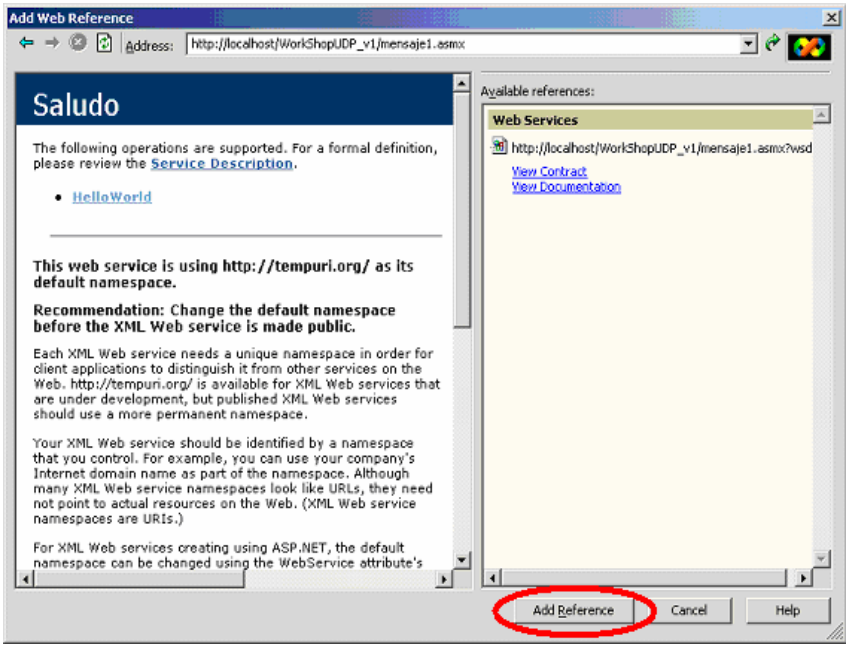
http://localhost/WorkShopUDP_v1/mensaje1.asmx



Pulsando <ENTER>, comprobamos la existencia del Servicio Web en la dirección ingresada.

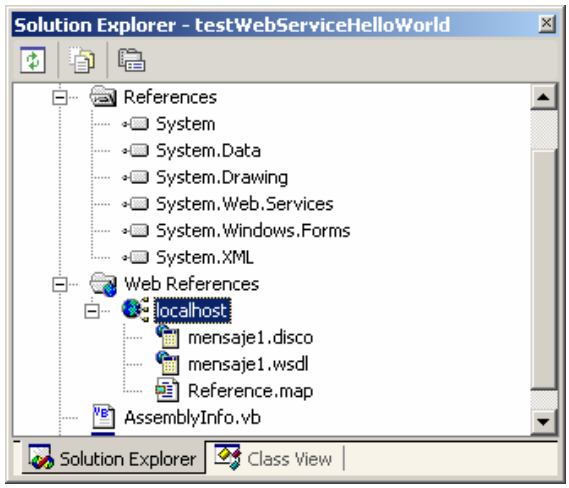


Agregamos la referencia al proyecto: Clic en “Add Reference”

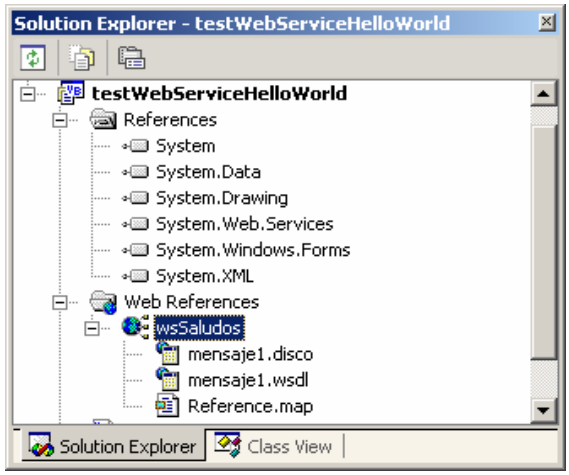


Comprobar en el “Solution Explorer” la referencia agregada, y cambiar el nombre de la carpeta “localhost” a “wsSaludos”

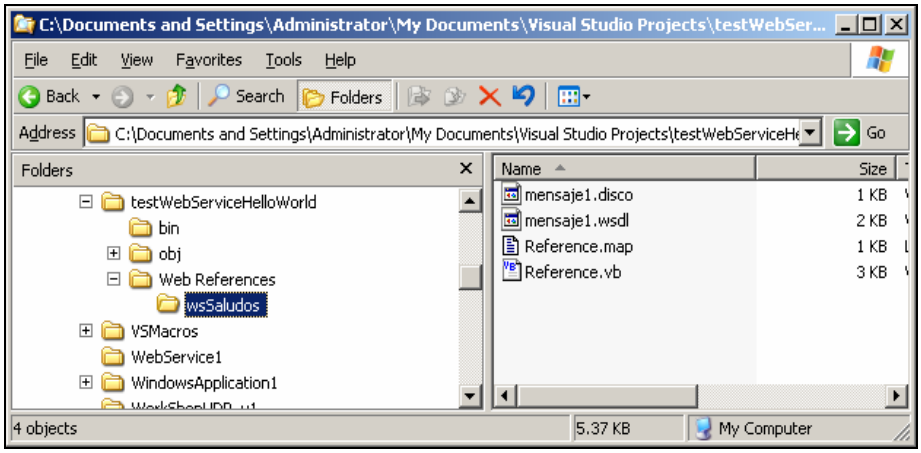
Antes:



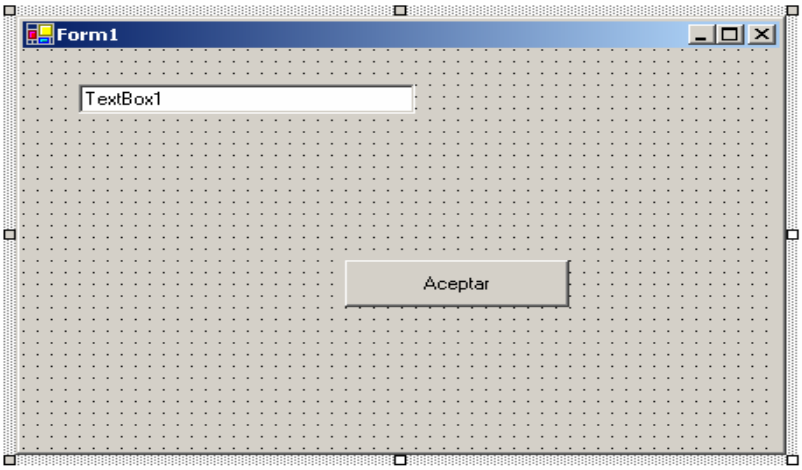
Después:



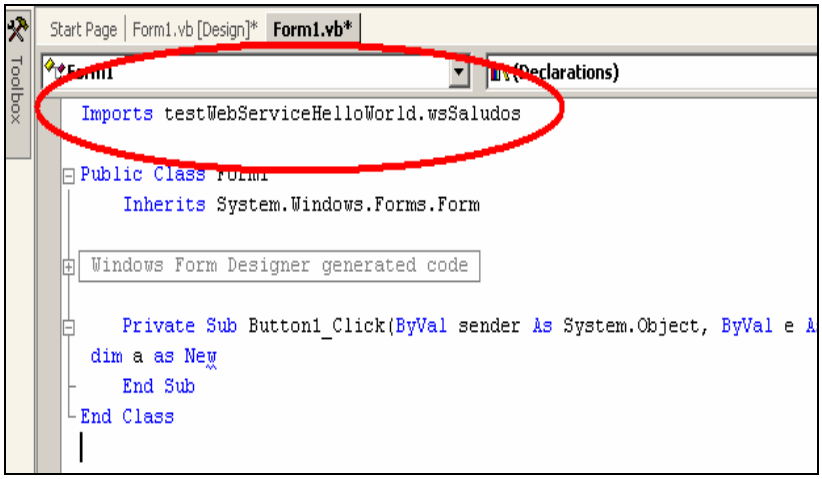
Comprobamos a través del explorador de Windows la existencia de los archivos “mensaje1.disco” y “mensaje1.wsdl” existen en el directorio “wsSaludos”



3.) Insertar un botón y un cuadro de texto al formulario



4.) En el código del formulario, importamos en espacio de nombres asociado a la referencia al Web service agregada.



5.) En el código de acción del botón, instanciamos un objeto de la clase “testWeb serviceHelloWorld.wsSaludos”, llamado “objWsSaludos”.

```
Dim objWsSaludo As New Saludo()
```

6.) Luego llamamos el método “HelloWorld” y asignamos su respuesta al TextBox1.

```
TextBox1.Text = objWsSaludo.HelloWorld()
```

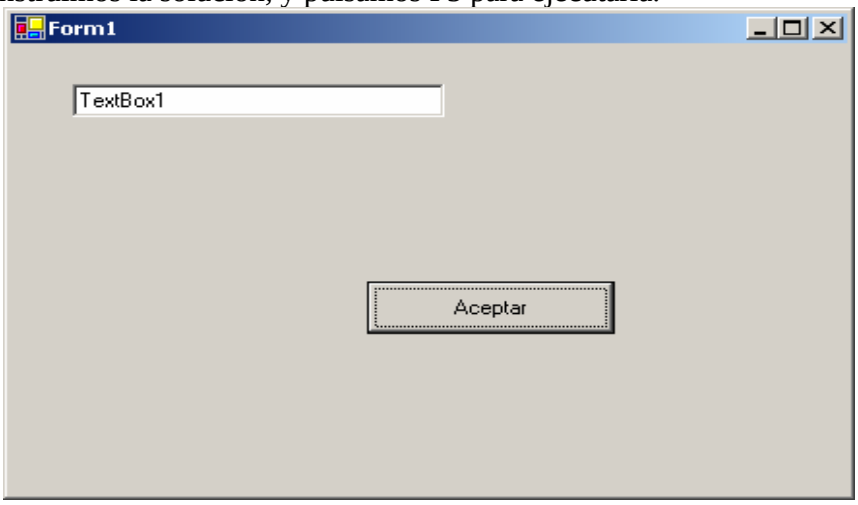
Finalmente el código queda como:

```
Imports testWeb serviceHelloWorld.wsSaludos
```

```
Public Class Form1
   Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

   Public Sub New()
      MyBase.New()

      'This call is required by the Windows Form Designer.
      InitializeComponent()

      'Add any initialization after the InitializeComponent() call

   End Sub

   'Form overrides dispose to clean up the component list.
   Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
      If disposing Then
         If Not (components Is Nothing) Then
            components.Dispose()
         End If
      End If
      MyBase.Dispose(disposing)
   End Sub

   'Required by the Windows Form Designer
   Private components As System.ComponentModel.IContainer

   'NOTE: The following procedure is required by the Windows Form Designer
   'It can be modified using the Windows Form Designer.
   'Do not modify it using the code editor.
   Friend WithEvents Button1 As System.Windows.Forms.Button
   Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
   <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
      Me.Button1 = New System.Windows.Forms.Button()
      Me.TextBox1 = New System.Windows.Forms.TextBox()
      Me.SuspendLayout()
      '
      'Button1
      '
      Me.Button1.Location = New System.Drawing.Point(184, 144)
      Me.Button1.Name = "Button1"
      Me.Button1.Size = New System.Drawing.Size(128, 32)
      Me.Button1.TabIndex = 0
      Me.Button1.Text = "Aceptar"
      '
      'TextBox1
      '
      Me.TextBox1.Location = New System.Drawing.Point(32, 24)
      Me.TextBox1.Name = "TextBox1"
      Me.TextBox1.Size = New System.Drawing.Size(192, 20)
      Me.TextBox1.TabIndex = 1
      Me.TextBox1.Text = "TextBox1"
      '
      'Form1
      '
      Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
      Me.ClientSize = New System.Drawing.Size(432, 273)
      Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.TextBox1,
Me.Button1})
      Me.Name = "Form1"
```

```
        Me.Text = "Form1"
        Me.ResumeLayout(False)

    End Sub

#End Region

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim objWsSaludo As New Saludo()
        TextBox1.Text = objWsSaludo.HelloWorld()
    End Sub
End Class
```

7.) Construimos la solución, y pulsamos F5 para ejecutarla.



Al pulsar el botón, se invoca el Web service y se asigna el resultado al cuadro de texto.